Automated Analysis for Digital Forensic Science

By

TYE BROWN STALLARD
B.S. (Willamette University) 1996

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee in charge

2002

Automated Analysis for Digital Forensic Science

Contents

Acknowledgments

No work is the product of an exclusively solitary effort and this thesis is no exception. Without the volunteer efforts of the technical community, this work would not have been completed. I'd like to thank Dan Farmer and Wietse Venema for The Cornor's Toolkit, Ernest Friedman-Hill for JESS, Donald Knuth for TEX, Leslie Lamport for LATEX, the Free Software Foundation and developers for Emacs and the rest of the GNU software suite, RedHat™ Inc. for their Linux distribution and the Linux kernel development team. Open source software is something technical researchers sometimes take for granted, without which progress in academia would be slower.

Many individuals helped me in this thesis. I'd like to thank Chris Wee and Becky Bace for the exceptional technical and corporate experience they shared with me. The Security Laboratory staff, Patty Graves, Jeff Rowe and Poornima Balasubramanyam, provided engaging brainstorming (including the half baked ideas), advice and conversation. My thesis committee, Karl Levitt, Matt Bishop and Felix Wu provided not only their funding and time, but also thought-provoking insights related (and unrelated) to this thesis. I'd like to thank Robin Roberts for her mentorship over the years. Most importantly, I'd like to thank my family for their consistent warmth and support.

Finally, I'd like to thank DARPA for funding this research.

Tye Brown Stallard
December 2002
Computer Science

Automated Analysis for Digital Forensic Science

Abstract

Flaws in system security may persist for the foreseeable future. Yet, software developers and system administrators are not learning from security mistakes because identifying the cause of a computer intrusion is time-consuming, tedious and unlikely to yield definitive results. Investigations are fraught with data volatility, privacy and legal issues as well. When intrusions *are* detected, computer forensics analysts are swamped in evidence because of the large volume of data encountered, the dearth of trained investigators and the lack of automated techniques to analyze computer crime data. An expert system with a decision tree that uses predetermined invariant relationships between redundant digital objects (like an application log entry and an audit trail) to detect semantic incongruities could augment a computer crime investigator's efforts. By analyzing data from a system and searching for violations of known data relationships, an attacker's changes to the system may be automatically identified. Examples of such invariant data relationships are provided, as are techniques to identify new, useful ones. A requirement for such a system is to have the evidence available in a standard machine-readable format. A prototype of this general approach has been written, integrating The Coroner's Toolkit and JESS, The Expert System Shell for the Java Platform, that automatically identifies files that have been modified, accessed or changed when their owners were not logged in. By automatically identifying relevant evidence, experts can focus on the relevant files, users, times and other facts first.

Chapter 1

Introduction

The best way to stop a computer attack is to prevent it from happening. If an attack does occur, being prepared for it by implementing preventative measures is the best way to effectively and efficiently react and recover. A clear policy of what is and is not allowed disambiguates the administrator's job. Consistently executed procedures of software maintenance and user management, and properly configured authorization and access control enforcement mechanisms prevent misuse. Logging mechanisms, whose logs are stored securely, empower the defender to look into a computer's or network's past to determine the causes and effects of computer misuse. Periodic backups will stop data loss and corruption. Unfortunately, these steps are not always taken. In a world of imperfect hardware, software and people, with budget, time and knowledge constraints, perfect security cannot be achieved. To understand security violations, experts require tools and techniques to augment their skills. This is the role of digital forensics.

When most people think of "forensics," they think of police detectives at the scene of a murder for example, collecting blood samples, fingerprints and taking pictures. A crime has occurred and a law enforcement agency is gathering evidence, identifying a suspect and building a case against the alleged perpetrator. This process is as old as the legal system. For common crimes, the problems and approaches to solving mysteries are familiar. The state establishes motive, means and opportunity for a suspect of a violation of a specific law, and tries him or her in court to obtain a conviction. Although technology crimes may appear novel, their nature remains the same. The motivations of criminals rarely change, but their methods do. So do the challenges to law enforcement. Since the buying habits of technology consumers dictate new and improved features, technology is usually designed with functionality and cost as primary concerns, not security. Yet, now that society increasingly relies upon proper operation of information technology, criminals are increasingly using and targeting computers. This leads to a dilemma for those trying to detect, prosecute or prevent a crime from occurring. They have the responsibility to stop crime, but not the means.

The perfect solution would be an unambiguous way to apply and enforce the law in a way that mistakes never occur. Of course this will never happen. Fallible technology is installed and operated until it breaks. Often, this is actually acceptable. For example, it is common knowledge that credit card fraud occurs, but financial companies, a technology-intensive industry, recognize, quantify and accept it. They do this because they calculate how much fraud and business interruption they can tolerate according to their business plan. Home users may not be so methodical, but go through a similar process. Due to the unreliability of computers, users learn not to depend on them. They will just reboot their computers if the software freezes. Unless there is a viewable side effect, most users would never know if their personal files had been stolen or maliciously, yet subtlety altered.

If computers will have imperfect security for the foreseeable future and crime will go unprevented, what can be done to at least learn from experience? Answering questions of method, intent, means, culpability, motive and loss resulting from cyber crime is the domain of computer forensics. Forensic science is the application of science to the law. Although no automated analysis tools that have knowledge of the semantics of the evidence are currently available, this thesis shows it is possible to apply techniques of computer science to the huge amount of digital evidence gathered in an investigation. Using automated data normalization and expert system reasoning, a program can calculate simple deductions that can help answer questions of illegal activity. Major uses of an expert system for computer forensics include investgation of federal, state, and local law violations as well and corporate policy and contract violations.

1.1 Thesis organization

This thesis is organized into seven chapters. Chapters 1 and 2 introduce the subject matter and context into which this thesis fits. Chapter 3 elaborates on issues upon which most digital forensic work has focused: gathering the input (raw data) to the forensic analysis process. Chapter 4, the core of this thesis, presents my general approach and will include specific examples, noting the various stages of an automated computer crime investigation. To show the feasibilty of this approach, chapter 5 presents a prototype and the results of its use in a simulated computer intrusion. Finally, chapters 6 and 7 discuss applications, perspectives and implications of this approach to computer forensics and some summary thoughts.

Chapter 2

Background

Digital forensic science, or computer forensics, is a new discipline in an old field: forensic science. Forensic science, defined as "the application of science and engineering to the law," is an established scientific discipline with a great deal of history. Due to society's increasing reliance on information technology and the inevitability of crime, the need for substantive advances in digital forensic science is important. The following is one definition of the discipline:

> The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitation or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disrupting to planned operations. - DFRWS [11]

Work has been done to apply existing forensic science techniques to the digital realm, but there is a great distance to go. The amount of evidence law enforcement alone has to process has increased every year. Supervisory Special Agent Mark Pollitt, of the FBI Laboratory's Computer Analysis Response Team (CART), stated at the 2002 University of Idaho Forensics Workshop that by the end of 2002, CART will have processed approximately 500 terabytes of digital evidence.

Currently the primary factor limiting the evidence process is the number of qualified technicians. The majority of digital evidence processed by law enforcement today is on computers used as *instruments* of traditional crimes, such as a threatening email rather than a threatening letter. Although a minority of prococuted cases, crimes in which the computer is the *target* are important as well. This difference in the number of cases is due to many factors including workload of investigators, familiarity with technology crime and likelihood of a successful conviction. This thesis will focus primarily on computers as the target of crime.

2.1 Scenario and Motivation

If you tell the truth you don't have to remember anything - Mark Twain

Suppose I, the network administrator responsible for computer security, get a telephone call informing me that one of my hosts is scanning and attacking other hosts on the Internet. Since the IP source address of the attacks to the victim may have been spoofed, to confirm the complaint, I will monitor the local network for malicious activity. Malicious network activity is detected. At this point, I must make a decision based upon available information, corporate policy and time constraints.

A known policy violation exists and the importance of the host to my organization is known. The likelihood of learning whether this incident was perpetrated by a single person or coordinated among multiple attackers is unpredictable, as is realizing that this attack may be a single instance of a set of related events. As the network administrator, I can choose between performing a full investigation, according to sound forensic evidence practices, or not. The typical answer is to investigate if the host is important, such as a mission critical server, or if valuable data is housed there. If the host belongs to an end user and user data can be recreated or restored, the operating system will just be re-installed.

Neither option is optimal. If we ignore the computer intrusion or reinitialize the system, no lessons can be learned. Perhaps there is a common software vulnerability that can be fixed. Perhaps there is a system configuration problem common to hosts throughout the network than can be exploited again. Maybe the user was tricked into installing a Trojan horse keystroke logger that delivered passwords or trade secrets to the company's competition. None of these possibilities can be ruled out if there is no evidence to examine. Just as importantly, experience with vulnerable software, poor practices and inadequate procedures cannot be shared with peers, within the company or in the community, if these problems are never pursued. Finally, corporate liability may be incurred. "Downstream liability," where a company is liable for the malicious activity of its users, authorized and unauthorized, creating damage elsewhere on the network is a possibility. A standard practice of ignoring malicious activity could also be construed as negligence or malfeasance of management if corporate operations fail.

On the other hand, given the state of information technology and likelihood for mischief, always performing a full investigation would be cost prohibitive. Each investigation will be time intensive and the local expertise of the company, a critical and finite resource, will be preoccupied with the investigation and not contributing to corporate productivity. Since proper seizure of data and documenting the state of the system is important, system down time could be a significant factor directly affecting customers and the productivity of employees. Finally, there is the potential for corporate liability to be incurred. The subject of an investigation may sue, for violation of an expectation of privacy for example, and if contraband is found, even unrelated to the investigation, the company is responsible for it.

The need exists for a diagnostic capability to rapidly answer operational questions and deal with the legal issues later. Just as fire fighters seek to save lives and property first, and legal staff determine culpability of suspects later, computer crime fighters must first perform triage to systems. With insights learned from formal, digital forensic investigations, in the future it may be possible to design systems with forensics in mind. More reliable, trustworthy and efficient mechanisms that aid the investigator would help. Common operational questions today though, are broad and ambiguous in nature:

- How did they get in?
- How long have they been in?
- What are they doing now?
- What did they get?
- How do we get them out?
- How can we prevent a recurrence?

- How can we deceive the attacker?
- How can we identify incriminating evidence on the attacker's host?

Answers to these questions may be approximated based upon evidence left behind by the attacker. The kinds of evidence available will be related to the sophistication of the attacker. A novice with little experience, no custom-built tools and no effort made to remove evidence will leave obvious signs. Anomalous log entries would be a simple example, but finding directories with the default names used by common root kits used to hide evidence is useful as well. `chkrootkit` [28] is an open source project that uses this signature based approach. An experienced attacker on the other hand, will attempt to create an illusion to the administrator that a security violation did not occur. Malicious alteration of the operating system's kernel code, through an unauthorized kernel module for example, will *corrupt* the results of even a trusted, statically compiled binary used to detect surreptitious activity. The attacker will also meticulously edit log files and other evidence of his presence. Hopefully, side effects of his presence though will still exist. Consequences of his activity, that he either doesn't realize or doesn't have the time and access privileges to eliminate, will remain. Only the perfect attacker will be able to perfectly recreate the state of the system as it was before the security violation.

2.2 Related Work

2.2.1 Existing digital forensic analysis

The current generation of forensics tools is based upon the hard drive as the basic unit of evidence. Many tools focus on duplication of data. To ensure the forensic integrity of evidence, a trustworthy image of *every* location data may reside on the disk is necessary. Work has been done on reliable imaging (not just duplication of allocated disk blocks), such as the Linux `dd` utility, and for specific hardware formats. Disk imaging tools have no analysis capability. The list of publicly available and peer reviewed automated analysis techniques is short. While this is not an exhaustive survey, the state-of-the-art automated forensics techniques are unsophisticated and evidence analysis is highly dependent upon the expertise of the examiner.

