A Formalization of Digital Forensics[1]

Ryan Leigland
University of Idaho

Axel W. Krings[2]
ID-IMAG, France

Abstract

Forensic investigative procedures are used in the case of an intrusion into a networked computer system to detect the scope or nature of the attack. In many cases, the forensic procedures employed are constructed in an informal manner that can impede the effectiveness or integrity of the investigation. We propose a formal model for analyzing and constructing forensic procedures, showing the advantages of formalization. A mathematical description of the model will be presented demonstrating the construction of the elements and their relationships. The model highlights definitions and updating of forensic procedures, identification of attack coverage, and portability across different platforms. The forensic model is applied in a real-world scenario with focus on Linux and OS X.

Introduction

Incidents of computer related crime continue to rise each year. The CERT Coordination Center reported over 135,000 incidents in 2003, a 67% increase from 2002 [1]. Consequently, the need for forensics techniques and tools to discover attacks is also rising. Many forensic investigators have developed ad-hoc procedures for performing digital investigations. The informal nature of these procedures can prevent verification of the evidence collected, and may diminish the value of the evidence in legal proceedings [2].

The science of digital forensics has been defined as “the process of identifying, preserving, analyzing, and presenting digital evidence in a manner that is legally accepted” [3]. It is sometimes referred to as forensic computing, computer forensics, or network forensics. Differences in nomenclature notwithstanding, the goal is to preserve evidence in a way that satisfies legal requirements. Thus, it is important to ensure that procedures are strictly defined and correct.

This paper presents a framework to formalize certain aspects of the forensic discovery process to address those concerns. Abstraction layers are applied to attacks and operating systems to allow for the construction of models. These

[1] This research was funded by the Scholarship for Service program from the National Science Foundation, the CNRS and Region Rhone-Alpes (Ragtime project).
[2] The author is on sabbatical leave from the University of Idaho.

models can then verify that the forensic procedures used in an investigation were thorough and complete. The framework also makes the task of maintaining a forensic procedure easier and less error-prone. Thus, the application of the framework can result in more robust and less work-intensive forensic procedures.

This framework exists as part of a larger effort to apply formalization to the science of digital forensics, e.g. [2] and [4]. The 2001 Digital Forensics Research Workshop (DFRWS) developed a six-stage process to describe the entire lifecycle of a computer forensic investigation, from identification through presentation [5]. The framework discussed here falls into the third stage, collection. The collection stage is described as "extracting or harvesting individual items or groupings [of evidence]" [4]. In this case, the framework formalizes the techniques used to extract or harvest evidence.

The clearest beneficiaries of this work are practicing investigators in the field who are tasked with maintaining procedures for investigating a variety of computer systems for an increasingly wide array of attacks. That task becomes much more manageable and robust by applying the principles outlined here. Of course, neither this nor any other technique can perform the job of a forensic investigator. But the framework can significantly improve effectiveness and make the job of the investigator easier and more thorough.

Background

There are limitations to current digital forensic techniques. At the DFRWS, Spafford listed four major deficiencies [5].

Procedural**:** In order to comply with traditional forensic requirements, all data must be gathered and examined for evidence. However, a modern computer system may yield many gigabytes of data to be analyzed. This presents challenges at all stages, from gathering the data to storing and finally analyzing the data. To date, no standard solution has emerged for handling this problem. One common approach is to extract only relevant information while the system is still running, which limits the amount of data gathered. This is called *live forensics*, and while it can help with these procedural problems by limiting the amount of data collected, there is a certain risk of data corruption.

Technical: Computer technology is a rapidly changing field, which means that computer crimes are also rapidly changing. In addition, the computer systems under investigation evolve more rapidly than the tools to examine them. The ubiquity of computers in today's society means that computer crimes can occur in all jurisdictions, from large urban centers to small towns that lack the resources to train investigators. These factors combine so that inexperienced investigators are forced to examine computer crimes with inadequate and outdated tools.

Social: A lack of standardization of procedures has led to uncertainties about the effectiveness of current investigation techniques. This in turn has led to sub-optimal use of resources, as data is gathered that may not be useful and stored for longer periods than are necessary. Additionally, privacy concerns about investigation suspects can hinder the forensic process. In short, it is still unclear how to think about digital evidence and its role in prosecuting crimes.

Legal: The use and bounds of digital evidence in legal proceedings is still unclear. Current techniques may not be rigorous enough to use in the courtroom.

In general, the methods of gathering evidence have been developed ad-hoc by forensic investigators in various locations, based on personal experience and expertise. In some cases, specific guides have been crafted to aid investigators (e.g. [12]). These represent an attempt to standardize practices for investigators who lack the necessary expertise to develop their own procedures. However, these solutions are not extensible and can be viewed as "one-shot" answers that cannot solve the general case.

Forensic Model

The goal of the model is to allow for a formalization of the forensic process applied to a compromised computer system. In brief, it entails a logical breakdown of a computer system into smaller system components that can be manipulated to create or modify forensic procedures. An attack on the computer system is characterized by the various components that it affects. By examining the appropriate components on a computer system, it is possible to determine whether a given attack occurred or not. This examination, referred to as a *forensic procedure*, is linked to the system components and thus to an attack. From this basis, ways to manipulate the computer system components and their corresponding forensic counterparts can be described to draw certain conclusions about forensic procedures.

The Process

As indicated before, the aim is to transform the notion of forensic procedure from a mere collection of steps to a mathematical relationship between a computer system and the attacks that can affect it. This mathematical relationship will be most useful if it can give a specific correspondence between the forensic procedure and the attacks it can discover. Further, that correspondence can be broken down by considering the following questions:

Question 1: Given a known attack, what is the forensic procedure to discover it?

New attacks on computer systems are being created and unleashed constantly. It is not obvious whether existing forensic procedures have the ability to detect these new attacks. Strictly defining the relationship between attacks and discovery procedures makes this clear.

Question 2: For a given forensic procedure, what known attacks can be identified? This is useful information for a variety of reasons. If an attack is discovered on a target computer, knowing the abilities of the forensics procedure that discovered it can give confidence in the results. The converse also holds true; if an attack is not discovered, a formal analysis of the forensics procedure can show whether the attack was not present or if the procedure was merely incapable of discovering it. In general, the current state of forensics does not provide a means for listing the abilities of a forensic procedure. A mathematical model can provide this**.**

Question 3: Given new information about a known attack, how can the forensic procedure be updated to discover it? A strictly defined relationship between the attack and the forensic procedure makes clear how to keep procedures current at all times.

Question 4: Given a new computer system (or operating system), how can a forensic procedure be created or ported it? As before, knowing the relationship between attack and forensics procedure can clarify this process. By enumerating the characteristics of the new system, it becomes possible to adapt existing procedures or create new procedures that can discover attacks on the new system.

The forensic framework presented here will provide the ability to answer all of the preceding questions. Keeping those questions in mind as the impetus for the framework can make the ensuing discussion easier for the reader to follow.

Assumptions

To alleviate potential confusion in the discussion, it is worth noting the difference between a forensics procedure and an intrusion detection system (IDS). An IDS is designed to detect attacks at the time they occur, including known and unknown attacks. This requires not only an understanding of known attacks, but also a way to recognize new attacks. In contrast, the forensics model assumes that an attack has been committed and has left some trace to be discovered. Then the forensic procedure attempts to discover which known attack has taken place. Thus, we must assume that the characteristics of the attack are already well understood at the time of the forensic investigation. This corresponds to the reality of computer forensics, in which organizations like CERT create exhaustive reports that are later used by forensic investigators. Further, it is left to these experts to discover and profile new attacks as they arise.

