RWTH Aachen University
Department of Computer Science
Dependable Distributed Systems Group

Diploma Thesis
in Computer Science

Towards Automating Analysis in Computer Forensics

Bastian Schwittay

August 23, 2006

First Examiner:
Prof. Dr. Felix Freiling

Second Examiner:
Prof. Dr.-Ing. Heiko Mantel

External Supervisor:
Olaf Lindner, Symantec Corporation

Erklärung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

August 23, 2006

Abstract

In the recent past, the number of attacks on computer systems that are motivated by profit has increased significantly, and such “cybercrime” is expected to dominate the threat landscape in the future. With criminal acts becoming more common in computer related incidents, the need for Computer Forensic experts to provide admissible evidence for these crimes also rises. Since automation of certain tasks will be inevitable to be able to deal with the increased demand for forensic analysis, this diploma thesis is concerned with possible methodologies for conducting automated forensic analysis. By defining a new process model for forensic investigations that incorporates Computer Forensics into an Incident Response framework, commonly used tools and Best Practices are presented. In order to further establish Live Response as an essential part of a forensic investigation, critical flaws in current methodology are discussed and improved techniques are proposed, which help to increase the reliability of automated Live Response and therefore of the whole investigative process. In this context a methodology for conducting rootkit detection on Microsoft Windows systems is developed, which is then integrated into the Live Response process. Altogether, this thesis will hopefully help to better understand the overall capabilities of automated forensic analysis methods and their limitations.

Zusammenfassung

Durch die ständig steigende Anzahl von Angriffen auf Computersysteme, die durch Profit motiviert sind, steigt auch der Bedarf an gerichtsverwertbaren Beweisen, um diese Vergehen erfolgreich aufzuklären. Die Computerforensik beschäftigt sich mit der Gewinnung solcher Beweise – und ist dabei zunehmend auf die Automatisierung von Arbeitsabläufen angewiesen, um der steigenden Zahl von Fällen Herr zu werden. Diese Diplomarbeit beschäftigt sich mit der Möglichkeit der Automatisierung von Teilen des Forensischen Prozesses, wobei sowohl bekannte als auch neue Techniken und Tools zur Automatisierung untersucht werden sollen. Diese Techniken werden anhand eines neuen Modells vorgestellt, das die Vorteile von Computerforensik und Incident Response, der wohldefinierten Reaktion auf sicherheitsrelevante Vorfälle, vereint. Dabei ist die Verlässlichkeit der Ergebnisse ein wesentlicher Punkt bei der Analyse der angewandten Methoden, daher werden Schwächen verbreiteter Tools und Methoden diskutiert und, falls möglich, Verbesserungsvorschläge eingebracht. Insbesondere wird hierbei die Durchführung der Live Response, der Datensammlung am laufenden Rechner, untersucht und diesem Zusammenhang eine Methodologie zur Erkennung von Rootkits für Microsoft Windows-Betriebssysteme implementiert. Durch Validierung soll außerdem die Glaubwürdigkeit automatisierter Analysen gestärkt werden und eine bessere Einschätzung der Leistungsfähigkeit und Grenzen solcher Methoden ermöglicht werden.

Contents

1 Introduction **9**

1.1 Motivation . . . 9

1.2 Problem Statement . . . 9

1.3 Results . . . 10

1.4 Outline . . . 10

1.5 Final Remarks . . . 11

2 Background **12**

2.1 Introduction . . . 12

2.2 Incident Response . . . 12

2.3 Computer Forensics . . . 16

2.4 Comparison and Discussion . . . 19

3 Common Process Model for Incident Response and Computer Forensics **21**

3.1 Introduction . . . 21

3.2 Common Process Model for Incident Response and Computer Forensics . . . 22

3.2.1 Phases of the Common Model . . . 22

3.2.2 Deliberating a full-scale forensic analysis . . . 23

3.3 Pre-Analysis Phase . . . 26

3.3.1 Pre-incident Preparation . . . 26

3.3.2 Incident Detection . . . 29

3.3.3 Initial Response . . . 30

3.3.4 Formulation of Response Strategy . . . 31

3.4 Analysis Phase . . . 32

3.4.1 Live Response . . . 33

3.4.2 Forensic Duplication . . . 34

3.4.3 Data Recovery . . . 39

3.4.4 Harvesting . . . 44

3.4.5 Reduction and Organization . . . 46

3.4.6 Analysis . . . 48

3.5 Post-Analysis Phase . . . 51

3.5.1 Report . . . 52

3.5.2 Resolution . . . 53

3.6 Discussion . . . 54

3.7 Summary 56

4 Live Response **57**
4.1 Introduction 57
4.1.1 Motivation 57
4.1.2 Live Response considerations 58
4.2 Best Practices and Standard Procedures 58
4.2.1 Live Response on Unix-based operating systems 58
4.2.2 Windows Live Response 63
4.3 Forensic Flaws in Current Practice 68
4.3.1 File metadata collection 68
4.3.2 Rootkit Detection 69
4.4 Improvements 70
4.4.1 File metadata collection 70
4.4.2 Rootkit Detection 71
4.5 Discussion 72
4.6 Summary 73

5 Windows Rootkits **75**
5.1 Introduction 75
5.2 Background: Windows Kernel and Rootkits 75
5.2.1 Windows Internals 76
5.2.2 Hooking Techniques 77
5.2.3 Direct Kernel Object Manipulation 79
5.3 Overview of Rootkits and Detectors 80
5.3.1 Rootkits used 80
5.3.2 Detection tools used 81
5.4 Dependability Experiments 82
5.4.1 Setup 82
5.4.2 Execution 83
5.4.3 Results 85
5.4.4 Analysis 88
5.5 Rootkit Detection Methodology 88
5.6 Discussion 91
5.7 Summary 93

6 Conclusion and Future Work **94**

A Sourcecode **97**
A.1 getmetadata.pl 97
A.2 getsignatures.pl 98
A.3 unknowns.pl 98
A.4 ir-script-linux.sh 99
A.5 ir-script-windows.bat 102
A.6 getmetadata-live.pl 104
bibliography 105

Chapter 1

Introduction

1.1 Motivation

In recent years, there have been significant changes in the field of computer security. Several new threats such as bot networks and modular malicious code have emerged and have become very popular among attackers. Well-known attack tools such as rootkits and trojans remain dangerous by constantly using new and improved techniques. Attackers have moved toward small and focused attacks, often targeting Web browsers or other client-side components. Since not all of these attacks can be prevented, Incident Response (IR) has become an important component of IT security management, because it provides procedures for detecting and containing computer security incidents to restore normal computing services as quickly as possible.

In addition to this, attacks and malicious code to commit *cybercrime* — criminal acts that incorporate a computer or Internet component — are expected to dominate the threat landscape in the future [67]. Many attacks nowadays are already motivated by profit, attempting to achieve financial gain by identity theft, extortion, fraud, or theft of confidential information (see e.g. [84], [85], [86], [88]). With criminal acts becoming more common in computer related incidents, the need for valid evidence for these crimes also rises. The field of Computer Forensics aims at providing such evidence.

1.2 Problem Statement

Due to the increasing number of cases, forensic analysis increasingly depends on *automation* to achieve results in a reasonable time. Another key challenge that computer forensics face concerns the *credibility* of an analysis. Often the credibility of an analysis is mostly based on the credibility of the expert conducting the analysis, rather than relating credibility to the methodology applied to investigate the incident. Therefore, in this diploma thesis, possible methodologies for conducting forensic analysis will be investigated, to study in which aspects forensic analysis can be automated while still yielding valid and reliable results.

1.3 Results

As a first result of comparing the processes of Incident Response and Computer Forensics respectively, a Common Model for Incident Response and Computer Forensics has been developed, which integrates forensic analysis practices into an Incident Response procedure. *Attacker threat level* and *potential damage* of an incident have been identified as major factors when planning the analysis phase of an investigation and it is shown how to evaluate them in order to develop a sensible response strategy. After introducing the Common Model, each step is described in detail and up to date techniques and tools to achieve the respective tasks are presented.

The "Live Response" step was then scrutinized in even more detail, focussing on the dependability and credibility of common automated tools and methods for analysis. This resulted in the detection of flaws in current methodology; e.g. when collecting file system metadata from a running Unix system using current techniques and tools, the timestamps of files and directories are updated, thus altering the original evidence in an undesirable way. To address these problems, improved techniques are proposed where possible, resulting in a more sound methodology for conducting Live Response during an investigation.

The third and last result is a methodology which allows detection of most publicly available rootkits as part of a Live Response process. After introducing the basic concepts of Windows rootkits and rootkit detection, an overview of currently available rootkits and detection tools is given, detailing 11 different rootkits and 12 currently available detection tools. By experimenting with the rootkits and detection tools, it could be shown that using a combination of 3 different detection tools, almost 100% of all rootkit installations can be detected.

1.4 Outline

In chapter 2, an introduction to both *Incident Response* and *Computer Forensics* will be given, and their similarities and differences will be pointed out. After this comparison, a common model for Incident Response and Computer Forensics will be proposed in chapter 3. By identifying the determining factors and parameters of the Incident Response process and the Computer Forensics process respectively, it will be shown how both of them can be combined into a single process model, flexibly describing an appropriate response to computer related security incidents. The different steps of the common process will then be described in detail, presenting methods and tools commonly used to fulfill the respective tasks, and discussing each step's significance within the new common model.

Chapter 4 will describe the step "Live Response" in even greater detail, trying to establish Live Response as a essential part of Computer Forensics. It will specifically focus on the main question of automating the analysis and validating the results. Techniques and tools commonly used in Live Response will be investigated to determine their strengths and weaknesses, and to explore the reliability of their results. Whenever possible improved techniques for Live Response will be proposed, resulting in a recommendation for a more reliable way of conducting Live Response using automated methods where applicable.

The growing threat of *rootkits*(see e.g. [83, 87]) will be studied in chapter 5. After giving some background information about the basics of current rootkit technology and the underlying concepts, several rootkits and rootkit detection tools will be presented. The detection tools will then be put to the test in experiments on real rootkit infected Windows computers, to see how they perform when facing different rootkits and operation system version. Ultimately, a methodology for rootkit detection as part of a Live Response will be proposed based on the results from the experiments.

The last chapter will summarize the whole thesis and its results, while also bringing up some open questions which could be dealt with in future work within the area.

1.5 Final Remarks

Throughout this thesis, a lot of different tools used in Computer Forensics will be introduced. Open-source tools will be preferred whenever possible, because they are easily obtainable and offer certain benefits over closed source or commercial software (see e.g. [63]), in particular they are free of charge. A lot of the tools used are part of The Sleuthkit [57], which is the leading open-source forensic toolkit. Whenever a command line tool is used, its name will be written in `verbatim` font, which is also used to mark the output of console commands. The sourcecode of certain perl scripts taken from [75] has been added in the appendix.

Chapter 2

Background

2.1 Introduction

This chapter will give an introduction to *Incident Response (IR)* and *Computer Forensics (CF)*, two main concepts of handling and investigating computer security incidents. A *computer security incident* can be defined as "a violation or imminent threat of violation of computer security policies, acceptable use policies, or standard security practices" [71]. Incident Response deals with computer security incidents in a well-defined manner to detect incidents, minimize the damage done to the organization, fix the weaknesses that were exploited and return to normal operations.

Computer Forensics is a scientific discipline which is concerned with the collection, analysis and interpretation of digital data connected to a computer security incident; it is sometimes also called *digital forensics*. Since any device, data or other resource subject to a forensic examination may be subsequently used as evidence in a court of law, Computer Forensics puts special emphasis on the correct treatment of potential evidence to prevent it from being altered or tampered with. Because of the same reason, any technique or tool used during an investigation has to meet strict standards, and any conclusions drawn must be scientifically reasonable. It is important to note that an Incident Response procedure can sometimes *include* a full forensic investigation, the conditions will be discussed in the next chapter.

The following sections will thus explain the process of IR and CF respectively in more detail in order to point out the basic differences and overlaps. Both sections will describe the process on a rather abstract level, without referring to implementation of certain steps or specific software to be used. This will be postponed until the presentation of the common model in chapter 3. Large part of this chapter is based on [78] and [66], so theses sources will only be cited if absolutely necessary.

2.2 Incident Response

As mentioned in the introduction, Incident Reponse is an organization's reaction to unlawful or unacceptable actions involving a computer or network compo-

nent. Instead of being caught unprepared and starting a chaotic and possibly devastating response, a systematic and well-organized approach should be used to react. Therefore incidents are usually handled by a so-called *Computer Security Incident Response Team*, or *CSIRT*, which is comprised of personnel with different qualifications that are needed during the response process, in particular people with legal and technical expertise. The CSIRT will then coordinate the appropriate response to an incident, effectively providing a *Computer Security Incident Response Capability (CSIRC).*
Some of the key benefits that an organized CSIRC offers are:

- Confirmation whether an incident actually occured
- Quick detection, containment and recovery
- Minimizing loss or theft of confidential information
- Ensuring proper handling of evidence and documentation
- Increasing protection against future attacks

Achieving all this can seem like an overwhelming task, especially when one considers that often incidents put a lot of pressure on the team responsible for the resolution of the problem, which could for example threaten the very existence of a company. Another characteristic of computer security incidents is their wide variety, which includes, but is not limited to:

- Denial of Service attacks
- Theft of confidential information
- Unauthorized access
- Extortion
- Identity Theft
- Spam or harassment
- Possession or distribution of child pornography

To be able to manage these different kinds of incidents and their unique complications regarding public law, business operations or even a company's reputation, the process of Incident Response has traditionally been broken down into a number of logical steps. The following model has been proposed in [78], which is a major reference in the field of Incident Response.

The methodology describes the process of Incident Response using seven different phases, which are also illustrated in Figure 1 on page 14. A short summary of each step will be given, more details can be found in chapter 3.

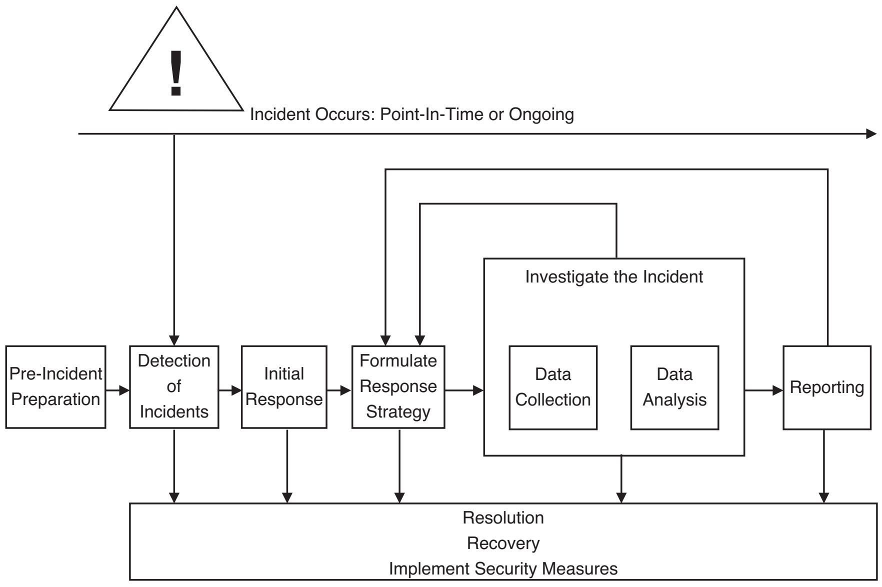


Figure 1: The Incident Response Process, taken from [78]

Pre-incident preparation

Pre-incident preparation is an ongoing phase, and it is actually the only one that takes place even before an incident occurs. Its purpose is to prepare the organization and the CSIRT for a possible incident. Preparing the organizion may involve implementing host- and network-based security measures, e.g. an Intrusion Detection System. To make sure that a CSIRT is able to handle an incident well, all hardware and software needed have to be provided. Appropriate training for the members of the CSIRT is mandatory to maintain an effective Incident Response Capability.

Detection of incidents

Since obviously no incident can be responded to until it is detected first, detection of incidents is a very important part of IR. Generally speaking, anyone from a system administrator to a regular employee with no technical background could detect an incident. That is why it is important to have clear guidelines for all members of an organization explaining how to react to suspicions that something might be wrong. Most importantly it should be known who should be notified and what details have to be recorded, so that the CSIRT may take over the IR process as quickly as possible to further investigate the potential incident.

Initial Response

During Initial Response, the collection of information concerning the incident that has started in the previous phase continues. The goal is to gather enough information to allow formulation of an adequate response strategy in the next step. Typically, the data that is collected in this step consists of interviews of any persons involved in reporting the suspected incident, and available network surveillance logs or IDS reports, that may indicate that an incident occured. At the end of this phase, the CSIRT should have confirmed that an incident actually occured, should have identified the type of incident and the affected hosts and users, and should be able to assess the potential impact or damage. This will allow to make an informed decision about a viable response strategy.

Formulation of response strategy

The goal of this phase is to formulate a response strategy which best fits the situation, thus "considering the totality of the circumstances" [78] that surround the incident. These circumstances include the criticality of the affected systems or data, what kind of attacker is suspected and what the overall damage might amount to. Also, an organization's *response posture*, which defines its policy regarding the response to computer security incidents, may have a large impact on the choice of a response strategy. Because an incident might also induce legal or administrative action, the development of the strategy has to involve people responsible for the respective areas, which means e.g. upper management executives or the Human Resources department.

Investigation of the incident

During the investigation of the incident, different types of evidence relevant to the incident, e.g. host- or network-based evidence, are collected in order to reconstruct the events that comprise the computer security incident. This reconstruction should provide explanations for what happened, when, how or why it happened and who is responsible. To achieve this, an investigation is typically divided into two steps: *Data Collection* and *Data Analysis*.

Examples of data collected during the Data Collection are host-based information retrieved from a live system, a duplication of a compromised host's harddrive or network surveillance logs. While collecting these information it is advisable to use forensically sound methods; this will be detailed later in this thesis.

In the Data Analysis step, the previously collected information is reviewed to uncover the details behind the incident. A thorough discussion of the methodology applied during analysis will follow in the section on Computer Forensics, and in even more detail in chapter 3. It should be noted that often the collection and analysis of relevant data may take turns, effectively turning the investigation into a series of feedback loops until the final result is obtained.

Reporting

After the investigation of a computer security incident is finished, all the findings and results have to be documented in a written report. In this report, all investigative activities have to be documented, and all conclusions drawn have to be explained. The report should be written in a concise, yet understandable way, so that even non technical readers can follow. Because the results of the report might be used as evidence during a lawsuit, the report should be able to hold up against legal scrutiny.

Resolution

The purpose of the Resolution phase is to take the right measures to contain an incident, solve the underlying problems that caused the incident and to take care that a similar incident will not occur again. All the necessary steps performed should be taken and their progress supervised to verify that they are effective. It is important that changes to the affected systems are only done after collecting possible evidence, otherwise that evidence might be lost. After the resolution of the incident is complete, it may be necessary to update security policies or the IR procedures, if the response to the incident exposed a weakness in the current Practices.

Most of the steps above will be included in the model proposed in chapter 3. The next section will introduce a Computer Forensics process model, and the common features of both will become clear.

2.3 Computer Forensics

Computer Forensics, or Digital Forensics, is a forensic science that deals with obtaining, analyzing and presenting *digital evidence*, which can be defined as “any data stored or transmitted using a computer that support or refute a theory of how an offense occured or that address critical elements of the offense such as intent or alibi.’ ’[66]. By employing accepted and proven techniques and principles, which are also applied in other forensic sciences, admissibility in a court of law and credibility of the evidence is achieved. This includes for example a high level of objectivity in every step of an investigation, and the use of reliable, repeatable and well-documented methods throughout the examination.

The Investigative Process Model [66] (see Figure 2 on page 17) which will be described now is in fact a more general approach to investigation of a computer-related offense, as it also includes steps which normally tend to fall into the responsibility of law enforcement personnel rather than the forensic analyst. Nevertheless it is a good choice for the purposes of describing an organized CF process and it will allow to show the similarities and differences between the CF process and the IR process described earlier in this chapter.

Just like in the previous section, each step in the process model will now be explained in an abstract way, this time pointing out the similarities and overlaps with the steps of the IR process model.

Incident alerts or accusation

The CF process starts with an incident alert or an accusation which could be an automated alert from an IDS or a concerned citizen reporting possible criminal activity. At this point, an initial assessment of the reliability of the source of the

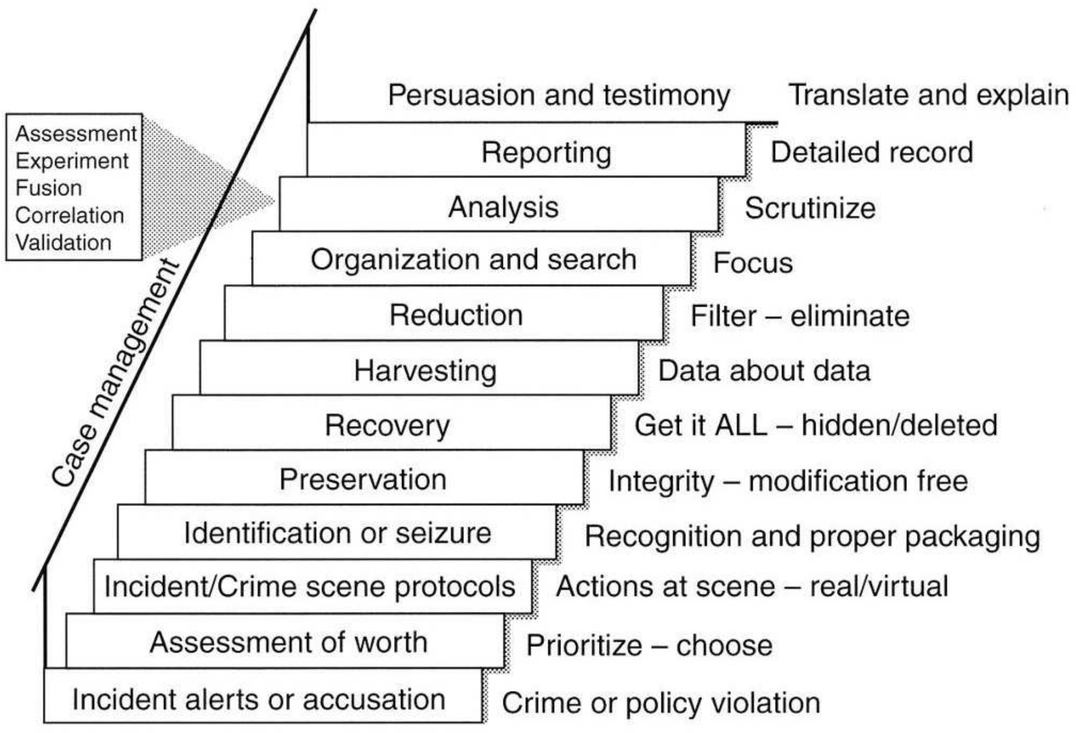


Figure 2: The Investigative Process Model, taken from [66]

alert is done, and some facts surrounding the suspected incident are gathered to get an idea of what the investigator is dealing with. Notice that there is no Pre-Incident Preparation Phase because an investigator is not necessarily working for the organization that reported the incident, but rather contacted when an incident is suspected.

Assessment of worth

During this phase, it is decided whether a detailed investigation should take place or if the suspicion of an incident that raised the alarm can be discarded. Generally this depends on how severe the problem appears to be, i.e. what the potential damage might be or whether the incident can be contained easily.

Incident/Crime scene protocols

If the decision in the previous step is to conduct a full investigation of the incident, all potential evidence at the crime scene has to be secured and documented using accepted protocols and procedures. The goal is to “freeze” the evidence in place and provide a “ground truth for all activities to follow” [66]. Actual analysis is not done at this point.

Identification or seizure

After securing the crime scene, all items that may contain potential evidence for a suspected incident have to be seized and a *chain of custody* for all evidence must be started.

There are numerous legal implications and restrictions to securing a crime scene and seizing material from a crime scene. Excellent resources regarding these two steps during an investigation are [92] and [91], which describe the recovery of digital evidence from law enforcement's point of view. This knowledge can also be valuable to a digital investigator and helping him to work together with the other parties involved in an investigation.

Preservation

After all potential evidence has been seized and is available to the digital investigator, duplicate copies of all data sources are made to ensure integrity of the original evidence. Actual analysis is only done on the copies, while the original evidence is securely stored and cataloged. This step is in fact the first step that is part of the actual digital forensic investigation, and it makes use of trusted and reliable tools to perform the duplication of evidence.

Recovery

The first step of a computer forensic analysis consists of recovering any data that may have been deleted, hidden, encrypted or is otherwise unavailable to the forensic investigator in its current state. The goal is to recover anything possible — "throwing out a large net" — so that the maximum of data is available for further analysis, hoping that it will contain valuable evidence.

Harvesting

During the Harvesting phase, metadata, i.e. data about data, are collected and used to organize the large amount of data that the last step produced. Often, files are grouped according to their filetype, or by their temporal relationship with respect to file timestamps. By slowly starting to make sense of all the available data, the reconstruction of events and the development of different hypotheses about what might have happened starts. The structured data that is the output of this phase represents the first step towards extracting the evidence out of all the data that was collected in the beginning.

Reduction

Since the volume of data under investigation may be very large, it is crucial for a forensic examiner to eliminate any unnecessary data in order to focus on the more important pieces of information. Still, this does not mean that evidence is reviewed based on content, but rather that data is eliminated based on filetype or by using hash databases of known good files. The desired result is "the smallest set of digital information that has the highest potential for containing data of probative value" [66].

Organization and search

Organization of the data extracted in the Reduction step helps to make the actual analysis easier and can also simplify referencing to specific data during the following phases. A searchable index is often used to allow efficient retrieval of data and thus speed up the analysis.

Analysis

In the Analysis step, all the data that has been prepared in the steps before will be analyzed in detail, this time also incorporating the actual content, e.g. the content of text files. Finding out how and why an offense occured is investigated by establishing links between different pieces of evidence, ultimately trying to identify the offender and offer comprehensive proof that supports the conclusion. With standard scientific methods such as showing correlation between certain events, experimenting with the data and rigorously validating the results, an investigator can present digital evidence that can be relied upon even under high standards such as that of a court of law.

Reporting

The final report should accurately document each step of the investigation, back up any conclusions drawn during the examination with evidence and report on the methods used to obtain the evidence. Whenever possible, accepted and known protocols and methods applied during the analysis should be referenced to increase the credibility of the investigation and its results.