Keyword search

The most common analysis technique is to search for strings of characters. Known as a keyword search, investigators assemble a list of words specific to the investigation and words commonly used in criminal investigations for example, and look for matches on digital media seized as evidence. By searching the physical media, the investigator not only searches every file in the file system, but the blocks marked as unused as well. In addition, this technique searches slack space; those data locations contained within the list of disk blocks allocated to a file, but outside its contents. This is data left over from deleted files, but whose disk blocks were recycled and not yet overwritten. Unfortunately for the investigator, the capacity of disk drives is growing at a much higher rate than their data transfer rates. Searching a single hard drive takes a long time. To aggravate this situation, as investigators learn more through the investigation, new words may be added to the "dirty word" list, which necessitates another search of the entire drive. An approach to mitigate the waiting time for keyword searches is to index all printable strings on the hard drive once. Although this takes hours, subsequent searches are fast and need not be predetermined.

Often, the most useful evidence collected is that which the suspect believed was deleted and forever unavailable to law enforcement. This easy source of incriminating evidence is an artifact of current software implementations and misconceptions of end users. Changes in either will make the job of the digital forensic evidence examiner much more difficult. With the conspicuous exception of the FBI [18], EnCase [35] by Guidance Software is the most popular forensic collection and analysis product of law enforcement. Guidance Software has a statement on the disk wiping utility that ships with Microsoft Windows XP operating system:

> The scrubbing feature is part of Windows XP, but it is not all that it was thought to be. It is a command line tool that is difficult to use, time consuming and nothing more than a good wiping utility. The average computer user will not know how to use it and even if it is used, evidence artifacts still remain in certain file systems. Guidance Software [24]

I find it a precarious situation for law enforcement to depend on the combination of the ignorance of users and the poor usability of disk wiping utilities to catch criminals.

“Thumbnail analysis”

For files that do not have printable strings, simple keyword searches do not work. Images are a typical example. The common technique for examination of images is to simply present thumbnails of the images to the investigator to quickly look at many images at a time. Though looking at an image does not take special technical skills, determining its relevance may. Using this approach, the only way to accelerate the investigation is to increase the number of investigators.

File extension verification

When suspects were found to be changing the file name extensions of files to obfuscate their content (specifically, changing an image file to appear to be an executable binary), a technique was developed that simply compared the first few bytes found in the file to the first few bytes expected based upon the file name extension. For example, for a file that has a `JPG` file name extension, one would expect the standard JPEG header at the beginning of its contents.

Checksum database

With thousands of files on a typical home computer or many more on an ISP’s server, an efficient method of identifying (and ignoring) unaltered, common files is needed. When suspects were found to be hiding evidence by renaming files to the same names found in standard operating systems or software applications, this problem was aggravated. By collecting a database of cryptographic hashes of files from legitimate software packages and comparing the hashes of files from the body of evidence, an investigator can identify and ignore common files. This idea was formalized and standardized in the National Software Reference Library (NSRL) at the National Institute of Standards and Technology (NIST). Law enforcement has similar databases of evidence found at crime scenes such as inks, fibers, firearms and fingerprints for example. To extend this idea, the Hashkeeper Database project [6] was created collecting cryptographic hashes of contraband files and has been successful in aiding investigators.

2.2.2 Research projects

There have been a number of projects that have applied artificial intelligence techniques to the prevention and detection of computer attacks, particularly audit trail analysis and intrusion detection. An example is eXpert-BSM [26] which is a forward-reasoning expert system designed for real-time analysis of Sun Solaris audit trail data (collections of system calls). It was designed for intrusion detection and *preventing* a security violation and is based upon a forward-chaining, rule-based knowledge base: P-BEST, which reasons directly from the raw data. If the costs of storing audit trail data were justifiable, it would be a rich source of data to learn the causes and effects of an attack that succeeded.

Few projects have focused on recovery from a successful attack. The "Diagnosis, Explanation and Recovery from Break-Ins" (DERBI) [42] project at SRI used a procedural reasoning system [29] to analyze data. If data was collected after an attack, it could search for patterns of attacker activity based on known *procedures* for subverting security. In addition, Elsaesser and Tanner present an approach [13] which automatically generates hypotheses of computer attacks, simulates them on the target configuration and applies plan recognition techniques [23] to search for supporting data.

My approach, which examines evidence for violations of specified relationships between data in existing software architectures, is a form of integrity checking first studied in the Clark-Wilson Integrity Model [4]. By applying that model to a database, for example, a formal specification of relationships between fields in tables of the database is checked to verify data consistency and database integrity. These are known as constrained data items (CDI). Tripwire [22] is an application of integrity checking to system security.

Chapter 3

A model for computer forensics

"Now, a few words on looking for things. When you go looking for something specific, your chances of finding it are very bad. Because, of all the things in the world, you're only looking for one of them. When you go looking for anything at all, your chances of finding it are very good. Because, of all the things in the world, you're sure to find some of them." - Darryl Zero [46]

A perfect computer attack would leave no unintentional evidence of its existence. The unknown or unintended side effects of the attacker's activity though, can give a defender an idea of what happened. An unknown configuration file of a hacker tool or forgotten log entry can be a significant clue. When an attacker attempts to hide his presence though, creating a realistic illusion may be difficult. A key contribution of this approach is to use multiple sources of such side effects and apply automated reasoning techniques to identify and characterize activity attributed to an attacker. Since human experts will always be required to deal with human attackers, due to the unwieldy amount of actual and potential data available to a computer crime investigator, automating as much of the data collection and analysis process as possible is critical. Reliable automated differentiation of relevant and irrelevant evidence would be an achievement.

To these ends, encoding an expert system with a decision tree that uses predetermined invariant relationships between digital objects to detect semantic incongruities could aid an investigator of computer crime. The first step is to identify those useful relationships between specific digital objects in a system that could indicate the activity of an attacker. Next is to collect that data from a potential or actual crime scene. Finally, automated data aggregation and analysis of the body of evidence will identify those specific data that violate the expected model of the system. Depending on the context of the application of this approach and the maturity of the automated reasoning knowledge base, the potential exists for automated response as well.

3.1 Data collection

Despite the common belief that the Internet provides anonymity, there is a great deal of data available to the investigator to answer relevant questions. On the other hand, it is the rare instance that enough evidence exists to prove beyond a reasonable doubt that a specific individual was responsible for some specific illegal activity. Fortunately, not all relevant questions are necessarily intended for criminal prosecution. Depending on the purpose of the investigation, characteristics of different types of data, such as volatility, may aggravate or mitigate the process. There are nontechnical factors to be considered as well. Much has been written about the legal issues that surround the collection of data to be used as evidence [30]. The applicable laws written long ago, are often difficult to apply to modern computers and their networks. The legal standard of "best evidence" for instance, is overkill for all but litigable purposes. Yet, there's no way to tell before the completion of an investigation if it's worth the time and effort to follow formal procedures.

3.1.1 Purpose, means and investigating agency

The purpose of the investigation will direct the types of data collected and affect the means by which it is collected. In the traditional forensic analysis example, law enforcement agencies will collect digital evidence in order to support a legal argument that a crime has occurred. Due to evidentiary rules, they physically seize the storage media and duplicate it. Analysts only examine copies of the original data and may be required to compare cryptographic hashes to the originals to verify that the data copied correctly. On the other hand, for an organization that only wants to stop a denial of service, it would be illogical to bring the system offline in order to physically isolate the potential evidence. Instead, they may use the data on the affected and related hosts directly (rather than a forensic copy for evidence purposes) to determine a solution. A company searching for a rogue, trusted insider, may search for incriminating evidence against a suspect, but not pursue litigation for fear of public humiliation. Data collected need not convince a jury, but only the target of the internal investigation. In yet another scenario, an insurance company may write a cybersecurity policy that reimburses a company for computer attacks but not for technical mistakes. They would only collect data to determine if the consequences of a loss of service were intentional or unintentional. Finally, an individual may intentionally allow his or her personal machine to be penetrated in order to learn the technique and underlying vulnerability that was exploited. Collecting access control logs would yield no new information because the owner is the only user. Core dumps, images of memory and error codes, for example, would help in locating the vulnerability in source code.

3.1.2 Characteristics of data to be collected as evidence

Physical location

Whether it is a distributed denial of service [17], a fast moving worm affecting thousands of hosts [37], or an astronomer tracking down a 75 cent accounting error leading to evidence of hacking and international espionage [39], system administrators use many sources of data to identify and implement a solution to a problem. RFC 3227 [2], "Guidelines for Evidence Collection and Archiving," identifies many locations of evidence:

```
Here is an example order of volatility for a typical system.

      -  registers, cache
      -  routing table, arp cache, process table, kernel statistics,
         memory
      -  temporary file systems
      -  disk
      -  remote logging and monitoring data that is relevant to the
         system in question
      -  physical configuration, network topology
      -  archival media
```

Volatility

Data volatility is a critical issue because highly relevant data, such as the owner of a malicious process, is easily lost. For reasons of storage capacity not all relevant data can be recorded in nonvolatile storage. For reasons of practicality though, storing all data ever created would not be reasonably searchable. System call audit trails, such as BSM [21] or WinNT C2 auditing, are examples of thorough, but unwieldy evidence trails. In a literal sense, they can explain everything that lead to the insecure state of the computer, but the relevant and irrelevant data cannot be distinguished and the data volume is enormous. As time passes, relevant information may be lost due to events such as memory being reused for another process or a system reboot. A common technique for attackers with unauthorized access is to hide a file by running a process with an open file descriptor. The file itself is then deleted. This allows the attacker's process to read and write data to the file system in a place the administrator cannot access through the file system. To the administrator, that location on the disk appears to be unallocated. When the process ends, the file descriptor is released and the location on disk is then vulnerable to being overwritten. Given the passage of enough time, though data in nonvolatile storage can be lost as well to events such as log file rotation and the reuse of archival media.

Ownership

To actually collect the data, its physical location is important, as is its volatility, but these are not the only factors. Gaining administrative access (e.g. identifying the owner, location and authentication credentials) is just a matter of policy and procedure, but is a real problem. Seeking the source or just documenting effects of malicious behavior often crosses administrative boundaries. Whether between departments, companies or even countries, those seeking potential evidence require the cooperation of those responsible for remote systems. If voluntary cooperation fails, law enforcement can exercise the privilege to force compliance. Of course, cooperation between law enforcement agencies, especially between sovereign nations, may not be fruitful either. For time-sensitive data (as most is), efficient cooperation not only lowers the cost and time of an investigation, but can actually be the difference between success or failure.

As in the nontechnical world, basing deductions on uncorroborated evidence is risky. In the technical realm, because data can be easily altered and the process automated, one must be skeptical of the veracity of all data. From an investigative perspective, evidence from more than one administrative domain that do not contradict eachother lends credence to the legitimacy of the evidence. A designated remote log host is an uncomplicated way to replicate logging data to a different administrative domain that creates many problems for the attacker hiding his presence. This requires prescience.

Trustworthiness

Since attackers can trivially alter data that may be used as evidence, the trustworthiness of the data is affected by the stage of the attack in which data collection occurs. Collecting data before an attack is a proactive way to be able to quickly react on reliable data. Administrators become familiar with events that normally occur and their relative frequency. If promptly detected, collecting data during an attack can provide the administrator the most relevant data with the least amount of overhead. Examples of such data include the attacker's actual keystrokes or just those network packets sent between the attacker and the compromised host. Collection of data only after an attack has occurred, which is the common case, causes complications. Questions of completeness and veracity can never be fully answered because the potential always exists that the attacker altered sources of evidence maliciously or normal processes altered the data *innocently*. The ability to determine what happened in the system depends on the sophistication of the attacker and the expertise of the investigator. Unfortunately, tools that cover the tracks of an attacker are much more common than those that uncover those tracks.