This also gives rise to the notion of *expert* versus *forensic investigator*. An expert discovers and describes attacks and can be thought of as the expert in an organization like CERT/CC, investigating and describing the attack. This person will likely not be involved with the creation or use of forensic procedures. A forensic investigator, on the other hand, will be actively involved in utilizing the forensic model. The term *computer and network forensic technician* is also used to describe the role of the investigator [6]. Understanding these roles can aid in comprehending the use and scope of the framework.

The term *attack* is used in a relatively broad sense. Since forensic investigations rely on interpreting evidence to discover a crime, computer forensics requires evidence to detect an attack [3, 7]. In a somewhat circular fashion, an attack is loosely defined here as an unwanted intrusion on a computer system that leaves evidence. While intrusions that leave no trace are indeed a problem, they are outside the bounds of forensics and thus have no bearing on the current discussion. Examples of attacks are viruses, Trojans, or denial-of-service bombardments.

Finally, it should be noted that the framework is generally designed for working in the context of a "live" discovery process (in which the target computer is still running). A discussion of the relative merits of live forensics is beyond the scope of this document. However, it is clear that the increasing complexity and size of computer systems ensures that live forensics will be an important task for the foreseeable future [3].

General Description

Though a rigorous description of the model will be presented later, it will be useful to describe the general characteristics first.

To answer the previously stated questions, the model defines abstract representations of attacks, computer systems, and forensic procedures, and formalizes their relationship. The representations are composed of abstract primitive elements that will be grouped and manipulated to achieve specified objectives. Further, mappings are defined to translate abstract elements into concrete, real-world actions, which are the specific steps taken by the investigator.

The abstract representations can be organized into the hierarchy shown in Figure 1. The relationship among attacks, forensic procedures, and actions will later be described within the context of OS independence and different operating systems. Forensic procedures are derived from attacks, in order to ensure the ability of the procedure to detect a given attack on an operating system. The forensics procedures are defined to be independent of the operating system, to

allow for translation of procedures across different operating systems. Forensic actions are specific to an OS and are derived from the forensic procedures. Thus, given the information about a specific attack, the model allows for the specification of a set of forensic actions sufficient to identify the attack. Alternatively, given a set of actions, the model allows for the identification of the attacks this set can detect.

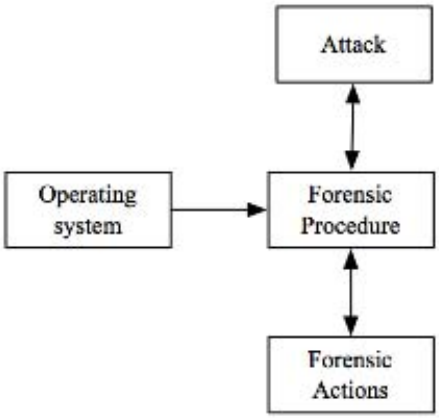

Figure 1: Model Hierarchy

Example: Trinoo

The model will be illustrated using the distributed denial of service (DDoS) client Trinoo [8]. Trinoo is an early DDoS tool that was active in the late 1990's, before being superseded by more sophisticated clients. Unable to spread by itself, Trinoo relied on scripts that exploited various known weaknesses. Trinoo has been chosen trinoo for a variety of reasons: it is very well understood, affects a small set of components, and works across various computer platforms.

Much of the following discussion will use the operating systems Linux and Mac OS X (BSD/Darwin) as examples. The particular variant of Linux is not important for our purposes. These systems were chosen because they are somewhat similar and thus are easy to contrast. Further, forensic processes on Linux are fairly well-defined, whereas on Mac OS X (referred to as OS X henceforth) they are less established. Finally, the attack Trinoo can affect both operating systems. Taken together, these factors provide an ideal setting for explaining and applying the framework.

It is not necessary for the reader to be familiar with the intricacies of either operating system. A basic understanding of UNIX-like operating systems will be helpful, but the only required knowledge is a general familiarity with the basic functions of operating systems. It is also not necessary for the reader to fully understand all the details of the Trinoo DDoS client. It suffices to know that Trinoo can be detected by looking for the “fingerprints” shown in Table 1.

Characteristic
The presence of a configuration file named "…"
An opening on port 31335
The presence of a process named "httpd"

Table 1: Trinoo Attack Characteristics

By themselves, none of the three indicators prove that Trinoo is installed on the system (especially since the process "httpd" is more likely to be a web server than a Trinoo daemon). When taken together, though, they can give evidence that Trinoo is in fact present. Since none of the indicators are definitive by themselves, all possible indicators must be investigated before drawing a conclusion.

Terms and Definitions

At the basis for the forensics model are two primitive types, i.e. *components* and *forensic primitives*. Informally, components are abstract elements that comprise a computer system, e.g. "password file", whereas forensic primitives are abstractions that examine the component for evidence, e.g. "examine password file." These basic types will be grouped into sets to represent attacks, operating systems, and forensics procedures, as will be described later.

In the notation used in this paper, singular primitives will be represented with lower case letters. Sets of primitives will be denoted with capital letters. Sets may be sub- or super-scripted to denote particular subsets. A set with no sub- or super-script represents the entire collection of primitives. For sets, a subscript always refers to the set associated with an attack. A superscript always represents a set associated with an operating system.

System Components

A system component is considered to be the basic building block of a computer system with respect to forensic discovery. Let c_i denote a system component. Components are abstract objects, e.g. "password file," that are not specific to a particular computer system or operating system. A particular computer or operating system will be composed of a particular set of components. For example, the operating system Linux has a password file along with many other components. Let C denote the set of all components on all operating systems, thus

$$C = \{c_1, ..., c_n\},$$

where n indicates the total number of components identified.

The choice of components is determined by the goals of system independence, as well as forensic relevance. For example, defining a system by its hardware is generally not useful for investigating network intrusions, but it is useful to

consider abstract software constructs as components. Recall that the attack Trinoo is partially characterized by which processes it runs. Thus another example component is "process table." This satisfies the goal of system independence since it is not necessary to know the implementation details of a process table on any particular operating system, such as Linux or OS X. It also satisfies the goal of forensic relevance, because it can contain evidence of an attack.

Given that component c_i is platform independent, it is obvious that not all operating systems contain all components. Let o^m denote an operating system, where m represents the specific operating system acronym. For example, o^{Linux} represents the Linux operating system. Henceforth, o^{Linux} will be shortened to o^L for reasons of brevity. Each o^m will correspond to a particular subset of C. Let C^m denote the set of components associated with o^m. Since C is the global set of all components, it must be the case that

$$C^m \subseteq C.$$

Set C can also be expressed as the union of C^m over all operating systems, i.e.

$$C = \bigcup_{OS} C^m.$$

Attacks

Let a_i denote an attack, where i indicates the specific attack name. For example, a_{Trinoo} represents the attack Trinoo. Henceforth, a_{Trinoo} will be shortened to a_T for reasons of brevity. The important aspect of an attack is the way that it can be detected. Components must have forensic relevance, or the ability to contain evidence of an attack. Thus, an attack is characterized by the components that contain evidence of that attack. Then each a_i is associated with a set C_i that contains these components, and

$$C_i \subseteq C.$$

An attack a_i on a specific operating system o^m defines a component set C_i^m, where i represents the attack a_i and m specifies the operating system. Thus, C_i^m is the set of c_j on o^m that are affected by a_i. The set C_i^m is the intersection of C_i and C^m, i.e.

$$C_i^m = C_i \cap C^m.$$

In order to identify whether a_i occurred on o^m we will have to examine the components in C_i^m with a forensic procedure.

