Automated Analysis for Digital Forensic Science: Semantic Integrity Checking

Tye Stallard and Karl Levitt
Department of Computer Science
University of California, Davis
One Shields Avenue, Davis, CA, 95616 USA
{stallard,levitt}@cs.ucdavis.edu

Abstract

When computer security violations are detected, computer forensic analysts attempting to determine the relevant causes and effects are forced to perform the tedious tasks of finding and preserving useful clues in large networks of operational machines. To augment a computer crime investigator's efforts, the approach presented in this paper is an expert system with a decision tree that uses predetermined invariant relationships between redundant digital objects to detect semantic incongruities. By analyzing data from a host or network and searching for violations of known data relationships, particularly when an attacker is attempting to hide his presence, an attacker's unauthorized changes may be automatically identified. Examples of such invariant data relationships are provided, as are techniques to identify new, useful ones. By automatically identifying relevant evidence, experts can focus on the relevant files, users, times and other facts first.

1. Introduction

Even without proper preparation for a computer attack, relevant evidence of a security violation can be automatically identified. If attackers make their presence clear of course, detection is not difficult. Yet, if they attempt to conceal their activity, they still have changed the state of the system, leaving footprints through the unknown side effects of their activity.

If cursory investigations were useful and identification of novel cases was more efficient, administrators and developers may find it valuable to spend the time to learn from their security mistakes. Automated analysis of technical evidence is an obvious approach. With an automated technique, system administrators who identify an anomaly may quickly make a preliminary diagnosis of their system. Beyond the savings of a forensic expert's time, for example in law enforcement, the repeatability of the investigative process 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 invariant relationships between digital objects that detect attackers become 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 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.

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 prosecuted 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 paper will focus primarily on computers as the target of crime.

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` [12] 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 never occurred. 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 may meticulously delete log files and other evidence of his presence. Yet side effects of his presence may still exist. Consequences of his surreptitious activity, that he doesn't realize were left behind, doesn't have the access privileges to eliminate, or simply doesn't have the time to remove, will remain. Only the perfect attacker will be able to perfectly recreate the state of the system as it was before the security violation.

Formalizing the intrusion response 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 [3], most of the advances have been at the evidence collection and preservation stages of an investigation. Concepts and approaches have been invented to manage complexity and arbitrary levels of abstraction in software development. An expert system can do this for the analysis stage of cyber crime investigations.

Few projects have focused on recovery from a successful attack. The "Diagnosis, Explanation and Recovery from Break-Ins" (DERBI) [17] project at SRI used a procedural reasoning system 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 [5] which automatically generates hypotheses of computer attacks, simulates them on the target configuration and applies plan recognition techniques [10] to search for supporting data.

On the other hand, the approach presented in this paper, 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 [2]. 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).

Currently, the intrusion response process is either dependent upon highly trained investigators (who become overloaded) or a proprietary product such as EnCase [8]. Current products designed for forensic analysis gather data according to procedures intended for law enforcement. Yet the sophistication of computer crimes, has greatly outpaced advances in the analysis of evidence limited to such low levels of abstraction. In general, investigators are now given too much data to analyze and the increasing trend will continue. 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.

2. Approach

The process is to first find data objects with redundancies that *must* exist in a secure system. The second step is to search though evidence collected from the system in question for contradictions. To extend the approach, it is possible to hypothesize potential scenarios, then search for evidence that rules out the incorrect ones.

Once collected and preserved, raw data on digital media must be aggregated in a rational way into more abstract objects. Without automated interpretation, many layers of abstraction will cause an investigator's time to become the bottleneck in an investigation. 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. Operating systems and their applications normally build digital objects (in this paper, understood to be abstractions) using lower level abstractions, Since this is usually invisible to the user, this step may seem obvious. Yet since these are the very mechanisms attackers subvert to hide their presence, the raw data must be independently interpreted and organized, and is therefore part of the forensic analysis process. If data is encrypted or steganographicly hidden, iteration between data collection and analysis steps of the forensic process must occur.

2.1. 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 though, 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.

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 make the data appear to be legitimate and self-consistent. Said differently, it would be more difficult for the attacker to make a realistic illusion of a normal system. For operational reasons though, separate administrative domains are not always implemented.

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, an explanation system can be queried to present the evidence and rules upon which a conclusion has been made.

An invariant relationship is a specification (a form of policy) between digital objects that holds true in for system operating in an authorized state. Of course the simplest invariant relationship between two digital objects is the identity property: 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.

The DERBI [17] system 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 timestamps must always be monotonically increasing. Chronological gaps and log entries out of chronological order are highly suspicious.

Tsutomo Shimimora describes in <u>Takedown</u> [13] 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 was 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.

While focusing on invariant relationships in common software is useful, discovering subtle redundancies in custom software and configurations will allow defenders to automate their "home-field advantage." These site-specific "semantic tripwires" or characteristics, not easily learned by attackers, can help incident handlers find an attacker's unintended side effects. Incident handling personnel 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.

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.