Abstraction

The type of data collected has a direct effect on the questions that may be answered. Data collected at a level of abstraction (one set of objects and the relationships between them), such as the users, terminals, and times in operating system login logs, may answer useful questions, but be unable to answer questions at a lower level of abstraction ("Why did the login daemon record these values in *this* login entry?"). Access logs for example, may allow one to determine what user accessed what resource at what time. Like credit card receipts and videotapes for a convenience store robbery, these logs can be useful in recreating a sequence of events that occurred during an intrusion. Error logs and audit trails (system call logs) on the other hand, can explain a sequence of events within an application. At this level of abstraction, it may be possible to determine what caused the application to crash or perform unauthorized actions. Network intrusion detection systems, firewall logs and stored packet streams are another level of abstraction and forms of auditing that are often in a different administrative domain than the potential victim hosts they are designed to protect. As Carrier [3] points out, some abstractions discard information from the things they represent whereas other abstractions lose no information: lossy versus non-lossy abstractions. A network intrusion detection alert, for example, simplifies the actual data packets the alert was based on, whereas an `ASCII` character requires all seven bits to be represented.

3.1.3 Legal issues

In traditional crime scene forensics, the process law enforcement agency follows to collect evidence is a highly formalized and regulated endeavor. To protect the rights of the innocent and guilty alike, rules of evidence have been established that manage what evidence may be taken into custody by the state, how it is handled and if it may be presented in court. Traditional crime scene forensic procedures, though, have not transferred well to computer crime. In a literal sense, digital objects exist in a physical sense in their given storage media. Yet as opposed to handwriting, bullet rifling marks or DNA samples, digital objects are infinitely malleable and perfectly replicable. Archival media, such as the Compact Disc-Recordable (CDR) standard, may be writable only once, but in an operating computer system, data on the hard drive, swap file, main memory, caches and registers can and do change frequently. In spite of the legal complications digital evidence poses, data is still collected for investigations with purposes other than criminal or civil litigation.

Law enforcement

In the United States, rules of evidence have numerous origins including English Common Law, the Bill of Rights, legislation and court case precedents. Common standards of practice that comply with existing law, have not yet matured in the collection of computer crime evidence. In the United States, under the Federal Rules of Evidence, and the rule of "Best Evidence" in particular, to prove the content of a document, the "original copy" is required. The Federal Rules address computer data specifically:

> "[i]f data are stored in a computer or similar device, any printout or other output readable by sight, shown to reflect the data accurately, is an "original." Fed. R. Evid. 1001(3) [30]

While many copies of the same data may be created by being output to some form "readable by sight" each is considered an "original" for legal purposes. An argument can be made that only the physical instantiation of computer data, its storage media, should be considered the original. The final interpretation was a compromise though, so that computer data could be useful as legal evidence.

> "While strictly speaking the original of a photograph might be thought to be only the negative, practicality and common usage require that any print from the negative be regarded as an original. Similarly, practicality and usage confer the status of original upon any computer printout." Advisory Committee Notes, Proposed Federal Rule of Evidence 1001(3) (1972). [30]

Finally, it is possible to present an automated analysis of evidence in court even though a person has not performed this part of the computer crime investigation.

> "Federal Rule of Evidence 1006 permits parties to offer summaries of voluminous evidence in the form of "a chart, summary, or calculation" subject to certain restrictions." CCIPS section: V.D.2 [30]

Non-law enforcement

Whereas law enforcement agencies have strict legal requirements for collecting and handling evidence, a company for example, with no intention of seeking legal relief, does not. If employees have an expectation of privacy or if the company is collecting data on behalf of a law enforcement agency, certain laws will still apply. Once a legal expectation of privacy has been eliminated, since the company owns all the computers and networks concerned, it owns and can collect the data as well. From the perspective of a computer crime investigator, more data to analyze is good.

Companies may actually have a legal disincentive to collect and store data. Their data backup and document retention (or deletion) policies exist to reduce corporate liability in the case of court subpoenas. When the company is aware of a policy or legal violation, there may be a responsibility to react. Although historical data may show the culpability of a suspect in a violation, it also may show the culpability of the corporation to different violations. Yet even though storage media and local area network bandwidth are inexpensive, since most data are useless, the liability and management costs outweigh the data's utility. System call audit trails are an example. So another rule of thumb is not to log any data that is not regularly used.

Organizations receive contradicting advice. During an investigation, collection of a great deal of data is advantageous. Between investigations though, time and budget constraints dictate a minimum level of data collection. Automatic, dynamic adjustment of the quality and volume of logging, analogous to a motion sensor that triggers a video camera, could meet these opposing requirements. Though legal standards for how to collect and preserve evidence are becoming common, answers to questions such as what data to collect, when to collect it and what level of abstraction of data to store are still an inexact art.

To address these issues, an automated approach, to reduce the challenge of data volatility, that is deterministic, for purposes of repeatability and consistency required in legal settings, would address difficult problems faced by investigators today. Knowledge engineers could design a knowledge base relating data from various levels of abstraction in a system, considering issues of trustworthiness based on the data's source, so that instead of re-learning how components of a system interact during every investigation, knowledge can be saved and reused. This allows experts to focus on new problems that arise and require judgment an automated system cannot provide.

Chapter 4

Approach

Currently, a computer forensic examiner begins to analyze data at the lowest layer of abstraction, physical locations on disk, and continues in an ad hoc process up the layers of abstraction of the operating system. While the brute force approach of examining every physical address on the media may ensure meticulousness, the cost in time and resources make it unusable in common scenarios. If investigating crime is more expensive than ignoring it, it will be ignored. If cursory investigations were possible and identification of novel cases was more efficient, administrators and developers may find it valuable to spend the time to learn from their mistakes. If identification of the most egregious violations of law was more efficient, caseloads for law enforcement, where evidence standards are the highest and budgets are limited, could be better prioritized. Automated analysis of technical evidence is an obvious approach.

Even if logging data and technical evidence are collected, if not examined, no insight can be gained. To analyze the body of evidence available, the investigator must first learn what information has been collected, then answer questions with it. Currently, this process is either dependent upon highly trained investigators (who become overloaded) or a proprietary product such as EnCase [35]. Current products designed for forensic analysis gather data according to procedures intended for law enforcement and provide arguably simplistic functionality for law enforcement specific purposes, which limit their utility. The sophistication of computer crimes, has greatly outpaced advances in the analysis of evidence at such low levels of abstraction. In either case, investigators are given too much data to analyze now and it will continue to increase. The solution is to automate simplistic tasks for the investigator and encode expertise into a program that can automatically make deductions that are relevant to an expert.

Obviously, detection of malicious activity is much more difficult when looking only at individual magnetic flux representing logical zeros and ones on physical digital media. So first, raw evidence must be aggregated in a rational way into more abstract objects. Ignoring hidden or encrypted data for the moment, even examining the contents of thousands of files may leave elementary questions unanswered if the investigator is not utilizing the correct programs to parse and interpret the data. Since operating systems and their applications normally build digital objects (in this thesis understood to be abstractions) using lower level abstractions, which is invisible to the user, this step may seem obvious. Yet since these are the very mechanisms attackers subvert, the raw data itself must be independently interpreted and organized and is therefore part of the forensic analysis process. At this point, standard analysis techniques may be applied. Then, using abstract digital objects that computer users take for granted, it is possible for a program to automatically search for and detect violations of invariant relationships between objects previously established by forensic experts. An invariant relationship is a specification (a form of policy) between digital objects that holds true in a system operating in an authorized state. A positive side effect of automating this process is that the assumptions and specific evidence leading to a deduction will be documented.

4.1 Data aggregation

Common standard practices after computer intrusions are to either reinitialize and reinstall the operating system or do a full forensically sound investigation. To accelerate a full investigation or when “best evidence” procedures aren’t required, such as an internal corporate investigation, or a personal system, evidence could still be stored and shared in a standard format and its analysis automated using the approach described herein. Caseloads for law enforcement, where evidence standards are the highest, may be prioritized as well, for example by identifying those cases with evidence to support a claim, and those cases that cannot be proved.

When solving a problem, such as how an attacker penetrated security countermeasures, experts don't usually think of computer data in terms of physical media. They think about files, users and activity like an InterRelay Chat (IRC) conversation. By sharing evidence at the level of abstraction desired, investigations would cost less and be completed faster. A standard set of objects or a common language for technical evidence would provide many benefits.

The process by which law enforcement collects and stores data that may be used as evidence is the legal standard of "best evidence." While the legal standard of "best evidence" has its merits, it is not ideal for all computer investigations. Even in investigations for law enforcement purposes where "best evidence" is the minimum standard, since evidence is only shared in its most basic form, its storage media, analyzing the evidence is time and labor intensive. Instead of only sharing the physical media, if a data standard was established to share evidence at higher levels of abstraction, software and organizations would gain benefits.

4.1.1 Example data-sharing standard

Though an exhaustive examination of the subject is beyond the scope of this work, a technical standard, such as an XML DTD, may include attributes of objects at various levels of abstraction such as the following Microsoft Outlook email example:

1. Raw data (logical ones and zeros only)
2. Aggregated data
    (a) Application
        i. Text of message
        ii. Sender/receiver
        iii. Sent/received time
        iv. Mail servers that delivered the message

v. ESMTP ID for each mail server

vi. Message-ID

vii. Content-type

viii. User-Agent

ix. Data specific to the ".pst" format

x. Embedded message

A. Forwarded message

B. Message replied to

C. Attachment

(b) Operating system

i. Host name, path to file and name of file

ii. Owner/group of file

iii. Modification/access/creation times of file

iv. Previous modification/access/creation times (from an MFT in NTFS)

v. Microsoft Universal Identifier (UID)

vi. Disk locations of data

vii. Audit trail information (userID etc.)

viii. Related temporary files

(c) Hardware

i. Unique identifier for storage media

ii. Make/model of storage media

3. Metadata

(a) Identity of individual who collected the data

(b) Time stamp of when it was collected

(c) Cryptographic checksum of raw data

(d) Tool that collected the original data

(e) Tool that aggregated (interpreted) the data

(f) Sources of data (Abstract concepts may be inferred from multiple sources.)

  i. Sender's host

  ii. Sender's mail server

  iii. Email attachment virus checker activity log

  iv. Receiver's host

4.1.2 Benefits

The definition of a language for technical evidence or the formalization of a standard set of digital objects at various levels of abstraction would separate the data aggregation process from its analysis. Automating the data aggregation process for forensic purposes will allow for more efficient use of an expert's time. Once in a standard machine-readable format, automated exchange of data can occur.

Once interpreted by trusted programs (as opposed to potentially subverted programs) and aggregated into a standard format, evidence could be shared yet the output of the analysis and the process by which it was analyzed could be concealed. Between prosecution and defense attorneys for example, this may be an important distinction. Both parties could have a common understanding about the facts of the evidence, but analyze it in different ways to support their respective theories. For example, two investigative firms are retained by opposing counsel. One has special expertise network flow analysis, whereas the other focuses on proprietary aspects of operating systems. Each has developed a valuable knowledge base of invariant data relationships in its domain of expertise. The evidence is not sensitive, but the knowledge bases are. Independent analysis of the same evidence could become at least feasible, if not common practice. Data could be shared and analyzed in different ways to corroborate conclusions independently.

In addition, new possibilities for evidence sharing policies could be created. Instead of disclosing either none of the data or all the data in its most raw form, by sharing data at the level of abstraction permitted, such as network events from an intrusion detection system, but not network packets, restrictions based upon that level of abstraction could be enforced. This could be useful when sharing evidence between federal, state and local law enforcement authorities or when two or more corporations need to cooperate in one context, but compete in others.