Forensic Primitives and Procedures

The final element of the model is the forensic primitive, denoted by f_j. Forensic primitives are the basic building blocks of a forensic procedure. Primitive f_j is an abstraction of the actual steps taken by a forensic investigator during the course of an investigation. Forensic primitive f_j is closely associated with its component c_j. Like components, forensic primitives are platform independent. For each c_j, there exists exactly one forensic primitive f_j that represents the investigation of that component. Thus, the mapping from components to forensic primitives is a bijection, i.e. one-to-one and onto. In English, a forensic primitive f_j might be said to "examine component c_j".

Let F be the set of all f_j. The number of elements in F is exactly the same as the number of elements in C, i.e. the cardinality of C. As a result,

$$F = \{f_1, .., f_n\},$$

where n indicates the total number of forensic primitives. Now a forensic procedure can be formally defined as a set of f_j. Let F_i be the set of f_j that correspond to c_j in set C_i. Set F_i is called the forensic procedure necessary to detect attack a_i. Finally, let F_i^m denote the forensic procedure necessary to detect attack a_i on OS o^m.

Forensic Actions

Forensic actions are defined as specific activities that an investigator can take. These may be operating system-specific command line instructions, or they may be some other physical step an investigator needs to perform. Each forensic primitive corresponds to one or more equivalent lists of actions for a given operating system. In order to fully carry out the investigation defined by a forensic primitive, all of the actions in one of the lists must be used.
Allowing multiple actions within an action list allows for the possibility that more than one action may be necessary to fully investigate a component. For example, fully examining the component "password file" not only requires an examination of the contents of the file, but also of the file metadata. Table 2 shows the forensic primitive and corresponding actions for the Linux component "password file," c_p^L.

Component	**Primitive**	**Action list**
c_p^L	f_p^L	cat /etc/passwd ls -l /etc/passwd

Table 2: Actions to Investigate Password File

In addition, there may be multiple alternative action lists for a given forensic primitive. This allows for the possibility of using different operating system tools to perform the same investigation. Which list is used can depend on institutional requirements or tool availability at a given time. Table 3 shows some alternative action lists for c_p^L.

Component	**Primitive**	**Action list 1**	**Action list 2**
c_p^L	f_p^L	cat /etc/passwd ls -l /etc/passwd	less /etc/passwd ls -l /etc/passwd

Table 3: Alternative Actions to Investigate Password File

In these examples, the actions in the alternative lists are all command-line instructions. This need not be the case. Some components may be physical objects that the investigator has to locate and examine. For example, if the computer is being investigated for possible use in an identity theft operation, then an attached scanner may provide supporting evidence. To examine the "scanner" component is to actually look for it.

Limitations

Before illustrating the framework in action, the reader should be reminded of the role of the framework in the larger world of computer and network security. This will reveal the limitations of this technique. As discussed earlier, the framework is not attempting to assume the role of either an IDS or a forensic investigator. Rather, it serves as a tool for managing forensic procedures in an effective and deterministic way.

In a real-world scenario, the many players will work together. Intrusion detection systems are created and deployed in order to detect attacks in real time, including known and unknown variants. In the case of a suspected successful attack, either these systems or some other technique will be used to discover that something actually has happened, although the specifics will likely not be known. At this time, the forensic investigator is called to perform diagnostics and discover the specifics. The investigator will use the forensics procedures. Finally, the investigator draws a conclusion about the nature of the attack based on the results obtained by following the forensics actions specified by the procedure.

It is important to see that the framework itself is incapable of drawing conclusions about a specific incident. However, the framework will ensure that the investigator has enough information about the incident to draw a correct and trustworthy conclusion. This is done by ensuring that the forensic process used was correct and thorough for the target computer system and potential attacks. Finally, it should again be emphasized that the framework will not aid in the detection of attacks that have not been previously described by an expert. For this reason, the investigator will still be required to use existing techniques for discovering new attacks or novel variations of existing attacks. Thus the framework is not a general "solution" to forensics, but rather a tool for field investigators.

Model Summary

The final relationship between the attack and a forensic process is summarized in Figure 2.

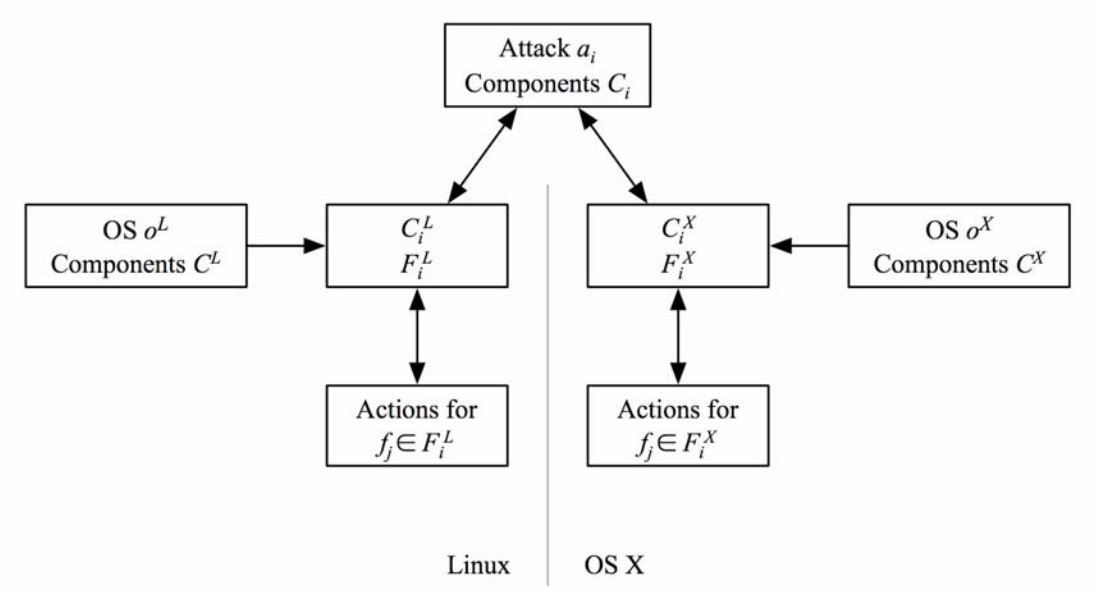

Figure 2: Hierarchy of Elements

This shows that the framework can be traversed to move from an attack to a detecting procedure, or move from a procedure to the attacks it can detect. The following scenarios are considered with reference to the four motivating questions in Section 3.

Scenario 1: A forensic procedure and actions to detect attack a_i can be derived as follows: The component set C_i defines C_i^m for a specific operating system o^m. This specifies the OS independent and dependent forensic procedure F_i and F_i^m respectively and finally the forensic actions for each f_j in F_i^m. It should be noted that for a new a_i some or all of the associated f_j in F_i^m may already exist due to previously defined attacks.

