The Role of Suspicion in Model-based Intrusion Detection

Timothy Hollebeek and Rand Waltzman
Teknowledge Corporation
4640 Admiralty Way, Suite 1010
Marina del Rey, CA 90292, USA
tholleb@teknowledge.com, rwaltzma@teknowledge.com

Abstract

We argue in favor of the explicit inclusion of suspicion as a concrete concept to be used in the analysis of audit data in order to guide the search for evidence of misuse. Our approach is similar to that of a human forensic analyst, who first notices details that seem slightly odd, and then investigates further and cross checks information in an attempt to build a coherent explanation for the observed details. We use deductive reasoning combined with expert knowledge about system behavior, potential attacks and evidence, and patterns of suspicion to link individual clues together in an automated way.

A prototype implementation that was designed based on these considerations is presented, including details of how suspicions and deductions are represented, and how these structures are updated as new evidence is discovered. Finally, we describe how this algorithm performs in practice on a realistic example where five discrete pieces of evidence are brought together automatically to create a unified and coherent description of what is believed to have occurred.

1. Introduction

Detecting intrusions into computer systems is one of the oldest research areas in computer security, and despite decades of research and increasingly sophisticated commercial systems, it remains a difficult problem. In the past decade, the amount of sensor and audit data available from systems has increased dramatically, but attacks are intrinsically difficult to recognize from raw sensor data, unless the sensor was explicitly designed with that class of attack in mind. Nevertheless, it has been widely recognized that intrusions do generally leave a variety of clues behind in the audit stream, and figuring out how to recognize those clues in a way that leads to a reasonably low false positive rate has been a popular research topic. We bring two new ideas to the table that we believe show promise in improving the state of the art: first, a deductive point of view, where clues are identified, their consequences investigated, and supporting evidence may or may not be found, and second, the inclusion of suspicion as an explicit concept that guides our search. We have developed these ideas into a working prototype system that we are in the process of evaluating. After describing our methodology, we present the results of our first test of this system with a realistic example.

Section two describes our philosophy of forensics and evidence, as well as the state of existing practice and research. In section three, we discuss deduction and modeling, including how uncertainty and suspicion figure into our approach. Section four discusses some concrete details of our algorithm. Section five describes the details of our implementation, while in section six we show our algorithm at work with a brief example.

2. Forensic Analysis

The philosophy and point of view we wish to adopt is that of a forensic analyst: we are interested in the collection of evidence of possible misuse, and putting those clues together into a coherent picture that describes what has occurred on the system in question. We assume that evidence is being collected and preserved in a secure way, so that the timeliness of the analysis and the level of scrutiny it receives can be varied as needed. However, since our forensic investigations will be at least semi-autonomous, we will assume sensors with useful audit capabilities are deployed on the system, and that evidence may be being analyzed in real-time as it comes in. In this case, the role of our investigations is not to assist a human forensic analyst, but to supply information and context to a separate component that may generate autonomic responses. We assume that questions of security policy, posture and response are the responsibility of that component and independent of the situational awareness supplied by our analysis engine.

2.1. Analysis of Evidence

We assume that the evidence we are analyzing is most likely fragmentary and incomplete. In the trivial case that a sensor explicitly detects an attack with high confidence (for example, an attachment is determined to be a known virus or a network packet contains an exploit for a known buffer overflow vulnerability), these events will be noted and added to the situational awareness, but analysis of such scenarios is fairly straightforward, and all the information needed to provide a concise and descriptive analysis of exactly what has occurred is already provided by the sensor itself. Instead, as in traditional forensics, we are interested in clues that may well seem fairly innocent when viewed individually, but suggest a coherent story of misuse or abuse when viewed in context. It is important to note that this more than just correlating alert information or collecting degrees of anomalous behavior in a "leaky bucket" until a threshold is reached; we are biased towards clusters of evidence that are consistent with a coherent "story" that explains the evidence in context of a larger pattern of behavior. This has two advantages: first, it avoids the statistical difficulties associated with the base rate fallacy that often plague intrusion detection schemes [Axelsson 99], and second, since our analysis explicitly relies upon analysis of evidence in context and with regard to the inferred pattern of activity, a concise and coherent description of what we believe is happening can be generated when the analysis succeeds. Furthermore, the entire deduction chain can be explained at each

level, down to the individual bits of concrete evidence that led to a particular deduction, and the role each piece of evidence played in leading the analysis engine to the conclusion it reached.