Persuasion and testimony

Sometimes it may be necessary for an examiner to testify in court or to answer to decision makers regarding the results of his report. Trying to explain the details of an investigation to a mostly non-technical audience can be difficult and therefore special techniques exist that allow an expert to do so in an understandable fashion.

The investigative process described above is a way of obtaining evidence in a reliable, accurate and repeatable way, basing all conclusions and findings upon facts and scientific reasoning. It will be shown how this strict scientific methodology used during the forensic analysis steps can be integrated into an Incident Response process and how it can improve the overall process.

2.4 Comparison and Discussion

In this chapter, two models for Incident Response and Computer Forensics respectively have been introduced and their different steps and procedures have been explained. There are several obvious differences between both approaches, since both emphasize different objectives when responding and investigating a computer incident or offense.

The Incident Response process clearly focusses on management issues and integration of the investigative process into a business environment. For this reason, it has to take into account things such as a possible hit to an organization's reputation and weighing off the monetary cost of an investigation against the potential damage the incident may cause. When proposing a common model for Incident Response and Computer Forensics in the next chapter, the management issues will be addressed in a similar way as in the Incident Response model described here. The different emphasis is also reflected in the way a response strategy is developed in the IR model, compared to the Assessment of Worth step in the CF model. While in the case of IR, there is a wide variety of possible response strategies, in the case of the Investigative Process the question at this point will simply be if it is necessary to assign additional resources to the case to investigate further, or if the investigation can be stopped altogether.

Another striking discrepancy between the IR and CF model is, that all the steps starting from Preservation up to the Analysis step in the CF process model correspond to just one step in the IR process model, the Investigation phase. This is because during Incident Response, the actual investigation can take on various forms or might also be left out altogether in some situations. Computer Forensics investigator on the other hand will always stress the importance of scientific and forensic methods during an investigation in order to obtain evidence that is admissible, i.e. that it can be presented in a court of law. For that reason the analysis steps are clearly seperated and structured in a subtle way to allow a systematic approach to an investigation. It is mandatory to use the scientific principle when drawing conclusions from the evidence, which means that not only has to be reconstructed what has happended, but also it has to be shown why other explanations can be ruled out; this practice is also known as *falsification.* The common model for IR and CF will incorporate these forensic principles and procedures into the investigation step that was described rather loosely in the IR model.

Chapter 3

Common Process Model for Incident Response and Computer Forensics

3.1 Introduction

In this chapter a Common Process Model for Incident Response and Computer Forensics will be proposed, which is based upon the two process models introduced in the previous chapter. The motivation for creating a Common Model (CM) that, in a way, merges two already existing processes was twofold.

On the one hand, Incident Response and Computer Forensics are two highly related topics, and it is questionable whether a strict separation makes sense at all. Both investigate a large number of different computer security incidents or offenses, sometimes use identical tools and methods and also share some of the key phases of the investigation, although they may set different priorities. Secondly, the specific view of the investigative process in IR and CF respectively may sometimes be too narrow to achieve optimal results in either case. A Computer Forensic investigation may lack the proper management that is enforced in an IR investigation, where the investigative efforts are coordinated with all parts of an organization, e.g. legal counsel, Human Resources and business executives. Without these, the "bigger picture" of an incident might not be seen.

On the other hand, the scientific standards of a forensic investigation can give benefit to an Incident Response procedure, as it promotes objectivity throughout the whole process and a precise and well-documented analysis. Because the large effort to conduct a full-scale forensic investigation may not be necessary in every case, the Common Model will identify certain parameters that determine the extent of an investigation.

After the Common Model has been introduced, the different steps will be explained in detail, discussing their function, methods and specific techniques that are used today. In each subsection the particular step's significance with respect to the Common Model, as well as its relation to similar steps that were part of the Incident Response or Computer Forensics process models will be discussed.

3.2 Common Process Model for Incident Response and Computer Forensics

The Common Process Model for Incident and Computer Forensics is a proposal for a new process model to investigate computer security incidents, and its aim is to combine the two concepts of Incident Response and Computer Forensics to improve the overall process of investigation. In fact the Common Model somewhat resembles a Computer Forensic investigation which is embedded into an Incident Response procedure.

3.2.1 Phases of the Common Model

The Common Model consists of three main phases: Pre-Analysis, Analysis and Post-Analysis (see also Figure 3).

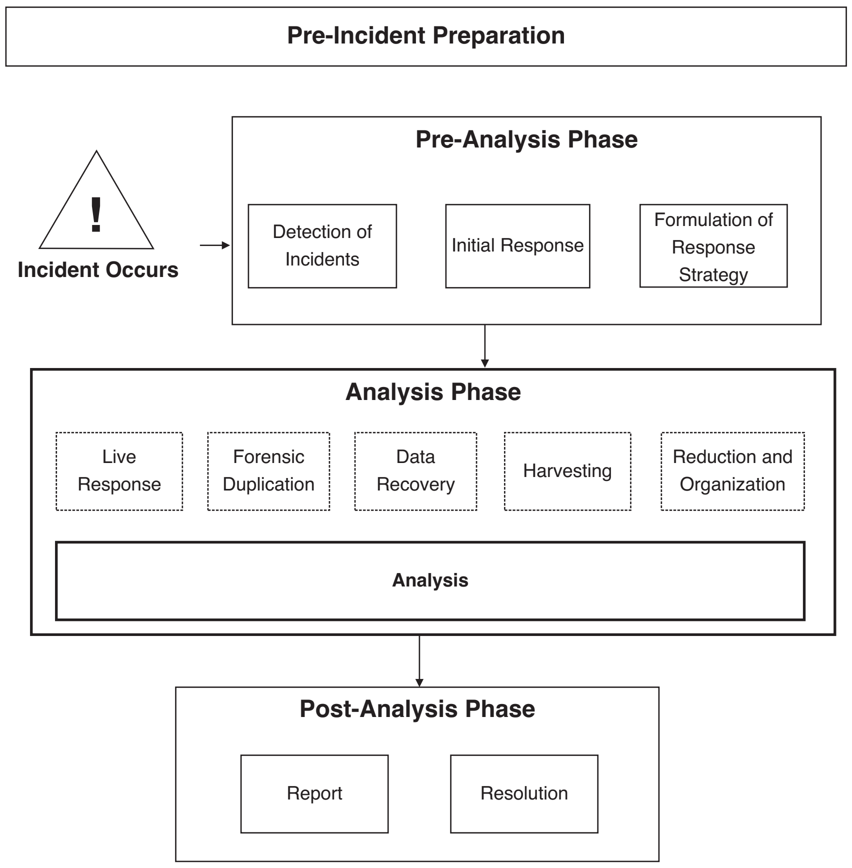


Figure 3: Common Process Model for Incident Reponse and Computer Forensics

Pre-Analysis Phase

The *Pre-Analysis Phase* contains all steps and activities that are performed before the actual analysis starts; in this case, analysis means that compromised hosts or data are reviewed in detail with the intention to reconstruct the reason for the computer security incident in question. Using this notion, a quick survey of the affected host during the *Initial Response* step does not qualify as actual analysis. The three steps of the Pre-Analysis Phase correspond to the respective steps in the Incident Response process model introduced in chapter 2, although the last step, *Formulation of Response Strategy*, differs slightly.

Analysis Phase

Actual analysis takes place in the *Analysis Phase*, which uses part of the Investigative Process Model's steps presented earlier in the Background chapter. The sequence of steps is based on the Computer Forensics process model, except for the step of *Live Response*, which did not appear explicitly in either of the two previous models. In short, Live Response means collecting information about an incident on hosts that are still running, i.e. “live” — as opposed to a “dead” analysis of devices after the host containing it has been powered down. Live Response has become increasinlgy popular over the last years because it offers certain benefits over a mere dead analysis; Chapter 4 will discuss Live Response in great detail.

Although the process model incorporates the steps that are usually done during a forensic investigation, such a full-scale approach might not be necessary in all cases. A response team will have to decide whether the cost and effort is justified in each specific situation, taking into account a lot of different factors, e.g. the severity of the incident or existing legal contraints. The determing factors which will influence this decision are explored in the section “Deliberating a full-scale forensic analysis”.

Post-Analysis Phase

Following the Analysis Phase, the *Post-Analysis Phase* is first of all concerned with documentation of the whole activities during the investigation in a written report; both classic models include this step. In addition to a written report, an examiner may also be asked to testify about the results of the analysis in a court of law, in which case he has to prepare properly to be credible and able to defend his work even under cross-examination. After the report has been completed, the incident has to be resolved by correcting the underlying problems that allowed the incident to happen; at the same time the incident handling procedures should be updated if necessary.

3.2.2 Deliberating a full-scale forensic analysis

After a computer security incident has been detected and some initial information concerning the incident has been collected, a suitable response strategy has to be chosen; this step is called *Formulation of Response Strategy*. In the case of the Incident Response process model, this simply means that after assessing the potential damage, criticality of the affected hosts and data and taking into account an organization's response posture, a strategy is formulated which seems to be the optimal way to resolve the incident. It has to be kept in mind, that a viable response strategy might in some cases be to just reinstall a compromised hosts's operating system, apply all patches and then resume normal operations again. Also, such a response strategy does not have to respect forensic principles, it may be perfectly adequate to disregard them altogether, depending on the particular incident.

In the Common Model, Formulation of Response Strategy includes an additional decision regarding the further investigation. In this step, an investigator or team of investigator has to decide, whether a *full-scale forensic analysis* is necessary. *Full-scale* means that the whole process of Live Response, Forensic Duplication up to the Analysis step as depicted in figure 3 has to be executed in full, without taking any shortcuts, and granting all the required resources to thourougly investigate the incident. *Forensic* means that in each step, forensic principles have to be respected, as introduced in the section on Computer Forensics.

It is clear that this type of investigation in general requires a lot of resources, because due diligence has to be invested for the analysis, and it may even cause downtime of its own, because hard drives have to be imaged, systems might have to be isolated or taken to the lab, and so on. This is why the decision to go for the full-scale forensic analysis should not be made lightly, but it should be made if the details of the incident in question call for it. The Common Model provides a comprehensive way to make the decision whether a full-scale forensic analysis is warranted or whether other response strategies may be more appropriate.

Determining factors

When planning analysis of a computer security incident, there are usually quite a lot of factors that might influence the decision about how to conduct the analysis. When deliberating the particular question of whether to conduct a full-scale forensic investigation, the Common Model focusses on two main factors, which will also be called "soft" factors, because they can only be estimated and depend largely on the properties of each particular incident:

- attacker threat level
- potential damage

The *attacker threat level* denotes the estimated threat of an attacker, which for this matter is any person who is directly responsible for the incident in question, whether it be by actively attacking a computer system or for example by violation of a security policy. Such a threat measure can be derived from a more detailed *attacker model*, which is a model of the attacker with his skills and intentions, thus allowing to assess the threat he poses for the organization under attack. In the Common Model, a high attacker threat level favors a full-scale forensic analysis, because a highly skilled attacker is likely to cover his tracks or even plant false clues. It may also be desirable to completely reconstruct the incident if a skilled attacker is involved, in order to find out what exactly the attacker did and how he did it; only a thorough forensic analysis can fulfill this requirement.

The *potential damage* of the incident under investigation is the second key factor to consider when considering the type of analysis to go for. Damage in this case means financial loss as well as loss of reputation or credibility, and since the actual damage of in incident is generally not known exactly in such an early state of the response process, it can only be approximated. Generally, a high potential damage will favor the decision for a full-scale forensic investigation, because in these cases, criminal prosecution or filing for civil complaint is a likely outcome. Hence, reliable and admissible evidence may be needed to fully resolve the incident.

With these two factors in mind, an abstract equation can be used to illustrate the decision:

$$AttackerThreatLevel \times PotentialDamage > Threshold$$

If this equation evaluates to true, a full-scale forensic analysis should be conducted. That means that the combination of a skilled attacker and high potential damage will influence the response strategy in a way that favors a forensic analysis, while in the case of an unskilled attacker or low potential damage, such an analysis is not justifiable.

Constraints

Apart from these soft factors, there are also two “hard” factors which are mostly independent from each other and from the soft factors, and which can call for a full-scale forensic investigation: an organization’s Response Posture and legal constraints.

A *Response Posture* describes an organization’s general stance towards responding to incidents. It contains general guidelines as well as prefered response procedures for a specific type of incident, e.g. what has to be done if confidential customer data has been compromised, or the organization’s web site has been defaced. It may also include recommendations regarding administrative actions should an employee violate a corporate policy, e.g. how to deal with theft of trade secrets or illicit internet use.

In some cases, a full-scale forensic analysis may be demanded explicitly by the Response Posture, for example because the organization pursues a “zero-tolerance” policy towards computer security incidents and will always try to get to the bottom of each case. In other cases a forensic investigation might be necessary because of the likely outcome of the investigation, in particular whenever the resolution could include administrative or legal action. In these cases admissible evidence is required, regardless of the attacker threat level or the potential damage of then incident, therefore it is a hard factor that can indicate the need for a full-scale forensic analysis.

Legal constraints can be another hard factor that will demand a full-scale forensic analysis of the incident. In some cases, an organization will choose to involve law enforcement and make a report against a suspected attacker. It may even be mandatory to report a suspected crime, e.g. the failure to notify the authorities about suspected possesion of child pornography may make an organization liable [78]. There have also been cases where the failure to objectively analyze an incident has led to negligence charges against the investigators [66]. Because of these possible complications, a well documented and objective approach to analysis of such incidents is necessary; that means a full-scale forensic analysis has to be conducted.

Summary

During the formulation of response strategy step in the Common Model, a decision has to be made about whether to conduct a full-scale forensic analysis of the incident. This decision will be influenced by two soft factors, the attacker threat level and the potential damage, and two hard factors, the Response posture and legal constraints. If the product of attacker threat level and potential damage is estimated to exceed an abstract threshold, a full-scale forensic analysis should be conducted. On top of that, an organization's Response Posture and certain legal constraints can call for a forensic analysis, independent of each other and the soft factors.
Having introduced the Common Model for Incident Response and Computer Forensics and the factors to consider when deliberating a full-scale forensic analysis, the specific steps of the Common Model will now be described in detail, pointing out their significance within the whole process as well as presenting current tools and methods that are commonly used.

3.3 Pre-Analysis Phase

Not surprisingly, the Pre-Analysis Phase (see figure 4) is comprised of all steps that are performed before the actual analysis of an incident starts. This includes Pre-incident Preparation, which is different from the rest of the steps because it is an ongoing phase, containing preparation procedures for a possible incident. Incident Detection is concerned with incident detection mechanism and procedures, and Initial Response deals with gathering initial information that allow to choose an appropriate response strategy in the Formulation of Response Strategy step.

3.3.1 Pre-incident Preparation

Goals

The goal of Pre-incident Preparation is to enable an organization to handle a computer security incident in a well-defined manner that allows quick and effective resolution. As mentioned before, Pre-incident Preparation is actually an ongoing phase where appropriate measures are taken to protect an organization

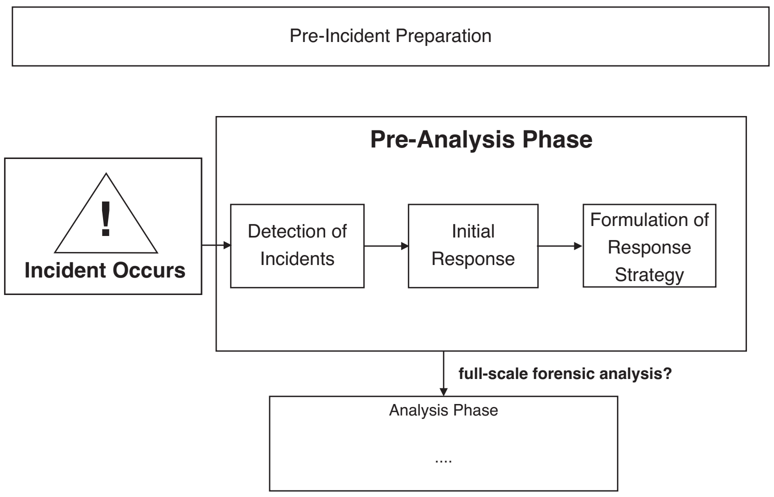


Figure 4: Pre-Analysis Phase of the Common Model

from the grave consequences of a possible incident. This includes strengthening the defenses of invididual hosts and networks, but since no perfect defense against incidents exists, Pre-incident Preparation also incorporates training incident responders and providing the technical infrastructure to pursue a successful investigation in the case of an incident. In addition to that, development of appropriate policies for incident response procedures has to be taken care of.

Methods, Tools and Best Practices

Usual measures taken during Pre-incident Preparation can be divided into two main categories: those that prepare the personnel that is responsible for responding to incidents, in particular the CSIRT, and those that concern preparation of the organization or environment that an incident may occur in.

All personnel involved in an incident's investigation have to be properly trained and equipped to be able to adequately respond to an incident. In addition to encouraging their personnel to voluntarily expand their knowledge, an organization should consider affording professional courses on Incident Response and related topics, as they are for example offered by SANS Institute [56], Foundstone [16] or Black Hat [3]. On top of that, all software and hardware needed for an investigation has to be up to date and ready when an incident actually occurs. Hardware that may be necessary in an investigation includes:

- analysis workstations with powerful CPU, large hard drives, DVD-RW, possibly tape drive
- extra connectors, cables, adapters for storage media
- forensic software dongles
- hardware write blocker
- blank CDs, DVDs, etc.

- digital camera
- toolkit
- flashlight
- folders, labels, markers, containers, paper, documentation forms

Regarding software, the following list can give an idea of what should be available and prepared in the case of an incident:

- several native operating systems on the analysis workstation, e.g. by using a multi-boot system
- forensic tools (Encase, FTK, TCT, Sleuthkit, etc.)
- boot disks
- software write blocker
- backup of complete setup to allow for quick recovery

For the preparation of an organization and the environment in which an incident may occur, classical host-based security measures like firewalls, organized backups or the use of strong authentication and encryption are mandatory. An excellent resource for host-based security measures is [69]. Complementing these, network-based monitoring and security mechanism should be implemented, using e.g. an Intrusion Detection System or similar methods. Network-based monitoring is described for example in [61]. With these precautions in place, a lot of incidents can be prevented, while others can be quickly and easily detected. Logging and monitoring can also help to collect valuable data while an incident is still ongoing and aid in the actual analysis steps. Regular vulnerability assessments and penetration tests should be performed to evaluate the quality of the security precautions in place.

Another aspect concerning the preparation of an organization for possible computer security incidents is the definition of proper policies. This includes both a general statement of an organization's response stance, called Response Posture, as well as policies and rules regarding the monitoring of end users or network traffic. It has to be assured, that a proper investigation can take place without breaching privacy rights or violating public law. So-called Acceptable Use Policies define what is considered an acceptable or unacceptable use of computer resources, and thus which behaviour can be considered a computer security incident. A good overview of the possible legal or business implications of an investigation can be found in [78], which stresses the importance of appropriate policies in an organization.

Summary

Pre-incident Preparation prepares an organization and its incident response personnel for investigation of computer security incidents or prevents them altogether. Enforcing a thorough preparation is a basic requirement for any organization that wants to build up an effective Incident Response Capability, thus making it a vital step to secure valuable data and computer systems.

3.3.2 Incident Detection

Goals

Incident Detection is about establishing proper incident detection guidelines, allowing for quick detection of computer security incidents. Proper notification and reporting procedures have to be in place, making it possible to transfer control over the remaining investigation to the CSIRT as soon as possible.

Methods, Tools and Best Practices

Incident Detection normally occurs whenever a person or security mechanism suspects an unauthorized or unlawful action involving a computer system or network. Actually suspicion of an incident can come from a lot of different sources, e.g.:

- End users
- Security personnel
- IDS
- System administrators
- Human Resources

This is why there have to be clear guidelines for everyone that may detect a possible incident, whom to contact and how to react in such a situation, so that no potential evidence is destroyed. If an incident is reported, some initial information concerning the detection should be recorded, including:

- Date and Time of detection
- Who is involved in the incident (by detection or otherwise)
- What kind of incident is suspected
- Which hosts/networks are involved

Ideally, appropriate forms for this checklist should be provided, to allow a standardized way of documenting the incident detection. Once completed, the CSIRT should be alerted and informed about the suspected incident, so that they can take control of any further investigation.

Summary

Detection of Incidents is one of the most important steps in an investigation of a computer security incident, because without it, there would not be an investigation at all and the incident would pass unnoticed. Yet it is also the phase during an investigation, during which the responsible experts, i.e. the CSIRT, have the least control over the incident response process [78]. Proper detection of incidents and documentation of initial details enables an organization to detect possible incidents reliably and lets the CSIRT start the next steps of an investigation in a sensible manner.

3.3.3 Initial Response

Goals

During the Initial Response step, the goal is to have the CSIRT confirm a suspected incident or discard the suspicion by collecting and reviewing information related to the incident. If the incident is confirmed, its type and scope should be determined, in order to be able to develop a suitable response strategy in the following step of the Pre-Analsis Phase. Initial Response also includes initializing well-documented containment measures to limit the potential damage of an ongoing incident.

Methods, Tools and Best Practices

In order to confirm a possible computer security incident and develop an approriate response strategy, different sources of information have to be collected. If possible, this information should be collected only from sources other than the actual victim or target of an attack, for example:

- Interviews of people involved in the incident
- Network-based evidence, e.g. IDS logs
- Firewall and router logs
- Classic physical surveillance data, e.g. security cameras or access card reader logs

Often network monitoring will be initiated to confirm an ongoing incident and possibly collect additional data. In rare cases, the Initial Response may also include procedures normally taken during *Live Response*; in this case all precautions that apply for Live Response also have to be respected in this earlier part of the process. If necessary, possible harm for an organization can be avoided by containing the incident, e.g. by removing compromised hosts from the network, initializing packet filtering at routers or firewalls or by keeping suspected persons away from the “crime scene”. As always all activities have to be documented accurately to maintain well-organized incident handling.

Once an incident has been verified, the information collected is used to estimate the incident’s impact on users, systems and business operations. This assessment will then be used to formulate an adequate response strategy.

Summary

During Initial Response data collection is started in order to verify whether an incident actually occured, or if the suspicion was just a false alarm. Through review of the information collected and subsequent assessment of the scope of the incident, formulation of a successful response strategy in the last step of the Pre-Analysis Phase becomes possible. Another benefit of this well-defined approach to initial handling of an incident is that it prevents rash or chaotic reactions and helps everyone involved to focus on the most important tasks at hand.

3.3.4 Formulation of Response Strategy

Goals

The goal of the Formulation of Response Strategy step is to determine the most appropriate strategy to handle the incident in question. In particular, the response team will have to decide whether a full-scale forensic analysis of the incident is warranted.

Methods, Tools and Best Practices

When trying to find an optimal response strategy to deal with a computer security incident, a lot of factors will influence the process of reaching a decision; this is also sometimes called "considering the totality of the circumstances" [78]. Some of the more obvious properties of a specific incident, which have an influence on the adopted response strategy are:

- Criticality of the compromised hosts/data
- Potential perpetrators
- Apparent skill level of the attacker
- Downtime caused by the incident
- Monetary loss

These are factors that are directly related to the incident and will influence how many resources are deployed to investigate the case and what kind of approach will be used to tackle the problem. Apart from these factors, there are also a number of more subtle ones that nevertheless can have a large impact on the choice of a response strategy:

- Political considerations, e.g. what happens if the incident becomes public
- Legal constraints, e.g. liability for not reporting certain incident to law enforcement
- Business objectives

- Acceptable Use Policy, monitoring policy, previous enforcement of policies

I should be noted that, in addition to determining the course of the Analysis Phase, planning administrative or legal action is also part of a Response Strategy.

In the Common Model, the Formulation of Response Strategy step is also the time when it has to be decided, whether a full-scale forensic analysis should be conducted. As described in section 3.2.2., a response team has to assess the *attacker threat level* and the *potential damage* of an incident. If the combination of both is high, i.e. either one is very high or even both, a full-scale forensic analysis should be seriously considered. Due to certain constraints symbolized by an organization's *Response posture* and possible *legal constraints*, a forensic analysis may be mandatory even if the soft factors would not indicate it.

Summary

Carefully planning the actual Analysis Phase of an investigation of a particular incident is a crucial step; only when deliberating all circumstances and the various influences, an appropriate response strategy can be developed. The choice of a specific response strategy will determine the further progression of the investigation, especially because the decision about initiating a full-scale forensic analysis is made during Formulation of Response Strategy. All following activities should be planned out to ensure a controlled Analysis Phase and a successful outcome.

3.4 Analysis Phase

During the Analysis Phase (figure 5), the actual analysis of the compromised computer systems takes place, as outlined in a response strategy that was developed in the previous phase. Starting with Live Response, data concerning the incident are first gathered while the computer systems in question are still running. The rest of the Analysis Phase deals with "dead" analysis of systems, i.e. when they have been powered down. After performing a Forensic Duplication of any storage media involved, several stages of data collection methods are applied in the Data Recovery and Harvesting steps. In the Reduction and Organization step, this collection of data is then reduced and organized to make it as useful as possible for the actual Analysis step, in which the interpretation of data and reconstruction of events takes place.

If the previous phase resulted in opting for a full-scale forensic analysis, then all of the steps of the Analysis Phase have to be performed without taking any shortcuts. If such a thorough investigation is not wanted, some of them may be skipped or shortened; this will be mentioned specifically for each step later on. Notice that some response strategies may not even demand any analysis at all and a response team will go straight to the resolution phases.

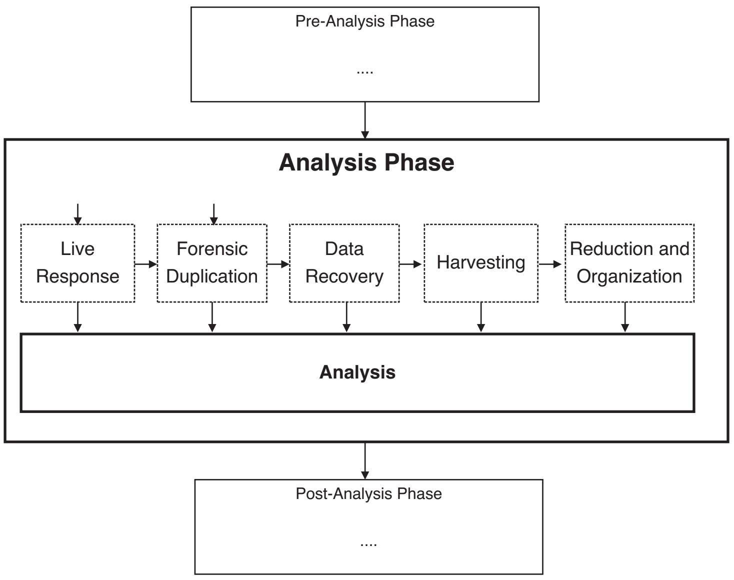


Figure 5: Analysis Phase of the Common Model

3.4.1 Live Response

Because Chapter 4 will contain a detailed discussion of current Best Practices for Live Response for both Windows and Unix systems, this section will only give a brief introduction into the methods for data collection and techniques. Considerations regarding the implications of working on a live evidence system will also be postponed until later, when Live Response techniques are critically analyzed to point out flaws in the methodology.