This process can be viewed as a traversal of the graph in Figure 3, which uses the example operating system o^L. It shows that the sets C_i^L and F_i^L are derived from C_i, and then the actions are derived from F_i^L.

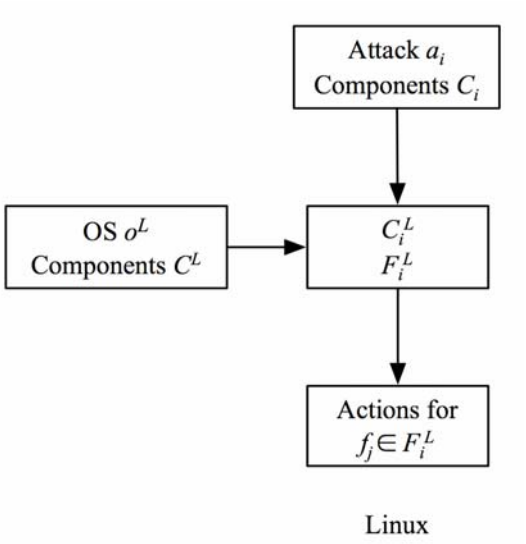

Figure 3: Creating a Procedure for Linux

Scenario 2: Given forensic actions, the attacks that can be identified are determined as follows: Each action identifies a forensic primitive f_j. Let X be the set of all these f_j. Each a_i can be detected for which $F_i^m \subseteq X$. Expressed with respect to forensic components, set X specifies its corresponding component set Y and all a_i can be detected for which $C_i^m \subseteq Y$. This process is referred to as *auditing*.

The graph in Figure 4 shows a representation of determining the attacks that can be detected for a forensic procedure for Linux. The actions are translated into forensic primitives, which then correspond to attacks.

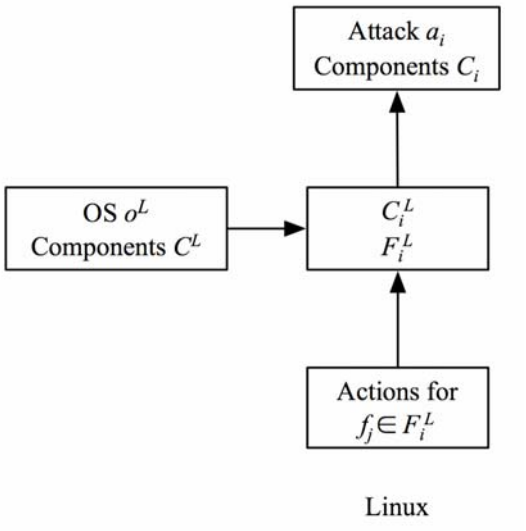

Figure 4: Auditing a Procedure for Linux

Scenario 3: Updating in general may affect C, C_i, C^m and/or C_i^m. A typical example might be that new information about a_i results in the introduction of a new component c_j in C_i. If $c_j \in C^m$, this will require an update of C_i^m. Conversely, if $c_j \notin C^m$ then C_i^m and thus F_i^m remain unchanged.

The process of updating a procedure is conceptually similar to creating a procedure, as shown in Figure 5. Like Scenario 1, the lower elements in the diagram are derived from the elements above it.

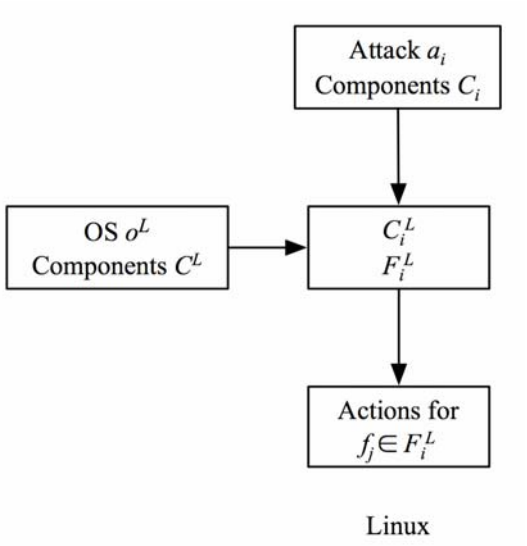

Figure 5: Updating a Procedure for Linux

Scenario 4: Porting forensics procedures from one operating system o^A to another OS o^B can be achieved using the following process. Procedure F_i^A is defined with respect to a_i. Therefore C_i is the OS independent component set that is the basis for specifying C_i^B and thus F_i^B. It should be noted that this scenario is very similar to Scenario 2 followed by Scenario 1.

A representation of this process is shown in Figure 6 and can be thought of as traversing the diagram in the direction of the arrows indicated. To translate from o^L to o^X, we translate from F_i^L to C_i^L, which allows us to determine C_i. This is then used to create C_i^X and thus F_i^X.

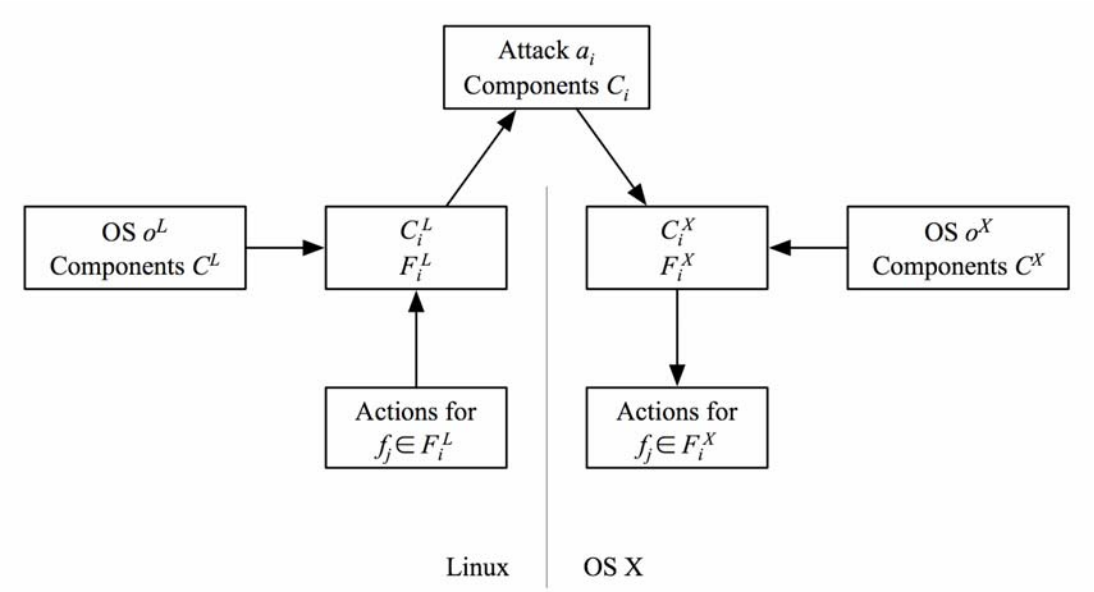

Figure 6: Porting a Procedure from Linux to OS X

Abstraction Levels

The level of abstraction used when choosing components can be matched to the goals of the forensic procedure. Depending on what type of evidence is desired, the component set *C* can be composed of broader or more fine-grained components. In all cases, the resulting forensic procedure will be able to detect the specified attacks. However, the attacks themselves will be at an equivalent level of detail as the components. This allows for the forensic procedure to be tailored for institutional requirements. As an illustration, we consider three different scenarios.