2.2. Existing Forensic Analysis Tools and Previous Work

It is interesting to note that there has been a divergence between intrusion detection tools and the tools used by those doing forensic analysis of systems, despite the fact that both are examining evidence to find indications of whether and how a system has been compromised. This is despite the fact that early intrusion detection research was inspired by attempts to automate tasks traditionally performed by system administrators, like monitoring the system log (see [Lunt 88] for a summary).

Forensic analysts generally use a wide variety tools, most of which can be described as evidence collection tools, and often were not explicitly designed for use as security tools. These tools bypass friendly interfaces in order to provide explicit information about the low level structure of various resources of interest. While tools packages specifically designed for forensic analysts do exist, these concentrate mainly on issues of evidence preservation, and provide minimal assistance in actually analyzing the evidence itself. In addition to those tools, a variety of configuration scanning (for example, COPS [Farmer 90]) and hardening tools have been developed, which analyze the security configuration of the machine. These are sometimes useful from a forensics point of view as they provide information about the security configuration of the machine. But there are surprisingly few tools that help a forensic analyst actually analyze the vast amount of evidence that can be obtained using these tools. Recently, some special case tools (like chkrootkit [Nelson04]) have started to appear, but these are highly specialized and tuned to detecting common clues and inconsistencies that indicate a system has been compromised.

Previous host-based intrusion detection work has focused mainly on analyzing system level events (see, for example [Hofmeyr98]) in order to detect unauthorized access or modification of the system. This is partly because for an external attack, the system itself is what comes under attack, but it is also partly because that is the only level where sensor data has been widely available. Intrusion detection can be partitioned into two main categories: anomaly detection and misuse detection. A great deal of research has been focused on anomaly detection as an intrusion detection technique, based on the assumption that even previously unanticipated attacks will cause the system to behave in a way that can be distinguished from normal behavior. A series of increasingly complex schemes have been investigated, ranging from simple n-gram based techniques [Forrest96] to Bayesian statistics [Anderson95, Porras97], data mining [Lee98, Barbara01], and neural nets [Ryan98]. While this research shows great promise in detecting anomalous application behavior and automatically learning how to do so from training data, all it can do is detect anomalous behavior. That anomalous behavior may be either benign and or it may be malicious. Unfortunately, benign anomalous behavior is several orders of magnitude more common than malicious behavior, leading to high false positive rates for anomaly detection schemes. This is a fundamental limitation of the anomaly detection approach [Axelsson99]. Furthermore, there is no guarantee that all malicious behavior is anomalous; even at the system level certain attacks like race conditions are expected to be missed by anomaly detection [Forest96]. In fact, a clever attacker can craft his attack in such a way that a known anomaly detection system will not consider it anomalous [Wagner02].

Misuse detection attempts to detect the attacks themselves [Garvey91]. A wide variety of methods have been investigated, ranging from pattern matching [Kumar94] to rules based on state transitions [Ilgun95, Eckmann00]. These approaches need sensor data from the level where the attack is actually occurring in order to reliably distinguish between attacks and normal behavior.

We believe that looking for clues that suggest an intrusion has taken place could potentially be easier than recognizing explicit attacks. The most troublesome characteristic of attacks is that they are more likely to take advantage of behaviors of the system that are complex, poorly specified, or even forgotten; indeed many attacks take advantage of the interaction of several complex features of the system. Understanding normal system behavior is difficult enough with current technology; understanding an attack, especially an unknown attack, well enough to recognize and detect it under these circumstances is extremely difficult.