Goals

Live Response is concerned with collecting data from “live” computer systems, that means the systems under analysis are still powered on and running, in contrast to “dead” analysis on systems that have been powered down. The goal is to collect *volatile* data, i.e. data that can not be recovered from a forensic duplication alone after having cut off the power supply. In addition to this it has been common to collect a number of non-volatile information too for convenience. All these data collection activities should modify the running system as little as possible so that the original evidence is not altered more than absolutely necessary.

Methods, Tools and Best Practices

Certain pieces of data on a running computer are only present in main memory, that means that they will be erased once the computer is switched off. There is no way to reconstruct these information later, including for example:

- system data and time
- list of current network connections and open TCP/UDP ports

- running processes
- open files

On Unix-based operating systems, most of these information can be collected using standard system tools such as `ps` or `netstat`. For Windows systems, a combination of built-in tools and freeware system administration tools can be used, e.g. from Sysinternals [46].

In addition to collecting these volatile data, it has become common to collect non-volatile information from running systems, usually out of convenience. Non-volatile data could also be recovered during a standard dead analysis, but collecting it from a live host may have certain advantages:

- transfer logfiles, e.g. `/var/log/messages`: allows to quickly assess the scope of an incident and its nature
- list login information with `w` or `who`: easy access to the related logfiles that would have to be parsed with considerable effort during a dead analysis
- checking for rootkits: if a rootkit is installed on an evidence system, the output of Live Response tools may be manipulated to hide certain information

The combination of volatile and non-volatile information collected in Live Response can often contain valuable clues for an investigator to estimate the extent of an incident and take measures to contain it.

Summary

Live Response has become an important part of forensic analysis, because it allows to gather data that would be lost forever when analyzing the hard disk image only. By collecting volatile and non-volatile data during this first step in the Analysis Phase, an investigator can already get a good idea of what kind of incident he is dealing with. This even allows the development of initial hypotheses regarding the cause of the incident, which can then be backed up or refuted by information extraced during the dead analysis steps, starting with Forensic Duplication.

3.4.2 Forensic Duplication

Goals

In the Forensic Duplication step, exact copies of all storage media that are involved in the incident are generated, while the original evidence has to stay unaltered. A chain of custody for all media is started and the original media are stored in a safe place along with the duplicates. This is done because the actual data analysis can not be performed on the original media, as it might be altered or corrupted, making it unacceptable as evidence and possibly voiding the entire analysis results that are based on it. For the same reason, every activity during Forensic Duplication has to be precisely documented.

Methods, Tools and Best Practices

When preparing for a forensic duplication, it is first of all important to prepare for duplicating a lot of different storage media, which may sometimes require special hardware. Some examples of possible media to copy are:

- IDE hard drives
- SCSI hard drives
- S-ATA hard drives
- USB or Firewire hard drives
- various types of RAID arrays using any of the hard drive types above
- tape drives
- floppy disks
- storage media such as CDs, DVDs etc.
- CF, SD, MMC etc. memory cards

In addition, hard drives must be removed from their hosts to image them, so bringing a screwdriver kit and other regular tools is wise. Because of the large variety of storage media that an investigator might encounter, a proper toolkit for Forensic Duplication should at least contain:

- extra power cables
- 40-pin/80-pin IDE cables
- 50-pin/68-pin SCSI cables
- SCSI terminators
- a selection of converters (i.e. IDE to Firewire, SCSI to USB etc.)
- hardware write blockers for IDE/SCSI
- large storage hard drives
- documentation material (labels, worksheets, forms)
- blank floppy disks, CDs, DVDs
- boot floppy disks or CD (with trusted OS and tools)
- network hub or switch, network cables

Doubtless the most common media to image are hard drives, and great care must be taken to preserve the original data while also creating an exact duplicate copy. Before starting the actual duplication, a chain of custody should be initiated for the evidence drives, if this has not already been done. All evidence has to be properly labeled and catalogued, for examples of appropriate documentation forms and procedure see e.g. [78]. The evidence drive should then be copied to a storage drive either by using special forensic duplication devices, e.g. a handheld device, or by using a forensic workstation with special duplication tools installed. Regardless of the technique applied, there a several requirements for any forensic duplication tool, which have been defined by the Computer Forensics Tool Testing Project of the National Institute of Standards and Technology (e.g. in [79]). On a relatively high level, these requirements can be summarized as:

- the tool must create a bit-wise copy (forensic duplicate) of the source
- the tool must not modify the source
- in case of read errors, the tool notifies the user of the error type and location in the source, and continues imaging
- in case of read errors, the tool uses a documented placeholder (usually zeros) in the destination in place of the inaccessible data

The first requirement can be quite tricky, because there are ways to manipulate the way a harddrive behaves so that is appears to be smaller than it actually is, allowing to hide data in the "invisible" part of the hard drive. A *Host Protected Area (HPA)* is an area of a hard drive that is not accessible by any means because it is hidden at the level of the ATA interface of the hard drive (see e.g. [64]). Such a HPA can be detected with several tools that compare the output of certain ATA commands; the Sleuthkit [57] contains the tool `disk_stat` to detect a HPA. The only way to access the data stored in the protected space is to change the ATA configuration of the hard drive itself, using for example the tool `disk_sreset` from the Sleuthkit or the freeware tool TAFT [47]. To prevent any damage to the hard drive and consequential loss of data, a drive with a HPA should first be imaged normally, then the HPA should be removed and the hard drive imaged again.

Another way to hide data on a hard drive is by using *Device Configuration Overlay (DCO)*, which essentially allows to define another disk area that cannot be accessed by normal means, and a hard disk can contain both a HPA and use DCO. The only free tool that can remove DCO known to the author is again TAFT. Before attempting to remove DCO from a hard drive and imaging it, the disk should first be imaged as described in the case of a HPA.

The requirement of not modifying the source medium can generally not be fulfilled using only software, therefore using a *hardware write blocker* is the best solution. This device is normally placed between the hard disk and the disk controller on the computer, and it prevents any write accesses to the harddrive.

Stand-alone solutions for duplicating hard drives often have a write blocker built-in.

When imaging a hard drive, there are generally two possible methods:

- remove source disk(s) from the affected host and connect to the forensic workstation
- boot the system containing the source from a live response CD with trusted tools

The first method should generally be preferred, as it is the easier one. Integrity of the evidence drive can be maintained by using a hardware write blocker, and then using a forensic duplication tool to image the drive. Several commercial toolkits such as EnCase by Guidance Software [19], Forensic Toolkit (FTK) by AccessData [1] and X-Ways Forensics by X-Ways Software Technology AG [60] have a built-in capability to create image files, and there are also a lot of specialized commercial imaging tools. There are also open source alternatives to the above commercial products, most importantly the Unix `dd` tool and variants like `dd_rescue` [11] or `dcfldd` [10]. Before imaging a hard drive, a cryptographic hash (e.g. SHA256) of the evidence drive has to be computed and stored. Using `dd`, an image can then be created in the following manner:

- `dd if=source_disk of=image.dd conv=notrunc,noerror,sync`

This will create a a file called `image.dd` which is an exact copy of the source drive, provided that no read errors occured. If there were read errors, the inaccessible sectors are filled with zeros in the image file. After the imaging process is completed, a cryptographic hash of the image file should be computed and compared to the one computed from the evidence drive earlier, to verify that an exact copy has been created. Later on this hash value can also be used to verify the integrity of the image file if needed. In case of read errors or bad sectors on the hard drive, the comparison may fail because of the zeros used to fill the gaps in the image. The tool `dcfldd` allows to compute and store MD5 hash values of variable sized blocks in a logfile while duplicating the drive, bypassing this problem:

-
  ```
  dcfldd if=source_disk of=image.dd conv=notrunc,noerror,sync
  ➥hashwindow=X hashlog=Y
  ```

This command computes an MD5 hash for every X byte block within the evidence drive and stores the list of hash values in a “hashlog” Y. To verify the integrity of the image file later, the following commands can be used:

-
  ```
  dcfldd if=image.dd of=/dev/null conv=notrunc,noerror,sync
  ➥hashwindow=X hashlog=Y_2
  ```
- `diff Y Y_2`

Another way of obtaining an image file of an evidence hard drive is to boot the system containing the hard drive in question from a boot CD with trusted tools, and transfering the data on the evidence drive to the forensic workstation via a network connection. This method is usually used only when the first method is not an option, e.g. when removing the hard drive is not possible, or a special piece of hardware to connect to the evidence drive is missing. When imaging RAID arrays or other logical volumes consisting of multiple hard drives though, this approach may sometimes be the only way to do the imaging; [64] is an excellent reference for acquisition of image files in more complex scenarios. Several pre-compiled Linux boot CDs with installed disk duplication tools are available, e.g. the KNOPPIX [27] Boot CD/DVD, which contains a lot of useful analysis and duplication tools. When booting from any boot CD, it should be made sure, that the evidence hard drive is not automatically mounted by the operating system, as this might modify its contents. After booting from the boot CD and connecting the evidence system to the forensic workstation via network, tools like `dd` can be used in connection with e.g. `netcat` to transfer the data to the forensic workstation:

- forensic workstation: `netcat -v -l -p port > image.dd`

- live system: `dd if=source_disk conv=notrunc,noerror,sync |`
  ➥ `netcat forensic_workstation_ip_address port`

Correctness of the imaging process should be verified using hash values exactly like already described above.

After an image file of an evidence drive has been created, the evidence drive and the drive containing the image file should be stored in a safe place; further analysis will be conducted on “working copies” of the image file which are created when needed for analysis. Having duplicated all relevant storage media, the investigation can then continue with the next steps.

Summary

Forensic Duplication is the first step of a dead forensic analysis of data, and it provides sound copies of all relevant storage media, that can be used in the analysis steps to come. Due diligence should be payed to hard drive duplication, because errors in this stage can discredit an entire investigation and ruin potential evidence. Because of the large amount of work that Forensic Duplication can include, it may in some cases be decided to be left out in favor of easier methods of data acquisition. If an organization just wants to find out what happened during a particular incident and does not care about the admissibility of potential evidence, performing a true forensic duplication may not be necessary and simpler methods may be applicable. During a full-scale forensic investigation however, it is mandatory to follow the guidelines summarized above in order to guarantee integrity of the original evidence and the duplicate copies.

3.4.3 Data Recovery

Goals

Working on the output of the Forensic Duplication step, mostly disk image files, Data Recovery is concerned with the recovery of data that in the current state of the image is not available for analysis. This includes recovery of deleted, damaged, hidden, or otherwise inaccessible data on a file system image, as well as uncovering data that is hidden otherwise, e.g. in unallocated space. The goal of Data Recovery is to "throw out a large net", that means to recover as much data as possible so that it is available in later stages of the analysis. Part of this section is based on [64], which is also a major reference for forensic file system analysis.

Methods, Tools and Best Practices

When a disk image has been acquired, there a usually multiple locations where data may have been hidden from the user's view. This description will focus on use of the open source Sleuthkit tools to recover data: commercial products like EnCase, FTK and X-Ways offer similar functionality and will often even recover deleted or hidden files automatically.

There a several tools in the Sleuthkit that aid in the analysis of a disk image by retrieving various parameter concerning partitions and file systems. The `mmls` tool works on a raw disk image (obtained by using dd on the evidence drive), and extracts information about the partitions on the hard drive. Typical usage of `mmls` may look like this:

- `mmls image.dd`
- Output:

```
     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000031   0000000031   Unallocated
02:  00:00   0000000032   0000062591   0000062560   DOS FAT16 (0x04)
```

This output states that there is one primary partition table with a single FAT 16 partitition, which spans from sector 32 to the end of the harddrive for a total of 62560 sectors; the 31 unallocated sectors in between are left blank because of alignment reason. Data could be hidden in these sectors, so an investigator should check if there is any data contained in them. In other cases, there may be more space unallocated to a partition, either between partitions or after the end of the last partition on the disk. A common way to hide data is to create an extra partition, store some data on it and then delete the partition. This will make the partition inaccessible with normal means from the operating system, but the data stored in the deleted partition will not be deleted and remains hidden. Unallocated disk space should thus be examined closely for hidden data.

Another way of hiding data outside of the actual file system is by using the *volume slack* of a partition to store data. The volume slack consists of the disk space between the end of a file system and the end of the partition in which the filesystem resides. To find out the size of a file system in a partition, the tool `fsstat` from the Sleuthkit can be used:

- `fsstat -f fat -o 32 image.dd`

- Output (truncated):
```
FILE SYSTEM INFORMATION
-----------------------------------------
[...]
File System Layout (in sectors)
Total Range: 0 - 62559
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 61
* FAT 1: 62 - 122
* Data Area: 123 - 62559
** Root Directory: 123 - 154
** Cluster Area: 155 - 62558
** Non-clustered: 62559 - 62559
[...]
```

As it can be seen in the output. `fsstat` reports the number of sectors that belong to the file system as 62560, because the total range is 0-62559. Since this is equal to the size of the partition the file system resides, no volume slack exists. If there is a difference between the two values, the volume slack space should be searched for hidden data.

Apart from the above methods of recovering data outside of the file system, there are also several ways to recover data inside it. The most obvious task is to recover deleted files, and there a number of free tools that allow achieve this. In the Sleuthkit, the tools `fls` and `icat` allow recovery of deleted files. While `fls` lists all file and directory names along with their inode (or similar metadata structure) number in a raw partition image, including those of deleted files and directories, `icat` allows to extract the contents of a file by inode number. Typical output of `fls`, in this case used on an image of an SD card used in a digital camera, looks like this:

- `fls -r image.dd`

- Output (truncated):
```
d/d 3:  DCIM
```

```
+ d/d 517:         11160606
++ r/r 1029:       PICT0418.JPG
++ r/r * 1030:     _ICT0434.JPG
++ r/r 1032:       .DS_Store
[...]
```

This output shows that there is a deleted file with inode number 1030, called `_ICT0434.JPG`; the first letter was replaced by an underscore when the file was deleted, since this is a FAT file system. To recover the deleted file, the `icat` command can be used is the following way:

- `icat image.dd 1030 > _ICT0434.JPG`

This will create a file `_ICT0434.JPG` containing the data extracted from the partition image for inode 1030. It must be noted though, that this kind of recovery will not always successfully recover deleted files. For instance, when a file is deleted, the data blocks containing the file content will be marked as *unallocated*, i.e. free, by the operating system. Whenever a new file is created, these data blocks could easily be reallocated and overwritten with different data. Once that happens, the file contents are permanently lost. In other cases, the file contents might be too scattered around the file system for easy recovery, or the inode information concerning the data blocks allocated to a file might be corrupted. Because recovering single files using the above method is very time-consuming, it is better to use the HTML interface to the Sleuthkit tools, Autopsy [2]. In addition to automating the usage of the Sleuthkit tools, it also offers various case management capabilities and reporting functions that make documentation easier.

Another method to recover deleted file contents involves the tools `unrm` and `lazarus` from The Coroner's Toolkit (TCT) [51]. Using `unrm`, the unallocated blocks in a file system, which is where deleted file contents may have survived deletion of the file, are extracted and stored in a file. Then `lazarus` is used on the output file, employing several methods to organize the unstructured data returned by `urnm` into meaningful files, that can then be reviewed by the investigator. The authors of `lazarus` described its full functionality in the appendix to [68], an detailed example of the rather complex usage of the two tools can be found in [80], "Hack 58".

When file recovery using the method described above does not succeed, the technique of *data carving* may be more successful. Data carving involves using heuristics to recover the content of files that would otherwise not be recoverable. For the FAT file system, the tool `fatback` [13] can be used to recover a lot of files that would otherwise be lost. Using `fatback` on the partition image creates an interactive prompt, which can then be used to copy deleted files from the image to separate files:

- `fatback SD_CARD.dd`

  ```
  No audit log specified, using "./fatback.log"
  ```

```
Parsing file system.
\ (Done)
fatback> ls
Sun Feb 13 16:32:50 2006          0 DCIM/
Sun Mar 14 19:42:48 2006          0 TRASHE~1/       .Trashes
Sun Jun  6 19:41:22 2006       6148 DS              .DS_Store
```

In order to recover the same file that was recovered using `fls` and `icat` before, the respective file has to be located and then copied to a separate file outside the image. Deleted files' names start with a `?` when using `fatback`, so recovering the file is done like this:

-
  ```
  fatback> cd DCIM

  fatback> cd 11160606

  fatback> cp ?ICT0434.JPG .
  Extracting cluster chain 244 to file ./?ICT0434.JPG
  "./?ICT0434.JPG": Unable to recover file entirely,  carving instead.
  ```

If necessary, `fatback` will use data carving techniques to recover deleted files, which will be logged in the audit file `fatback.log`. Another tool to recover files using data carving is `foremost` [14], which uses file signatures for file recovery. A signature file that contains headers and footers of known file types is used to carve data based on this information, often allowing recovery of partially corrupted files or files which are not referenced by any inode data structure but are just present in unallocated space. Usage of `foremost` is pretty straightforward, as it comes with a sample configuration file containing file signatures for most common file types.

- `foremost image.dd`

will create a new folder named `foremost-output`, that contains a large number of files; usually there will be several unusable files, in cases where `foremost` tried to recover files based on their signatures but did not succeed. Generally the rate of success is comparable to that of `fatback`, with the added benefit that `foremost` can work on file systems other than FAT too.
Commercial forensic software packages often use techniques similar to that of the open source tools, but that cannot be said with certainty because their file recovery source code is not publicly available. An interesting fact is, that sometimes the commercial tools will be able to recover certain files that the open source alternatives are not able to recover, and vice versa [75]. Therefore it is advisable to use different tools in combination to be able to recover as many files as possible.

Apart from deleted files, there may also be chunks of data hidden in *file slack* space. Since space on hard disks is allocated only in multiples of the block size, e.g. 512 byte, and some files may not occupy an exact multiple of the block size, there is usually a little bit of space wasted in the last block occupied by a file. This slack space could be used to hide data, although it is very fragile and could be easily overwritten. The tool `dls` from the Sleuthkit can extract all file slack space into a file, which can then be analyzed. Commercial forensic toolkits routinely examine slack space and will reconstruct hidden data if possible. In a very thourough investigation, slack space should be examined closely for hidden data.

Apart from these rather obvious ways to hide data, there are also a number of very obscure places where a skilled attacker could hide data (again [64] is the landmark work to refer to), e.g.:

- in unallocated clusters following the partition map
- in unused fields in partition entry structure/disk label structure
- in altered partition type fields
- additional data on individual disks in RAID volumes which does not belong to the RAID volume
- data units manually added to the list of bad data units in a file system
- alternate data streams on NTFS systems

There are even more clever ways to hide data, see for example [77] or [72]. Only a very skilled investigator who devotes a lot of time will be able to detect this level of data hiding techniques, which are sometimes even called "Anti-Forensics". Recovering data hidden in this way will mostly consist of manually scrutinizing the data with only a few low-level tools like `dd` and a hex editor, or custom-made experimental tools. If the suspected attacker skill level is estimated to be extremely high, or if the data that might be hidden is of extreme value, looking for hidden data should be expanded to the obscure locations mentioned above and in the cited material.

Summary

Data Recovery is an important step in any incident analysis, because it provides a basis for all successive step to come. If a file or piece of information has not been recovered during Data Recovery, the chances that it will become a part of the analysis are small, and even if it is discovered later this could complicate the whole process as previous analysis results might have to be reevaluated. Thus in a full-scale forensic analysis, considerable effort should be put into Data Recovery and the documentation of all activities involved. If the response strategy adopted is exclusively aimed at recovery of deleted data, the methods metioned above to find hidden data can be skipped. In general, the higher the attacker threat level is estimated, the more likely it is that advanced data hiding techniques have been used to hide data or cover the tracks of a crime; in such a case even the more obscure ways of data hiding should be investigated.

3.4.4 Harvesting

Goals

During Harvesting, the investigator begins to gather *metadata* (data about data) about the preserved material which is the output of the Data Recovery step. This allows to structure the largely unorganized material based on certain criteria, typically file timestamps, permissions and other file attributes, and the file type. This structure will help during the rest of the analysis, because often sets of data with certain properties "seem or are known to be related to the major facts of the case or incident known to this point in the investigation" [66].

Methods, Tools and Best Practices

There are several different classes of metadata that can be harvested from the recovered data. One of the most important classes are file attributes, namely:

- file name and size
- modification, access, change (MAC) times
- user/group ownership
- permissions
- inode number

The Perl script `getmetadata.pl` (sourcecode listed in Appendix A.1) that is included in [75] can be used to collect all of the above mentioned metadata for all files in a partition image, and it will in addition also compute an MD5 hash value for each file. Before being able to use it, a specially formated `fls` output file has to be created first using:

```
• fls -r -p -l -m / -f file_system_type partition_image >
  ➥ part_fls.txt
```

This command will produce an output file called `part_fls.txt` which contains various metadata for each file in a partition image in a format such as in this example output:

```
• 0|/DCIM/11160606/PICT0418.JPG|0|1029|33279|-/-rwxrwxrwx|1|0|0|0|
  ➥ 762344|1149544800|1149615232|1149615232|512|0

  0|/DCIM/11160606/_ICT0434.JPG (deleted)|0|1033|33279|-/-rwxrwxrwx|0|0|0|0|
  ➥ 831346|0|1149615560|1149615560|512|0
```

The `getmetadata.pl` script will then make use of this file to extract the metadata, compute the hash values and display the whole information in a human-readable format

- `./getmetadata.pl part_fls.txt file_system_type partition_image >`
  `➥ metadata.txt`

Metadata is displayed in the output file in the following format:

-
  ```
  MD5|Filename|Inode|Mode|ModeString|User|Group|Size|ATime|ADate|MTime|
  ➥ MDate|CTime|CDate
  ```

  ```
  50a87c5fe2ce7bae9a8613e27917f696|/DCIM/11160606/PICT0418.JPG|1029|
  ➥ 33279|-/-rwxrwxrwx|0|0|762344|00:00:00|06-06-2006|19:33:52|06-06-2006|
  19:33:52|06-06-2006
  ```

  ```
  c4479a3e1fb1f1b6dbc1f7f8d8c6cdaa|/DCIM/11160606/_ICT0434.JPG (deleted)|1033|
  ➥ 33279|-/-rwxrwxrwx|0|0|831346|01:00:00|01-01-1970|19:39:20|06-06-2006|
  19:39:20|06-06-2006
  ```

This collection of metadata can then be used in further analysis, e.g. by importing it into a spreadsheet program and grouping files according to user and group ownership. Hash values are also used in the next step of Reduction and Organization to reduce the number of files that have to be analyzed.

To group files according to file type a very similar script can be used, that uses the output file of `getmetadata.pl`. Using the Unix tool `file`, the script `getsignatures.pl` (listed in Appendix A.2) checks for known file signatures and outputs the file type after the already extracted metadata:

- `./getsignatures metadata.txt file_system_type`
  `➥ partition_image > sigmeta.txt`

- Output:

  ```
  50a87c5fe2ce7bae9a8613e27917f696|/DCIM/11160606/PICT0418.JPG|1029|
  ➥ 33279|-/-rwxrwxrwx|0|0|762344|00:00:00|06-06-2006|19:33:52|06-06-2006|
  ➥ 19:33:52|06-06-2006|JPEG image data, EXIF standard 0.73, 10752 x 2048
  ```

  ```
  c4479a3e1fb1f1b6dbc1f7f8d8c6cdaa|/DCIM/11160606/_ICT0434.JPG (deleted)|1033|
  ➥ 33279|-/-rwxrwxrwx|0|0|831346|01:00:00|01-01-1970|19:39:20|06-06-2006|
  ➥ 19:39:20|06-06-2006|JPEG image data, EXIF standard 0.73, 10752 x 2048
  ```

The file type information can then be used to group files of a certain type or class, e.g. when investigating a case of suspected distribution of child pornography one would focus on image and video files, whereas in case a computer intrusion with a suspected rootkit installation the investigator would rather focus on executables or driver files.

All the task described above can also be performed using the commercial forensic toolkits EnCase and FTK, typically the metadata is even displayed by default and multiple ways to evaluate them are available. At the end of this step, organized sets of digital data grouped by metadata information have been formed and ease the further analysis.

Summary

By collecting metadata during Harvesting the large and unstructured body of data that was the output of the Recovery step can be organized to make it more useful for the following analysis. In this step, the actual details of the case are for the first time really incorporated in the Analysis Phase, since the type of metadata used and the way in which the data are grouped with respect to it can be adjusted to the specific case. For some response strategies, Harvesting may be left out because only the content of the data and not the metadata are important for the analysis. A full-scale forensic anlysis should always include a Harvesting step, since metadata information – in particular MAC timestamps – are often the best way to reconstruct a series of events in an incident. The computing of hash values can also be used in the Reduction and Organization step that follows to eliminate irrelevant data, which is crucial in a thourough investigation that typically produces a lot of raw data in the Recovery and Harvesting steps.

3.4.5 Reduction and Organization

Goals

During Reduction and Organization all data that can be identified as irrelevant to the case is eliminated, and the remaining data is organized to alleviate access to the data later. The goal is to reduce the large set of structured data that was the output of the previous step to “the smallest set of digital information that has the highest potential for containing data or probative value” [66], and then to organize the data to allow efficient search, identification and reference to relevant data in the Analysis step and the whole Post-Analysis Phase.

Methods, Tools and Best Practices

To reduce the amount of data, the most widespread way is to eliminate known good files by comparing MD5 or other cryptographic checksums of the data in question with MD5 checksums of known files recorded in a database. For instance, there are hundreds of files in a standard Windows installation which are the same for every installation, and if they are unaltered, they can be safely disregarded in the further analysis. Cryphographic checksums like MD5, SHA1 or the newer SHA256 can be used to detect any difference in the contents of those files, because a matching checksum represents an overwhelming possibility that the file is unmodified, i.e. that it can be eliminated.