Low-level components

This first case is the example that has been used throughout this paper: discovering traditional intrusions into a computer system. This task may be performed by network system administrators or law enforcement organizations. As has been shown, detecting an attack of this nature involves low-level activities, such as examining temporary file space, checking for certain processes, or looking for network connections. Minor differences in these components can lead to completely different conclusions about the nature of the attack. For example, we have seen that a listening process on port 31335 can be linked to an instance of Trinoo on the target computer. But a listening process on port 31336 may mean nothing at all. Thus, the acquired digital evidence must be examined for extremely fine distinctions in order to draw the proper conclusions. And thus the corresponding components must be able to gather enough details to differentiate the data properly. Table 4 presents some example components for this type of low-level examination.

Component
Temporary file space
Ports
Password file

Table 4: Example Low-Level Components

Mid-level components

A more high-level approach can be used for the situation described by Goldman and Mackenzie [9]. They discuss the nature of computer abuse in the setting of a college campus. In this case, the investigation would consist of scanning a particular computer to see if it violates university policy or law in the areas detailed in [9]. These abuses include such things as harassment, hacking, and copyright violation (e.g. sharing copyrighted music, or "file-sharing").

Note that while the term *attack* was used earlier, here it is more appropriate to consider the term *computer abuse*. The change in semantics does not alter the use of the model, but it does help to understand the use of the model in this context.

It is clear that examining which specific ports are active will not help when looking for copyright violations. Here we must broaden the scope of the components to account for broader nature of the desired evidence. Table 5 presents some examples components for this level of investigation.

Component
MP3 music files
Email messages
Network activity

Table 5: Example Mid-Level Components

Recall that the components themselves are not incriminating; they merely have the ability to contain evidence. Thus the possession of music files, as listed in Table 5, is not evidence of wrong-doing. However, a thorough investigation into possible computer abuses must examine music files to determine if a copyright violation has occurred.

High-level components

Finally, an example of a high-level approach is given by the National Institute of Justice's Electronic Crime Scene Investigation (ECSI) handbook [10]. This is a guide used by federal law enforcement organizations for examining computers for evidence of a variety of crimes. A thorough demonstration of the model used in conjunction with the ECSI handbook will be presented in the MAC OS X case study . However, a brief description is provided here.

The handbook is designed to be used for all types of crimes that may involve a computer, even if the computer itself is only incidental to the crime. These crime categories include issues such as extortion, identity theft, and child abuse. The ECSI guide also describes where evidence of these crimes might be found on a computer involved in the crime investigation. These locations can then be used as a basis for creating components. Table 6 presents some examples of components for this level of investigation.

Component
Address book
Scanner
Email messages

Table 6: Example High-Level Components

Again, note that the term *attack* can be replaced with term *crime* in this context. To properly investigate the crime of identity theft, the ECSI handbook recommends looking for an attached scanner (for making fake documents). Thus, a complete forensic procedure should include some investigation of the component "scanner," which would be performed by physically looking for a scanner. Compared with the previous examples, this represents a very high level of abstraction. Yet these components are compatible with the model, and are useful when considering the ECSI definition of computer-related crimes.

Model Contributions

Now that a definition for the model has been presented, we can show how this addresses the limitations of current forensics processes.

Procedural: The procedural problem referred to the unmanageable amounts of data on modern computer systems. By applying the model, the forensic procedure can target the specific relevant components, thereby gathering only the appropriate information. This limits the amount of data and reduces the data management problem.

Technical: Rapid changes in technology lead to the technological problem. The model allows forensic procedures to be modified quickly in response to changes in technology, thus allowing investigators to keep up. It also gives non-technical investigators (e.g. local police departments) a way to create and maintain forensic procedures without requiring high levels of technical skill. In addition, the model is well suited to implementation in software. Modifying procedures to adapt to technological changes can then become a matter of "plug and play."

Social: The social problem referred to uncertainties about the capabilities of current processes. The tight coupling of attack components and forensic

procedures means the capabilities of forensic procedures is completely defined (or can be derived as needed). Since the model guarantees that the correct evidence has been gathered, the need to preserve extraneous data is eliminated.

Legal: While the legal requirements for digital evidence are still uncertain, it is clear that the formal approach of the model increases confidence in the results. For any conclusion reached during a forensic investigation, it can be proven that all of the necessary evidence was collected. In a legal context, this helps to eliminate doubt that might otherwise hamper prosecution.

Applications

The following sections will use the techniques described in low, mid, and high-level components scenarios described above to create and modify forensic procedures for Linux and OS X with respect to the DDos client Trinoo.

Creating a Forensic Procedure

Consider the case of creating a forensic procedure to detect Trinoo on Linux. In the notation of the model, we wish to create the forensic procedure F_T^L to detect attack a_T on operating system o^L. This procedure will provide the evidence necessary to prove or disprove the presence of a_T on o^L, requiring that the procedure examine every "fingerprint" that Trinoo leaves. The enumeration of these characteristics has been fully performed by an expert, e.g. CERT. For Trinoo, the list is shown in Table 7, which has been restated from Table 1 for convenience.

Characteristic
The presence of a configuration file named "…"
A listening process on port 31335
The presence of a process named "httpd"

Table 7: Trinoo Attack Characteristics

The first step in the process is to determine which components a_T affects. Each of the characteristics in Table 7 is mapped to a component, as can be seen in Table 8.

c_i	Component	Characteristic
c_1	File system	The presence of a configuration file named "…"
c_2	Ports	A listening process on port 31335
c_3	Process table	The presence of a process named "httpd"

Table 8: Trinoo Components and Characteristics

This leads to the specification of C_T, the platform independent component set for Trinoo, i.e.

$$C_T = \{c_1, c_2, c_3\}.$$

Next, we must determine which of these components are relevant to the target operating system o^L. We will use the Linux-specific component set C^L to make that determination. Since the set C^L is too long to enumerate here, it should be observed by the experienced reader that all three components in C_T are indeed present on all Linux systems. By performing a set intersection, we are led to the set C_T as it applies to o^L, i.e.

$$C_T^L = C_T \cap C^L = \{c_1, c_2, c_3\}.$$

The next step is to translate the components into corresponding forensic primitives. Since forensic primitives are abstract and correspond directly to components, this process is simple. We merely substitute the forensic primitives for their component counterparts. Thus the forensic set for attack a_T with respect to o^L is

$$F_T^L = \{f_1, f_2, f_3\}.$$

Finally, we can translate the primitives in the new forensic procedure into actual forensic actions. These are the steps that an investigator will perform when attempting to detect a_T on o^L. For Linux, we arrive at the list of actions shown in Table 9.

f_i	Action
f_1	find / -name '...'
f_2	lsof -i
f_3	ps -ef

Table 9: Forensic Actions

By performing the actions in Table 9, an investigator can detect Trinoo on a Linux machine. Note that this does not mean that the model will be able to detect a Trinoo client in a particular investigation. Rather, it means that the performed investigation will have gathered enough information to draw a correct conclusion about the presence or absence of a Trinoo client. It is up to the investigator to interpret the data and make the final determination.