> The national discussion, development and adoption of common data-sharing and communications protocols would greatly improve the ability to share information across jurisdictions. A tightly integrated data-sharing approach, engineered into the next generation of investigative solutions, would provide the foundation for national cyber-attack information databases. Michael Vatis [45]

Organizations with different priorities could collect low-level technical evidence with relevance to their respective goals (See section 3.1.1). While law enforcement organizations will collect technical data with the purpose of solving and prosecuting a serious crime with significant damages, a company in the course of normal operations will not. A company will collect data in order to maintain operations, optimize them or prevent exfiltration of proprietary data. A risk analysis of what data to collect, preserve and analyze should eventually reflect its impact on the company's profit and loss statement. On the other hand, a loose collection of computer security aficionados, with no corporate revenue stream to protect, may be more interested in collecting forensic data based upon the popularity of software with a security flaw and the impact on the community. Below is a non-exhaustive list of purposes for collecting data with forensic investigation applications:

- Seriousness of a committed crime
- Effect on corporate revenue (or profit & loss statement)
- Cost of replacement (of a proprietary data set for example)
- Simplicity of solution (in order to solve the easy problems first)
- Potential for loss of life
- Potential opportunity cost (of a stolen proprietary data set for example)
- Popularity of software with a security flaw

If organizations that collect data for different purposes were able to export it to a common format, an analysis tool would not have to support countless formats specific to hardware, operating systems, file systems and applications. Likewise, given a set of data described by this common forensic standard, it could be analyzed by different tools, written and operated for different purposes.

4.1.3 Iteration between data analysis and collection

Searching for hidden meaning, such as a steganographically hidden message, is obviously data analysis, but the output could be considered part of the process of data collection. The hidden message is at a higher level of abstraction than the document or image that conceals it. The description of a Microsoft Word document, an abstract object itself, is really the aggregation and interpretation (or analysis) of blocks and sectors on disk which are just objects of a lower level of abstraction. Even the interpretation of bits into `ASCII` characters is a type of analysis. So normalizing raw data in order to construct an XML document of evidence *is* a type of analysis. A more esoteric example, but one that I have encountered, was an email in a Microsoft Outlook inbox with an attachment of a presentation slide of an image of a screen shot of a spreadsheet. Although this may appear to be a contrived and therefore anomalous example, because email was the means the organization used to store and share business documents, it was not unreasonable. Without automated interpretation, enough layers of abstraction, even with benign intent, will cause an investigator's time to become the bottleneck in an investigation.

4.2 Identification of invariant relationships

Mechanisms such as audit trails, firewall logs, IDS and application logs, and accounting records, intended for security enforcement or monitoring purposes obviously provide a direct means for the investigator to learn the sequence of events leading to a security breach. Yet these mechanisms may not have been installed and configured in the first place, they may have been maliciously disabled or their output may have been deleted through log file rotation, or altered by an attacker's log wiping program. Modern computers, through their operating systems and applications, store massive amounts of information even in a default configuration. File system metadata, log files from the default logging configuration and application-specific file formats are examples. Unlike a database with an efficient data model that has been put into third normal form, there are redundancies in this information. By looking for these redundancies, and the mechanisms from which they arise, it is possible to elucidate relationships in the data that should always hold true.

4.2.1 Examples

Identity property

Of course the simplest invariant relationship between two digital objects is the identity property. In other words, for a given object such as a log file entry, a copy of it exists elsewhere. Use of this property with log file centralization is particularly effective when an attacker adds, deletes or modifies suspicious log entries on the penetrated host. In this scenario, it is easy for the investigator to compare the log file and the remote copy on a trusted centralized log server, searching for differences. If an unauthorized difference exists, since the assurance of the logging mechanism has been compromised, future log entries from that host cannot be considered trustworthy because entries may have been forged, edited or eliminated.

File access times

An operating system's file system stores metadata useful to the computer crime investigator. In Linux's EXT2 and UNIX's UFS for example, a file's metadata, such as owner, permissions and last access time, are stored in a data structure known as an inode. For every file, the modification time (`mtime`), access time (`atime`) and time of the last change of the inode (`ctime`) are recorded. With knowledge of the semantics of each of these fields, an inherent redundancy becomes apparent: although a file's inode may change at any time, since a file modification changes the modification time, the inode itself must change as well. Therefore, for a given file, the `ctime` field must never be less than the `mtime` field.

wtmp and utmp

The DERBI [42] system, designed at SRI, noted duplication of some, but not all, data in the `utmp` and `wtmp` files in Solaris UNIX. These files are accounting records that collect the log in and log out times of each user, in addition to system reboots and shutdowns. If an attacker modifies one and not the other, or modifies both, but inconsistently, his presence may be detected. Unfortunately, the mechanism that records these data is common to both, so if it is compromised, both logs will be consistently unreliable. Another simple, but effective invariant DERBI noted was that log entry time stamps must always be monotonically increasing. Chronological gaps and log entries out of chronological order are highly suspicious.

Shrinking log files

Tsutomo Shimimora describes in Takedown [33] an elegant invariant relationship about log files; they should never grow smaller. It was effective because he configured them to be backed up on a log host and able to compare the size of a log file to a copy from some period in the past. Only attackers would edit them in a way that decreased their size.

Sequential inodes

When an attacker subverts the security of a computer, a common technique is to replace system binaries with Trojan horses that conceal the attacker's presence. A `rootkit` replaces important operating system programs such as `last`, `ps`, `netstat` and `lsof` used by the administrator with different programs with the same name that appear to behave correctly, but prevent detection of the attacker. To do so, the attacker first uploads the source code and compiles and creates the Trojan horse. Or, he just uploads the precompiled binary, then moves the original and renames the Trojan horse. In either case, the file system of the host to be subverted allocates new disk space for the Trojan horse. Robert Lee, a speaker at SANS 2001, Baltimore and formerly of the Air Force Information Warfare Center, noticed the file system in many operating systems allocates disk space and the associated inodes in sequential order. The side effect is that files created at the same time, such as initial installation, have inode numbers in a small range. When an attacker replaces a system binary, well after system installation, the inode number allocated to the Trojan horse is much greater than the system binaries in that directory. Thus by examining the inodes allocated to system binaries, it is possible to detect unauthorized substitutions. Similar conclusions might be drawn from the allocation of process ID numbers or disk blocks. To generalize this result, by examining the mechanisms at a lower layer of abstraction in a system, patterns that occur normally and have been violated may lead to useful deductions at a higher level of abstraction.

4.3 Automated evidence analysis

The process to automate evidence analysis is to identify relevant evidence then reason with it for the purposes of the investigation. While users, administrators and experts will have relevant information; automatically identifying data that contradicts itself because of an attacker's activity will aid the investigator. Based on these contradictions and other readily available information, it may be possible to identify the possible explanations and hopefully the most reasonable one. The process is similar to software debugging:

> Fortunately, most bugs are simple and can be found with simple techniques. Examine the evidence in the erroneous output and try to infer how it could have been produced. Look at any debugging output before the crash; if possible get a stack trace from a debugger. Now you know something of what happened, and where. Pause to reflect. How could that happen? Reason back from the state of the crashed program to determine what could have caused this. Brian Kernigan [25]

4.3.1 Expert systems

An expert system is a program designed to simulate a human expert in a given subject domain. It has an *inference engine* that works with a *knowledge base* that contains *facts, rules* and *relationships* about the subject matter. Two algorithms an inference engine may use to calculate conclusions are *forward chaining* and *backward chaining.*

> Definition 4.5: Forward-Chaining
>
> Inference strategy that begins with a set of known facts, derives new facts using rules whose premises match the known facts, and continues this process until a goal state is reached or until no further rules have premises that match the known or derived facts. - Durkin [8]
>
> Definition 4.7: Backward-Chaining

Inference strategy that attempts to prove a [sic] hypothesis by gathering supporting information.

A backward-chaining system begins with a goal to prove. If first checks the working memory to see if the goal has been previously added. ... The system then checks to see if the goal rule's premises are listed in the working memory. Premises not listed then become new goals (also called *subgoals*) to prove, that may be supported by other rules. This process continues in this recursive manner, until the system finds a premise that is not supported by any rule - a primitive. - Durkin [9]

4.3.2 Contradiction detection

Once you have eliminated the impossible, whatever's left, however improbable is the truth. - Sherlock Holmes [7]

By making simple deductions and highlighting suspicious data, an expert system will allow investigators to focus their attention more efficiently. Data can be suspicious either with or without analysis. An example of suspicious evidence without analysis would be the traditional approach of identifying a signature of a common hacker technique such as naming a directory one or more space characters (e.g. "/dev/   "). The name appears blank and authorized users tend to overlook the directory's presence thus hiding the hacker's activity.

Evidence may also be suspicious through analysis, whereby all legitimate reasons for its presence are eliminated and there still not being an explanation. In other words, if all the evidence is of legitimate activity, semantic contradictions shouldn't exist. To identify these contradictions, I use a forward chaining, rule-based, expert system. Its working knowledge is the body of evidence and the invariant relationships between digital objects are encoded into the expert system's knowledge base. The ontology is based upon the objects experts use to understand the system in question: memory usage statistics at the kernel level, users, privileges and files for an operating system, network events for a network intrusion detection system, tables and transactions for a database, for example. At whatever level of abstraction, an object has a related context, or using Minksy's terminology [27], frame. The expert system searches through the data, eliminating those that conform to a known legitimate specification or invariant relationships, and highlights exceptions. These exceptions are semantic contradictions. The implication, once all legitimate reasons have

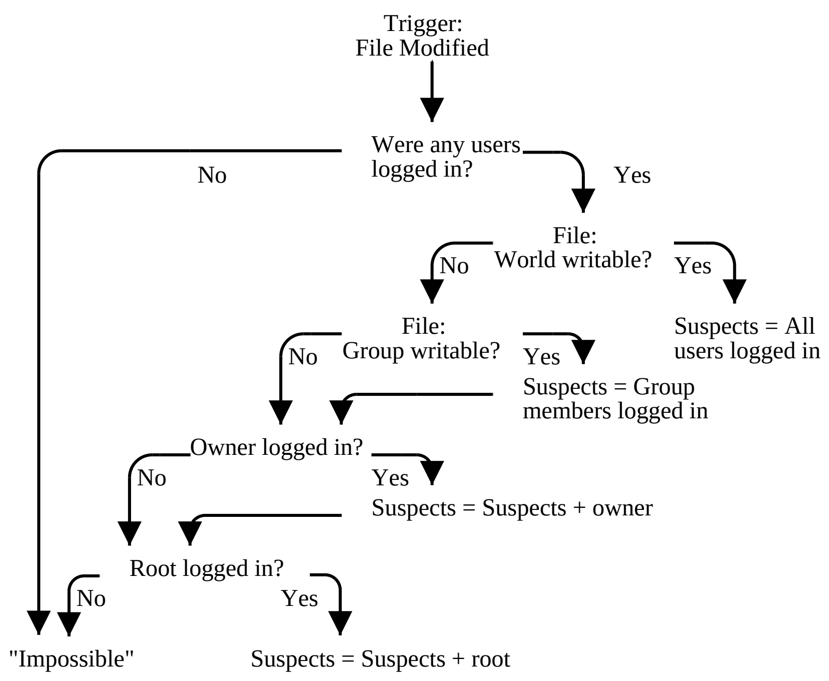
Trigger:
File Modified
Were any users logged in?
No
Yes
File: World writable?
No
Yes
Suspects = All users logged in
File: Group writable?
No
Yes
Suspects = Group members logged in
Owner logged in?
No
Yes
Suspects = Suspects + owner
Root logged in?
No
Yes
"Impossible"
Suspects = Suspects + root

Figure 4.1: Prototype decision tree

been eliminated, would be that an attacker has subverted the system's security by altering login logs, or had unauthorized privileges, for example. Figure 4.1 is an example of such a decision tree with the goal of determining the set of users who may have changed the contents of a file. In this case, the reason to initiate the analysis, or "trigger," was the fact that the file in question was modified, but this could be for any reason: manual or automated. For example, the administrator may have a hunch or anonymous tip there is a problem, or an intrusion detection system triggered the forensic expert system to search for suspicious activity. If no users are found (i.e., the lower left branch is reached), that file is considered suspicious. A specific suspicious datum will have significance to investigators based upon the *goal* of the investigation and the nature of the anomaly.