There are several different large collections of precomputed hash values for known files, e.g.:

- The National Software Reference Library [54] by the NIST
- Maresware Hash Set CD [28] (commercial)
- Hashkeeper Group [20]
- KnownGoods Database [53]

- Dan Farmer's FUCK baseline collection (discontinued, but available via archive.org [7])

- Solaris Fingerprint Database [40]

It is important to keep in mind, that these hash databases contain hash values of *known* files, not necessarily of known good files only. They can not be used to simply eliminate any matching file from further analysis, but rather to identify files and then choose whether they can be safely eliminated. There are also hash sets of known child pornography images, in a way "known bad" files, that can be used to scan through large image sets that were formed in the preceding steps.

In either case, an investigator will probably assemble and use a database suitable for the actual case using these available resources and perhaps even creating a custom hash set. If a hash set containing only hashes of known good files is available, the script `unknowns.pl` (listed in Appendix A.3) can be used to extract the names of all files whose MD5 checksum does not equal that of a known file. To use the script, the hash set `MDHashDatabase.txt` has to contain only the hash values, each on a single line and sorted alphabetically; it also uses the `metadata.txt` file that was created in the Harvesting step:

- `./unknowns.pl metadata.txt MD5HashDatabase.txt > unknowns.txt`

This will filter the metadata file by eliminating all files whose MD5 checksum matches one that is contained in the hash database file. To make sure that no unknown file is eliminated by chance, the list of files to be excluded should always be manually reviewed, to avoid accidentally eliminating important data.

Using the commercial EnCase and FTK, precompiled hash databases such as the NSRL can be easily imported and used to filter the data in a forensic image. They also include functionality to build new hash sets from scratch, for example by importing complete sample installations of different operating systems and then computing and storing the hash values in a central database.

After having reduced the data to a set that is believed to hold the highest potential for evidence, it is a good idea to organize the data in preparation for the actual analysis later on. This can mean that extracted files are physically group by putting certain types of files into distinct folders, e.g. folders named "images", "executables", "logfiles" and so on. Another way to group data is by *tagging* it, which means that different pieces of data can be labeled with one or more attributes out of a set of predefined keywords (tags). If the amount of data under review is very large, one might even use a database approach to manage the data, in order to allow quick retrieval and identification and make it easier to reference these data in the Post-Analysis Phase. For the same reasons, a searchable index for the data is often built to allow an efficient review of the data.

Summary

In the Reduction and Organization step, the final preparations for the Analysis step are made, by reducing the amount of data that has to be analyzed in detail and by organizing the data in meaningful ways to facilitate efficient analysis. The Reduction part will most likely be part of every investigation, since in most cases of computer security incidents, most of the data involved can be safely disregarded and thus all resources can be focussed on analyzing the really important pieces of data. Nevertheless, attention has to be payed that no vital data is eliminated by accidentally excluding it. Organizing the data after having reduced them is also a task that will be done in most investigation, sometimes even implicitly because investigators tend to organize data already as they recover, harvest or reduce it. In a full-scale forensic investigation it is even more important to use sound means of grouping and retrieving the data; in the Post-Analysis Phase, referencing the data in a report or testimony is a major part of justifying and substantiating the findings of the investigator and his interpretation of the case.

3.4.6 Analysis

Goals

After having recovered, harvested, reduced and organized the data related to an incident in the previous steps of the Analysis Phase, the actual reasoned analysis start in the Analysis step. An investigator develops a detailed reconstruction of the events that comprise the incident and tries to answer the questions of what happened, when and how did it happen, and who is responsible. During an investigation of suspected criminal activity, the perpetrator has to be identified, along with determining the means, motivation and opportunity. By reviewing the actual contents of the data, different pieces of evidence are correlated to establish links between them. By carefully documenting the activities and validating the results, this results in a thorough reconstruction of the incident based on objectivity and scientific principles.

Methods, Tools and Best Practices

There is a wide variety of possible ways to achieve the goal of event reconstruction, because every investigation has his own special properties and technicalities that influence the analysis. To guarantee an objective analysis and interpretation of the results, the scientific method has to be applied, that means an investigator should test multiple theories regarding an incident and try to disprove them, instead of trying to verify them. By eliminating theories that conflict with intermediate results, a remaining theory has a high possibility to represent a correct reconstruction of the events.

Another very important property of analysis results is that they are *repeatable*, meaning that any observer could make the same observations as the investigator using the same methods. To allow a smooth and accurate Post-Analysis Phase, every action taken, technique applied or tools used should be documented precisely and immediately, along with its results and impact on the considered theories.

For these reasons, rather than trying to list all the available tools and Best Practices for analysis, it is better to consider some basic techniques that should be used to achieve repeatabiliy and objectivity of the analysis [66]; some examples of actual Best Practices and tools will be given afterwards:

- Assessment of content and context of digital data objects
- Experimentation with new or unusual techniques
- Fusion of different pieces of evidence to form a conclusive argument; also Fusion can mean incorporating evidence from sources other than the digital evidence
- Correlation of related evidence to establish cause-and-effect relationships between certain pieces of evidence and events
- Validation of the findings to produce reliable and repeatable results

An example of Assessment of digital evidence are string searches, that are routinely used to search digital data sets for certain keywords. Open source tools like `strings` and `grep` can be easily used to search for keywords in a complete disk image or in a group of files. Attention has to be payed to compressed and encrypted files, as strings included in them can normally not be searched for; in these cases, decompressing and decrypting these files has to be performed before they can be searched for occurences of the keywords. Using the HTML graphics interface Autopsy for the Sleuthkit, keyword searches and regular expression searches can be conducted quickly with the added opportunity to look for strings in unallocated space too. Commercial forensic toolkits usually offer the same features.

The analysis of unknown binary files, i.e. executables, is an area in which often new or untested methods are used and experimentation is necessary to uncover the purpose of the unknown files. Generally speaking there are two approaches to analyzing binary files, static and dynamic analysis. In static analysis, the binary is only analyzed while it is not running, e.g. using a disassembler and an number of assorted Unix tools. Dynamic analysis incorporates running the program in a controlled environment and often involves using a debugger to single-step through the instructions of a binary file, inspect memory registers and alter the program flow to discover an unknown binary's purpose. To get an idea of the amount of work and experimentation that may be involved in the art of reverse engineering a binary file, see for example the Honeynet Scan of the Month Challenges 32 [21] and 33 [22].

Fusion is often used to bring together network-based evidence with host-based evidence and find conclusive explanations of the events that comprise the incident. Usually the Unix tool `tcpdump` is used to capture network traffic and store it in a capture file. There are four different classes of network-based evidence that can be collected and analyzed [61]:

- full-content data
  - all network data is captured
  - everything that traveled the network can be reconstructed, e.g. with `tcpflow` [49]
- session data
  - analyze TCP sessions (incoming/outgoing ports, amount of data transferred), e.g. using `tcptrace` [50]
  - correlate sessions to identify an attacker's behaviour
- statistical data (e.g. using `tcpdstat` [48])
  - count number of unique TCP flows or number of observed IP adresses
  - break down traffic into the different protocols
- alert data
  - use an IDS to check traffic online or run it on capture files offline, e.g. with `snort` [39]
  - check alert message explanations and note possible security breaches for later analysis

Apart from these automated tools, detailed analysis of captured network data is often done using Ethereal/Wireshark [59].

Putting together the network-based evidence with the host-based evidence produced during the analysis, it is often possible to reconstruct events that could not have been explained with one type of evidence alone. Often, an activity timeline is constructed to represent the incident in a chronological way, especially in case of computer intrusions, where there is usually network-based evidence for the intrusion and host-based evidence for the changes made to the attacked hosts. Another possible way to fuse evidence is combining physical with digital evidence, consider e.g. a piece of paper found at the desk of a suspected attacker with passwords written by hand on it, which can then be used to decrypt encrypted files in a disk image.

Fusion is closely related to Correlation, but correlating evidence means more than putting events in a chronological context — it means that cause and effect of different events are investigated to establish a causal relationship. In the example above, consider that a forensic document examiner can prove that the suspect wrote the passwords on the piece of paper himself; this would mean that he most likely knew about the encrypted files and had access to them. Correlating with additional evidence might then make it possible to prove that the suspect encrypted the files himself, perhaps even created them and so on.

The examples above represent only a small selection of tools and Best Practices that are commonly used in the Analysis step, other topics that have not been mentioned are reconstructing web browsing or e-mail activity, dealing with proprietary formats of data, forensics of PDAs etc. There is basically no limit of what can be done in the Analysis step, as long as the guiding principles described above are respected, and the investigator maintains an objective standpoint throughout the whole process.

With respect to the individual tools and methods used during the Analysis, requirements for admissibility, i.e. the validity of evidence in a court of law, depend on the local legislation. As a guideline, the so-called *Daubert Standard* for the admissibility of scientific evidence should be consulted; the Daubert Standard was introduced by a U.S. Supreme Court of Justice ruling in the case *Daubert vs. Merrell Dow Pharmaceuticals, Inc.* [9] and describes a set of requirements for scientific evidence to be admitted in court. Among other requirements it demands that a tool or technique has been subject to peer review and publication, that the error rate is known and that the technique is generally accepted by the relevant scientific community. For this reason established and well-known tools and techniques should be prefered in any case, and if previously unknown and untested methods have to be used, due diligence should be payed to documentation and justification of the technique in order to achieve credibility.

Summary

During the Analysis step the actual reasoned scrutiny of all evidence collected and prepared so far takes place. Using the scientific method and accepted methods like correlation, fusion and validation, valid and admissible evidence is produced that reconstructs the events that make up the computer security incident. The Analysis step will usually be part of any investigation of a computer security incident. In case of a full-scale forensic analysis, all of the above principles and requirements have to be fulfilled to achieve admissibility of the analysis results. A more relaxed investigation may borrow from the techniques, but does not have to be that strict in its interpretation of for example the Daubert Standard. Regardless of the scope and type of the investigation though, objectivity should be maintained through the whole process, and proper documentation has to be performed to achieve credibility of the results and ease the activities perform in the Post-Analysis Phase.

3.5 Post-Analysis Phase

The Post-Analysis Phase (figure 6) starts when all activities regarding the collection and analysis of digital evidence have ended, and the objectives set by the response strategy for the Analysis Phase have been fulfilled. The remaining tasks concern producing a report which documents the whole investigative process, and taking measures to resolve the underlying issues of the incident that was investigated. This can mean that a forensic examiner has to testify in court about his work on the incident; in these cases, proper preparation to withstand cross-examination from an oppposite legal party is necessary.

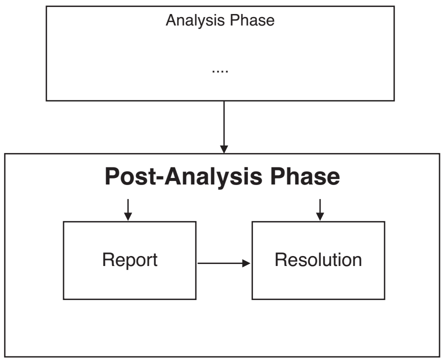


Figure 6: Post-Analysis Phase of the Common Model

3.5.1 Report

Goals

The Report step deals with writing a precise report that describes the details of an incident, is understandable to non-technical readers or executives and meets the legal standard for admissibility in court. It contains all of the documentation material that (hopefully) was produced during the Pre-Analysis and Analysis Phase respectively, and assembles the different pieces into a comprehensive overview of the whole case, while pointing out the most important analysis results and implications for resolution of the incident.

Methods, Tools and Best Practices

If documentation of the previous steps has been done in a proper way, then the Report step consists mainly of summarizing the already documented activities in a clear and concise report. Before the actual investigative procedures are described, a computer forensic report should give a short “Executive summary” of the incident. This summary should include a list of all the people involved in the incident, a list of the most significant results of the investigation and recommendations for resolution of the incident.

The report should then proceed to describe the course of the investigation, starting with the Detection of the Incident step, i.e. at the very beginning of the whole process. The reasons for starting an investigation and whether to conduct a full-scale forensic investigation should be clearly stated, and the Response Strategy should be included. After that a summary of the steps in the Analysis Phase is given, e.g. which steps were taken and which tools or methods were used for a specific task. Original documentation, sourcecode of any scripts used or lenghty output of commands should be put into attachments or appendices, so that the flow of the report does not get interrupted easily. It is a good idea to include any piece of data that had any influence on the conclusions drawn during the investigation, because it makes the report self-contained and more credible.

When presenting the results of the analysis, great care should be exercised to explain how the results were achieved, what other possible theories were investigated and why the analysis ultimately ended with the conclusions documented in the report. In this step, an investigator should point out where he used scientific reasoning based on correlation, fusion and how he validated his findings, in order to show that the results are reliable. All this should be written in a clear and intelligible style, so that not only forensic experts, but also management executives or legal personnel can comprehend the implications of the results and use them in a sensible way. The same applies if the forensic analyst is summoned as an expert witness in a lawsuit.

Summary

In every investigation, regardless of its scope or the amount of resources put into it, a written report should be created that documents the entire investigation and its results. Such a report not only enables an organization to initiate the appropriate measures to resolve the incident in the Resolution phase, it also helps to avoid similar problems in the future and to optimize the Incident Response procedures themselves.

Writing a good forensic report is one of the most difficult tasks of the whole investigative process — on one hand it has to contain all the technical evidence and scientific reasoning to support its results, on the other hand it is most of the time not directed at technical experts and thus has to convey its results in an easily understandable manner. If an examiner is asked to testify about the investigation in a court of law, he should prepare properly to defend his analysis methods and the results, which can be difficult when being attacked by an opposing legal party in cross-examination.

By promoting accurate and proper documentation throughout the whole process of investigation, documenting immediately and in a well-defined way, some of these difficulties can be avoided. If documentation of the previous steps is not sufficient then writing an accurate report can become near impossible, jeopardizing the credibility of the whole analysis and possibly ruining all the work put into the investigation. For the same reasons, an expert witness's credibility can crumble if an opposing party can raise questions about an examiner's methods and diligence, which could lead to exclusion of the analysis results from the trial.

3.5.2 Resolution

Goals

The goal of Resolution is to contain the problem that caused the incident, solve it and take appropriate measures to keep it from occurring again in the future. In case of an ongoing incident, these measures may also be taken prior to conclusion of the analysis, but if possible they should only be put in place after all potential evidence has been secured for further analysis, because the evidence will generally be altered by the implemented security measures. To verify that the chosen security mechanism have been integrated successfully and that they are effective, the implementation should be supervised and the effectiveness of the measures should be validated in the future.

Methods, Tools and Best Practices

Usually the methods used to resolve an incident are well-known host- and network-based security measures that are adjusted to the specific situation. If it turns out that a computer was infected with an internet worm then removing the worm completely and then applying patches for the exploited vulnerabilities will solve the problem. In case of a massive Distributed Denial of Service attack on a company's webserver, then reconfiguring the network ingress routers may resolve the situation, depending on the type of DDoS attack.

Whatever steps are taken to resolve the incident, every step should keep the top priorities of the affected organization in mind, e.g. whether quick recovery to normal business operations is most important, or if a thorough investigation is desired. After having identified the cause of the incident and the underlying problems, suitable security measures should be determined, according to the priorities mentioned above. These host- and network-based security mechanism should then be implemented as soon as possible, but without risking the loss of valuable information or evidence. To make sure that all measures work as intended, their effectiveness should be validated and documented. After having resolved the individual incident completely, it may be necessary to adjust policies or incident handling procedures, if the investigation of the incident revealed a flaw in the current methodology. This "Lessons Learned" effect is crucial to improve the incident handling process and use experience from past cases to prepare better for possible incident to arise.

Summary

The Resolution step deals with the resoluting of the incident and the elimination of the underlying problems that caused it. Although it is listed as the last step in the Common Model, steps to resolve or contain an incident may actually be taken very early into an investigation, generally to avoid harm by an ongoing incident. Whenever possible, this should only be done after all relevant data have been collected in an unmodified state. If it is absolutely necessary, affected systems can then be isolated from the network, switched off or in maybe repaired without losing too much information regarding the incident. A Resolution step will be part of any investigation that resulted in identification of one or more security issues that should be addressed, even if the analysis itself did not obey to forensic standards. As mentioned before, sometimes the focus simply lies on solving the problem at hand rather than accurately reconstructing and investigating an incident to produce admissible evidence. In either case, a "Lessons Learned" effect can give valuable clues to improve overall security and incident response procedures.

3.6 Discussion

The Common Model for Incident Response and Computer Forensics offers a way to conduct proper Incident Response while applying principles known from Computer Forensics during the actual analysis phase. In this way it addresses the shortcomings of both seperate models, effectively integrating a forensic analysis into an Incident Response framework. This model makes use of the management procedures developed in Incident Response process models [78], which mainly concern the Pre-Analysis and Post-Analysis phases. Typically these measure like Pre-Incident Preparation and Incident Detection are not included in traditional investigative models, since those tend to approach the problem from a law enforcement's point of view.

On the other hand, the Incident Response process model introduced in chapter 2 only describes the Analysis Phase rather loosely, basically leaving it up to an investigator which steps to take and how to conduct the analysis. But in many cases, an incident investigation requires a much more thorough and structured type of investigation, because the possible outcome might include legal or administrative action, or the potential damage to the organization under attack is so high, that they want to find out exactly what caused the incident to prevent similar incidents in the future. In these cases, a well-defined and accepted process model like the Investigative Process model [66] offers an ideal way to conduct an investigation that satisfies forensic standards and will be able to yield admissible evidence of high reliability. For this reason the Common Model's Analysis phase consists of the classical investigative steps which comprise a full-scale forensic analysis, and it also includes Live Response as another method of data collection which has proven to be inevitable in modern incident investigations.

Since a full-scale forensic analysis is not required or adequate in every single incident, the Common Model uses a number of factors related to the particular incident to decide whether such a full-scale forensic analysis should be afforded. There are two "hard" factors, which can influence this decision independently of everything else: legal constraints and an organization's Response Posture. It has been described how both can make an analysis that complies with forensic standards mandatory. On top of that there are two "soft factors", the estimated Attacker Threat Level and the Potential Damage of the incident. If the abstract equation

$$AttackerThreatLevel \times PotentialDamage > Threshold$$

becomes true, then a full-scale is justified and should be seriously considered. This equation has been introduced only to illustrate the decision process, reality the "Threshold" value or the Attacker Threat Level can hardly be displayed as absolute values.

Nevertheless this equation is related to the well-known *Risk Equation* which is often used to define risk [55]:

$$Risk = Threat \times Vulnerability \times Cost$$

The Threat value corresponds approximately to the Attacker Threat Level and the Cost value corresponds to Potential Damage. Vulnerability normally denotes the likelyhood that a particular attacks succeeds, which in the case of Incident Response has in fact already happened; therefore this factor could be set equal to 1, and the two equations would draw even nearer to each other.

In this sense, the value that determines whether a full-scale forensic analysis is necessary equals Risk under the condition that a vulnerability has already been exploited.

To show which techniques are commonly applied in each step of the Common Model, examples of methods, tools and Best Practices have been given. This can of course not be an exhaustive listings of all available tools and methods, but rather it shows which techniques and tools are currently relied upon to automate certain tasks and which are generally accepted in the forensic community.

3.7 Summary

This chapter introduced the Common Model for Incident Response and Computer Forensics which is a new way of conducting Incident Response, with focus on complying with scientific forensic principles in the Analysis Phase. Each step's significance within the whole process has been discussed, and commonly used tools as well as tried and accepted Best Practices have been introduced for each step. The model has been designed to flexibly include a the full-scale forensic analysis, and a method for determining in which cases this is adequate has been proposed. It was shown that the Common Model's way of assessing the need for forensic analysis is comparable to measures of risk in Risk Management. In the next chapter, the step of Live Response, which was previously left out of the description of the Common Model, will be described in great detail. This will aim at establishing Live Response as an inevitable part of forensic analysis. On top of that it will be shown that current Live Response practices may be flawed and it will be shown how they can be improved to make the Live Response process more accurate and reliable.

Chapter 4

Live Response

4.1 Introduction

When investigating an incident or computer crime, classical Computer Forensics would usually start with pulling the plug of all computer systems involved, and then creating forensic duplications of all hard drives in the respective systems. This kind of analysis is commonly called "dead" analysis. In recent years, this principle has been reconsidered, and methods of data collection on systems that are still running have become popular. In contrast to dead analysis, this is called "live" analysis, and all activities concerning data collection on live systems are summarized under the term *Live Response*.

4.1.1 Motivation

The reason why Live Response has become an important part of forensic analysis is that when powering down a computer system, a lot of volatile information is lost, because the RAM is cleared. This information will not be available for further analysis, although it may contain valuable clues regarding the incident, and there is sometimes no other way to obtain it other than collecting it on the running system. Volatile information that can be collected during Live Response includes:

- system date and time
- list of current network connections and open TCP/UDP ports
- running processes
- loaded kernel modules
- open files

Especially the information regarding network connections and ports, and running processes can be crucial information that allows to figure out what caused an incident or how a system was compromised.

In addition to the collection of volatile data, it has also become common to collect certain non-volatile data too, i.e. data that could also be recovered in dead analysis. Sometimes this is done out of convenience, e.g. because certain logfiles can easily be recovered from a running system with special commands, but in a dead analysis the investigator would have to parse the logfile format with large effort. In some cases file formats are even proprietary so no publicly available parsers do exist, yet it is very easy to recover the files on a running system with the right tools.

Another reason for collecting non-volatile data is that this information will already be available in a very early stage of an analysis and can improve containment of the incident. With the amount of data that can already be recovered during Live Response, performing a forensic duplication can become redundant, because enough information to explain and resolve the incident is already there. On the other hand, if Live Response yields unexplainable results or raises further suspicions, the decision to perform a forensic duplication and thorough dead analysis is strengthened.

4.1.2 Live Response considerations

A severe problem with Live Response techniques is that they will generally alter the running system's state, which contradicts the general paradigm of forensics to never modify potential evidence. Issuing commands on an evidence system will change that system's RAM contents, create processes, and even the hard drive may be modified, e.g. by unnoticed creation of cache files by the operating system. On top of that, software on a possibly compromised host can not be trusted, since it may have been manipulated to return false output, crash the operating system, or worse.

In order to be able to conduct effective Live Response anyway, special techniques are used to modify the system under investigation *as little as possible* while still collecting accurate information from the host. If the changes that the use of a Live Response tool will make to the target system are well understood and documented, its use can be admitted even if it alters evidence slightly. This tradeoff is justified by the fact, that the information gained by conducting Live Response is often so valuable, that a small and controlled modification of the evidence is negligible.

In the next section, techniques to collect volatile and non-volatile data from live systems will be introduced, for both Windows and Unix-based systems. Methods to reduce the forensic flaws of Live Response – modification of the evidence and missing trust in the underlying operating system – to a minimum will be given for both operating systems by demonstrating how to assemble a Live Response toolkit.

4.2 Best Practices and Standard Procedures

4.2.1 Live Response on Unix-based operating systems

This section introduces common methods to perform Live Response on Unix-like operating system including BSD, Linux, Solaris and Mac OS X. Some of the tools or techniques may not be applicable to all of these operating systems, either by design or because there does just not exist a version of a tool for this operating system yet. Because of constant development of operating systems and the tools, it can not be guaranteed that a particular tool or method described here will work in all circumstances. Most of the software used for the Live Response procedures for Unix-based operating systems is available from the GNU Project [52] website.

Volatile data

There are a large number of tools that can be used to recover volatile information from a running system. Most of them concern the state of network connections or processes, but there are also some more general information that may be useful in an investigation. All of the tools to collect volatile information are standard system tools on Unix/Linux systems, the following list includes the most common tools used and a small description of each:

- `date`: prints out the current system date and time
- `hostid`: prints out the host ID, a numeric identifier for the current host
- `hostname`: prints out the current hostname
- `rpcinfo -p 127.0.0.1`: prints out a list of all RPC (remote procedure call) services registered on the current host
- `netstat -an`: prints out a list of current network connections and open TCP/UDP ports
- `netstat -rn`: prints out the internal routing table
- `ifconfig`: prints out network interface parameters
- `lsof`: prints out list of open files and executables opening them
- `ps -aux`: prints out list of running processes
- `lsmod`: prints out list of loaded kernel modules
- `mount`: prints out list of mounted file systems
- `df`: prints out information about mounted file systems

Analyzing this information can already give an investigator valuable clues and can help to confirm suspicions about the incident. For example, if suspecting that an attacker installed a backdoor on the victim host, the list of open ports obtained with `netstat` and the list of running processes obtained using `ps` may reveal the listening backdoor. The `lsof` output may further confirm this conclusion. In another case, an attacker may have manipulated a host's internal routing table or the network adapter's configuration to intercept traffic to and from the victim host, which could be revealed by the output of `ifconfig` and `netstat`. It is up to an investigator how to use the information gained using the commands listed above, and it certainly depends on the kind of incident that is investigated.

There are also a number of more complicated ways to assess the volatile data on a running computer system, mostly this concerns analysis of the RAM of the host. Depending on the operating system there are several different ways to read main memory, most notably using the devices `/dev/mem` and `/dev/kmem`. The tool `memdump` [29] offers the ability to capture the complete main memory of a running system, avoiding some of the potential problems that normally arise when working with the memory devices. For detailed information about how to perform full memory dumps, refer to [68]. Another possible way to look for information is the `/proc` virtual filesystem, which serves as an interface to internal kernel data structures. Data in the `/proc` filesystem is only partially human-readable, the manpage of proc displayed with `man proc` contains information about the structure of this virtual filesystem. Because of the complexity of the methods described above, direct RAM capturing and review of the `/proc` filesystem should only be done by very skilled investigators and only in cases where a lot of resources and time can be allocated to the analysis of this data.