Auditing a Forensic Procedure

Another application of the model is auditing an existing procedure to determine if it can detect a given attack. In this example, we want to see if a certain procedure for Linux can detect Trinoo. Auditing is useful for proving the effectiveness of a procedure or for applying structure to a previously unstructured forensic procedure. This application will be conceptually similar to reversing the steps described above. Assume that Table 10 shows F^L, which is currently assumed to be the complete set of actions used for detecting attacks on Linux.

Action
find / -name '...'
ps -ef
who -al

Table 10: Linux Forensic Actions

Comparing Table 10 to Table 9, it is obvious that not all of the steps required to detect Trinoo are present; i.e. the command "lsof" is missing, and a new command has been added. This new step has been inserted for purposes of illustration, and could be involved with detecting a different attack. Using the model, we can identify the missing command.

The first step is to convert the actions into forensic primitives. This is accomplished by merely matching the actions to their corresponding primitives from the existing primitive set, e.g. Table 9 identifies the first two primitives f_1 and f_3. Table 11 shows primitives and their associated actions.

f_i	Action
f_1	find / -name '...'
f_3	ps -ef
f_4	who -al

Table 11: Primitives and Actions

From this set of primitives, the list of corresponding components is generated. This will show which components on the target system will be examined when the forensic process is applied. Again, this is a mere matching of forensic primitives to components, drawing from a previous enumeration. Table 12 shows the matching components and primitives.

f_i	c_i
f_1	c_1
f_3	c_3
f_4	c_4

Table 12: Forensic Primitives and Components

Now the component set in Table 12 can be compared to a known attack, which is Trinoo in this case. As seen earlier, an expert has determined that Trinoo affects the components c_1, c_2, and c_3. Comparing C_T^L to the list in Table 9, it is clear that the procedure does not detect all aspects of Trinoo, e.g. c_2 is missing. From this we conclude that when the procedure is performed on an actual compromised system, it may not be able to conclusively prove or disprove the existence of Trinoo. It is a straightforward matter to add the correct entries to our forensic process to make it fully able to detect Trinoo.

Updating a Forensic Procedure

There are a number of reasons for updating a forensic procedure. A common case is when new information is discovered about an existing attack. This may require that new primitives be added to the existing procedure. For this example, we assume the scenario from Section 5.2, where

$$F^L = \{f_1, f_3\}.$$

We will treat F^L as if it were previously assumed complete for a_T, so that

$$C_T = \{c_1, c_3\}.$$

Assume that the attack expert has discovered a new component that a_T affects, namely c_2. If $c_2 \in C^L$, then this new information is relevant to o^L and the existing procedure must be corrected. The updated procedure F'^L is created by inserting a primitive to detect the new component, in this case f_2. The model expresses this as

$$F'^L = F^L \cup \{f_2\}.$$

The process performed in this section updates a procedure to account for new information about an attack. It works equally well for correcting a deficient procedure, as determined by the process described in Section 5.2.

Porting a Forensic Procedure

A final application of the model is to take an existing forensic procedure for one OS and create a forensic procedure for another OS in such a way as to guarantee that both procedures can detect the same attacks. Here, we will take our example forensic procedure on Linux and apply it to OS X.

As with attacks, there has been an expert determination of the components of this new operating system. Thus the set C^X is known. This example assumes that a forensic procedure exists for Linux, but that the attacks that the F^L procedure can detect are unknown. To generate a procedure for OS X that can detect the same attacks first requires that this attack set be determined. Then it is straightforward to create the new procedure for OS X, following the steps outlined in previous examples.

The first task is to determine which attacks the existing F^L can detect. Assuming F^L from "Creating a Forensic Procedure," we have

$$F^L = \{f_1, f_2, f_3\}.$$

For each known a_i, we check if it can be detected by testing if $F_i^L \subseteq F^L$. The next step is to generate the forensic procedure F^X for OS X. This is identical to the process performed in creating a forensic procedure, for all attacks that can be detected by F_L. In short, the attack components are mapped into forensic primitives, which can then be translated into actions.

Case Study: Mac OS X

This section presents a comprehensive example of an application of the model. The techniques used here are the same as those described in the above examples. In this case, rather than focusing on a particular attack, many types of intrusions will be considered on a Mac OS X system. The end result will be a specific forensic procedure that can be used by field investigators for performing live forensics on OS X. The generated procedure will only use standard tools available on the operating system.

A significant difference between this case study and the previous examples is that now high-level components (rather than low-level components) with respect to the ECSI [10] are considered. As indicated before, the ECSI is a reference for crime scene investigators that provides many categories of computer crimes and how to investigate each type of crime, and is used by the FBI and other federal agencies. We will use this guide as a basis for developing the forensic procedures in this case study. In previous examples, the expert that enumerated the attack characteristics was CERT or some equivalent. Here it will be the National Institute of Justice. The ECSI handbook is designed for dealing with computer systems as part of a larger investigation. Thus the forensics procedure developed in this section will also serve as a complement to traditional forensic techniques.

There are several reasons why the ECSI has been chosen as the source for the case study. First, it illustrates that the particular choice of the components is not central to the model. The case study will show that the model is a tool for dealing with abstractions, and is not itself specific to any given set of primitives. In addition, the use of the ECSI data shows how the model can be incorporated into a real-world forensics scenario. Finally, this case study will generate a working forensic procedure that can be used in a wide variety of situations dealing with computer crime, which increases its value as a practical tool.

Attack Coverage

The first step in creating the forensics procedure is to determine the desired attack coverage. As in the previous examples, this is a list of intrusions that we wish to detect. This case concerns with a broader range of computer crimes, so it will be more general than the traditional notion of hacks. Given the scope of this case study, the term *crime* will be used interchangeably with *attack* henceforth. Table 13 lists the crime categories that are identified in the ECSI handbook.

Attack	Description
a_1	Auction Fraud
a_2	Child Exploitation/Abuse
a_3	Computer Intrusion
a_4	Death Investigation
a_5	Domestic Violence
a_6	Economic Fraud
a_7	E-Mail Threats/Harassment
a_8	Extortion
a_9	Gambling
a_{10}	Identity Theft
a_{11}	Narcotics
a_{12}	Prostitution
a_{13}	Software Piracy
a_{14}	Telecommunications Fraud

Table 13: Computer Crimes

These crimes will comprise the attack coverage of the case study. In most of the attacks in Table 13, a computer is not the focus of the investigation. Rather, the evidence gathered on the computer will be used to support a broader investigation.

OS X Components

In order to correlate these attacks with the operating system, we must create the list of components. For OS X, i.e. o^X, the component list will be C^X. Table 14 contains a listing of components, which were drawn from ECSI data.