2.2. Automating the expert

> Once you have eliminated the impossible, whatever's left, however improbable, is the truth. [4]

Evidence may be suspicious through analysis, whereby all legitimate reasons for its presence are eliminated and there still not being an explanation. To identify these contradictions, this approach uses a forward chaining, rule-based, expert system. Its working knowledge is the body of evidence. 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, and tables and transactions for a database, for example. At whatever level of abstraction, an object has a related context, or using Minksy's terminology [11], frame. The expert system searches through the data, eliminating those that conform to a known legitimate specification or invariant relationship, and highlights exceptions. These exceptions are semantic contradictions. Figure 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.

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, it is useful in only a limited context. Like any program, a forward chaining program's working knowledge consumes memory and its inference calculations con-

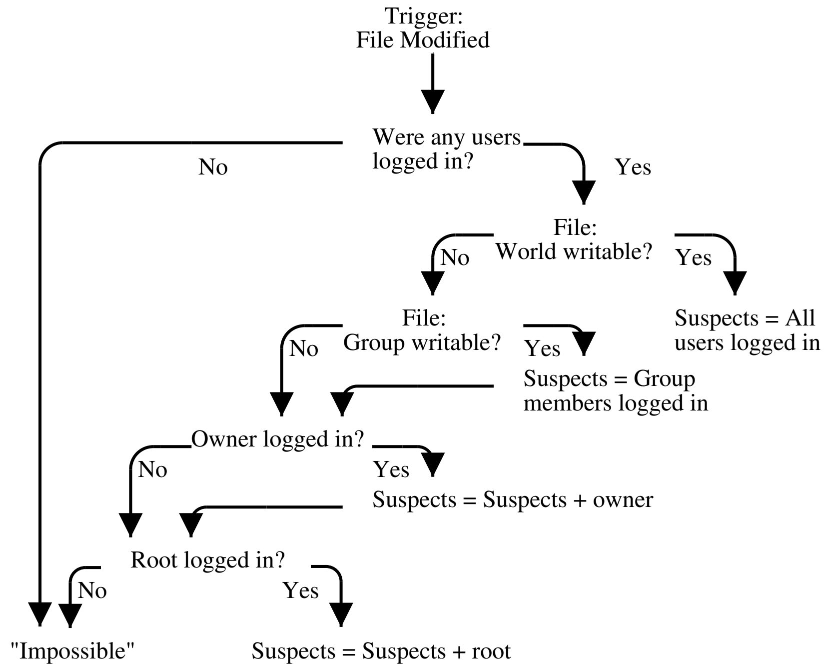

Figure 1. Prototype decision tree

sume computational resources. Since most evidence is not directly relevant to the purposes of an investigation, if forward chaining was the only approach used throughout 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.

2.3. Hypothesis testing

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 [5], to calculate the implications of those deductions 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 [18]. 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 a purely ad hoc process of collecting data, a goal-oriented expert system can help prioritize tasks for investigators based on heuristics. For example, the most easily available data, the most volatile data (that will soon disappear), or that data which will make deductions with strong assurance shall be collected first.

While not an easy task (as shown in [5]), formalizing the reasoning process for the investigation is important. Meticulously eliminating incorrect possible explanations for an event, leaving only the most reasonable explanation is a crit-

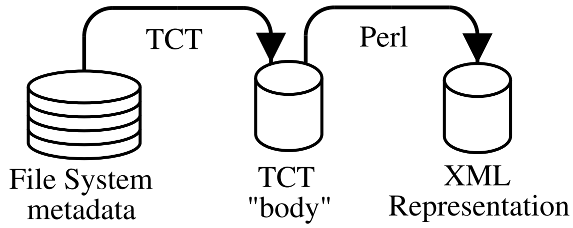

Figure 2. Data flow through the prototype implementation

ical process.

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." 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 [16].

3. Prototype

To confirm the utility of a forward chaining expert system in this context, a prototype was written. 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. 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.

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

Table 1. Types of JESS facts in the prototype

A prototype of this contradiction detection approach was implemented by using a collection of `C` programs and `Perl` scripts called "The Coroner's Toolkit" [6] (TCT) to automate data collection and an expert system called The Rule System for the Java Platform (a.k.a. JESS) [7] to automate its analysis. TCT can gather evidence including the record of user login sessions in the file `lastlog` and thirteen fields of metadata from every file on the host in a file called `body`). A Perl script parses the `lastlog` and `body` files, then produces an XML structured document. Figure 2 is a diagram of the flow of forensic evidence to a standard format.

The prototype then parses this XML document and asserts two kinds of facts in JESS's working knowledge (See Table 1). 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 "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 1. Differences include the "trigger" and file permissions; the prototype analyzes *all* `setuid` files (not just those that were modified) and does not consider a file's permission mode settings in its checks. When any of these invariant relationships (Table 2) are violated, a message is printed highlighting the exception. Figure 3 is a graphical representation of this process.