Non-volatile data

Apart from these volatile data a lot of non-volatile data is usually collected too, that could also be obtained in a dead analysis by analyzing the forensic duplicates of the hard disks. In some cases the live collection is preferable, because special tools display data in a human-readable way, which would not be easy to reproduce during a dead analysis. In other cases data is collected simply because it is then already available very early in the investigation, allowing to quickly assess an incident and contain it if necessary. Activities commonly performed include:

- `uname -a`: acquire system version and patch level
- `rpm -qa`: obtain package information (only possible if RPM package management is used)
- `w, who`: list users currently logged on
- `last`: list history of logins
- `lastcomm`: list process accounting logs (only possible if process accounting is enabled)

The information about the system version and installed packages can be used if a system has obviously been compromised by using an exploit of some software vulnerability. In this case, knowing about the exact operating system version and installed software can help to narrow down the list of possible exploits that may have been used. Information about logins can help to identify unexpected users or may even reveal a rogue login sessions that is still active. In addition to these information regarding the operating system itself, typically some data is collected about the file system of the host in question:

- `find / -printf "%m;%Ax;%AT;%Tx;%TT;%Cx;%CT;%U;%G;%s;%p\n"`
  prints out file name, MAC timestamps, ownership information and file size of every file in the file system
- `find / -type f -xdev -exec md5sum -b {} \;`
  prints out MD5 checksums for every file in the file system
- review `/var/log/messages`, `/var/log/secure`
- review `/etc/passwd`, `/etc/groups`, `/etc/inetd.conf` and other important configuration files
- review user history files, e.g .`~/bash_history`
- possibly transfer any suspicious file

The listing of file attributes can be used for correlation with other clues that have been found earlier, e.g. by comparing timestamps of suspicious files with login times of users. Using the `find` command it is also possible to create a hash database of all the files on the system, making it easy to quickly check if a certain file has been modified by checking available hash databases, which were introduced in Chapter 3. Logfiles often contain a lot of information about the activities performed on a computer, including those of an attacker. There is also a number of important configuration files like `/etc/passwd` that are very often involved in the manipulation of a victim host and that should be checked routinely when investigating a computer security incident.

Armed with the volatile and non-volatile information gathered through the steps described above, a lot of incidents can already be properly classified and contained before even starting the dead analysis of the incident.

Preparing a Linux Live Response toolkit

This section will show a common way to prepare a Live Response toolkit that addresses the two main issues of Live Response, namely the modification of potential evidence and the lack of trust in the operating system that is used to run the Live Response tools.

One of the first things to realize is that during the whole Live Response no files must be created on the evidence system at all, because this is certainly an unnecessary modification of the system's hard disk. This is why normally only command-line tools are used, because creation of files can be controlled pretty well. A common way to record the output of the commands issued on the running system is to connect it with a network cable to a forensic workstation, e.g. a laptop, and then send the output of a command over the network to the forensic workstation, where it is stored. All this can be done using the network tool `netcat`, which is first used to make the workstation listen to incoming connections on a specific port and store all incoming data in a file:

- forensic workstation: `netcat -v -l -p port > $COMMAND.txt`

Then the command that corresponds to `$COMMAND`, e.g. `ps` is executed on the evidence system and its output is sent via the network to the forensic workstation.

- evidence system: `$COMMAND | netcat forensic_workstation_ip port`

After the whole output has been sent to the forensic workstation, the connection can be terminated and the file `$COMMAND.txt` will contain the exact output of the command executed on the evidence system. In unsafe environments, it is probably better to use an encrypted version of `netcat` such as `cryptcat` [6], which can be used exactly like `netcat` and has the additional benefit of providing strong encryption for the data transmitted. Now that a safe way to store the output of Live Response tools without modifying the evidence hard disk has been established, the executable files themselves have to be prepared.

First of all, no binary that is already installed on the evidence system should be used at all, because it might be modified to return false information, crash the operating system or worse. This is why all Live Response tools have to be trusted tools, stored on read-only media, which are then run from the media during the process. In addition to this, most executable files on Unix systems are *dynamically linked*, which means that certain standard libraries are loaded from the operating system at runtime to reduce the file size of the binary. This effect is undesirable during a Live Response for two reasons:

- Accessing the libraries on the evidence system changes the file timestamps on those files
- All files on the evidence system may potentially be modified in a malicious way

To avoid access to the library files stored on the evidence system, all executables should be *statically linked*, i.e. that all libraries needed to run the binary are compiled into the binary file and it does not need to access additional files at runtime. To obtain static binaries of all the Live Response tools, an investigator will most of the time have to download the source code of the tool and manually perform a static compilation of the binary. A good description of how to compile statically can be found in [75], if problems occur then usually trying different modifications of the accompanying "Makefile" will succeed. If it is not possible at all to compile a static binary of the tool, then an alternative should be considered; if there is none, then the `ldd` command can be used to find out which libraries the binary loads at runtime, and these libraries should then be copied to the Live Response CD as well.

Having prepared the Live Response toolkit, the tools can then be used to collect data and transmit it to the forensic workstation as described above. To alleviate the process, the next section will describe a Live Response script (adapted from [75]), that automates this process.

Linux Live Response script

To show how the process of Live Response data collection can be automated, this section will introduce a Live Response script, that has been introduced in [75], see Appendix A.4. This bash script called `ir-script-linux.sh` is designed to be run with root privileges from a CD which contains all trusted binaries and eventually needed libraries in separate folders:

- `# ls -l`

-
  ```
  drwxr-xr-x  59 root  root  2006  17 Jul 21:30 bin
  -rwxr-xr-x   1 root  root  4672  17 Jul 21:30 ir-script-linux.sh
  drwxr-xr-x  14 root  root   476  17 Jul 21:30 lib
  ```

All the trusted binaries are renamed by using the prefix `t_`, e.g. the `date` tool will be named `t_date`, to avoid mixing up the trusted tools with untrusted ones from the evidence system. After defining some variables, the script then prints out a header for each command, executes the command using a trusted `bash` shell (`bin/t_bash`) and then prints out a footer afterwards. The complete output of the script can then be transferred to the forensic workstation with `netcat`:

- forensic workstation: `netcat -v -l -p port > live_response_data.txt`
- evidence system: `./ir-script-linux.sh | ./t_netcat forensic_workstation_ip port`

Scripts like these represent the current standard in automated Linux Live Response, as they collect data with a minimum of modification to the evidence operating system and with relatively little effort. Similar scripts could be designed to work on BSD, Solaris or Mac OS X operating systems, since not all commands are available for every platform, or some commands use different command line options.

More thorough investigation of the evidence system, such as performing main memory dumps or similar techniques should rather be done manually, since although an error in the Live Response script will normally only trigger an error message and nothing more, an error when reading or working with main memory can easily crash the running operating system. Therefore before trying more advanced data collection techniques, such a script should be used to safely collect the more easily obtainable information.

4.2.2 Windows Live Response

This section introduces common methods to perform Live Response on Microsoft Windows operating systems, focussing on systems at least as recent as Windows 2000, e.g. Windows XP and Windows 2003 Server. Generally tools for one of these platforms will run on all of them, but because Microsoft continuously upgrades their operating systems, this can not be promised for all tools used in this discussion. A lot of the tools used during Windows Live Response can be obtained from the Sysinternals website [46], which offers an excellent selection of free commandline tools, in particular the PsTools suite [43].

Volatile data

Generally the volatile data that is usually collected during a Windows Live Response is not very different from the data collected during a Linux Live Response, except for a few special information specific to Windows, e.g. the concept of "services". Not all information can be collected using built-in tools alone, therefore several free third-party tools are used:

- `time, date`: prints out system date and time
- `netstat -an`: prints out a list of current network connections and open TCP/UDP ports
- `netstat -rn`: prints out the internal routing table
- `doskey /history`: prints out command history of current shell
- `fport` [16]: prints out a list of executables opening TCP/UDP ports
- `nbtstat -c`: prints out a list of current NetBIOS connections
- `psloggedon`: prints out a list of users currently logged on (PsTools)
- `pslist`: prints out a list of running processes (PsTools)
- `psservice`: prints out a list of running services (PsTools)
- `psfile`: prints out a list of remotely opened files (PsTools)
- `at`: prints out a list of scheduled jobs
- `ipconfig /all`: prints out current network configuration

These data can be used in the same way as already described for the Unix/Linux case, by trying to link open ports to running processes, network connections etc.

Like in the Unix case, there are some advanced techniques for data collection on live Windows systems too, which mainly concern capturing contents of the system's main memory. The tool `userdump.exe` [32] can be used to create a "dump" file for a running process, which contains the contents of the memory a process occupied in main memory. Because the output of `userdump.exe` can not be sent via the network with netcat directly, a network share should be created on the forensic workstation which can then be accessed from the evidence system to store the dump file. Dump files can be examined in a lot of ways, for example with the Microsoft utility `dumpchk.exe` [31] or by using `strings` [45] to extract all human-readable strings in the file.

Apart from these process memory dumps, it is also possible to perform full system memory dumps with the help of a special enhanced `dd` [15] version for Windows, which has been modified to be able to capture main memory. Such a full system memory dump can often contain data that would not be recoverable even with Live Response techniques, e.g. residual data from terminated processes or destroyed files (see [75] for a detailed example). However, before attempting to use these advance techniques, it is better to capture all the data using the basic tools so that they are not lost in case of a system crash.

Non-volatile data

Non-volatile data collected from Windows systems again resembles the data that can be collected from Unix/Linux systems, with a few additions and of course special commands to retrieve the various logfiles. Some of the tools are part of the Windows NT Resource Kit (NTRK) [33], which is sold by Microsoft and includes some useful tools for Windows administration and management. Non-volatile data that should be captured from a Windows system during Live Response include:

- `psinfo -h -s -d` : obtain system version and patch level
- `find c:\ -printf "%m;%Ax;%AT;%Tx;%TT;%Cx;%CT;%U;%G;%s;%p\n"` prints out file name, MAC timestamps, ownership information and file size of every file in the file system, using `find` from the UnxUtils [58], a set of Unix utilities ported to Windows
- `regdmp.exe`: prints out the registry hive file (included in NTRK)
- `auditpol.exe`: prints out information about the auditing policy of the system (NTRK)
- `ntlast`[16]: prints out a history of logins
- `psloglist -s -x [security|application|system]` : prints out the security/application/system event logs (PsTools)
- `pwdump` [35]: prints out user account information, including password hashes
- possibly transfer any suspicious file
- transfer available application-specific logfiles

Depending on the auditing policy, logfiles can be avaluable source for information, as well as application specific logfiles like the IIS [24] logfile of a web server, which is often targeted in computer intrusion attacks. Since the registry file of a Windows system is the central file which controls administration of the system, it should also be reviewed for obvious signs of manipulation, e.g. a lot of pieces of malware create additional services in the `HKLM\Software\Microsoft\Windows\CurrentVersion\Run` registry key.

The combination of volatile and non-volatile data gathered from a Windows system can often help a lot in the reconstruction of an incident and its containment, just like with Unix/Linux systems.

Preparing a Windows Live Response Toolkit

This section will describe how to prepare a Live Response Toolkit for Windows, trying to minimize the impact of Live Response activities on the running system, and to avoid using untrusted resources on the evidence system. First of all, output of commands can be transferred from the evidence system to the forensic workstation using a Windows version of `netcat` [34] in almost the same way as in the Unix case:

- forensic workstation: `nc.exe -l -p port > $COMMAND.txt`
- evidence system: `$COMMAND | nc.exe forensic_workstation_ip port`

This allows to review the response tools' output on the forensic workstation without unnecessary modification of the evidence system.

Regarding the preparation of the executable files themselves, there is no equivalent way to static compilation of the binaries like on the Unix platform, particularly because the sourcecode for the Windows tools is rarely publicly available. Most Windows executables make use of shared libraries, which are called *Dynamic Link Libraries (DLL)* in Windows and contain pre-compiled functions that can be used by several different programs just by loading the respective DLL. Usually there are hundreds or even thousands of different DLL files on a standard Windows installation. When a program is started, it loads parts of the DLL files into his own memory space and is able to use the functions the DLL provides as if they were included in the executable file itself.

To avoid access to possibly manipulated DLL files on the evidence system by the Live Response tools, all DLL files that each program calls upon execution have to be included in the same folder as the executable file. This solves the problem because any DLL files called will first be searched for in the current folder, and afterwards in the operating system's `C:\Windows\System32` folder (on standard installations), which is to be avoided.

To find out which DLL files are called by a certain executable, the tool Filemon [42] can be used. Filemon monitors and logs any system call that involves accessing, reading from and writing to a file on the hard drive, and it will thus show any DLL file that an executable reads from the evidence system's hard drive. The methodology to determine an executable's dependencies would then be [78]:

- Start monitoring with Filemon
- Run the executable
- Stop monitoring and record the DLL files that were accessed

All DLL files logged should then be copied into the same folder as the Live Response tool. Double-checking with Filemon to confirm that all necessary DLL files have been copied is a good idea. Repeating the process of monitoring with Filemon and copying the DLL files for every tool in the Live Response toolkit will result in a large number of DLL files along with the execuables in one folder, which can then be copied to a Live Response CD.

There is one thing to watch out for when preparing or using Live Response tools on systems with Windows XP or newer installed: the *Windows File Protection (WFP)* [30]. WFP is a mechanism to protect certain critical system files, e.g. some core operating system components like `Kernel32.dll` or `NTDLL.dll` from tampering. Whenever any executable is run, these core DLL files are touched, i.e. opened, but not read or modified, which updates these files' last access times. These accesses will be shown by Filemon when monitoring an executable, but copying the DLL files to the exectuable's directory will not stop this (see [75] for more details). For the most part, these accesses can be safely ignored, but an investigator should still document the changes done to the access times of those DLL files on the evidence system.

Having prepared the tools and assembled the required DLL files, the toolkit can now be used with a Live Response script to collect data in an automated fashion.

Windows Live Response Script

The Windows Live Response script introduced in this chapter has been proposed in [75] and represents the current standard in automated live data collection for Windows systems using open source or free tools. The batch-file `ir-script-win.bat` (Appendix A.5) has to be run with Administrator privileges from the Live Response CD, from a directory which contains the script, all executables and any DLL file identified in the previous preparation of the toolkit. As usual the files are prefixed with a `t_` to avoid confusing them with programs installed on the evidence system (X is the drive letter assigned to the drive the Live Response CD has been inserted in).

- `X:\ dir`

-
  ```
  07.07.2006  22:41    <DIR>          .
  07.07.2006  22:41    <DIR>          ..
  07.07.2006  02:45             3.365 ir-script-windows.bat
  07.07.2006  04:00            23.040 psapi.dll
  07.07.2006  04:00           401.408 t_cmd.exe
  07.07.2006  03:07            61.440 t_nc.exe
  [...] (truncated)
  ```

The script should then be started from the CD using the trusted `t_cmd.exe`, and its output is transmitted via a network to the forensic workstation.

- forensic workstation: `nc.exe -l -p port > live_response_data.txt`

- evidence system: `ir-script-win.bat X | t_nc.exe forensic_workstation_ip port`

Using this script, a large amount of valuable live data can be easily collected from a running system with as little modification as possible.

Advanced techniques to acquire process dumps or even full memory dumps should not be used in such an automated approach, but rather manually after the Live Response script has been run. This will allow to gather the volatile and non-volatile information the tools in the script provide, before attempting the memory capturing techniques that always involve the risk of crashing the operating system.

4.3 Forensic Flaws in Current Practice

As it was mentioned before, the central paradigm of Live Response is that any modification to the evidence system while collecting data has to be kept to a minimum. Only if the possible changes of a certain method for live data collection are small, well-understood and properly documented, its use can be justified from a forensic, and especially, from a legal standpoint.

To meet this high standard, every method or tool used should be carefully analyzed to assess its potential to alter evidence. In the preceding chapter, Live Response scripts proposed in [75] were used for both Windows- and Unix-based operating systems to automatically collect data from a running system. Given the fact that the authors of this work are renowned forensic experts, these scripts can be considered Best Practices when it comes to Live Response. When analyzing the scripts' effects on live evidence systems however, it turned out that there were forensic flaws in them, concerning both the modification of the original evidence and the dependability of the results of the Live Response process.

4.3.1 File metadata collection

The Linux Live Response script uses the following command to print out a list of all files with their metadata:

- `./t_find / -printf %m;%Ax;%AT;%Tx;%TT;%Cx;%CT;%U;%G;%s;%p\n`

When called like this, the `find` command will list all files in the file system and print out the respective file metadata, i.e. the permissions, MAC timestamps, used and group ownership, file size and file name. The problem is that `find`, when it recursively traverses the file system, accesses all directories on the way, thus updating their A-times, i.e. the last access times. This flaw also applies to the Windows version of the Live Response script. Certainly altering the last update time of all directories is an undesirable modification of the evidence system's hard disk.

Collecting the file MD5 checksums of all files in the file system is also done usind the `find` command:

- `t_find / -xdev -type f -exec t_md5sum {} \;`

Again, `find` recursively traverses the file system, this time executing the md5sum command on each file to compute its MD5 hash checksum. This actually updates the A-time of every directory and every file that is hashed, basically leveling the whole last access time information on the file system. When testing the effect of the script on a Linux workstation, it turned out that a combined 96% of all A-times where updated when using this script. This is a serious modification of the original evidence that should absolutely be avoided.

It should be noticed that when find is used as above, the find command will record the "right" A-time of the directories and files *before* updating it, that means that although the A-times are updated by `find`, the correct times are recorded in the Live Response script output. Nevertheless this modification can lead to problems, for example when extracting file metadata later during dead analysis in the Harvesting step. Together with the Live Response script output, there will then be a discrepancy regarding the A-time information, because the "real" A-times are only recorded in the Live Response script output. An opposing legal party will certainly question the validity of the original timestamps, since they do no longer exist in the duplicated disk image, i.e. the principle of repeatability of the analysis is violated. This flaw could then be used to argue that the whole Live Response procedure was not forensically sound and altered the evidence system in such a way, that the dead analysis results cannot be relied on either. By discrediting one small part of the analysis process, the complete analysis results' credibility can be ruined.

4.3.2 Rootkit Detection

Whereas the file metadata collection issue raises questions about the forensic validity of common techniques in Live Response, the question of rootkit detection touches the reliability of live data collection. A *rootkit* is a program or set of programs that allows an attacker to maintain an unnoticed presence on a compromised system, enabling him to hide files, processes, ports, drivers and so on, often including a backdoor for continued access. The earliest and least sophisticated rootkits that were observed simply consisted of a set of manipulated standard tools like `ls`, `rm`, `mv`, `ps`, `netstat` etc. that replaced the original versions of the same tools, filtering the output to hide certain information. To beat this kind of rootkits, using trusted tools from a Live Response CD as described earlier is already enough, because this practice does not rely on untrusted tools.

Recent rootkits use methods far more advanced than the simple command replacement technique to subvert the operating system. Since rootkits will be discussed in detail in chapter 5, it suffices to say at this point that even using trusted tools from a Live Response CD will not guarantee accurate results when run on a system infected with a modern rootkit. This means that the output of virtually every Live Response tool mentioned so far in this thesis can be manipulated; files, open ports, network connections, processes and services, login sessions can all be hidden or shown in an altered way. In particular the files and processes belonging to the rootkit will be hidden, making its detection more difficult. The bottom line is, when using the Live Response script on a running system that is infected with a rootkit, and the investigator is not aware of this, output of Live Response tools cannot be trusted and must be regarded as completely unreliable.

One could argue that although a rootkit can hide certain information on a running system, a dead analysis will always be able to recover all information that the rootkit hid, and it will also reveal the presence of the rootkit. First of all, any volatile data that is manipulated is not reconstructable during a dead analysis, so the otherwise valuable information about running processes, open ports etc. is lost, unless special techniques to correct the rootkit's manipulations are used. Secondly, rootkits exist that leave no trace on the file system at all, running completely in main memory; these rootkits will be undetectable on a file system image, and if their presence is not noticed during Live Response, there is a severe danger that an investigation will be using false data collected during Live Response without ever knowing.

4.4 Improvements

4.4.1 File metadata collection

It was shown that collectiong file metadata or hash checksums from a live system using a recursive `find` command greatly modifies the original evidence and has serious implications. Ironically, a technique used in dead analysis can be adapted for live analysis to avoid these complications.

When discussing metadata collection during the Data Recovery step of the Analysis Phase, it was shown how open source tools from the Sleuthkit and a small Perl script can be used to extract all file metadata information from a forensic image of the evidence system's hard disk:

- `fls -r -p -l -m / -f file_system_type partition_image >`
  `➥ part_fls.txt`

- `./getmetadata.pl part_fls.txt file_system_type partition_image >`
  `➥ metadata.txt`

While experimenting with the `getmetadata.pl` script, the result was that it can just as well be used to collect metadata from a live system instead from a partition image. The `getmetadata.pl` script has to be slightly modified to account for the fact that the output of the `fls` command can not be stored on the evidence system's hard drive. In addition, since the command is executed on a untrusted operating system, trusted statically linked versions of all commands, including the `perl` interpreter, are used.

Working with the new `getmetadata-live.pl` (Appendix A.6), which is placed into the same directory as the Linux Incident Response script `ir-script-linux.sh`, the file metadata would now be collected with the following command, where `/dev/raw_parttition` stands for the raw partition device file of the partition which contains the file system in question:

- `./bin/t_fls -r -l -p -m / /dev/raw_partition | ./getmetadata-live.pl`
  `➥ file_system_type /dev/raw_partition`

The code in ir-script-linux.sh that collects the metadata and MD5 checksum information can thus be removed from the script. Because the name of the raw partition device and the file system type may be different from system to system, the above command has to be issued manually in a trusted shell on the evidence system, and its output has to be transmitted to the forensic workstation using netcat as usual. In this way, the original Live Response script output file will not contain any file system information, but a seperate file containing the output of the command above will be available.

Like it has already been said, using the original ir-script to collect file metadata and MD5 checksums of all files in a file system resulted in approximately 96% of all A-times being updated. When doing the same experiments with the new method using getmetadata-live.pl, only about 0,07% of all A-times were updated, which can be attributed to normal background operating system activities. By comparing the output of both methods on a real Linux workstation, it could also be shown that the outputs of both methods are absolutely equivalent. Thus, the new technique has greatly reduced the impact of the Live Analysis activities on the original evidence while retaining its reliability.

4.4.2 Rootkit Detection

As explained above, rootkits can severely limit a Live Response procedure's reliability and its result can become worthless. Therefore detection of rootkits should be performed to find out whether a rootkit is installed; if a rootkit is detected, an investigator can decide to perform the Live Response anyway, skip it altogether or he can try to use advanced techniques that may be able to overcome the rootkit's subversion. In any case, rootkit detection can provide the investigator with more information about an incident and enables him to make an informed decision about how to proceed.

On Unix-based systems, the tool freeware chkrootkit [5] can be used to check a a running systems for signs of a rootkit infection; currently chkrootkit can detect 60 different rootkits. It consists of a number of executable files and makes use of some external binaries, which of course have to be trusted version from the Live Response CD. To be able to use chkrootkit, the tools awk, cut, echo, egrep, find, head, id, ls, netstat, ps, strings, sed and uname are necessary. These programs are compiled into static binaries and stored on the Live Response CD, along with the statically compiled chkrootkit binaries (see [4] for build instructions). Once the Live Response CD has been prepared like that, the following code can be added to ir-script-linux.sh to perform rootkit detection as part of the Live Response process:

```
CHKRK="$BIN_PATH/chkrootkit"
## Scanning for rootkits

console "- running chkrootkit..\n"
echo "# Running chkrootkit #"
$CHKRK -p $BIN_PATH
echo "## END ##"; echo ""
```

The output of `chkrootkit` is then included in the Live Response output file created by transfering the script's output to the forensic workstation with `netcat`. If a rootkit is detected, the investigator will have the opportunity to reassess the reliability of the collected information; generally he should lower his trust into the results and not take critical decision based on the Live Response data, since they could be manipulated by the rootkit.

On the Windows platform, several rootkit detection tools exist, yet none of them can be easily integrated into the Live Response script like in the case of Linux. Chapter 5 will develop a detailed methodology for rootkit detection on Windows, therefore the discussion of these techniques is postponed at this point. It should just be said, that rootkit detection on Windows will involve using multiple tools with a GUI (Graphic User Interface), because adequate command line tools to fulfill the task do not exist yet.

4.5 Discussion

This chapter introduced a methodology for Live Response which represents the current standard in collecting data from live systems. Live Response is the only way to preserve certain pieces of volatile information from an evidence system, e.g. a list of open ports, of all running processes or the exact system time. In addition to this, some non-volatile information is also collected, even if they could also be recovered from a disk image of the system's hard drive during dead analysis. Their collection during Live Response is justified by the fact, that very often live collection is considerably easier, e.g. in the case of login logfiles in a specific format, which can easily be collected using the `w` command, whereas parsing the file during Data Recovery later would be quite involved. On top of that, the sooner crucial information about the incident is known, the sooner an Incident Response Team can take appropriate measures to contain the incident to prevent further damage.

Because all Live Response activities take place on an actually running system, there are two main concerns regarding the validity of Live Response practices:

- Applications run on a live system can not be trusted because the system might be compromised
- Running applications potentially alters the live system's state

To address the first issue, all tools during the process are *trusted* tools stored on a read-only Live Response CD. In the case of Unix-based operating systems, statically linked binaries are compiled to aviod using possibly compromised library files on the evidence system; for the Windows platform, the required DLL files for all the executables are also stored on the CD.

The second issue can not be taken care of as easily, because there is no way to completely prevent applications run on a live system from modifying the system's state. To minimize the impact of live data collection procedures on the evidence system, creating files on the evidence system's hard drive is completely omitted, all output is transferred to a forensic workstation via a network connection. All tools and methods have to be carefully examined to assess the extent of their modification of the evidence system.

This thesis proposed two improvements to current practice, the incorporation of rootkit detection into Live Response and an alternative way of collecting file metadata and hash checksums. It was shown how undetected presence of a rootkit can undermine trust into the results of Live Response, and even the whole investigation. For Unix-based systems a modification to the Live Response script was introduced which integrates rootkit detection into the already well-established process of Live Response; the next chapter will propose an equivalent approach for Windows-based systems.

In the case of file system metadata collection, experiments were conducted with current best practices that resulted in detection of a flaw in the data collection method: the unwanted modification of file timestamps. This issue has been addressed by modifying an already known method for metadata collection during dead analysis and showing that it does not share the same flaws, thus increasing the forensic integrity of file metadata collection.

Together, these two changes to current Live Response procedures result in a more reliable and reasonable way of conducting automated data collection on live hosts.

4.6 Summary

This chapter gave a detailed survey of data collection from live systems, called Live Response. By introducing the notion of volatile and non-volatile data, the motivation to conduct Live Response was given, while also discussing possible side-effects of working on live evidence systems and weighing them up against the advantages.

Afterwards current Best Practices for Live Response procedures were given for both Windows and Unix-based operating systems. During the preparation of an appropriate Live Response toolkit, accepted ways of mitigating the risk of modification of the original evidence were used, resulting in a toolkit of trusted tools that keeps changes to the live system to a minimum. Live Response scripts for both platforms were introduced, which are based on scripts proposed in [75].