Component	Description
c_1	Account data
c_2	Accounting/bookkeeping software
c_3	Address books
c_4	Calendar
c_5	Chat logs
c_6	Cloning software
c_7	Configuration files
c_8	Credit card reader/writer
c_9	Date and time stamps
c_{10}	Diaries
c_{11}	Digital cameras
c_{12}	Erased internet documents
c_{13}	Executable programs
c_{14}	E-mail
c_{15}	Graphic editing and viewing software
c_{16}	History log
c_{17}	Images
c_{18}	Image players
c_{19}	Internet activity logs
c_{20}	Internet browser history/cache files
c_{21}	IP address and user name
c_{22}	Movie files
c_{23}	Scanners
c_{24}	System files
c_{25}	Temporary files
c_{26}	User-created files and directories

Table 14: Components

As discussed earlier, these components are abstract and can contain evidence of a crime. Further, the level of abstraction of the components has been chosen to correspond with the abstraction level of the attacks under consideration. Since our attack list in Table 13 is at a rather high level, the component list will also be at a high level. Finally note that for this case study, all components in the global set *C* are also in C^X, i.e. $C^X = \{c_1,...,c_{26}\}$

Correlating Crimes and Components

For each of the crimes a_i in Table 13, we need to generate the associated component list C_i. Table 15 shows the component lists for each a_i, using definitions from the ECSI handbook.

Attack	Components
a_1	$c_1, c_2, c_3, c_4, c_5, c_{11}, c_{14}, c_{17}, c_{19}, c_{20}$
a_2	$c_5, c_9, c_{11}, c_{14}, c_{15}, c_{16}, c_{17}, c_{19}, c_{22}, c_{26}$
a_3	$c_3, c_7, c_{13}, c_{14}, c_{19}, c_{21}, c_{25}$
a_4	$c_3, c_{10}, c_{14}, c_{17}, c_{19}$
a_5	c_3, c_{10}, c_{14}
a_6	c_3, c_4, c_{14}, c_{19}
a_7	$c_3, c_{10}, c_{14}, c_{17}, c_{19}$
a_8	$c_9, c_{14}, c_{19}, c_{25}$
a_9	$c_3, c_4, c_{14}, c_{16}, c_{18}, c_{19}$
a_{10}	$c_8, c_{11}, c_{12}, c_{14}, c_{23}, c_{24}$
a_{11}	c_3, c_4, c_{14}, c_{19}
a_{12}	c_3, c_4, c_{14}, c_{19}
a_{13}	$c_5, c_{14}, c_{19}, c_{26}$
a_{14}	c_6, c_{14}, c_{19}

Table 15: Component Lists

This table shows the components that can provide evidence of the various types of crimes. If our generated procedure investigates the appropriate components, then it will be able to thoroughly investigate the crime in question.

For each attack, the component list needs to be adjusted to account for the characteristics of o^X, to create C_i^X. This is performed by deriving $C_i^X = C_i \cap C^X$.

In this case, the components in the attack sets are all present on o^X, so the results will be identical to Table 15.

Forensics Procedure

The next step is to create the forensics processes to detect each type of crime. Recall that a forensic primitive f_i corresponds exactly to a component c_i, and represents the investigation of that component. Table 16 lists F, the set of all forensic primitives.

Primitive	Description
f_1	Search account data
f_2	Find accounting/bookkeeping software
f_3	Examine address books
f_4	Examine calendar
f_5	Examine chat logs
f_6	Find cloning software
f_7	Examine configuration files
f_8	Look for credit card reader/writer
f_9	Get date and time stamps
f_{10}	Examine diaries
f_{11}	Look for digital cameras
f_{12}	Find erased internet documents
f_{13}	Find executable programs
f_{14}	Examine e-mail
f_{15}	Find graphic editing and viewing software
f_{16}	Examine history log
f_{17}	Find images
f_{18}	Find image players
f_{19}	Examine internet activity logs
f_{20}	Examine internet browser history/cache files
f_{21}	Find IP address and user name
f_{22}	Find movie files
f_{23}	Look for scanners
f_{24}	Examine system files
f_{25}	Examine temporary files
f_{26}	Find user-created files and directories

Table 16: Forensic Primitives

Using the primitives in Table 16, we can generate the forensic procedure necessary to detect each attack. Thus for each a_i, we will create F_i^X.

Substituting the forensic primitives in Table 16 for the components in Table 15, we arrive at the list of all F_i^X shown in Table 17.

Procedure	Primitives
F_1^X	$f_1, f_2, f_3, f_4, f_5, f_{11}, f_{14}, f_{17}, f_{19}, f_{20}$
F_2^X	$f_5, f_9, f_{11}, f_{14}, f_{15}, f_{16}, f_{17}, f_{19}, f_{22}, f_{26}$
F_3^X	$f_3, f_7, f_{13}, f_{14}, f_{19}, f_{21}, f_{25}$
F_4^X	$f_3, f_{10}, f_{14}, f_{17}, f_{19}$
F_5^X	f_3, f_{10}, f_{14}
F_6^X	f_3, f_4, f_{14}, f_{19}
F_7^X	$f_3, f_{10}, f_{14}, f_{17}, f_{19}$
F_8^X	$f_9, f_{14}, f_{19}, f_{25}$
F_9^X	$f_3, f_4, f_{14}, f_{16}, f_{18}, f_{19}$
F_{10}^X	$f_8, f_{11}, f_{12}, f_{14}, f_{23}, f_{24}$
F_{10}^X	f_3, f_4, f_{14}, f_{19}
F_{12}^X	f_3, f_4, f_{14}, f_{19}
F_{13}^X	$f_5, f_{14}, f_{19}, f_{26}$
F_{14}^X	f_6, f_{14}, f_{19}

Table 17: Forensic Procedures

Since the goal of this process is to create F^X, the forensic procedure that can detect all of these attacks, we combine the individual forensics procedures, so that

$$F^X = \bigcup_{i=1}^{14} F_i^X .$$

Forensic Actions

Finally, we can translate the forensic primitives into actual system dependent forensic actions. Recall that the model allows alternate action lists, each of which gathers the same data in a different way. This allows the investigator to choose a set of actions that fulfills institutional requirements. For simplicity, however, we have only listed one such alternative here, using only standard tools. Table 18 presents the forensic primitives and their corresponding actions. The material in this table was drawn from Apple's technical documentation [11], existing Linux forensic actions [12], and online resources [13].

Primitive	Actions
f_1	cat ~/Library/Keychains/*
f_2	ls –lR /usr/bin ls –lR /Applications
f_3	cat ~/Library/Addresses/*.addressbook
f_4	cat ~/Library/Calendars/*.ics
f_5	find / -name *.chat
f_6	ls –lR /usr/bin ls –lR /Applications
f_7	tar –cf - -C /etc tar –cf - -C ~/Library/Preferences
f_8	system_profiler Look for credit card reader/writer
f_9	ls –lR /
f_{10}	ls –lR ~/Documents
f_{11}	system_profiler Look for digital cameras
f_{12}	tar –cf - -C ~/.Trash
f_{13}	ls –lR /usr/bin ls –lR /sw/bin ls –lR /Applications
f_{14}	cat ~/Library/Mail/*/*/mbox
f_{15}	ls –lR /usr/bin ls –lR /Applications
f_{16}	cat ~/.bash_history cat ~/Library/Safari/*.plist
f_{17}	find / -type f -exec file {} \; | grep image
f_{18}	ls –lR /usr/bin ls –lR /Applications
f_{19}	cat ~/Library/Safari/*.plist
f_{20}	tar –cf - -C ~/Library/Caches/MS\ Internet\ Cache . tar –cf - -C ~/Library/Caches Safari .
f_{21}	ifconfig –a uname –a
f_{22}	find / -type f -exec file {} \; | grep "ASF|movie"
f_{23}	system_profiler Look for scanners
f_{24}	cat ~/Library/Logs pstat -fstv
f_{25}	tar -cf - -C /tmp/ .
f_{26}	ls –lr ~/Documents ls –lr ~/Images ls –lr ~/Movies ls –lr ~/Pictures

Table 18: OS X forensic actions

In the case of alternative action lists, the choice of which to use depends on the context of the investigation. Commonly when investigations are performed on live systems, it is desirable to modify as little as possible on the target system. Thus, any data collected is not stored locally, but rather sent over a network connection to the investigator's computer. In that context, interactive commands are less desirable, because they do not work well over a network. When specifying the actions, commands are chosen that do not require interaction or expectations about the terminal environment. For example, "cat" instead of "vi" has been used for examining files, even though "vi" may in fact provide an investigator with more utility.