> A problem that requires an expert around fifteen minutes to solve is a reasonable problem for an expert system. If the problem is more complex, try to break it into sub-topics, each of which you could solve with a single expert system. - Durkin [10]

I believe this is a good general heuristic for designing a knowledge base because the problem to be solved is not so complicated the knowledge is huge, yet not so small, a small external program would be easier to implement. The expert system's purpose, at the contradiction detection stage, is to eliminate irrelevant data. Since a forward chaining inference engine will find all true deductions of all the facts is useful in only a limited context. Like any program, a forward chaining program's working knowledge consumes memory and its inference calculations consume computational resources. If forward chaining was the only approach used throughout an investigation, since most evidence is not directly relevant to the purposes of an investigation, the expert system would be swamped calculating the logical implications of irrelevant data (recursively). The investigator will want to focus the process to the relevant facts.

4.3.3 Hypothesis testing

> Debugging involves backwards reasoning, like solving murder mysteries. Something impossible occurred, and the only solid information is that it really did occur. So we must think backwards from the result to discover the reasons. Once we have a full explanation, we'll know what to fix and, along the way, likely discover a few other things we hadn't expected. Brian Kernigan [25]

To extend this approach, the iteration between forward and backward chaining inference engines may prove most useful. Based upon the contradictory evidence identified by one or more forward chaining inference engines, the use of a backward chaining expert system, such as automated diagnosis [13], to calculate the implications of those deductions (with relevance to the investigating agency) would be useful to direct the search for and limit the collection of evidence. For a given set of facts, the hypothesis with the most supporting evidence could be pursued until evidence is found that refutes its assertion, or the investigator determines the purposes have been met. The knowledge base would consist of hypotheses and the modes to collect evidence that support them.

For example, a forward chaining expert system may be able to deduce that a file was changed because an unauthorized user had `root` privileges. A backward chaining expert system with the goal of identifying the damage consequences of a policy violation could use this deduction to calculate that `root` privileges on that host would imply the user had obtained root access to certain other hosts in the network such as in NetKuang [47]. This deduction by the goal based, backward chaining system would accelerate the search for related evidence. A forward chaining approach would need to examine all the files on all the hosts (including obviously unrelated hosts) to come to a similar conclusion.

One piece of data, or working knowledge, can be associated with multiple goals. With multiple goals, users or investigating agencies may prioritize them differently. As the knowledge base becomes more elaborate, and the expert system has multiple courses of action to choose from, the investigator can assert a preliminary hypothesis and then verify it later. A scenario such as "Let's assume now that this email is authentic and corroborate it later with evidence from a different jurisdiction," is commonplace. At any point during the investigation as in any explanation system, the expert system can present the evidence and the reasoning process upon which a conclusion has been made.

Instead of an ad hoc process of collecting data, an expert system can help prioritize tasks for investigators. It can be goal-oriented, based on heuristics. For example, the most easily available data, administratively speaking (i.e. Section 3.1.2), is collected first, or the most volatile data that will disappear (i.e. Section 3.1.2) is collected, or that data which will make deductions with strong assurance is collected (i.e. Section 3.1.2).

Corroboration

For another example, if a deduction is reached and the investigator identifies that its assurance is a high priority, to corroborate the facts, the expert system could increase the priority of the search for supporting evidence.

Data sources from more than one administrative domain will provide more reliable and less refutable deductions. For example, one may correlate audit trail entries to an application level log, or network connection log entries to the output of a network intrusion detection system. Technically, this is not complicated: send a duplicate of a logging entry to a designated logging host. An attacker would require access at the user and `root` level, or access on two separately secured machines for example, to alter the data to make the evidence appear legitimate. Said differently, it would be more difficult for the attacker to make a realistic illusion of a normal system. For administrative or operational reasons though, separate protection domains are not always implemented. Hence the analysis of data in one administrative domain, and verifying that it is "self-consistent," is of practical utility.

If data sources from more than one administrative domain cannot be aggregated, then the potential exists for the attacker to have created a complete body of evidence that is self-consistent. Either evidence leading to the cause of the policy violation could be eliminated or it may have been doctored to lead to a false conclusion. This is why it is important to document the evidence that leads to a conclusion. If that evidence is later found to be untrustworthy, so must the conclusion. Again, at any point, the explanation system can be queried to present the evidence and rules upon which a conclusion has been made.

4.4 Response

When an implmentation of this approach is used on a body of evidence, how to interpret the results and how to act on the output are important issues. There are number of reasons a piece of evidence may be identified as suspicious. Some reasons are errors in the knowledge base. Others are related to the data with which the expert system reasons or the mechanisms that collect it.

As the knowledge base is being developed, the primary reason for apparently suspicious evidence will be an inadequate model of the system. Since a model is a simplification of the relationships of components in a system, an oversimplified model will not take into account common legitimate scenarios of evidence. The best invariant relationships specified in the knowledge base will be general and simple in nature and are only violated by users with ill intent. Unfortunately, specifying the behavior of an operational system, be it a small, familiar, system binary or a confederation of networks of interconnected hosts, requires time and expertise. Specification of all possible relationships between data in the system may not be reasonable (or cost effective) for anything more complicated than an academic exercise. Checking all possible relationships of data in a body of evidence would also be computationally expensive. Thus it is important for the expert system to be an aid to an expert rather than a substitute for one. So, in the case of an inadequate model, the expert would identify that the evidence anomaly was in fact a normally occurring event, and add new information to the expert system's decision tree to take account of this new situation.

Since experts are fallible, and do not always agree for that matter, the knowledge base may not be completely correct. Since an expert system can present each fact and each reason that led to a deduction, it will be possible for an expert with a better understanding to identify an error of commission or substitution by a faulty knowledge base. Checking that the invariants are correct may be done by profiling system behavior or by more formal methods. Errors of omission may be more difficult to recognize if not impossible. The absence of a relevant data relationship may lead to a false deduction as well. If a piece of evidence is found to be either normal or anomalous, and the expert system is found to have made an incorrect deduction, the knowledge base would need to be updated and augmented to address the mistake.

If the knowledge base is complete and data is identified as suspicious, it may be due to the fact that it is unauthentic, incomplete or has been tampered with. Or as in a structured database, there has been an unauthorized add, delete or update transaction respectively. Since the specific anomalous data has been identified, as has the reason why it is anomalous, a response can be designed for the investigative purposes that consider that set of circumstances. For example, if an important file has been modified, yet no users appear to have been active at the time, searching the system for other activity at the time or examining the activity of that file's owner more closely may give more clues. For an online scenario in which the investigative agency wants to learn more about the attacker's activities, an unobtrusive response would be to activate more detailed logging of the components of the system that have been determined to be anomalous. It may be reasonable to automate this response if the expert system is running without human participation, and particularly useful if the knowledge base can use this new information to make new deductions or corroborate existing ones. Gathering more facts in an ongoing attack can allow administrators to customize their response. For example, rather than remove the penetrated host from the network, disabling the penetrated user account or deactivating the faulty service may impose a less significant impact on the user community. Finally, by comparing the current set of facts to a database of similar facts, such as previous hacking attempts and penetrations, it may be possible to characterize the attacker's expertise, intentions, or origin as well as to identify lessons learned from previous experience in dealing with this class of, or individual, attacker.

Yet another possibility exists to explain anomalous data despite the above. Perhaps the expert system has been operating correctly and the knowledge base is complete. In addition, by verifying the identity property of the data with a designated secure logging host for instance, the data has been determined to be complete and authentic. The possibility still exists that the mechanism that generates the data or that which the mechanism monitors has been tampered with. This is evidence of tampering and implies a specific level of access [5].

So by using invariant data relationships, aggregating data into objects and concepts that match those in an expert system's knowledge base, automated analysis can occur. This augments an expert's capabilities allowing a more targeted, assured, and (or) timely response.

Chapter 5

Prototype

Since a file on a host can typically be modified only when its owner is logged in, one can check that its modification time (from its inode) is during a login session of its owner (from `lastlog`). If not, the file, its owner and the modification time may be considered suspicious and worthy of attention. On a system with many users, gigabytes of files and years of operation, identifying relevant evidence quickly is crucial. As data becomes more volatile, time becomes more important. This assertion of course is a simplification. When the owner is not logged in, if `world` or `group` writable, others are able modify it. In addition, a `cron` job, the `root` user or its daemon processes may modify it, and `setuid` programs may further complicate matters. Common valid exceptions may be added to the knowledge base. As more expertise is encoded and automated, an expert's time can be spent on more novel situations. Although useful evidence may have been deleted legitimately (e.g., log rotation implemented by a circular buffer) or the real user may have subsequently overwritten the modification time, only a perfect attacker will leave no evidence whatsoever. The key is to automate the search for impossibilities based upon the semantics of normally recorded data.

5.1 Architecture

I have implemented a prototype of this approach using a collection of `C` programs and `Perl` scripts called "The Coroner's Toolkit" [14] (TCT) to automate data collection

struct stat {		
dev_t	st_dev;	/* device */
ino_t	st_ino;	/* inode */
mode_t	st_mode;	/* protection */
nlink_t	st_nlink;	/* number of hard links */
uid_t	st_uid;	/* user ID of owner */
gid_t	st_gid;	/* group ID of owner */
dev_t	st_rdev;	/* device type (if inodedevice) */
off_t	st_size;	/* total size, in bytes */
unsigned long	st_blksize;	/* blocksize for filesystem I/O */
unsigned long	st_blocks;	/* number of blocks allocated */
time_t	st_atime;	/* time of last access */
time_t	st_mtime;	/* time of last modification */
time_t	st_ctime;	/* time of last change */
};		

Table 5.1: `stat` data structure for EXT2 file system

$md5,	MD5 hash	$file,	File name
$st_dev,	Device	$st_ino,	Inode number
$st_mode,	Protection	$st_ls,	Permissions as viewed by `ls`
$st_nlink,	Number of hard links	$st_uid,	User ID of owner
$st_gid,	Group ID of owner	$st_rdev,	Device type
$st_size,	Total size in bytes	$st_atime,	Time of last access
$st_mtime,	Time of last modification	$st_ctime,	Time of last change
$st_blksize,	blocksize for fileysstem	$st_blocks	Number of blocks allocated

Table 5.2: Fields for each file in TCT's "body"

and an expert system called The Rule System for the Java Platform (a.k.a. JESS) [16] to automate its analysis. TCT can gather evidence including the record of user login sessions in the file `lastlog` and thirteen fields of metadata (Table 5.1) from every file on the host in a file called `body` (Table 5.2). My Perl script parses the `lastlog` and `body` files, then produces an XML structured document. Figure 5.1 is a diagram of the flow of forensic evidence to a standard format.