Afterwards two weaknesses in the methodology were discovered; the first one concerned the collection of file metadata and hash checksums, where the method used severely altered the original evidence by updating the last access times of all files and directories. This made it impossible for an investigator or an independent expert to arrive at the same results in a later phase of the analysis, thus violating the principle of repeatability of the analysis. The second weakness that was discussed is the missing rootkit detection routines in the Live Response procedures. It was shown how a rootkit infection can seriously distort live data collection results and thus make the whole work worthless.

To adress the improper file metadata collection, an already known way to collect these information has been adapted to be applicable to live data collection. In order to integrate rootkit detection into the Live Response procedures, the Live Response script for Linux has been updated to include the use of a special rootkit detection tool, `chkrootkit`. A solution for rootkit detection on Windows systems will be developed in the next chapter, where the topic of Windows rootkits is explored in more detail. Implementing these changes in the Live Response practices has improved the overall reliability of the results and has reduced the inherent forensic flaw of Live Response, the modification of the evidence.

Chapter 5

Windows Rootkits

5.1 Introduction

This chapter will take a detailed look at rootkits for Windows operating systems, their basic principles and techniques, as well as ways to detect rootkits despite their stealth capabilities. A rootkit can be defined as "a set of programs and code that allows a permanent and undetectable presence on a computer" [73]. Typically rootkits are used to hide certain files, processes and other information on a running system, making them one of the most dangerous classes of malware, because they can exist on a compromised system unnoticed for years. Additional functionality like a backdoor or keyboard sniffers can often be added, turning rootkits into the ultimate attacker's tool. By examining current rootkits and detection tools and putting them to the test, it will be shown how rootkits can nevertheless be detected, resulting in a methodology to safely detect publicly available rootkits.

5.2 Background: Windows Kernel and Rootkits

To understand how rootkits work it is first of all important to understand the basics of the Microsoft Windows kernel architecture and the way the operating system works beyond the normal graphical user interface. The techniques used by rootkits to subvert the operating system are fundamentally different from the way normal software operates. Often they manipulate internal operating system tables that are used to control the execution flow of other software on the system, causing the rootkit code to be executed instead of the original code. In other cases rootkits will filter or forge the output of system calls to hide themselves or other objects and yet another class of rootkits modifies memory contents directly to fool the operating system.

Because of the complexity of the underlying technology, it will hardly be possible to explain all details of the operating system mechanism completely. Therefore this introduction will only focus on a few central concepts that allow presentation of the most common rootkit techniques, namely *Hooking* and *Direct Kernel Object Manipulation*. For detailed description of Windows operating system internals, refer to [89] or [73]; the latter can also be considered to be a reference work on rootkits.

5.2.1 Windows Internals

Access control at hardware level

The first important concept to understand is how *access control* is realized on the x86 processor family, which is the processor type used in most personal computers today. In this context, access control means how the hardware controls which kind of instructions may be used by a particular process, which areas of memory or files may be modified, and how hardware components may be accessed and written to. Without access control at the hardware level, any process would theoretically be allowed to access and modify every file, alter the memory contents belonging to another process and issue every CPU instruction, no matter whether that could disturb other processes or even crash the whole system.

The x86 processor type has provided the capability for access control for a long time, in the form of four privilege levels called *rings*, where *Ring 0* is the most privileged and *Ring 3* is the least privileged (see figure 7). Since Windows NT, Microsoft operating systems use the two rings Ring 0 and Ring 3 for access control. All user-mode programs, i.e. most normal applications, run in Ring 3 and have only limited access to memory, and cannot access hardware directly; this will be prevented by the CPU itself. Ring 0 contains all software that runs with full system priviliges, for example the Hardware Abstraction Layer (HAL), device drivers, IO and memory management – essentially all components of the operating system kernel. Code running in Ring 0 can access the whole memory space and is able to execute privileged instructions, e.g. allowing access to hardware components like graphic cards or the keyboard. Programs running in Ring 3 are often called *userland programs* and those running at Ring 0 are called *kernel programs*.

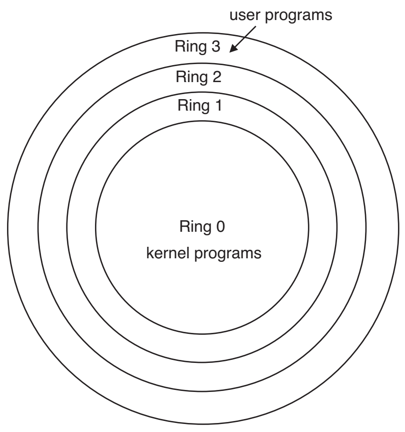


Figure 7: Ring architecture of the Intel x86 processor family

Since even userland applications may have to access hardware from time to time – e.g. when using a system administration programm to install new device drivers for a graphics card – special mechanism exist that allow a userland program to cross the barrier to Ring 0 and use the otherwise unavailable functions in a controlled way. Some rootkits exploit this capability to plant their own device driver (a `*.sys` file) containing their malicious code in Ring 0, allowing them to run at full privilege level; these rootkits are also called *kernel rootkits*. It is important to notice that any Ring 3 detection tool will have a large disadvantage against a kernel rootkit, because the two are basically not competing with each other on equal terms. Because a Ring 0 rootkit can control the environment in which other software runs, it can achieve stealth and avoid detection in a very effective way.

The System Service Dispatch Table

Whenever a userland program wants to use a function that is only available to Ring 0 programs, it issues a so-called *system call*. A system call interrupts execution of the user program and transfers execution control to the kernel, which will then process the requested service as indicated by the system call number stored in the CPU register `EAX`, possibly using a set of input parameters received from the userland application. After the system call has been processed, execution of the userland program is resumed, which can now use the system call's results for further operation.

On Windows, there is a large number of these service functions provided by the kernel, and each function is identified by a unique system call number. Whenever a system call is made, a special kernel routine called `KiSystemService` is executed in Ring 0, which reads the system call number from the `EAX` register and looks up the matching function in a table, the *System Service Dispatch Table (SSDT)*. The SSDT contains the memory addresses of all functions that correspond system calls, making it one of the central points of execution flow control in the kernel. Thus the SSDT naturally represents an opportunity for malicious modifications by a kernel rootkit; the technique for SSDT manipulation will be described in more detail later.

5.2.2 Hooking Techniques

Altering the execution flow of programs is one of the most prominent techniques of rootkits to subvert the Windows operating system. By having code that belongs to the rootkit execute instead of the original, a rootkit can manipulate the operating system and the programs that are run on it to achieve stealth, i.e. operate completely unnoticed and hidden from the user. The various techniques of rerouting the execution of programs are commonly referred to as *Hooking*. It should be noted that often legitimate software uses Hooking techniques too, so the presence of a hook on a system does not automatically mean that a rootkit is installed.

IAT Hooking

The easiest form of hooks manipulate the way in which individual applications are executed, in particular how they import library functions, that are typically provided on Windows operating systems in the form of DLL files. Whenever an application is run, a list of required library files included in the application executable file is used to load the matching library files into the application's memory space completely. Apart from the list of required DLL files, the executable also contains a structure that stores the specific functions inside the respective library files that the application needs. These functions' locations in memory are determined and a special table called *Import Address Table (IAT)* is loaded with the library functions' names and the corresponding address of the function in the application's memory space.

A rootkit that has access to an application's address space can manipulate the IAT in order to make its own code execute instead of the original library function. It will do this by overwriting the address of the IAT functions with the address of the rootkit function's location in memory space, allowing the rootkit to manipulate the parameters or output of the original function. There are a number of userland rootkits that will use IAT hooking to manipulate the IAT of every application running on a system, which enables them to hide their presence globally across the whole operating system. Figure 8 shows how IAT Hooking alters the execution path of a program.

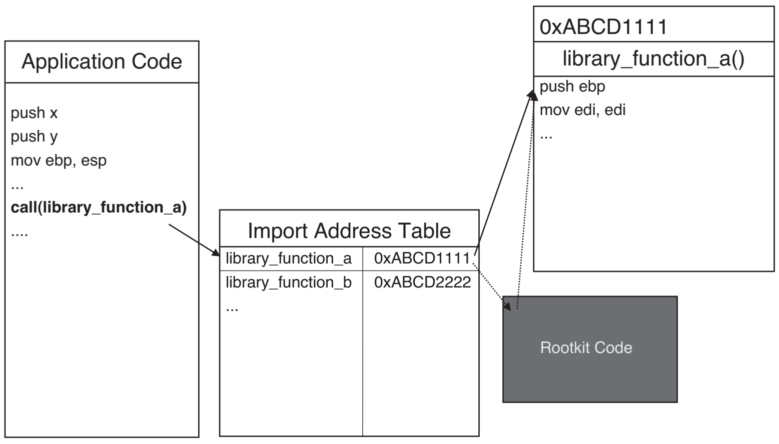


Figure 8: IAT Hooking – The solid line represents the normal execution path, the dotted line shows the hooked path

Inline Function Hooking

A slightly more elegant hooking technique does not overwrite the IAT address of the library function to be hooked, but instead modifies the function's code directly in memory. Such an *inline function hook* will usually patch the first few instructions of the hooked function with a special jump instruction to the rootkit code, which will then be executed before the original code. Usually the rootkit code will then call the original function, because when execution of the original function has completed, control will return to the rootkit code, allowing to modify the results of the library function. Inline hooks have several advantages over standard IAT hooks, mainly because the technique avoids problems when the function is not called using the IAT, because no matter how a process calls the respective function, the inline hook is always effective.

SSDT Hooking

Whereas IAT and Inline Function Hooking all take place in the userland realm, there are also techniques to hook execution in the kernel. As previously explained, one of the central tables that control execution of kernel functions is the System Service Dispatch Table. Whenever a system call is performed, the kernel function `KiSystemService` takes control over the execution and looks up the memory address of the kernel function corresponding to the system call number in the SSDT.

A rootkit can subvert this mechanism by exchanging the original function's address with the address of the rootkit's hook function. The hook function can then imitate the original function but choose to filter out certain information, e.g. a function that reports a list of open ports could be replaced by a similar function that does the same but does not display a certain range of ports, which is used by a built-in backdoor of the rootkit. Overwriting the addresses in the SSDT is generally a lot harder than IAT Hooking, but the details have intentionally been left out in this context to be able to just explain the rough concepts. A detailed description of the method can be found in [73], figure 9 is a simplified representation of the technique.

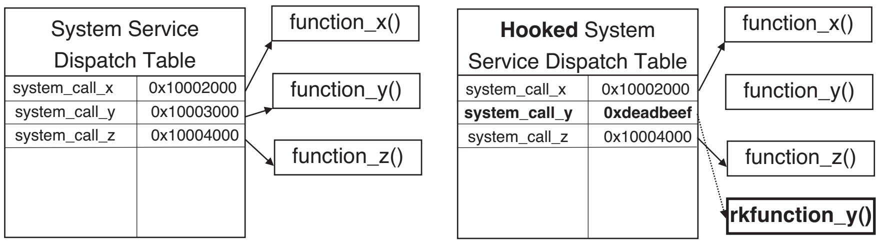


Figure 9: SSDT Hooking – left: before hooking, right: after hooking

5.2.3 Direct Kernel Object Manipulation

Hooking techniques are a solid technique of subverting an operating system, but their main drawback is, that a hook can usually be detected and there are several tools for hook detection, some of which will also be introduced in later sections. An advanced technique for subversion, that e.g. allows to hide processes, directly manipulates the very data structures in kernel memory that keep track of the state of the operating system, therefore it is called *Direct Kernel Object Manipulation (DKOM)*.

Windows keeps a number of undocumented data structures in kernel memory which for example contain a list of running processes, of threads scheduled for execution etc. Part of DKOM techniques is therefore trying to understand the structure of these kernel objects to be able to manipulate them without making the operating system unstable or even crashing it completely. The technique then consists of careful modifications to the in-memory data structures to hide certain objects from the user. For example, processes are recorded in a doubly-linked list, so that if the offset of the forward and backward pointers to the next or previous process in the list is known, it is possible to exclude a process from the list with simple pointer reorganization. Because the scheduling of execution of processes does not depend on a process being present in that list, this technique hides the process successfully (e.g. from the Task Manager), but the process is still executed unnoticed (see figure 10 for a simplified model of the process hiding technique).

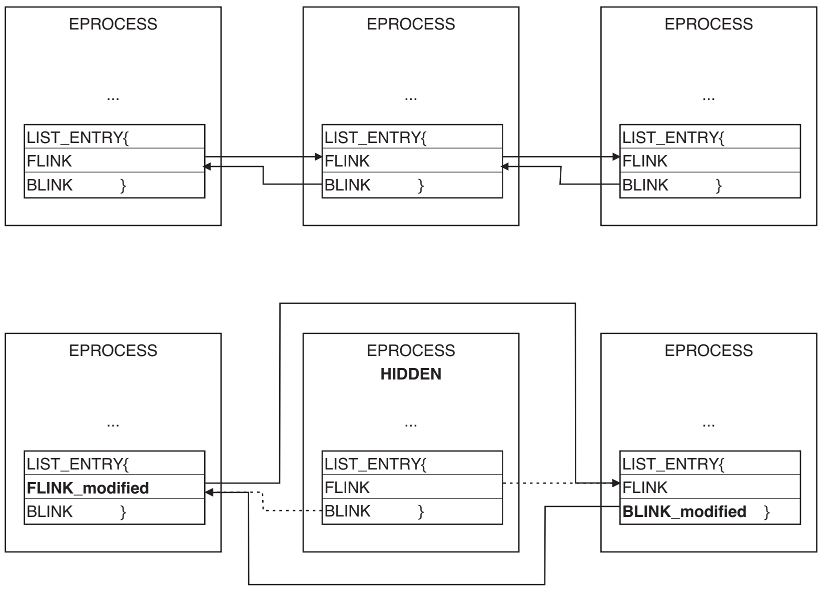


Figure 10: Process Hiding with DKOM [73]– The upper figure shows a part of the original linked list of processes, the lower figure shows how the middle process is hidden by pointer manipulation

DKOM has certain benefits over Hooking, especially because it is very hard to detect, since directly modifying the raw main memory contents with a Ring 0 rootkit can not be controlled by any built-in security mechanism in Windows. The disadvantage of DKOM is its extreme instability, since there is no documentation about the actual structure and usage of the kernel objects by the operating system, and often even minor operating system upgrades change the way they look like and how the operating system uses the kernel objects. Nevertheless DKOM has become a concept that has to be considered one of the most sophisticated and dangerous rootkit technologies that exists.

5.3 Overview of Rootkits and Detectors

5.3.1 Rootkits used

For the experiments on rootkit detection, a number of publicly available rootkits were used, all of which can be obtained via the rootkit.com website [38].

This includes userland rootkits as well as kernel rootkits, and rootkits which use Hooking techniques as well as some that use DKOM to hide their presence. The userland rootkits included in the experiments were AFX Rookit, Hacker Defender (also called HxDef), He4Hook, NtIllusion and Vanquish. All of these rootkits offer about the same capabilities of hiding processes and files, sometimes drivers or services; the details will be described in the next section.

Kernel rootkits included in the experiments were FU, FUTo, phide and HideProcessHookMDL (HPHM), where the latter is more of a proof-of-concept device driver that allows to hide processes by using SSDT Hooking. It was chosen nevertheless because no other rootkit could be included that uses this technique. FU, FUTo and phide all use DKOM techniques to hide processes, FU and FUTo even offer functions to elevate a process's privileges and also alter other kernel objects than the linked list of processes.

There were also some related pieces of software, namely cfsd and Klog, which were included in the trials although they are not rootkits, because they use very similar techniques. Cfsd is a file system driver, which can be used to misrepresent the contents of a file system, for example allowing to hide certain files. Klog is a device driver that acts as a keyboard sniffer, i.e. it logs everything typed into the keyboard into a file without being noticed.

5.3.2 Detection tools used

After selecting the rootkits for the experiments an assortment of rootkit detection tools was assembled to be able to test the various technologies used by rootkits against the different kind of detection tools. Among them were two tools that explicitly look for hooks, VICE [38] and Respendence Software Rootkit Hook Analyzer (RKHA) [36]. VICE detects both userland and kernel hooks, and can discover normal IAT hooks, inline function hooks, SSDT hooks and also more advanced hooks, while Rootkit Hook Analyzer only detects kernel hooks.

Another class of detection tools directly looks for hidden objects on the running operating system, the ones used for this thesis are F-Secure Blacklight [12], RKDetector [37], Rootkit Revealer [44], and UnhackMe [18]. They are generally run once and the output is a list of all hidden objects found. Blacklight can detect hidden processes and files, RKDetector and UnhackMe detect hidden processes and will sometimes also alert because of suspicious services and corresponding registry keys. Rootkit Revealer can detect hidden files and registry keys, including those belonging to a system service, making it possible to also detect hidden services indirectly.

It is known that Blacklight and Rootkit Revealer use a *cross-view detection* approach, which identifies hidden objects by comparing the output of high-level API reporting functions with results gained from parsing the appropriate structures on a very low level. For example, to find hidden files, a cross-view detection tool would first use the Windows API to request information about the contents of the file system. It would then use low-level methods of parsing the file system itself, and compare the output with the API output, uncovering any files that were hidden at API level. A cross-view approach only works if the subversion of the rootkit does not manipulate the operating system at the same or even lower level than the detection tool uses for its baseline data collection. Apart from this limitation, cross-view detection is one of the most effective ways to find rootkits, because it is "generic" in the sense that is does not scan for specific rootkit signatures or for signs of a data hiding technique like hooking.

The tools IceSword [23] and DarkSpy [8] are both freeware tools that offer the possibility to look for rootkits in an interactive fashion. For example both offer a function to view a list of currently running processes, like the Windows Task Manager, or an Explorer-like application to browse the file system or the Windows registry. The difference to the ordinary reporting and browsing tools is that they use very advanced techniques to gather the data about the respective objects, often exploiting the way in which rootkits typically hide processes or files. Although IceSword and DarkSpy may sometimes highlight hidden processes using similar techniques as the cross-view detection tools, generally they have to be used in an interactive fashion by an investigator who systematically looks for sings of a rootkit.

To complete the set of detection tools there are also a number of command line tools that cannot be classified like the tools above, these are modgreper, flister and System Virginity Verifier (all from [25]). Flister is a tool that can be used to detect hidden files, exploiting some common bugs in rootkits when using file hiding techniques. Modgreper scans kernel memory for hidden modules, which can sometimes reveal the presence of a rootkit that uses a hidden module in the kernel to do its work. The System Virginity Verifier (SVV) written by Joanna Rutkowska is a unique tool, which is related to the concept of *Explicit Compromise Detection* [81], which means that the integrity of critical operating system elements is checked to detect a possible compromise. SVV compares the code sections of kernel modules in memory, i.e. drivers and DLLs, with their representation on the file system, where the files are stored. If discrepancies are detected between the stored file and its image in memory, SVV evaluates the type of the change and outputs an infection level that denotes the severity of the modification, and whether it is likely to be a malicious modification.

To see how standard Antivirus Software works against rootkits, additional tests were made with Norton Internet Security 2005 (NIS), and to see how some standard Live Response tools are influenced by the presence of a rootkit, some more tests were conducted with `netstat`, `fport`, `pslist`, `psservice`, `find` and `regdmp`.

5.4 Dependability Experiments

5.4.1 Setup

Test platforms

To get an idea of how rootkits work on different version of the Windows operating system, all experiments were executed on four different operating systems, namely on Windows 2000 Service Pack 4 with all official update as of June 1, 2006, Windows XP without any updates, Windows XP Service Pack 2 with all updates and Windows 2003 Server Service Pack 1 with all updates. All systems were standard installations, with a minimum of required device drivers installed to be able to run properly. After the setup, all systems were kept offline and all software was copied from a CD to the test platform, to avoid accidental compromise by a real attack.

Compatibility issues with rootkits

There were significant compatibility problems with the rootkits, in particular, the only rootkits that could be installed on Windows 2003 were Hacker Defender, Klog and Vanquish. Other notable problems occured with cfsd and He4Hook, none of which could be successfully installed on any of the test platforms; it is not clear whether this was a known issue or caused by incorrect use of the supplied files, because these rootkits come with no documentation at all. On top of that, NtIllusion did not work properly on any operating system but Windows XP Service Pack 2, HideProcessHookMdl did not work on Windows 2000 and the AFX Rootkit could not be executed on Windows XP Service Pack 2.

Compatibility issues with detection tools

There were also minor compatibility issues with some of the detection tools, although they appeared to be far more robust than the rootkits themselves. In particular, VICE does not execute properly on Windows XP Service Pack 2; the problem seems to be the latest patch to the .NET framework, which is used by VICE. In addition to these incompatibilities, SVV does not work properly on Windows XP Service Pack 2 with the latest patches installed as of June 1, 2006. Although it does execute normally and outputs a result, it also outputs an error message and it can be clearly seen from the resuts that it did not operate at full effectiveness. The last incompatibility was with DarkSpy and Windows 2000 SP4: when starting DarkSpy, the operating system would simply crash immediately with a Blue Screen, making it impossible to test DarkSpy on this platform.

5.4.2 Execution

Configuration of the rootkits

To start the experiments the rootkits were, if necessary, configured using the included documentation to use all the capabilities for stealth that they provide. Specifically, this means:

- If the rootkit offers a file hiding capability, a sample folder and files that should be hidden were created.
- If the rootkit offers a process hiding capability, an appropriate process that should be hidden was started.

- If the rootkit offers hiding of network ports, an open port was created using a listening `netcat` session.
- If the rootkit offers a registry key hiding feature, sample registry keys and values to hide were created.
- If the rootkit allows hiding of services and drivers, sample objects of the respective type to be hidden were configured.

Some rootkits hide files or processes by using a *magic string*, i.e. if a file or process name contains a specific string, it will be hidden; AFX, NtIllusion, Vanquish, and HideProcessHookMdl use this technique. Hacker Defender has a real configuration file which allow definition of different magic strings for different classes of objects to be hidden, and it allows to define network port ranges to hide. FU, FUTo and phide can hide processes by referring the process's identification number, or PID. The keyboard sniffer Klog did not have to be configured.

Table 1 gives an overview over the hiding capabilities of the different rootkits.

	Files	Processes	Registry Keys	Ports	Services	Drivers
AFX Rootkit	X	X	X	X	X	–
FU	–	X	–	–	–	X
FUTo	–	X	–	–	–	X
Hacker Defender	X	X	X	X	X	X
Klog	–	–	–	–	–	–
NtIllusion	X	X	X	X	–	–
Vanquish	X	X	X	–	X	–
phide	–	X	–	–	–	–
HideProcessHookMdl	–	X	–	–	–	–

Table 1: Hiding capabilities of rootkits: an X denotes that the rootkit can hide objects of the respective type

Rootkit installation

After configuration the rootkits were installed using the included loader programs, which is mostly a simple EXE file that will load the rootkit into memory and start it, which results in immediate hiding of the desired objects. In this case, *hidden* means that files are no longer visible in the Windows Explorer, processes are invisible in the Task Manager, registry keys are invisible in the Registry Editor, and open network ports are not detectable with `netstat`. Drivers and services were checked using the System Information tool included in Windows.

Generally, the rootkits succeeded in hiding the respective objects they were designed to hide very well, all rootkits that have the capability the hide processes, files, registry keys or network ports worked exactly as expected, hiding the target elements from the applications mentioned above. The only exception is AFX Rootkit, which did not hide network ports as it was supposed to do. Vanquish and Hacker Defender use a special service to operate, and they both succeeded in hiding “their own” service. Driver hiding, which is offered by FU, FUTo and Hacker Defender, did not work at all; all loaded rootkit drivers were still visible with the System Information program.

For the test involving Norton Internet Security 2005, the installation was actually done in two different orders, on the one hand by installing the rootkit first and then NIS, and on the other hand by installing NIS first and then the rootkit. The Live Response tools were all run at once using a small batch script after the rootkit was installed and started.

Application of Detection Tools

After having installed a single rootkit and having documented its stealth properties, the rootkit detection tools were run to see if they would detect the hidden objects and the rootkits themselves respectively. The output was then recorded and it was checked whether all hidden objects that the detection tool can detect have actually been detected; of course it makes no sense to blame a tool for not detecting a hidden process, if it is not designed to do so. Having documented the results for the detection tool, it was then uninstalled if necessary, and any drivers it used were unloaded. After a system reboot, the state prior to testing the previous detection tool was reestablished, if necessary by restarting the rootkit and reconfiguring the sample processes and network ports. Then the next detection tool was tested in the exact same way as described above.

5.4.3 Results

Differences between operating system versions

As a first result, there were absolutely no significant differences in the detection performance between the different operating system versions used during the experiments. In particular, there was no instance, in which a detection tool detected any rootkit modification successfully on one operating system, but failed on another one. The only differences were the compatibility issues of some rootkits and detection tools that were also described in detail in the Setup section. For this reason the results will be mainly be evaluated for the Windows XP SP2 system, which was compatible to most rootkits and detection tools and can thus be regarded as the reference operating system concerning the results of the rootkit detection experiments. For the AFX Rootkit and VICE, which were incompatible with Windows XP SP 2, the results on Windows XP without any Service Packs were used.

Experimental Results

It turned out that rootkit detection was not a “black-and-white” issue, but that in some cases detection tools would only partially detect the rootkits modifications. In this context, complete detection would mean, that the detector detected all changes that it is designed to detect. Table 2 gives an overview over detection tools and the kind of objects they were designed to detect.

	Files	Processes	Registry Keys	Ports	Services	Drivers	Hooks
Darkspy	X	X	–	X	–	X	–
Flister	X	–	–	–	–	–	–
F-Secure Blacklight	X	X	–	–	–	–	–
IceSword	X	X	X	X	X	X	X
modgreper	–	–	–	X	X	–	–
RKDetector	–	X	–	–	X	X	X
RKHA	–	–	–	–	–	–	X
RK Revealer	X	–	X	–	–	–	–
SVV	–	–	–	–	–	X	X
UnhackMe	–	X	X	–	X	–	–
VICE	–	–	–	–	–	–	X

Table 2: Rootkit Detectors: an X denotes that the detector can detect hidden objects of the respective type

A scale with three values was used to rate the amount of success that was achieved. A success variable s was used to account for the effectiveness of the detection tool as folllows

- $s = 0$: The detection tool did not detect anything.
- $s = 1$: The detection tool detected some rootkit modification or hidden objects, but not everything.
- $s = 2$: The detection tool detected all possible rootkit modifications and hidden objects.

The results can be seen in table 3. A "–" means that the tool in question was not applicable, because it was not designed to detect the objects the particular rootkit hid. It is obvious, that the Klog keyboard sniffer does not really fit into this selection, because it does not actually hide anything. This issue will be discussed after the analysis of the experimental results.