Some of the actions in Table 18 are actual command line commands, but some are physical actions that must be taken. For example, c_{23} specifies the component "scanners," where scanners may provide evidence of an identity theft operation. The investigator must therefore look for one.

Again, the actions in Table 18 represent merely one possibility. Adding extra items to an action list as needed is completely supported by the model. The minimum requirement of an action list is that it can provide evidence to prove or disprove the presence of a crime. However, an investigator may find value in augmenting an action list to gather further data useful for pursuing their investigation.

Finally, Table 18 shows only commands that are native to OS X. It should not be inferred, though, that the actual binaries are resident on the computer under investigation. A proper investigation would use binaries that are known to be uncompromised, typically on a write-only media such as a CDROM.

Final Results

Combining all the actions in Table 18, we arrive at the actual process that an investigator can follow to detect the crimes in Table 13. The results are shown in Table 19. Each line represents an action that must be performed.

Final List of Actions
cat ~/Library/Keychains/*
ls –lR /usr/bin
ls –lR /Applications
cat ~/Library/Addresses/*.addressbook
cat ~/Library/Calendars/*.ics
find / -name *.chat
tar –cf - -C /etc
tar –cf - -C ~/Library/Preferences
system_profiler
Look for credit card reader/writer
ls –lR /
ls –lR ~/Documents
Look for digital cameras
tar –cf - -C ~/.Trash
ls –lR /sw/bin
cat ~/Library/Mail/*/*/mbox
cat ~/.bash_history
cat ~/Library/Safari/*.plist
find / -type f -exec file {} \; | grep image
cat ~/Library/Safari/*.plist
tar –cf - -C ~/Library/Caches/MS\ Internet\ Cache .
tar –cf - -C ~/Library/Caches Safari .
ifconfig –a
uname –a
find / -type f -exec file {} \; | grep "ASF|movie"
Look for scanners
cat ~/Library/Logs
pstat –fstv
tar -cf - -C /tmp/ .
ls –lr ~/Documents
ls –lr ~/Images
ls –lr ~/Movies
ls –lr ~/Pictures

Table 19: Final Action List

This is a process that is suitable for performing a real investigation on a live OS X system. The process in Table 19 is adequate for investigating the crimes in Table 13. However, it could be improved in several ways. Many of the actions are the simplest possible, and they could be augmented with further steps or replaced with a custom tool. In addition, simplifying assumptions were made about the user environment. Extra steps would be required to account for more complicated scenarios. Finally, the component list presented in the ECSI handbook takes a simplistic view of computer systems. Lowering the abstraction

level of the components might better describe modern operating systems, and create a more robust forensic procedure. All of these suggested improvements are easily performed using the model, by adding the appropriate information to any of the tables.

Conclusion

The problem of computer crime is rapidly growing, requiring increasing expertise in the areas of attack detection and prosecution. Many law enforcement organizations use ad-hoc or informal procedures for performing investigations, which can limit confidence in the investigation results.

This paper has presented a comprehensive technique for generating and modifying computer forensic procedures using the attributes of attacks and the targeted computer system. This is done with a model that specifies mathematically the relationship between attacks and abstract system components.

To demonstrate the use of the model, several examples are given that show the model in various scenarios. Specifically, methods are shown for creating, updating, auditing, and porting a forensic procedure. Using the model and following the examples, it is possible to create and maintain forensic procedures that are precise and accurate.

About the Authors

Ryan Leigland received his M.S. in computer science from the University of Idaho in 2004. His main research interest is in computer security and computer forensics and he has been funded under the National Science Foundation's Scholarship for Service program. His practical experience includes an internship at the NASA Office of Inspector General, Computer Crimes Division, where he was involved in forensic investigations. He can be reached at ryanl@cs.uidaho.edu.

Axel W. Krings is an Associate Professor of Computer Science and is currently on sabbatical from the University of Idaho at the ID-IMAG in Grenoble, France. He received his M.S. in Electrical Engineering from the FH-Aachen, Germany, in 1982, and his M.S. and Ph.D. degrees in Computer Science from UNL, in 1991 and 1993, respectively. Dr. Krings has published over 60 journal and conference papers in the area of Survivability and Fault-Tolerant Systems, as well as Computer and Network Security. His current research in System Survivability is

funded by CNRS and INRIA. Dr. Krings is a senior member of the IEEE, the IEEE Computer Society and the IEEE Reliability Society. He can be reached at axel.krings@imag.fr

References

[1] Computer Emergency Response Team, *CERT/CC Statistics 1988-2003*, Carnegie Mellon University Software Engineering Institute, CERT Coordination Center, 22 January 2004. Available: http://www.cert.org/stats

[2] B. Carrier, *Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers,* International Journal of Digital Evidence, Vol. 1 Issue 4, Winter 2003.

[3] R. McKemmish, *What is Forensic Computing?*, Australian Institute of Criminology Trends and Issues, Number 118, June 1999. Available: http://www.aic.gov.au/publications/tandi/tandi118.html

[4] P. Stephenson, *Modeling of Post-Incident Root Cause Analysis,* International Journal of Digital Evidence, Vol. 2 Issue 2, Fall 2003.

[5] Digital Forensic Research Workshop, *A Road Map for Digital Forensics Research 2001*, Digital Forensics Research Workshop, 6 November 2001. Available: http://www.dfrws.org/dfrws-rm-final.pdf

[6] R. Erbacher et al, *Computer Forensics Education,* IEEE Security and Privacy, July/August 2003.

[7] H. Axlerod and D. Jay, *Crime and Punishment in Cyberspace: Dealing with Law Enforcement and the Courts*, Proceedings of the 27th Annual ACM SIGUCCS Conference on User Services,1999.

[8] D. Dittrich, *The DoS Project's " trinoo " distributed denial of service attack tool*, Available: http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt

[9] K. Goldman and E. Mackenzie, *Computer Abuse, Information Technologies and Judicial Affairs*, Proceedings of the 28th annual ACM SIGUCCS Conference on User Services, 2000, pp. 170-176.

[10] National Institute of Justice, *Computer Forensic Testing Tool Program*, 2004. Available: http://www.ojp.usdoj.gov/nij/sciencetech/cftt.htm

[11] Apple Computer, *Technical Notes*, Apple Computer. Available: http://developer.apple.com/technicalnotes/

[12] B. Grundy, *The Law Enforcement and Forensic Examiner Introduction to Linux: A Beginner's Guide,* January 2004. Available: http://www.linux-forensics.com/linuxintro-LEFE-2.0.5.pdf

[13] Mac OS X Hints, *Mac OS X Hints*, 2004. Available: http://www.macosxhints.com/