My prototype then parses this XML document and asserts two kinds of facts in JESS's fact base (See Table 5.3). After input validation checks (e.g., Verify $A < B$), the knowledge base directs the JESS engine to check that the last modification time for every file was during a login session of its owner (e.g. Verify $A < C$ and $B > C$). It ignores files last modified before the first entry in `lastlog` by adding a login session from time

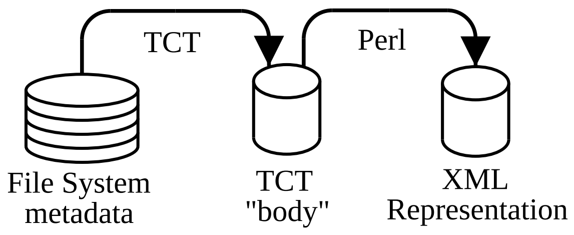

Figure 5.1: Data flow through the prototype implementation

"User X was logged in from time A to time B"
"File Y, owned by X, was modified at time C"

Table 5.3: Types of JESS facts in the prototype

"0" to the first session's login time to JESS's working knowledge (asserted facts). The same is done for file access times and the time of last change to the inode. The prototype uses a reasoning process similar to that illustrated in Figure 4.1. Differences include the "trigger" and file permissions; the prototype analyzes *all* files (not just those that were modified) and does not consider a file's permission mode settings in it's checks. When any of these invariant relationships (Table 5.4) are violated, a message is printed highlighting the exception. Figure 5.2 is a graphical representation of this process.

$user$	$owns(file)$	
$loginTime(user)$	$\leq accessTime(file)$	$\leq logoutTime(user)$
$loginTime(user)$	$\leq modifyTime(file)$	$\leq logoutTime(user)$
$loginTime(user)$	$\leq changeTime(file)$	$\leq logoutTime(user)$

Table 5.4: Relationships between data from evidence

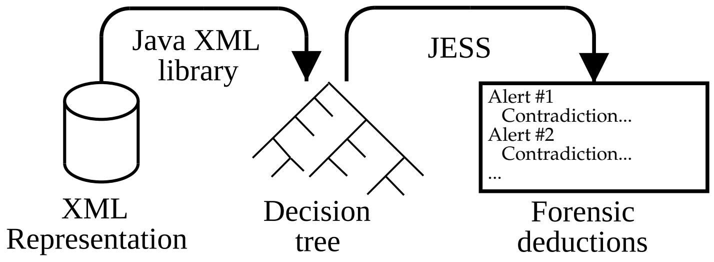

Figure 5.2: Process to produce deductions

5.2 Experiment

5.2.1 Research goal

An idea that is not usable has no practical impact. As a proof-of-concept, if this prototype can determine which files an unauthorized user modified, accessed or changed, while mostly ignoring those the user did not, the feasibility of this idea will be demonstrated.

5.2.2 Scenario

The scenario that guides the test of the prototype is that of a UNIX host, in a default configuration with network access, that does not send its logging data to a remote host. An attacker has successfully penetrated its security by simply sniffing the `root` user's password from the network and logged in. A common technique to avoid detection is to use `root` privileges to delete `syslog` entries recorded by the vulnerable root application during the buffer overflow and edit `lastlog` to remove the record of the victim's session. In this case the logs do not need to be edited since the misuse is difficult to detect. Although a clear text, remote, `root` login session itself may be considered suspicious today, on the network and at the host, it appears a normal, legitimate login has occurred. The attacker downloads, compiles and installs a malicious loadable kernel module (`adore` [38]) to hide his files on the hard drive and his processes from the administrator and is careful not to modify any files that a file integrity checker may check. Further suppose the network administrator later detects a violation in policy such as port scanning activity originating from the compromised host. Even though the obvious logs show no suspicious activity (and may have been doctored anyway), other evidence the attacker could not hide remains, from which useful questions may be answered.

5.2.3 Apparatus

The "victim" host has an Intel-based CPU with RedHat$^{\text{TM}}$ Linux version 7.3 installed and fully patched as of 12 December 2002. The Coronor's Toolkit [14] version 1.09 is installed and is configured with an unroutable network address (`192.168.50.51`). The attacker's tools include `netcat` [19] version 1.10 for network access and the `adore` [38] loadable kernel module to subvert system calls by utilities designed to detect the attacker's presence.

The digital forensic analysis host is also an Intel-based CPU with RedHat$^{\text{TM}}$ Linux version 7.1 installed (IP address `192.168.50.50`). The Coronor's Toolkit [14] version 1.09, in conjunction with Java XML libraries and The Rule System for the Java Platform (JESS) [16] version 5.1, are installed which are used by my prototype implementation, `lead`.

5.2.4 Procedure

The following steps implemented the above scenario:

1. Install operating system and TCT on the "victim" host.
2. Simulate normal operation (update packages, login/logout multiple times).
3. Assume attacker gained access by using a network sniffer to steal the `root` password and logs in as `root`.
4. Attacker downloads `netcat`[19], sets up an unauthorized backdoor and logs out.
5. Attacker uses the `netcat` backdoor to gain unauthorized access and uploads `adore`.
6. Attacker compiles and runs `adore`, then hides suspicious directories and processes.
7. Simulate hacker activity while concealed.

8. Administrator notices suspicious activity and discovers subverted host.
9. Administrator logs in as the real `root` user and collects TCT body of evidence.
10. Administrator uses `netcat` to move TCT body of evidence to a remote host for analysis.
11. Forensic analyst runs my prototype (`lead`) on the body of evidence.

5.2.5 Results

Of the three inode times in each of 64 programs in TCT's `body.S` file, the subset of all `setuid` programs on the file system, only the access times on the programs used by the simulated attacker were highlighted. Yet there was one unexpected result: during the period only the attacker was logged in, yet hidden, a file was accessed that he didn't touch. With regards to performance of JESS (written in Java), on a 747 Mhz Intel Pentium III CPU, the Linux `time` utility, averaged over ten runs, averaged 6.165 seconds elapsed real time between invocation and termination. It averaged 4.632 seconds elapsed user CPU time and averaged 0.121 seconds system CPU time. The XML formatted input file containing data on 64 files and 25 login sessions was 31146 bytes in size.

Example output

The following is output from the program indicating user "0" (a.k.a `root`) was the only "suspect" logged in when both were *changed* and *modified*, but no users were logged in when they were *accessed* (a semantic contradiction):

```
File /usr/bin/rsh owner 0 changed 1023294327
  Suspects: 0
File /usr/bin/rsh owner 0 accessed 1040018508
  No users logged in at time 1040018508
File /usr/bin/rsh owner 0 modified 995975720
  Suspects: 0

File /usr/bin/rlogin owner 0 changed 1023294327
  Suspects: 0
File /usr/bin/rlogin owner 0 accessed 1040018508
  No users logged in at time 1040018508
```

```
File /usr/bin/rlogin owner 0 modified 995975720
  Suspects: 0
```

There was one message from the program that initially appeared to be a false positive, but was later found to be correct (true positive). Yet it demonstrates the complexity of the system and the difficulty for an intruder to leave no traces:

```
File /usr/lib/sendmail owner 0 accessed 1040020537
  No users logged in at time 1040020537
```

With the fact that the simulated attacker did not open that file or use a program related to the mail subsystem, this message was perplexing. Upon further investigation by a person, it was found that `/usr/lib/sendmail` is a *link* that was accessed while no users were logged in. Its target though, (through a chain of 3 links) `/usr/sbin/sendmail.sendmail`, was last accessed at `1040019174` when `root` *was* logged in. If the `sendmail` daemon, running between `root` login sessions, accessed the link, it would have accessed the target as well. If a `cron` job had run `updatedb` to update the database used by `locate`, all the files would have been accessed at the same time. An interesting possibility lies in the fact that when a user (or intruder) exercises the *file name completion* feature in a user shell such as the Bourne-Again Shell (`/bin/bash`) or Turbo C shell (`/bin/tcsh`), the shell program updates a file's inode access time if it is a *link*, but not otherwise. This is not a complete reason since `/usr/lib` is not a common directory in the `$PATH` environment variable. The explanation is that the attacker executed `ls -R`, a recursive listing of `/usr/lib` and updated the access time field of the link's inode. Given this evidence and reasoning process, it would be another reason for the investigator to be suspicious about activity on that host at that particular time. Similarly, directory access times are updated via the use of `cd`. With this experience, new objects could be instantiated in the knowledge base to consider this new situation, thereby automating it in the future.

Example working knowledge

The following are two exemplars of JESS facts about file access times (seconds since the epoch, 1 Janurary 1970):

```
(mactime (file /usr/bin/rsh)
(uid 0) (modify 995975720) (access 1040018508) (change 1023294327) )

(mactime (file /usr/bin/rlogin)
(uid 0) (modify 995975720) (access 1040018508) (change 1023294327) )
```

The following are four exemplars of login session JESS facts including the session fabricated to ignore files before the first login session entry:

```
(session (login 0) (logout 1039567500) (uid 0))
(session (login 1040020260) (logout 1040020402) (uid 0))
(session (login 1040018760) (logout 1040020260) (uid 0))
(session (login 1040018280) (logout 1040018280) (uid 0))
```

Summary

So although the knowledge base is an initial, yet useful attempt, an experienced hacker can still be detected. If the attacker installs a malicious kernel module that hides his presence from all system utilities (`last`, `ps`, `netstat`, `lsof`, etc.), opens and modifies no files, the access time to a directory's inode is still updated when browsing its contents. Selecting just these anomalous directories and sorting them by time can provide a time line of events relevant to an investigation. Although avoiding detection is still possible, attackers will need to write, upload and *always* use their own tools they know do not change file system inodes. Journaling file systems will further complicate their attempt to hide.

Chapter 6

Implications

Technology-savvy criminals exploit mistakes in the security of systems to their own advantage. Their activity is camouflaged in the "glitches" that people expect from unreliable technology and is hidden from experts by the closed nature of many software systems (often due to proprietary intellectual property) that prevent understanding of their internal operations and correct behavior. Experts can only attempt to recreate the error or fault in order to identify its preconditions, and in doing so, obfuscate or eliminate evidence relating to malicious activity. To begin the process of learning from experience, and for users, operators, owners, and law enforcement to recover from a violation of law or policy, experts must demystify these faults, errors and failures [12]. Formalizing this process for consistent repeatability purposes and automating it for practical ones is critical. While work has been done to frame the problem of digital forensics, most of the advances have been at the evidence collection and preservation stages of an investigation. Yet it makes sense that when technology is used to manage data and its security is subverted, that technology be used to manage the data relating to the crime itself. Concepts and approaches have been invented to manage complexity and arbitrary levels of abstraction in software development. An expert system can do this for cyber crime investigations.

6.1 Approach

A major contribution of this thesis is that even without proper preparation for a computer attack, evidence of a security violation can be automatically identified. If attackers make their presence clear, detection is not difficult. If they make the effort to conceal their activity, they still have changed the state of the system, leaving footprints through the unknown side effects of their activity.

By specifying relationships (before or after an attack) specific to their system (i.e. Section 4.2), defenders can automate their “home-field advantage.” These site-specific characteristics, not easily learned by attackers, can help incident handlers find an attacker’s unintended side effects. Incident handling personel will be able to automate decision making for commonly encountered scenarios thereby accelerating the search for evidence and solving more incidents or understanding them better.

With this automated technique, system administrators who identify an anomaly may be able to make a preliminary diagnosis of their system. If they were able to quickly search for and analyze suspicious data, even data unrelated to the anomaly involved, and see an anomaly with security significance, instead of debugging a potential configuration error for example, they could recognize the anomaly as evidence relating to a computer intrusion.

Beyond the savings of a forensic expert’s time, for law enforcement, the repeatability of the investigative process (i.e. Section 4.3.2) at the technical level is important. Instead of an opinion based upon a person’s best effort and limited resources, the reasoning process and the evidence upon which the deductions were made are documented, transparent and deterministic. Even if well known invariant relationships between digital objects are common knowledge, they can still prove effective. As opposed to static signatures such as which ports are open and closed in a firewall policy, where attackers can learn and trivially change their activity to avoid detection, an attacker must change the system so that a set of digital objects fails to behave in an expected manner, which this approach may detect.