Antivirus Tests

Regarding Norton Internet Security, there were major differences when comparing its performance when the rootkit was installed first against the case in which NIS was installed first. When NIS was installed first, it would detect all files belonging to the rootkits and quarantine them, making it impossible to even configure or install the rootkits. Results were different however when the rootkits were installed before NIS. A pre-install scan performed by NIS did not detect anything, and the software would be installed normally; except when Vanquish was installed, which made it impossible to install NIS on top. It is

	AFX	FU	FUTo	HxDef	Klog	NtIllusion	Vanquish	phide	HPHM
Darkspy	2	2	2	2	–	2	2	2	2
Flister	2	–	–	0	–	2	2	–	–
Blacklight	2	2	0	2	–	2	2	2	2
IceSword	2	2	2	2	–	2	2	2	2
modgreper	–	–	–	1	–	–	–	–	–
RKDetector	2	2	0	0	–	2	0	2	2
RKHA	–	–	–	–	–	–	–	–	2
RKRevealer	2	–	–	2	–	0	2	–	–
SVV	2	–	–	2	–	1	2	–	1
UnhackMe	2	0	0	2	–	1	2	0	2
VICE	2	–	–	2	–	–	2	–	2

Table 3: Experimental Results for s

not clear whether this is a desired function of the rootkit, which would generally make sense, or just coincidence. After rebooting to finish the installation, the Auto-Protect feature would then generally detect the rootkit during a full system scan, quarantine the related files and stop it from operating. Of course any modifications that NIS is not designed to detect remained unnoticed, such as processes hidden with FU, FUTo, phide and HideProcessHookMdl, or the keyboard sniffer Klog.

In the case of Hacker Defender though, NIS did sometimes not detect the rootkit at all, allowing it to run unnoticed in the background. It would fail to quarantine all files that belong to the rootkit installation, or not find any of them during a full system scan. Despite extensive tests of NIS when confronted with Hacker Defender, a pattern of behaviour could not be identified; NIS reacted almost randomly to the same situation under the exact same circumstances. In at least one case, it was possible to have Hacker Defender run in the background, hiding files, processes etc., and NIS would not detect it in any way, neither with its Auto-Protect feature or manual full system scans. But under the same circumstances, NIS would also sometimes completely detect and disable the rootkit; it was not possible to discover the reason for this seemingly random behaviour. Full detection was achieved in roughly 80% of the experiments, while in the remaining 20% of all cases, NIS did either not completely detect the rootkit or fail completely.

Tests with Live Response tools

The results of the Live Response tools `netstat`, `fport`, `pslist`, `psservice`, `find`, and `regdmp`, included in the experiments can be summarized quickly: *none* of them reported any of the objects that were hidden by the rootkits. The only way that these tools could be used to indicated a potential rootkit installation, was when a hidden process had a non-hidden port open; in this case, the tools would detect the open port, but not the process, a discrepancy that could alert an investigator if he carefully analyzed the output of the tools.

5.4.4 Analysis

The experimental results show that there were several excellent tools that were able to detect all rootkit modifications, but there were also some that did not perform well. The two interactive tools DarkSpy and IceSword performed excellent, both reaching a mean score of 2, the maximum possible. F-Secure's cross-view detection tool Blacklight also detected all modifications except processes hidden by FUTo, which is an expected result, because FUTo is a modified version of the FU rootkit, specifically designed to be able to evade detection by Blacklight [17] .VICE also perfectly detected the hooks installed by the hooking rootkits, although it has previously been mentioned that identifying malicious hooks can sometimes be diffcult, because a lot of software, especially Antivirus software, uses benign hooking.

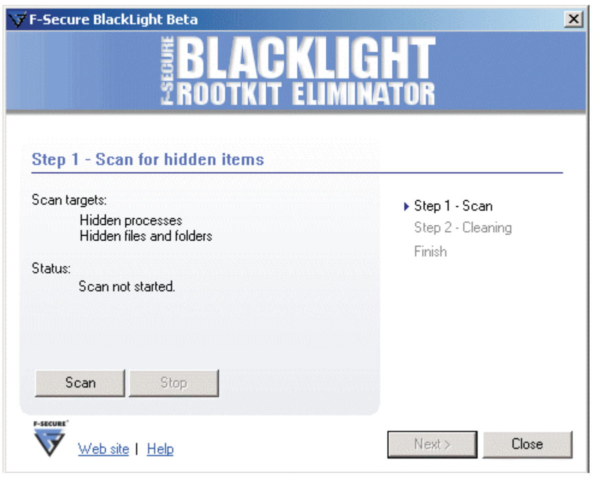
Other tools that performed quite well were SVV with an average score of 1.6 and Rootkit Revealer (1.5), along with the file lister flister (average score of 1.5). The tools Rootkit Detector and UnhackMe did not perform satisfactorily, with scores of 1.25 and 1.125 respectively; in addition to the lower average results, these tools did not detect a number of rootkits at all, making their use questionable and the results not very reliable. The very specialized tools modgreper and Rootkit Hook Analyzer (RKHA) could not be evaluated properly based on the experiments that were conducted, because they were each only applicable in one case, which is certainly not representative of the tools' reliability.

Norton Internet Security offers good protection against publicly available rootkits if it is already installed on the system that is under attack. Even if the rootkit had the advantage of being installed first, NIS would detect it in all cases except when facing Hacker Defender, which lead to random detection failures. The fact that NIS detected the publicly available rootkits is not surprising because their source code is known and thus they are included in the set of malware signatures that are routinely checked for by antivirus products. During the experiments it also became obvious that the keyboard sniffer Klog would not be detected, simply because it does not attempt to hide anything; this result was therefore expected.

The fact that the Live Response tools tested failed completely in trying to report the hidden objects does not surprise either: rootkits are specifically designed to subvert these tools' mechanisms, which generally means that they subvert at least the high level-API reporting functions used by the Live Response tools. This results shows that rootkit detection has indeed become very important in forensic analysis because the standard tools used in Live Response procedures do not work properly. Adressing this need for rootkit detection, a methodology for rootkit detection has been developed, which will be described in the next section.

5.5 Rootkit Detection Methodology

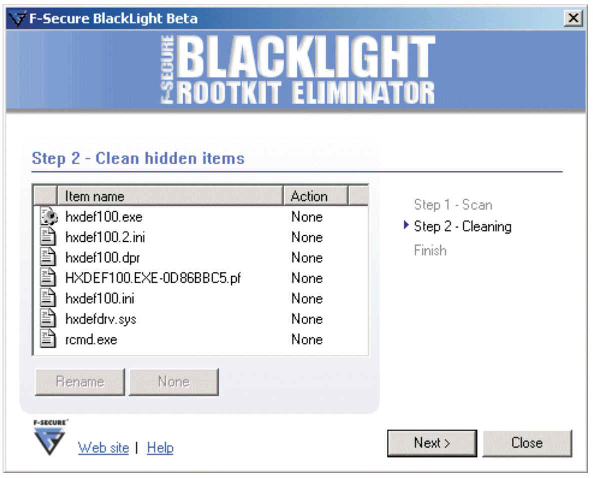
Based on the results of the experiments, a methodology for rootkit detection has been developed, which consists of using a combination of different detection tools to add redundancy to the detection procedures. The proposed methodology uses three tools: F-Secure Blacklight, IceSword and SVV. This combination was chosen for several reasons; first of all, all three tools achieved good average results in the experiments, reliably detecting rootkit modifications.



Scan targets:
Hidden processes
Hidden files and folders

Status:
Scan not started.

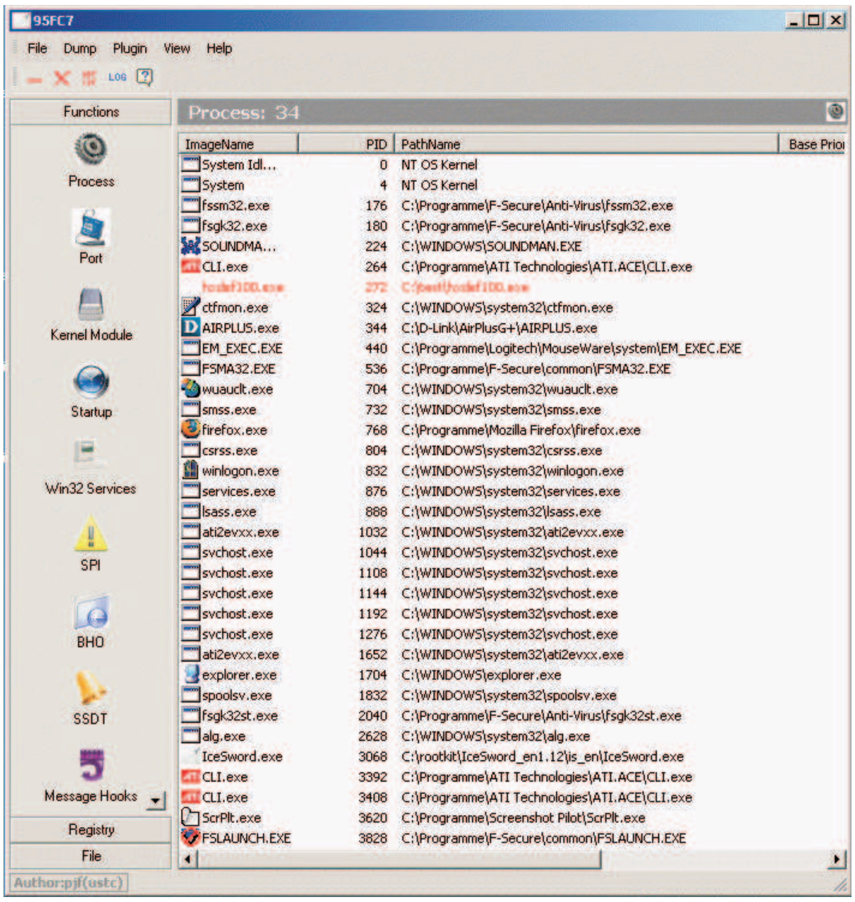
Figure 11: F-Secure Blacklight



Item name	Action
hxdef100.exe	None
hxdef100.2.ini	None
hxdef100.dpr	None
HXDEF100.EXE-0D86BBC5.pf	None
hxdef100.ini	None
hxdefdrv.sys	None
rcmd.exe	None

Figure 12: F-Secure Blacklight detects a Hacker Defender rootkit

Secondly, these three tools represent three different approaches to rootkit detection. Blacklight uses a cross-view based detection approach, it offers a simple user interface, which allows to scan for hidden objects with a single mouse click (see figure 11). After checking the system, Blacklight outputs a list of hidden objects found (figure 12). Usage of IceSword differs in the way that it offers information about certain operating system elements in an interactive



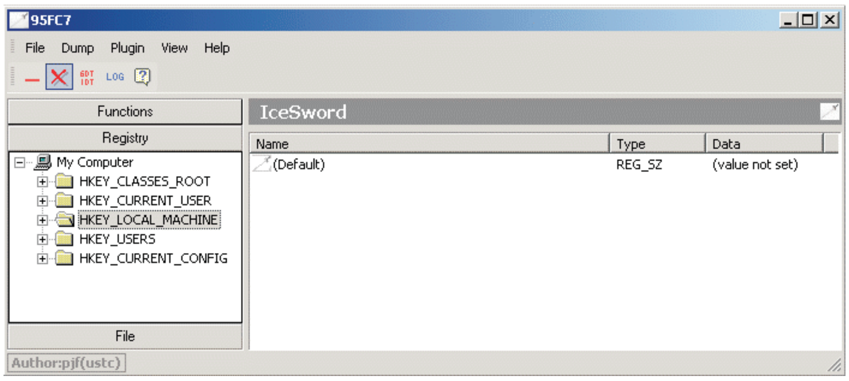
Process: 34

ImageName	PID	PathName	Base Prioı
System Idl...	0	NT OS Kernel	
System	4	NT OS Kernel	
fssm32.exe	176	C:\Programme\F-Secure\Anti-Virus\fssm32.exe	
fsgk32.exe	180	C:\Programme\F-Secure\Anti-Virus\fsgk32.exe	
SOUNDMA...	224	C:\WINDOWS\SOUNDMAN.EXE	
CLI.exe	264	C:\Programme\ATI Technologies\ATI.ACE\CLI.exe	
hxdef100.exe	272	C:\test\hxdef100.exe	
ctfmon.exe	324	C:\WINDOWS\system32\ctfmon.exe	
AIRPLUS.exe	344	C:\D-Link\AirPlusG+\AIRPLUS.exe	
EM_EXEC.EXE	440	C:\Programme\Logitech\MouseWare\system\EM_EXEC.EXE	
FSMA32.EXE	536	C:\Programme\F-Secure\common\FSMA32.EXE	
wuauclt.exe	704	C:\WINDOWS\system32\wuauclt.exe	
smss.exe	732	C:\WINDOWS\system32\smss.exe	
firefox.exe	768	C:\Programme\Mozilla Firefox\firefox.exe	
csrss.exe	804	C:\WINDOWS\system32\csrss.exe	
winlogon.exe	832	C:\WINDOWS\system32\winlogon.exe	
services.exe	876	C:\WINDOWS\system32\services.exe	
lsass.exe	888	C:\WINDOWS\system32\lsass.exe	
ati2evxx.exe	1032	C:\WINDOWS\system32\ati2evxx.exe	
svchost.exe	1044	C:\WINDOWS\system32\svchost.exe	
svchost.exe	1108	C:\WINDOWS\system32\svchost.exe	
svchost.exe	1144	C:\WINDOWS\system32\svchost.exe	
svchost.exe	1192	C:\WINDOWS\system32\svchost.exe	
svchost.exe	1276	C:\WINDOWS\system32\svchost.exe	
ati2evxx.exe	1652	C:\WINDOWS\system32\ati2evxx.exe	
explorer.exe	1704	C:\WINDOWS\explorer.exe	
spoolsv.exe	1832	C:\WINDOWS\system32\spoolsv.exe	
fsgk32st.exe	2040	C:\Programme\F-Secure\Anti-Virus\fsgk32st.exe	
alg.exe	2628	C:\WINDOWS\system32\alg.exe	
IceSword.exe	3068	C:\rootkit\IceSword_en1.12\is_en\IceSword.exe	
CLI.exe	3392	C:\Programme\ATI Technologies\ATI.ACE\CLI.exe	
CLI.exe	3408	C:\Programme\ATI Technologies\ATI.ACE\CLI.exe	
ScrPlt.exe	3620	C:\Programme\Screenshot Pilot\ScrPlt.exe	
FSLAUNCH.EXE	3828	C:\Programme\F-Secure\common\FSLAUNCH.EXE	

Figure 13: IceSword Process browser: IceSword highlights a process hidden by Hacker Defender

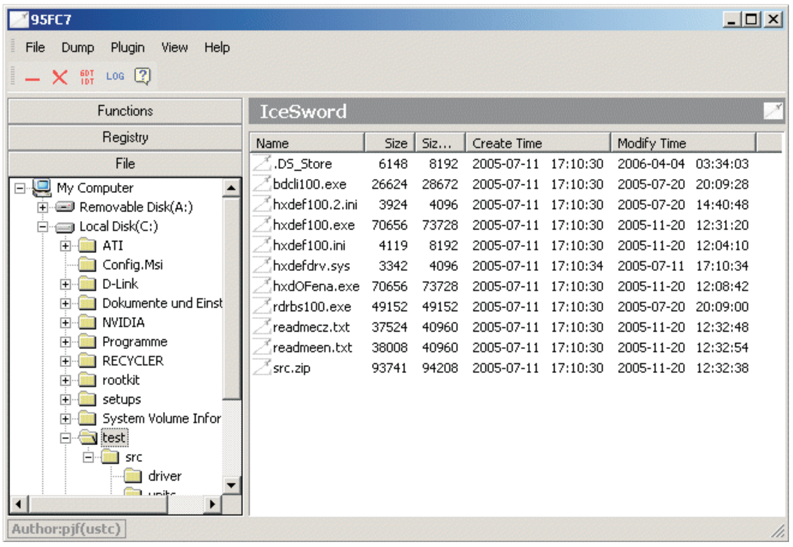
way; it offers a function that resembles the Task Manager to review running processes (figure 13), a browser for the Windows registry (figure 14) and the file system (figure 15), which resembles the Windows Explorer, and it also includes function to view loaded device drivers, services, the SSDT and even more. IceSword does not offer a convenient scanning option such as Blacklight, but if an investigator knows what to look out for, it is a very flexible and powerful tool. SVV was chosen to be included in the methodology because it represents yet another technique to detect rootkits, by checking important operating system elements for their integrity (a sample output of SVV can be seen in figure 16). This technique requires the most technical knowledge about the concepts and techniques used by rootkits, but for an experienced investigator it is one of the most powerful and advanced tools to analyze a rootkit infection.

This comparison of the three tools has also shown that the methodology is applicable for forensic examiners with different levels of understanding; Blacklight does not require any special background to be used, IceSword becomes more effective in the hand of an experienced investigator, and SVV offers detailed information which will be most helpful to a very skilled analyst. By correlating the output of all three tools, the reliability of rootkit detection is



Name	Type	Data
(Default)	REG_SZ	(value not set)

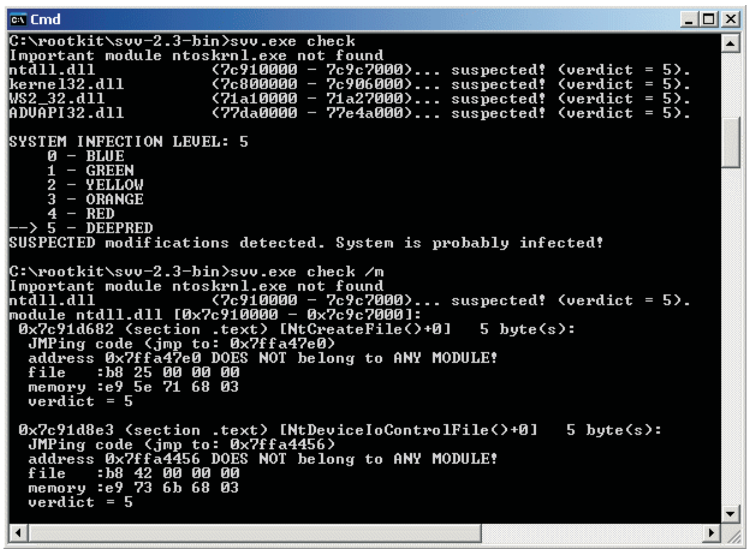
Figure 14: IceSword's Registry Browser is similar to the standard Windows registry editor



Name	Size	Siz...	Create Time	Modify Time
.DS_Store	6148	8192	2005-07-11 17:10:30	2006-04-04 03:34:03
bdcli100.exe	26624	28672	2005-07-11 17:10:30	2005-07-20 20:09:28
hxdef100.2.ini	3924	4096	2005-07-11 17:10:30	2005-07-20 14:40:48
hxdef100.exe	70656	73728	2005-07-11 17:10:30	2005-11-20 12:31:20
hxdef100.ini	4119	8192	2005-07-11 17:10:30	2005-11-20 12:04:10
hxdefdrv.sys	3342	4096	2005-07-11 17:10:34	2005-07-11 17:10:34
hxdOFena.exe	70656	73728	2005-07-11 17:10:30	2005-11-20 12:08:42
rdrbs100.exe	49152	49152	2005-07-11 17:10:30	2005-07-20 20:09:00
readmecz.txt	37524	40960	2005-07-11 17:10:30	2005-11-20 12:32:48
readmeen.txt	38008	40960	2005-07-11 17:10:30	2005-11-20 12:32:54
src.zip	93741	94208	2005-07-11 17:10:30	2005-11-20 12:32:38

Figure 15: IceSword File Explorer shows files that were hidden in Windows Explorer

increased, because even if a rootkit manages to evade one of the tools, it can still be detected by another one that uses different methods for detection. None of the rootkits tested can evade the combination of Blacklight, IceSword and SVV, and only an extremely sophisticated rootkit could hope to avoid detection by all of the three tools together. Although it is difficult to quantify the success rate of detection when using this methodology for an arbitrary rootkit infection, it is reasonable to assume that when facing a rootkit that uses techniques similar to the ones tested in this thesis, the rate of detection is close to 100%. Asssuming that the majority of rootkits used today fall into this category – which appears reasonable – the likelihood to detect an actual rootkit detection is equally high.



```
C:\rootkit\svv-2.3-bin>svv.exe check
Important module ntoskrnl.exe not found
ntdll.dll             (7c910000 - 7c9c7000)... suspected! (verdict = 5).
kernel32.dll          (7c800000 - 7c906000)... suspected! (verdict = 5).
WS2_32.dll            (71a10000 - 71a27000)... suspected! (verdict = 5).
ADVAPI32.dll          (77da0000 - 77e4a000)... suspected! (verdict = 5).

SYSTEM INFECTION LEVEL: 5
    0 - BLUE
    1 - GREEN
    2 - YELLOW
    3 - ORANGE
    4 - RED
--> 5 - DEEPRED
SUSPECTED modifications detected. System is probably infected!

C:\rootkit\svv-2.3-bin>svv.exe check /m
Important module ntoskrnl.exe not found
ntdll.dll             (7c910000 - 7c9c7000)... suspected! (verdict = 5).
module ntdll.dll [0x7c910000 - 0x7c9c7000]:
 0x7c91d682 (section .text) [NtCreateFile()+0]   5 byte(s):
  JMPing code (jmp to: 0x7ffa47e0)
  address 0x7ffa47e0 DOES NOT belong to ANY MODULE!
  file   :b8 25 00 00 00
  memory :e9 5e 71 68 03
  verdict = 5

 0x7c91d8e3 (section .text) [NtDeviceIoControlFile()+0]   5 byte(s):
  JMPing code (jmp to: 0x7ffa4456)
  address 0x7ffa4456 DOES NOT belong to ANY MODULE!
  file   :b8 42 00 00 00
  memory :e9 73 6b 68 03
  verdict = 5
```

Figure 16: System Virginity Verifier alerts changes to important system elements by Hacker Defender

5.6 Discussion

To explore the problem of rootkit detection from a forensic investigator's point of view, experiments were conducted to test the effectiveness of rootkit detection tools against some of the most popular publicly available rootkits today. This was justified by the fact that normal Live Response tools fail to uncover the modifications done to operating systems by modern rootkits, which could also be shown impressively by testing common Live Response tools on the rootkit infected systems.

All experiments were planned and the test platform carefully prepared to allow objective and accurate testing of the tools, and to gain a representative view of their performance on the Windows operating system platform, multiple different operating systems were included in the tests. A selection of detection tools that use different methods for detection was chosen, among which were cross-view based detection tools and tools that use Explicit Compromise Detection (ECD) to detect rootkits.

It could be shown that there exist a number of very good rootkit detection tools, but also that some are not reliable enough to be used in a forensic investigation. By choosing a combination of three detection tools which achieved good results in the experiments and which each represent a different approach to rootkit detection, redundancy has been added to the process of rootkit detection, resulting in a reliable and accurate methodology. Using F-Secure Blacklight, IceSword and System Virginity Verifier, all publicly available rootkits can be detected safely, and depending on the skill of the investigator, this toolkit, especially IceSword and Blacklight, can also be used to analyze the rootkits in more detail.

There is one thing to keep in mind when using IceSword and several other rootkit detection tools: a special device driver, for IceSword it is called `IsPubDrv.sys`, is often copied to the systems hard drive to allow the detector to function properly. This of course contradicts the principle of not creating files on an evidence system, because this is a severe modification of the evidence. The principle problem here is, that in order to be able to detect a rootkit's subversion, the detection software has to be able to deal with the problem on the same level, i.e. using a Ring 0 kernel component. Device drivers are the standard way to gain access to Ring 0, and that is why the creation of a single file can be tolerated in this context; otherwise the rootkit would have an implicit advantage, because it can manipulate the operating system at a lower level than the detection tool can reach.

5.7 Summary

In this chapter, the concepts and principles of Windows rootkits, one of the most dangerous classes of malware today, have been introduced and the implications for forensic analysis have been discussed, in particular in the context of Live Response. Since the reliability of Live Response is severly crippled in a situation where an evidence system is infected with a rootkit, a sound methodology for rootkit detection is mandatory. To develop an adequate methodolgy, a survey of common rootkits and detection tools was compiled to conduct a series of experiments to evaluate the quality of the detectors. Rootkits and detection tools were put to the test to determine the reliability of the individual tools when facing diverse rootkits that also use different concepts to function, e.g. Hooking and DKOM.

After having identified the best tools for rootkit detection, a rootkit detection methodology was defined by combining the tools F-Secure Blacklight, IceSword and SVV. It was shown that the combination of these tools represents a reliable method to detect rootkits, using redundancy and multiple different approaches to achieve reliable Windows rootkit detection. This methodology can then be combined with the usual Live Response procedures, more specifically, rootkit detection should be performed before all other data collection procedures. If a rootkit is detected, the investigator can decide whether or not to collect data using the standard Live Response tools, but in any case the data should be considered not trustworthy and critical decisions should not be based solely upon this untrusted information. Detection of a rootkit can also be a reason to use advanced Live Response techniques like collecting main memory dumps, and it can influence an investigator's assumptions about the responsible attacker. In any case it allows an investigator to reach more informed decisions about how to continue with the investigation of the respective incident.

Chapter 6

Conclusion and Future Work

In this diploma thesis, methodologies for conducting forensic analysis were investigated, focussing on ways to automate analysis while still achieving valid and reliable results. For this reason, process models for Incident Response and Computer Forensics respectively were introduced, which represent the two most important concepts of for investigation of computer security incidents. By comparing these two process models, it could be shown that Incident Response and Computer Forensics are two closely related concepts, and that they could be integrated into a single process model, which will benefit from the advantages of both models but will not share the unique disadvantages.

For this reason, a Common Model for Incident Response and Computer Forensics was developed, which integrates a forensic analysis into a Live Response framework, carefully evaluating the need for full-scale forensic analysis by considering the circumstances of the particular incident. Each step of the new model was explained in detail by first presenting commonly used tools and Best Practices and then discussing the step's significance within the whole process. This lead to a thorough understanding of the crucial parts of the investigation of an incident, as well as an overview over practical methods used during the analysis.

The step of Live Response was then dealt with in more detail, discussing its significance and why it has become a mandatory part of a modern forensic investigation. Live Response is comprised of various methods to safely collect information from live systems, as opposed to the traditional forensic analysis which only works on "dead" evidence. It could be shown that sufficiently reliable techniques exist that allow to gather valuable information from live systems without disturbing them unnecessarily. When examining commonly accepted Live Response practices it was discovered that some of them are not as forensically sound as they seemed, modifying the evidence system in an undesirable way by updating file timestamps on the hard drive. An improved method was proposed which does not share this flaw but yields equivalent results. The missing rootkit detection routines in current Live Response procedures were motivation to investigate the problem of rootkit detection on Windows operating systems.