The hypothetical 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

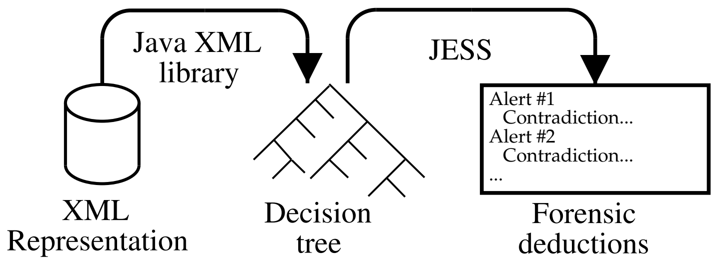

Figure 3. Process to produce deductions

$user$ owns($file$)
$login(user) \leq access(file) \leq logout(user)$
$login(user) \leq modify(file) \leq logout(user)$
$login(user) \leq change(file) \leq logout(user)$

Table 2. Relationships of times as JESS facts

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 sessions. 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 legitimate login has occurred. The attacker downloads, compiles and installs a malicious loadable kernel module (`adore` [15]) to hide files on the hard drive and processes from the administrator and is careful not to modify any files that a file integrity checker may monitor. Further suppose the network administrator later detects a violation in policy such as port scanning activity originating from the compromised host.

3.1. Apparatus

The "victim" host has an Intel-based CPU with RedHat™Linux version 7.3 installed and fully patched as of 12 December 2002. The Coronor's Toolkit [6] version 1.09 is installed and is configured with an unroutable network address. The attacker's tools include `netcat` [9] version 1.10 for network access and the `adore` [15] 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™Linux version 7.1 installed. The Coronor's Toolkit [6] version 1.09, in conjunction with Java XML libraries and The Rule System for the Java Platform (JESS) [7] version 5.1, are installed which are used by the prototype.

3.2. 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` [9], sets up an unauthorized back door and logs out.
5. Attacker uses the `netcat` back door 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 the prototype expert system on the body of evidence.

3.3. Prototype output

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 when only the attacker was logged in and hidden, there was a file that was accessed yet he didn't intentionally touch.

- 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, 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.

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 6.2 seconds elapsed real time between invocation and termination. It averaged 4.6 seconds elapsed user CPU time and averaged 0.1 seconds system CPU time. The XML formatted input file containing data on 64 files and 25 login sessions was 31146 bytes in size.

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

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))

Although the knowledge base is an initial attempt, an experienced hacker can still be detected. 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.

3.4. Future work

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, 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, yet 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. 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 [14]. A different way would be to find two or more objects related to the same condition; potentially an attacker would change one and not the other. This is similar to the confinement problem [1]. 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.

4. Conclusion

The approach presented in this paper shows it is possible to answer useful forensic questions by using data from common mechanisms that may not have been intended for security. It is possible to do so with *no* preparation *before* an attack. This *post facto* analysis is an extensible approach that can be evolved to include more semantic data relationships specific to host, network and application abstraction layers. In addition, these relationships may be able to be detected and verified in an automated fashion in systems that have functional redundancies. Although it is possible that the minimum amount of information may not exist to completely solve the crime, an imperfect attacker *will* leave clues.

References

[1] M. Bishop. *Computer Security: Art and Science*, pages 439–472. Pearson Education, Inc., 2003.

[2] 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.

[3] DFRWS. DFRWS A roadmap for digital forensics research, 2001. Digital Forensic Research Workshop, Utica, New York.

[4] S. A. C. Doyle. *The Hound of the Baskervilles*. The Strand Magazine, 1901.

[5] C. Elsaesser and M. Tanner. Automated diagnosis for computer forensics. Technical report, The Mitre Corporation, 24 September 2001. Cited 1 June 2003 http://www.mitre.org/ work/tech_papers/ tech_papers_01/ elsaesser_forensics/.

[6] D. Farmer and W. Venema. The Coroner's Toolkit. Online, 1999. Cited 1 June 2003 http://www.porcupine.org/forensics/tct.html.

[7] E. J. Friedman-Hill. Jess, The Rule System for the Java Platform. Technical report, Sandia National Laboratories, Livermore, CA, 2002. Cited 1 June 2003 http://herzberg.ca.sandia.gov/jess.

[8] Guidance Software. EnCase. Online, 2003. Cited 1 June 2003 http://www.guidancesoftware.com.

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

[10] 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.

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

[12] N. Murilo and K. 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 1 June 2003 http://www.chkrootkit.org.

[13] T. Shimomura and J. Markoff. *Takedown*. Warner Books, December 1996.

[14] D. A. Simovici, D. Cristofor, and L. Cristofor. Impurity measures in databases. *Acta Informatica*, 28(5):307–324, 2002.

[15] Stealth (stealth@teamteso.net). Adore Linux Kernel Module, 2001. Cited 1 June 2003 http://www.team-teso.net/releases/adore-0.42.tgz.

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

[17] D. W. M. Tyson. DERBI: Diagnosis, Explanation and Recovery from computer Break-Ins. Cited 1 June 2003 http://www.ai.sri.com/derbi/, 2000.

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