Given data on enough incidents over time, throughout the company, system owners may be able to gather meaningful statistics on the assets, threats and vulnerabilities with the most risk for the organization. With this data, they may allocate resources more efficiently and learn from mistakes. Surveys of people in similar industries reporting hacker activity and impact is no substitute for data from an operational environment. Similarly, software developers can learn from mistakes. Data from an operational environment on how their software is used and abused would provide invaluable feedback to the development process to make software better.

6.2 Future work

6.2.1 Invariant discovery

Identifying new semantic redundancies, such as between logs of popular applications and operating system logs, would create a more realistic model of an operational system. The development of techniques to determine the behavior of digital artifacts left by programs is important to this process. As stated before, the best invariant relationships specified in the knowledge base will be general and simple in nature and are only violated by users with ill intent. Many invariant relationships, though, will be the obvious ones that a person would not bother to check throughout the system, but once automated, are effective. To find non-obvious relationships, the knowledge base designer would need to identify an object, monitor its behavior and the behavior of related objects. The best digital objects will be nonvolatile, verifiably authentic, complete and unaltered, but useful information may still be derived from less than ideal sources. An approach to finding these relationships would be to find a mechanism that affects two or more objects that would reasonably be expected to be found after a successful computer intrusion. Functional dependencies may lead to data redundancies [34]. A different way would be to find two or more objects related to the same condition and potentially an attacker would change one and not the other. This is similar to the detection of covert channels. The knowledge base engineer is attempting to detect information flow between objects in a system and use this knowledge of redundancy to detect an attacker's presence and activity.

"Wisdom and Sense" [44] is an anomaly based intrusion detection approach that could have applications in this context. By training such a system to look for anomalies in a data set, such as digital objects in a system that may affect each other, it builds a list of rules and their preconditions for "normal data." Instead of using these rules looking for anomalies directly, these rules for "normal data" could be used to identify invariant relationships between data. This automated approach to generate relationships between objects will most likely produce many relationships that are not very useful, since the semantics of the data fields are not recognized.

6.2.2 Hypothesis Testing

While not an easy task (as shown in [13]), formalizing the reasoning process for the investigation is important as well (See section 4.3.3). For law enforcement purposes, meticulously eliminating incorrect possible explanations for an event, leaving only the most reasonable explanation is a critical process. To be able to answer these questions, data must exist and identification of that data for a particular question should be formalized. Yet data is collected and preserved with different purposes in mind [1].

Companies that use technology do not focus on how that technology might be misused. They focus on optimally using their technology investment to make a profit by monitoring its operations and maintenance. Companies are focused on their technology assets: *what* may have been misused, *what* needs to be fixed and likely *when* it was misused to determine the extent of the damage. When something malicious occurs, their highest priority is to become operational again. Yet in the same situation, law enforcement typically prefers not to be involved unless there is a reasonable assurance the case will be solved successfully, including asset recovery. From their perspective, whether the company makes a profit or not is less important than solving the crime. Law enforcement is focused on the threats to the technology: *who* attacked or misused the system and *why*, arguably a more difficult problem. A third perspective exists as well. The software developer or system designer is interested in making the next version better or improving the existing system. They are interested in *how* the security was exploited. Once they understand how, a patch can be implemented and installed. Developers are focused on the vulnerabilities in the system. To answer these questions, the debugging process requires very different types of data.

So while each of these classes of investigators will ask similar questions in an investigation, each has different goals. Because of these different goals, the priority of each question, which is the process of confirming or refuting an hypothesis, will differ. If each of these classes were enumerated and formalized, they could be encoded into an expert system's knowledge base to be automated:

- Evidence that answers questions
- Questions to confirm or refute an hypothesis
- Hypotheses that explain a crime for the investigation's purpose
- Purposes and goals of investigations

Some questions will be easier to answer than others due to the data available. The data collection policy, be it regular auditing and backups or freezing the state of a computer after an intrusion, can change according to the circumstances. The investigation's purpose may affect the process of deduction as well. If the quality of the evidence is in question and strength of deductions is not irrefutable, corroboration of two or more sources may be required. Again, the amount and type of audit and log collection in operation will affect what questions can be answered. Examples of data collection profiles:

- Operating and maintaining the state of system
- Debugging a problem
- Documenting attack in progress
- Documenting effects of an attack

Investigators should be able to produce evidence supporting the working hypothesis. For example, "Evidence found on the suspect's computer contained a program, known to be used by hackers, that leaves a specific file on the victim's computer, such as *this* file found in the evidence on the penetrated host." To show that other explanations are less convincing or impossible, alternative hypotheses, and the evidence that refutes them, would be produced as well. "The network intrusion detection system and firewall logs recorded a network connection from the suspect's computer's address during the time in question and because of the network configuration, no one else could have done that."

If the purpose of the investigation is to identify the perpetrator for example, an hypothesis, explaining the perpetrator's method and activities, would be identified, and alternative explanations ruled out. The attacker's method and activities may be modeled on experience with other attackers. Such a model may be based upon attack modeling languages such as JIGSAW [40] and techniques of automated diagnosis [13].

6.2.3 Model of attacker

Various attackers will leave different traces of their activity, but classes of attacks will have common characteristics. By having a model of the attacker, the evidence may lead to the attacker's intentions and to where other evidence may be found. A denial of service "zombie," waiting for instructions for example, would have a much different set of evidence than an intermediate (pass through) hacking host or host that is the origin of the attack with all the attacker's tools, output from those tools and conversations with friends. An unplanned ("hit and run") attack is very different from a meticulous and tenacious attacker that targets his victim, or a legitimate user with malicious intent. These models can help characterize the threat.

6.2.4 Fault tolerance

Instead of a law enforcement perspective, suppose the availability of data and functionality is of primary importance. The owner of a given system wants to maintain operations. Corroboration of deductions is less important than minimizing the time of diagnosis and implementation of an effective reaction. This approach of detecting semantic incongruities based on invariant relationships between digital objects in a system can be applied to automate failure diagnosis. Based upon the known symptoms of the fault, automated testing of hypotheses of the cause may also augment the skills of an analyst.

In fault tolerant systems, redundancy of components is a common approach. When a fault occurs, identification and replacement of the faulty component is automated. This redundancy can lead to specifications of invariant relationships of redundant data in the system that can be checked for validity, and in turn identifying the component in an invalid state.

> ...a *system* is: an entity having interacted or interfered, interacting or interfering, or likely to interact or interfere with other entities. ...a *component* is another system, etc. The recursion stops when a system is considered being *atomic*: any further internal structure cannot be discerned, or is not of interest and can be ignored. Laprie [12]

Independent components that have redundant data could be considered to be in separate administrative domains at that level of abstraction in the system. In the HACQIT [32] architecture for example, redundant hosts are independent. This independence aids the detection of semantic incongruities (and specification violations) since an attacker would need access to both hosts in order to eliminate evidence from both administrative domains. Yet identifying a faulty component is different than identifying the cause of the component's failure.

> It could be argued that introducing the phenomenological causes in the classification criteria of faults may lead recursively "a long way back", e.g., why do programmers make mistakes? why do integrated circuits fail? The very notion of fault is *arbitrary*, and is in fact a facility provided for stopping the recursion....In our view, recursion stops at *the cause which is intended to be prevented or tolerated.* Laprie [12]

Identifying the cause of the component's failure would require data from a lower level of abstraction; data that documented the states of the component and its sub-components leading up to the failure. Since a component is a system itself, the process would lead recursively until *"the cause which is intended to be prevented or tolerated"* is identified, which is based upon the goals of the investigating agency. Example goals of an investigation:

1. Detect a system intrusion

2. Identify mechanism associated with the violation

3. Identify last known good state (time)

4. Identify data sets used during that time

For any system an attacker is able to penetrate, that system lies in a larger context ("super-system") and is subject to verification of invariant relationships between digital objects in its environment. Said differently, seen from the perspective of a digital forensic investigator, an attacker that compromises the security of a component (e.g., a host) in a system (e.g., a network), may be detected by analyzing the component's behavior in the context of the rest of the system. So while a system can mask the faults of its components, it cannot mask its own faults. This is good news for defenders because attackers would need control (and total knowledge) of the whole "system" in order to create a convincing evidence trail that would mislead investigators.

By understanding a fault, one is more likely to be able to prevent it in the future. By reducing system functionality, reliability, security or efficiency in an automatic fashion, the impact of a system violation may be reduced if not eliminated. Calculating a timely, accurate diagnosis is the best chance for implementing an optimal response and system reconstitution.

Chapter 7

Conclusion

Interestingly, however, destroying or modifying data to hide evidence can leave significant marks as well – sometimes more telling than if they had left the system alone. Consider the physical world – anyone walking on a snowy walkway will obviously leave footprints. If you see the walkway clear of any tracks, it might make you suspicious: "Did someone brush away all traces of activity?" As we all know from programming experience, it is significantly easier to find a problem or bug if we know something is wrong than if you're simply presented with a program. Dan Farmer and Wietse Venema [15]

7.1 Results

The main result of this thesis is that despite the challenges faced by an individual attempting to determine what sequence of events led to an security violation, it is possible to automate the analysis of digital evidence to identify relevant data left by an attacker who attempts to leave no trace. Rarely are approaches presented for incident recovery in which no preparation is required *before* an attack.

Although it is by far more effective to instrument preventative measures before a system is brought into an operational state and vulnerable to attack, for reasons of budget, schedule or ignorance, this is not often done. The minimum amount of information may not even exist to completely solve the crime. So the computer crime investigator's job will always be difficult and crime is unsolvable in the general case.

By using data from common mechanisms that may not have been intended for security, I have shown it is possible to deduce security-related conclusions and answer useful questions related to a computer intrusion, particularly when intruders attempt to erase their activity. Automating the analysis of such data is crucial. This *post facto* analysis, I believe, is an extendible approach that can be evolved to include more data relationships at the host, network and application specific levels. In addition, these relationships may be able to be detected and verified in an automated fashion in systems that have functional redundancy.

7.2 Extensions

> There is not only the effect of the criminal on the scene to be considered, but also the manner in which the scene may have imparted traces to the criminal.
> Nickell & Fischer, Crime Scene Investigations [31]

It may be possible to apply lessons learned from the collection of physical evidence in the field of criminal forensics to the digital realm. With the assumption that there is a suspect identified, techniques may be developed to confirm or refute his involvement in the crime. Yet although users may not leave physical evidence in cyberspace, they may leave the digital equivalent in the side effects of their actions to the hosts and networks they use. The key is to find those side effects left by malicious users that they do not or cannot remove or obfuscate beyond recognition.

> Forensic scientists have almost universally accepted the Locard Exchange Principle. This doctrine was enunciated early in the 20th Century by Edmund Locard, the director of the first crime laboratory, in Lyon, France. Locard's Exchange Principle states that with contact between two items, there will be an exchange.
> John Thornton [41]

Questions relating to the limits of solving computer crime have practical importance, but do not yet have answers. Identifying *useful* questions that provably can or cannot be answered would be an accomplishment. Two related questions would be, "Given certain facts, what questions can you answer and to what level of assurance?" and "Given a set of questions, what is the minimum set of facts required?" The general problem of determining what sequence of events occurred based on the artifacts left behind in the system is a theoretical area that may be worthy of further investigation.

Bibliography

[1] Matt Bishop, Christopher Wee, and Jeremy Frank. Goal oriented auditing and logging. Cited 17 November 2002 http://seclab.cs.ucdavis.edu/papers/tocs-96.pdf, Submitted to IEEE Transactions on Computing Systems, 1996.