The basic principles of rootkits and Microsoft Windows internals were introduced to be able to show why rootkits are one of the most dangerous and sophisticated classes of malware and why they are likely to stay a big concern in the future. An overview of currently available detection tools and popular rootkits was given in order to prepare a series of experiments to explore how well rootkit detection tools can really detect different rootkits. These experiments were then conducted, resulting in a detailed record of the effectiveness of the detection tools. Based on the results, a methodology for rootkit detection was proposed, which was able to detect 100% of all rootkit installations in the tests, and which is expected to work well even when facing different rootkits. This combination of three different tools that use different approaches to detection and which complement each other well can now be used in Windows Live Response situations to check for the presence of rootkits on an evidence system. Its results are both reliable and accurate, yielding lots of information about the rootkit's modifications to the operating system.

By developing a rootkit detection methodology and by improving currently common Live Response techniques, the overall reliability and dependability of Live Response, and therefore of the whole forensic analysis process has been improved. Nevertheless there are a lot of possibilities to continue the work begun in this thesis, aiming at improving the reliability of forensic analysis while using automated techniques. Each step in the Analysis Phase could be scrutinized just like the step of Live Response, which was the focus of this work.

Recently there have been a number of researchers who exposed flaws in common Best Practices for Computer Forensics, and the term "Anti-Forensics" has been coined to label this class of attacks [77, 72]. These attacks focus specifically on weaknesses in accepted forensic tools, such as EnCase, and how they can be exploited to cause incorrect analysis results. These examples show that trust in forensic tools that automate certain tasks always has to be justified, and that a lot of commonly accepted tools might have to be reevaluated in light of the recent results.

Although this thesis was able to propose a sound methodology for rootkit detection, there are a number of factors that have to be considered when discussing the danger that rootkits represent. First of all, there are possibly a number of rootkits that are not publicly available, but that are instead sold for considerable amounts of money. There was no way to obtain samples of these rootkits, and it cannot be said with certainty that the rootkit detection methodology would also detect all modifications of commercial rootkits. Secondly, a lot of authors of public rootkits offer the option to pay for a modified version of their public rootkit, which is then promised to be able to evade all common detection tools. Again, no examples of customized rootkits were available for test, and although the methods for detection with the proposed combination of tools may be generic enough to catch even modified rootkits, no real guaranty can be given. The principle of Explicit Compromise Detection seems to be the most promising approach to detection of advanced rootkits, and there is still a lot of work to do in this area [81]. It would also be desirable to automate rootkit detection further, by trying to find ways to use the detection tools in a pre-defined way to check for signs of a rootkit infection. Up to now rootkit detection consists mostly of manual application of the detectors; it should be explored how this can be alleviated while retaining the good rate of detection that manual techniques achieve. The Open Methodology for Compromise Detection [26] is a promising research approach in this matter.

Another major reason for concern is that rootkit technology is rapidly advancing, and detection tools that may work well today are likely to fail when confronted with newer rootkits. Recently the prototype of a rootkit named Shadow Walker [90] has been presented, which subverts the Windows memory manager to achieve stealth, a method which is completely undectable by today's rootkit detection tools. Lately an independent security researcher has announced the development of a "hypervisor" rootkit called Blue Pill [25], which uses virtualization techniques provided by modern CPUs to subvert the operating system at an even lower level than the kernel itself.

These problems and examples show that the race between rootkit developers and rootkit detection developers is no different to the race between virus authors and antivirus providers, and that constant research is necessary to keep up with the latest developments to assure the security of computer systems and the dependability of Computer Forensic analysis in the future.

Appendix A

Sourcecode

This appendix contains the sourcecode of the scripts used in this thesis.

A.1 getmetadata.pl

```
### getmetadata.pl ###

#!/usr/bin/perl

open MACTIMES, $ARGV[0];

print
"MD5|Filename|Inode|Mode|ModeString|User|Group|Size|ATime|ADate|MTime|
➥ MDate|CTime|CDate\n";

while (<MACTIMES>) {
        chomp;
        @macrow = split( /\|/, $_ );

        if (@macrow[1] =~ m/\(deleted-realloc\)/g) {
                $md5sum = '';
        } else {
                $md5sum = `icat -f $ARGV[1] $ARGV[2] @macrow[3] | md5sum -b`;
                $md5sum =~ m/^(\S+)\s+.*$/g;
                $md5sum = $1;
        }

@atime = localtime( @macrow[11] );
@mtime = localtime( @macrow[12] );
@ctime = localtime( @macrow[13] );

@atime[4] += 1;
@atime[5] += 1900;

@mtime[4] += 1;
@mtime[5] += 1900;
@ctime[4] += 1;
@ctime[5] += 1900;

$atimeasc = sprintf("%02d:%02d:%02d", @atime[2], @atime[1], @atime[0]);
$adateasc = sprintf("%02d-%02d-%d", @atime[4], @atime[3], @atime[5]);

$mtimeasc = sprintf("%02d:%02d:%02d", @mtime[2], @mtime[1], @mtime[0]);
$mdateasc = sprintf("%02d-%02d-%d", @mtime[4], @mtime[3], @mtime[5]);

$ctimeasc = sprintf("%02d:%02d:%02d", @ctime[2], @ctime[1], @ctime[0]);
$cdateasc = sprintf("%02d-%02d-%d", @ctime[4], @ctime[3], @ctime[5]);

print
"$md5sum|@macrow[1]|@macrow[3]|@macrow[4]|@macrow[5]|@macrow[7]|@macrow[8]|
➥ @macrow[10]|$atimeasc|$adateasc|$mtimeasc|$mdateasc|$ctimeasc|$cdateasc\n";

}
```

A.2 getsignatures.pl

```
### getsignatures.pl ###

#!/usr/bin/perl

open METADATA, $ARGV[0];

print "MD5|Filename|Inode|Mode|ModeString|User|Group|Size|ATime|ADate|MTime|
➥ MDate|CTime|CDate|Signature\n";

while (<METADATA>) {
        chomp;
        $sigdata = $_;
        @sigrow = split( /\|/, $sigdata );

        if (@sigrow[1] =~ m/\(deleted-realloc\)/g) {
                $signature = '';
        } else {
                $signature = `icat -f $ARGV[1] $ARGV[2] @sigrow[2] | file -`;
                $signature =~ m/^[^:]+:\s+(.*)$/g;
                $signature = $1;
        }

print "$sigdata|$signature\n";
}
```

A.3 unknowns.pl

```
### unknowns.pl ###

#!/usr/bin/perl

open EVID, $ARGV[0];

while (<EVID>) {

        $evid = $_;
        @evidrow = split( /\|/, $_ );
        $evidmd5 = uc( @evidrow[0] );

        if (@evidrow[0] =~ m/\w/g) +
                $results = ‘grep -l $evidmd5 $ARGV[1]‘;
                if ( !($results =~ m/$ARGV[1]/g) ) {
                        print $evid;
                }
        } else {
                print $evid;
        }
}
```

A.4 ir-script-linux.sh

```
### ir-script-linux.sh ###
#!./bin/t_bash
# Environment variables
FIND_PATH="/"
BIN_PATH="./bin"
LD_LIBRARY_PATH="./lib"
export LD_LIBRARY_PATH
#More variables
#FIND_FLAGS: permissions, 3 time stamps, user, group, size, file name
FIND_FLAGS="-printf %m;%Ax;%AT;%Tx;%TT;%Cx;%CT;%U;%G;%s;%p\n"
PASSWD_PATH="/etc/passwd"
GROUP_PATH="/etc/group"
INTED_PATH="/etc/inetd.conf"
DATE="$BIN_PATH/t_date"
CAT="$BIN_PATH/t_cat"
FIND="$BIN_PATH/t_find"
UNAME="$BIN_PATH/t_uname"
IFCONFIG="$BIN_PATH/t_ifconfig"
HOSTNAME="$BIN_PATH/t_hostname"
LAST="$BIN_PATH/t_last"
W="$BIN_PATH/t_w"
WHO="$BIN_PATH/t_who"
RPCINFO="$BIN_PATH/t_rpcinfo"
PS="$BIN_PATH/t_ps"
RPM="$BIN_PATH/t_rpm"
NETSTAT="$BIN_PATH/t_netstat"
DF="$BIN_PATH/t_df"
MOUNT="$BIN_PATH/t_mount"
LSMOD="$BIN_PATH/t_lsmod"
HOSTID="$BIN_PATH/t_hostid"
MD5SUM="$BIN_PATH/t_md5sum"
LSOF="$BIN_PATH/t_lsof"
```

```
#function to send messages to console
function console {
 printf "$1" >&2;
} # console

console "Starting Live Response Script for Linux.\n";
echo "====  Live IR Script v. Linux 2.6  ===="
echo ""

console "- storing system date..\n"
echo "# System Date - Start #"
$DATE
echo "## END ##"; echo ""

## Storing system information

console "- obtaining host ID..\n"
echo "# Hostid #"
HOSTIDSTRING=‘$HOSTID‘
echo $HOSTIDSTRING
echo "## END ##"; echo""

console "- storing hostname..\n"
echo "# Hostname #"
$HOSTNAME
echo "## END ##"; echo ""

console "- storing system ID (uname)..\n"
echo "# Uname -a #"
$UNAME -a
echo "## END ##";echo""

console "- storing IP configuration..\n"
echo "# IP Configuration #"
$IFCONFIG -a
echo "## END ##";echo""

console "- storing uptime and w information..\n"
echo "# w and uptime #"
$W
echo "## END ##";echo""

console "- storing logged in users (who)..\n"
echo "# who #"
$WHO
echo "## END ##";echo""

console "- storing wtmpx via \"last\"..\n"
echo "# last #"
$LAST
echo "## END ##";echo""

console "- storing netstat..\n"
echo "# netstat -an #"
```

```
$NETSTAT -an
echo "## END ##";echo""

console "- storing routing table..\n"
echo "# netstat -rn #"
$NETSTAT -rn
echo "## END ##";echo""

console "- storing RPC information..\n"
echo "# rpcinfo #"
$RPCINFO -p 127.0.0.1
echo "## END ##";echo""

console "- storing process list (ps -eaf)..\n"
echo "# ps -eaf #"
$PS -eaf
echo "## END ##";echo""

console "- storing list of all files.. This is going to take a while.\n"
echo "# File Listings #"
(echo "permissions;access date;access time;modification date;modification time;
➥change date;change time;user ownership;group ownership;file size;file name";
➥$FIND $FIND_PATH $FIND_FLAGS)
echo ""
echo "## END ##";echo""
console "Done. (finally)\n"

console "- storing MD5 sums for all files.. This is going to take a while.\n"
echo "# MD5 sums #"
$FIND $FIND_PATH -xdev -type f -exec $MD5SUM {} \;
echo ""
echo "## END ##";echo""
console "Done. (finally)\n"

console "- storing lsof information..\n"
echo "# lsof #"
$LSOF
echo "## END ##";echo""

## Store files for further review

console "- storing file (/etc/passwd)\n"
echo "# /etc/passwd #"
$CAT $PASSWD_PATH
echo "## END ##";echo""

console "- storing file (/etc/group)\n"
echo "# /etc/group #"
$CAT $GROUP_PATH
echo "## END ##";echo""

console "- storing file (/etc/inetd.conf)\n"
echo "# /etc/inetd.conf #"
```

```
 $CAT $INETD_PATH
echo "## END ##";echo""

## OS Configuration and Patches

console "- storing Linux package information..\n"
echo "# RPM #"
$RPM -qa
echo "## END ##";echo""

console "- storing kernel module information..\n"
echo "# lsmod #"
$LSMOD
echo "## END ##";echo""

## File System information

console "- storing list of mounted file systems..\n"
echo "# mount #"
$MOUNT
echo "## END ##";echo""

console "- storing file system utilization stats..\n"
echo "# df -k #"
$DF -k
echo "## END ##";echo""

console "- storing system date - shutting down the script\n"
echo "# System Date - end #"
$DATE
echo "## END ##";echo""

echo "====  Live Response Script - Linux  ===="
console "## Job Complete\n"
echo""
```

A.5 ir-script-windows.bat

```
@echo off
echo ==== Live IR Script for Windows ====
echo

set IRPATH=%1:\windows_live_response

echo **********************
echo ***** Start Date *****
echo. | date

echo **********************
echo ***** Start Time *****
echo. | time

echo **********************
echo ***** netstat -an ****
```

```
%IRPATH%\t_netstat -an

echo **********************
echo ***** netstat -rn ****
%IRPATH%\t_netstat.exe -rn

echo **********************
echo ***** FPort **********
%IRPATH%\t_fport.exe

echo **********************
echo ***** pslist *********
%IRPATH%\t_pslist.exe

echo **********************
echo ***** nbtstat -c *****
%IRPATH%\t_nbtstat.exe -c

echo **********************
echo ***** psloggedon *****
%IRPATH%\t_psloggedon.exe

echo **********************
echo ***** File Times *****
echo permissions;access date;access time;modification date;modification time;
➥change date;change time; user ownership;group ownership;file size;file name

for %%d in (c d e f g h i j k l m n o p q r s t u v w x y z) do IF EXIST %%d:\
➥%IRPATH%\t_find.exe %%d:\ -printf "%%m;%%Ax;%%AT;%%Tx;%%TT;%%Cx;%%CT;
➥%%U;%%G;%%s;%%p"\n
echo **********************

echo **********************
echo ***** auditpol *******
%IRPATH%\t_auditpol.exe

echo **********************
echo ***** ntlast *********
%IRPATH%\t_ntlast.exe *****

echo **********************************
echo ***** Security Event Log *********
%IRPATH%\t_psloglist.exe -s -x security

echo *************************************
echo ***** Application Event Log *********
%IRPATH%\t_psloglist.exe -s -x application

echo ********************************
echo ***** System Event Log *********
%IRPATH%\t_psloglist.exe -s -x system

echo **********************
echo ***** psinfo *********
%IRPATH%\t_psinfo.exe -h -s -d
```

```
echo **********************
echo ***** psfile *********
%IRPATH%\t_psfile.exe

echo **********************
echo ***** psservice ******
%IRPATH%\t_psservice.exe

echo **********************
echo ***** at *************
%IRPATH%\t_at.exe

echo **********************
echo ***** pwdump6 ********
%IRPATH%\t_pwdump.exe

echo **********************
echo ***** regdmp *********
%IRPATH%\t_regdmp.exe

echo **********************
echo ***** IP Config ******
%IRPATH%\t_ipconfig.exe /all

echo **********************
echo ***** End Time *******
echo. | time

echo **********************
echo ***** End Date *******
echo. | date

echo ==== Live IR Script for Windows ====
echo ## Job Complete
echo
```

A.6 getmetadata-live.pl

```
### getmetadata-live.pl ###

#!./bin/t_perl

print
"MD5|Filename|Inode|Mode|ModeString|User|Group|Size|ATime|ADate|MTime|
➥ MDate|CTime|CDate\n";

while (<STDIN>) {
        chomp;
        @macrow = split( /\|/, $_ );

        if (@macrow[1] =~ m/\(deleted-realloc\)/g) {
                $md5sum = '';
        } else {
                $md5sum = `./bin/t_icat -f $ARGV[0] $ARGV[1] @macrow[3] |
➥./bin/t_md5sum -b`;
                $md5sum =~ m/^(\S+)\s+.*$/g;
                $md5sum = $1;
        }
@atime = localtime( @macrow[11] );
@mtime = localtime( @macrow[12] );
@ctime = localtime( @macrow[13] );
@atime[4] += 1;
@atime[5] += 1900;
@mtime[4] += 1;
@mtime[5] += 1900;
@ctime[4] += 1;
@ctime[5] += 1900;
$atimeasc = sprintf("%02d:%02d:%02d", @atime[2], @atime[1], @atime[0]);
$adateasc = sprintf("%02d-%02d-%d", @atime[4], @atime[3], @atime[5]);
$mtimeasc = sprintf("%02d:%02d:%02d", @mtime[2], @mtime[1], @mtime[0]);
$mdateasc = sprintf("%02d-%02d-%d", @mtime[4], @mtime[3], @mtime[5]);
$ctimeasc = sprintf("%02d:%02d:%02d", @ctime[2], @ctime[1], @ctime[0]);
$cdateasc = sprintf("%02d-%02d-%d", @ctime[4], @ctime[3], @ctime[5]);
print
"$md5sum|@macrow[1]|@macrow[3]|@macrow[4]|@macrow[5]|@macrow[7]|@macrow[8]|
➥ @macrow[10]|$atimeasc|$adateasc|$mtimeasc|$mdateasc|$ctimeasc|$cdateasc\n";
}
```

Bibliography

[1] AccessData Corporation. http://www.accessdata.com.

[2] Autopsy Forensic Browser. http://www.sleuthkit.org/autopsy.

[3] Black Hat. http://www.blackhat.com/.

[4] Building a Security Audit Toolkit. http://www.netadmintools.com/art279.html.

[5] chkrootkit – locally checks for signs of a rootkit. http://www.chkrootkit.org/.

[6] Cryptcat. http://farm9.org/Cryptcat/.

[7] Dan Farmer's FUCK Database. http://web.archive.org/web/20050309151003/www.fish.com/fuck/.

[8] DarkSpy 1.02. http://d.hatena.ne.jp/tessy/20060417.

[9] Daubert v. Merrell Dow Pharmaceuticals, Inc. http://supct.law.cornell.edu/supct/html/92-102.ZS.html.

[10] dcfldd. http://dcfldd.sourceforge.net/.

[11] dd_rescue. http://www.garloff.de/kurt/linux/ddrescue/.

[12] F-Secure – Blacklight. https://europe.f-secure.com/blacklight/.

[13] fatback – DMZS-Biatchux Bootable CD Distro. http://sourceforge.net/projects/biatchux/.

[14] foremost. http://foremost.sourceforge.net/.

[15] Forensic Acquisition Utilities. http://users.erols.com/gmgarner/forensics/.

[16] Foundstone, Inc. http://www.foundstone.com/.

[17] FUTo. http://uninformed.org/?v=3&a=7&t=sumry.

[18] Greatis Software – UnhackMe. http://www.greatis.com/unhackme/.

[19] Guidance Software, Inc. http://www.guidancesoftware.com.

[20] Hashkeeper Group. http://groups.yahoo.com/group/hashkeeper/.

[21] Honeynet Scan of the Month Challenge 32. http://www.honeynet.org/scans/scan32/.

[22] Honeynet Scan of the Month Challenge 33. http://www.honeynet.org/scans/scan33/.

[23] IceSword 1.18. http://soft.patching.net/list.asp?id=25.

[24] Internet Information Services. http://www.microsoft.com/windowsserver2003/iis/default.mspx.

[25] Invisiblethings.org. http://invisiblethings.org.

[26] ISECOM – OMCD. http://www.isecom.org/projects/omcd.shtml.

[27] Knoppix - Live Linux Filesystem On CD. http://www.knopper.net/knoppix/.

[28] Maresware Hash Set CD. http://www.dmares.com/maresware/hash_cd.htm.

[29] memdump. http://www.porcupine.org/forensics/forensic-discovery/.

[30] Microsoft Knowledge Base: Description of the Windows File Protection Feature. http://support.microsoft.com/kb/222193/EN-US/.

[31] Microsoft Knowledge Base: How to Use Dumpchk.exe to check a memory dump file. http://support.microsoft.com/kb/156280/EN-US/.

[32] Microsoft Knowledge Base: How to use the Userdump.exe tool to create a dump file. http://support.microsoft.com/kb/241215/EN-US/.

[33] Microsoft Windows NT Resource Kit. http://www.microsoft.com/resources/documentation/windowsnt/4/server/reskit/en-us/default.mspx?mfr=true.

[34] netcat. http://www.vulnwatch.org/netcat/.

[35] pwdump Home Page. http://www.foofus.net/fizzgig/pwdump/.

[36] Resplendence Software – Rootkit Hook Analyzer. http://www.resplendence.com/hookanalyzer.

[37] Rootkit Detector. http://www.rootkitdetector.com/.

[38] rootkit.com. http://www.rootkit.com/.

[39] Snort – the de facto standard for intrusion detection/prevention. http://www.snort.org/.

[40] Solaris Fingerprint Database. http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl.

[41] Symantec. http://www.symantec.com.

[42] Sysinternals – Filemon. http://www.sysinternals.com/Utilities/Filemon.html.

[43] Sysinternals – PsTools Suite. http://www.sysinternals.com/Utilities/PsTools.html.

[44] Sysinternals – Rootkit Revealer. http://www.sysinternals.com/Utilities/RootkitRevealer.html.

[45] Sysinternals – strings. http://www.sysinternals.com/utilities/strings.html.

[46] Sysinternals Freeware. http://www.sysinternals.com/.

[47] TAFT – The ATA Forensics tool. http://vidstrom.net/stools/taft/.

[48] tcpdstat. http://staff.washington.edu/dittrich/talks/core02/tools/tools.html.

[49] tcpflow – TCP Flow Recorder. http://www.circlemud.org/~jelson/software/tcpflow/.

[50] tcptrace. http://jarok.cs.ohiou.edu/software/tcptrace/.

[51] TCT – The Coroner's Toolkit. http://www.porcupine.org/forensics/tct.html.

[52] The GNU Operating System. http://www.gnu.org.

[53] The KnownGoods Database. http://www.knowngoods.com/.

[54] The National Software Reference Library. http://www.nsrl.nist.gov/.

[55] The Risk Equation. http://www.icharter.org/articles/risk_equation.html.

[56] The SANS Institute. http://www.sans.org/.

[57] The Sleuthkit. http://www.sleuthkit.org.

[58] UnxUtils – Native Win32 ports of some GNU utilities. http://unxutils.sourceforge.net.

[59] Wireshark, formerly Ethereal. http://www.wireshark.org/.

[60] X-Ways Software Technology AG. http://www.x-ways.net.

[61] Richard Bejtlich. *The Tao of Network Security Monitoring*. Addison-Wesley, 2004.

[62] James Butler and Greg Hoglund. VICE - Catch the Hookers! `http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf`, 2004.

[63] Brian Carrier. Open Source Digital Forensic Tools: The Legal Argument. *@stake Research Report*, 2002.

[64] Brian Carrier. *File System Forensic Analysis.* Addison-Wesley, 2005.

[65] Brian Carrier and Eugene H. Spafford. Getting Physical with the Digital Investigation Process. *International Journal of Digital Evidence*, 2, 2003.

[66] Eoghan Casey. *Digital Evidence and Computer Crime - 2nd Edition.* Academic Press, 2004.

[67] Symantec Corporation. Internet Security Threat Report Volume IX – July 05 - December 05, 2006.

[68] Dan Farmer and Wietse Venema. *Forensic Discovery.* Addison-Wesley, 2004.

[69] Simson Garfinkel, Alan Schwartz, and Gene Spafford. *Practical Unix & Internet Security, 3rd Edition.* O'Reilly, 2003.

[70] Alexander Geschonneck. *Computer Forensik.* dpunkt.verlag, 2005.

[71] Tim Grance, Karen Kent, and Brian Kim. *Computer Security Incident Handling Guide.* National Institute of Standards and Technology, 2004.

[72] The Grugq. The art of defiling: Defeating forensic analysis. `http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-grugq.pdf`, 2005.

[73] Greg Hoglund and James Butler. *Rootkits - Subverting The Windows Kernel.* Addison-Wesley, 2005.

[74] Barnaby Jack. Remote Windows Kernel Exploitation: Step into the Ring 0. `http://www.eeye.com/~data/publish/whitepapers/research/OT20050205.FILE.pdf`, 2005.

[75] Keith J. Jones, Richard Bejtlich, and Curtis W. Rose. *Real Digital Forensics.* Addison-Wesley, 2005.

[76] Mark Keith, Clint Carr, and Gregg Gunsch. An Examination of Digital Forensic Models. *International Journal of Digital Evidence*, 1, 2002.

[77] Vincent Lui and Francis Brown. Bleeding Edge Anti Forensics. `http://www.metasploit.com/projects/antiforensics/InfoSecWorld%202006-K2-Bleeding_Edge_AntiForensics.ppt`, 2006.

[78] Kevin Mandia, Chris Prosise, and Matt Pepe. *Incident Response & Computer Forensics - 2nd Edition.* McGraw-Hill, 2003.

[79] NIST - Computer Forensics Tool Testing Project. *Disk Imaging Tool Specification Version 3.1.6.* http://www.cftt.nist.gov/DI-spec-3-1-6.doc, 2001.

[80] Kyle Rankin. *Knoppix Hacks – 100 Industrial Strength Tips and Tools.* O'Reilly, 2004.

[81] Joanna Rutkowska. Rootkit Hunting vs. Compromise Detection. http://www.invisiblethings.org/papers/rutkowska_bhfederal2006.ppt, 2005.

[82] SANS. *Computer Security Incident Handling: Step by Step - Version 1.5.* The SANS Institute, 1998.

[83] Securityfocus. Hidden DRM code's legitimacy questioned. http://www.securityfocus.com/news/11352, 2005.

[84] Securityfocus. Wide-scale industrial expionage using Trojan horses in Israel. http://www.securityfocus.com/archive/1/401130, 2005.

[85] Securityfocus. Blue Security folds under spammer's wrath. http://www.securityfocus.com/news/11392, 2006.

[86] Securityfocus. Seven arrested in online fraud crackdown. http://www.securityfocus.com/news/11385, 2006.

[87] Securityfocus. Symantec closes potential hiding hole. http://www.securityfocus.com/brief/102, 2006.

[88] Securityfocus. Veterans Affairs warns of massive privacy breach. http://www.securityfocus.com/news/11393, 2006.

[89] David Solomon and Mark E. Russinovich. *WIndows Internals.* Microsoft Press, 2005.

[90] Sherri Sparks and James Butler. Shadow Walker - Raising The Bar For Windows Rootkit Detection. *Phrack*, 59, 2005. Article 8, http://www.phrack.org/show.php?p=63&a=8.

[91] UK Association of Chief Police Officers. *Good Practice Guide for Computer based Eletronic Evidence.* National Hi-Tech Crime Unit, 2003.

[92] U.S. Department of Justice. *Electronic Crime Scene Investigation: A Guide for First Responders.* National Institute of Justice, 2001.

All URLs in this bibliography were valid as of July 10, 2006.