[2] D. Brezinski and T. Killalea. RFC 3227 Guidelines for Evidence Collection and Archiving. Cited 25 July 2002 ftp://ftp.rfc-editor.org/in-notes/rfc3227.txt, 2002.

[3] Brian Carrier. Defining digital forensic examination and analysis tools. In *Digital Forensic Research Workshop*, 2002.

[4] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *IEEE Symposium on Security and Privacy*, pages 184–194, 1987.

[5] Frederick B. Cohen. A note on detecting tampering with audit trails. Technical report, Fred Cohen & Associates, 572 Leona Drive, Livermore, CA 94550, USA, 1995. Cited 31 October 2002 http://www.all.net/books/audit/audmod.html.

[6] Brian Deering. Hashkeeper database. Technical report, U.S. National Drug Intelligence Center, 2002. Cited 30 December 2002 http://www.hashkeeper.org.

[7] Sir Arthur Conan Doyle. *The Hound of the Baskervilles.* Berkley Pub Group, 1993.

[8] John Durkin. *Expert Systems: Design and Development*, page 100. Prentice Hall, 1994.

[9] John Durkin. *Expert Systems: Design and Development*, page 106. Prentice Hall, 1994.

[10] John Durkin. *Expert Systems: Design and Development*, page 37. Prentice Hall, 1994.

[11] Gary Palmer editor. DFRWS A roadmap for digital forensics research, 2001. Digital Forensic Research Workshop, Utica, New York.

[12] Jean-Claude Laprie editor. *Dependability: Basic Concepts and terminology.* IFIP WG 10.4, August 1994 draft.

[13] Christopher Elsaesser and Michael Tanner. Automated diagnosis for computer forensics. Technical report, The Mitre Corporation, 24 September 2001. Cited 25 July 2002 http://www.mitre.org/support/papers/tech_papers_01/elsaesser_forensics/index.shtml, Submitted to IEEE Transactions on Systems, Man, & Cybernetics (Part A).

[14] Dan Farmer and Wiese Venema. The Coroner's Toolkit. Online, 1999. Cited 25 July 2002 http://www.porcupine.org/forensics/tct.html.

[15] Dan Farmer and Wietse Venema. Forensic computer analysis: An introduction, September 2000.

[16] Ernest J. Friedman-Hill. Jess, The Rule System for the Java Platform. Technical report, Sandia National Laboratories, Livermore, CA, 2002. Cited 25 July 2002 http://herzberg.ca.sandia.gov/jess.

[17] Steve Gibson. Distributed reflection denial of service. Technical report, Gibson Research Corporation, 27071 Cabot Road Suite 105, Laguna Hills CA, 92653 USA, February 22nd, 2002. Cited 12 November 2002 http://grc.com/dos/drdos.pdf.

[18] Guidance Software & FBI. Private communication, 2002.

[19] Hobbit (hobbit@atstake.com). Netcat 1.10 for Unix, 20 March 1996. Cited 15 December 2002 http://www.atstake.com/research/tools/nc110.tgz.

[20] Bill Hutchison. Adding chkrootkit to your unix auditing arsenal. Online. Cited 31 October 2002 http://rr.sans.org/malicious/chkrootkit.php.

[21] Sun Microsystems Inc. SunSHIELD Basic Security Module Guide, Solars 7. Part No. 805-2635-10.

[22] Tripwire Incorporated. Tripwire. Online. http://www.tripwire.com or http://sourceforge.net/projects/tripwire.

[23] H. A. Kautz. A formal theory of plan recognition and its implementation. In J. F. Allen, H. A. Kautz, R. Pelavin, and J. Tenenberg, editors, *Reasoning About Plans*, pages 69–125. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.

[24] Richard Keightley and Kimberly Stone. Can computer investigations survive Windows XP? An examination of Windows XP and its effect on computer forensics. Technical report, Guidance Software, 572 East Green Street #300 Pasadena, CA 91101, USA, December 2001. Cited 31 October 2002 http://www.guidancesoftware.com/whitepapers/XPWhitepaper.shtm.

[25] Brian W. Kernighan and Rob Pike. *The Practice of Programming.* Addison-Wesley, 1999.

[26] Ulf Lindqvist and Phillip A. Porras. eXpert-BSM: A host-based intrusion detection solution for Sun Solaris. *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, pages 240–251, December 10-14 2001.

[27] Marvin Minsky. A framework for representing knowledge. In *The Psychology of Computer Vision*, pages 211–277. McGraw Hill, New York, 1975.

[28] Nelson Murilo and Klaus Steding-Jessen. chkrootkit v. 0.37. Technical report, Pangeia Informatica LTDA, SRTVS 701 Ed Palacio do Radio II s. 304, Brasilia, DF, 70340-000, BR, 2002. Cited 31 October 2002 http://www.chkrootkit.org.

[29] K. L. Myers. User guide for the procedural reasoning system. Technical report, Artificial Intelligence Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA, 94025, 1997.

[30] United States Department of Justice Computer Crime and Intellectual Property Section. Searching and seizing computers and obtaining electronic evidence in criminal investigations. Online. Cited 31 October 2002 http://www.usdoj.gov/criminal/cybercrime/searching.html.

[31] Charles O'Hara. *Fundamentals of Criminal Investigation*, page 23. Charles C Thomas Pub Ltd, 6th edition, October 1996.

[32] J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich, and K. Levitt. The design and implementation of an intrusion tolerant system. *International Conference on Dependable Systems and Networks*, 2002.

[33] Tsutomu Shimomura and John Markoff. *Takedown.* Warner Books, December 1996.

[34] Dan A. Simovici, Dana Cristofor, and Laurentiu Cristofor. Impurity measures in databases. *Acta Informatica*, 28(5):307–324, 2002.

[35] Guidance Software. EnCase. Online. Cited 25 July 2002 http://www.guidancesoftware.com.

[36] Guidance Software. EnCase FAQ. Technical report, Guidance Software, 572 East Green Street #300 Pasadena, CA 91101, USA, December 2001. Cited 31 October 2002 http://www.guidancesoftware.com/whitepapers/XPWhitepaper.shtm.

[37] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in your spare time, 2002. To Appear in the Proceedings of the 11th USENIX Security Symposium (Security '02).

[38] Stealth (stealth@team-teso.net). Adore Linux Kernel Module, 2001. Cited 15 December 2002 http://www.team-teso.net/releases/adore-0.42.tgz.

[39] Clifford Stoll. *Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage.* Pocket Books, 1989.

[40] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *Proceedings of the New Security Paradigms Workshop, Cork Ireland*, Sept. 19-21, 2000.

[41] John I. Thornton. The general assumptions and rationale of forensic identification. *Modern Scientific Evidence: The Law And Science Of Expert Testimony*, 2, 1997.

[42] Dr. W. Mabry Tyson. DERBI: Diagnosis, Explanation and Recovery from computer Break-Ins. Cited 31 October 2002 http://www.ai.sri.com/˜derbi/.

[43] Mike Uschold and Michael Grüninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[44] H. S. Vaccaro and G. E. Liepins. Detection of anomalous computer session activity. In *IEEE Symposium on Security and Privacy*, pages 280 –289, 1989.

[45] Michael Vatis. Law enforcement tools and technologies for investigating cyber attacks: A national needs assessment. Technical report, Institute for Security Technology Studies, Dartmouth, 45 Lyme Road, Hanover, NH 03755 USA, June 2002. Cited 31 October 2002 http://www.ists.dartmouth.edu/lep/lena.htm.

[46] Jake Kasdan (WGA). *Zero Effect.* Columbia Pictures Corporation, 1998. Filmstrip.

[47] Dan Zerkle and Karl Levitt. NetKuang - a multi-host configuration vulnerability checker. In *Proceedings of the Sixth USENIX UNIX Security Symposium*, July 1996.

Appendix A

An example rule base

```
(deftemplate mactime
   (slot file)
   (slot uid)
   (slot modify)
   (slot access)
   (slot change)
)

(deftemplate session
    (slot login)
    (slot logout)
    (slot uid)
)

(defquery users-logged-in
    "Finds all users logged in at the specified time"
    (declare (variables ?t))
    (session (uid ?u) (login ?i&:(>= ?t ?i)) (logout ?o&:(>= ?o ?t)))
)

(defrule m-chain-start
    (mactime (file ?f) (uid ?u) (modify ?m))
    =>
    (printout t "File " ?f " owner " ?u " modified " ?m crlf)
    (bind ?e (run-query users-logged-in ?m))
    (bind $?U (create$ ))
    (while (?e hasMoreElements)
        (bind ?uid (call (call (call ?e nextElement) fact 1) getSlotValue uid))
        (if (not (member$ ?uid ?U)) then
            (bind $?U (create$ $?U ?uid))
        )
    )
    (if (> (length$ $?U) 0) then
        (bind $?S (create$ ))
        (assert (test-ownership ?f ?u (implode$ $?U) $?S))
     else
        (printout t "  No users logged in at time " ?m crlf)
    )
)

(defrule a-chain-start
    (mactime (file ?f) (uid ?u) (access ?m))
    =>
    (printout t "File " ?f " owner " ?u " accessed " ?m crlf)
    (bind ?e (run-query users-logged-in ?m))
    (bind $?U (create$ ))
    (while (?e hasMoreElements)
        (bind ?uid (call (call (call ?e nextElement) fact 1) getSlotValue uid))
        (if (not (member$ ?uid ?U)) then
            (bind $?U (create$ $?U ?uid))
        )
    )
    (if (> (length$ $?U) 0) then
        (bind $?S (create$ ))
        (assert (test-ownership ?f ?u (implode$ $?U) $?S))
     else
        (printout t "  No users logged in at time " ?m crlf)
    )
)

(defrule c-chain-start
    (mactime (file ?f) (uid ?u) (change ?m))
    =>
    (printout t "File " ?f " owner " ?u " changed " ?m crlf)
    (bind ?e (run-query users-logged-in ?m))
    (bind $?U (create$ ))
    (while (?e hasMoreElements)
        (bind ?uid (call (call (call ?e nextElement) fact 1) getSlotValue uid))
        (if (not (member$ ?uid ?U)) then
            (bind $?U (create$ $?U ?uid))
        )
    )
    (if (> (length$ $?U) 0) then
        (bind $?S (create$ ))
        (assert (test-ownership ?f ?u (implode$ $?U) $?S))
     else
        (printout t "  No users logged in at time " ?m crlf)
    )
)
```

```
(defrule chain-test-owner
    ?ret <- (test-ownership ?f ?u ?U $?S)
    =>
    (retract ?ret)
    (bind $?Um (explode$ ?U))
    (if (member$ ?u $?Um) then
        (bind $?S (create$ $?S ?u))
    )
    (assert (test-root ?f ?U $?S))
)

(defrule chain-test-root
    ?ret <- (test-root ?f ?U $?S)
    =>
    (retract ?ret)
    (bind $?Um (explode$ ?U))
    (if (and (member$ 0 $?Um) (not (member$ 0 $?S))) then
        (bind $?S (create$ $?S 0))
    )
    (if (and (member$ root $?Um) (not (member$ root $?S))) then
        (bind $?S (create$ $?S root))
    )
    (if (and (member$ Administrator $?Um) (not (member$ Administrator $?S)))
then (bind $?S (create$ $?S Administrator))
    )
    (assert (suspects ?f $?S))
)

(defrule chain-print-suspects
    ?ret <- (suspects ?f $?S)
    =>
    (retract ?ret)
    (if (> (length$ $?S) 0) then
        (printout t "  Suspects: " (implode$ $?S) crlf)
     else
        (printout t "  No one was logged in!" crlf)
    )
)
```