On the other hand, in both the real and virtual world, it is very difficult to do anything without leaving some evidence behind. In addition, attempts to conceal or remove evidence generally create new evidence, and if detected, this evidence gives strong evidence about the perpetrator's intent. Even in the virtual world, where creating exact replicas and performing irreversible deletions are theoretically much easier, secure deletion tools are often detected due to the patterns they leave over what was erased or the traces left behind on the system even after the tool is uninstalled. Security is often difficult because the defenses must be perfect, while the attacker needs to find only one flaw. An emphasis on forensics as a second line of defense reverses the burden, by requiring the attacker and his tools to be perfect, while the defender needs only a few clues to recognize an intrusion is underway.

2.3. Motivation and Goals

These observations have lead us to build a system that identifies individual clues based on a concrete model of suspicion and then connects them deductively based on its understanding of the system and potential patterns of intrusive behavior. At first, such a system would act mainly as an aid to forensic analysts looking through audit logs, as well as assisting security professionals in detecting and understanding intrusions.

A secondary motivation is based on our belief that the biggest barrier to useful autonomic responses is a lack of information about context and intent with respect to detected intrusive events. Without such information, responses are blind and may well do as much damage as they prevent. It is important to respond to situations, not single events, and we hope that such an evidence analysis system will be able to create the necessary situational awareness to allow effective responses to be planned and executed.

To test these ideas, we have begun building a system capable of reasoning about evidence of intrusions based on our notions of

suspicion, and have started performing experiments to see how feasible this approach is.

3. Deduction and Modeling

While it may seem appealing to build a system that requires no knowledge or understanding of the system it is protecting and can detect attacks independent of any understanding of potential security threats, a system which does take this information into account almost certainly will be more effective. Certainly, a human analyst who is asked to detect intrusions without any information about potential threats and the system itself would object to this restriction and recognize that it would hinder his ability to concentrate on relevant information and make reasonable deductions. While a generic system that does not rely upon such information may be able to more easily detect certain classes of unknown attacks, it is necessarily blind to other classes, depending on what mechanism allows attacks to be seen by such a generic system. This trade-off is explicitly stated in early anomaly detection papers (e.g. [Forrest 96]), but is curiously absent from many more recent ones. By adopting a deductive approach to evidence, we restrict ourselves to scenarios involving evidence we can see and actions we understand. It seems unlikely one can do better at detecting new attacks than that, except under special circumstances.

It has long been understood that deductive reasoning is the foundation for forensics (interestingly, Sir Arthur Conan Doyle's Sherlock Holmes was inspired by medical diagnosis, another field where deductive reasoning plays a crucial role). It may be particularly effective on computers, due to their deterministic nature, though with increasing system complexity this difference may be slowly fading. In fact, deductive reasoning has already been applied to the problem of evidence evaluation, though in a different context [Keppins03]. The success of such methods in analyzing hypotheses about real world crime scenarios suggests to us that similar methods are likely to prove fruitful when applied to analysis of intrusions.

With respect to the models we base our deductions on, we view them as necessarily incomplete. A model that faithfully represents every aspect of a system is no longer a model, but a simulation. Therefore we must expect that deductions based on the models may prove false in certain cases. We rely on the models only for information about the relationship between deductions, the likely intent of inferred actions, and suggestions about what evidence *may* mean. We are interested in context and explanations, not ironclad proofs. We expect most attacks to occur at the periphery of our understanding, and hence at the periphery of our models; in fact, in the future we plan to take advantage of this to investigate more fully evidence that lies near the limits of our understanding. Suspicion, after all, is closely related to ignorance, and ignorance is a fundamental facet of understanding that should not be.

3.1. Uncertainty

As mentioned earlier, we view incompleteness as a necessary characteristic of our models, and so we need to be able to deal with uncertainty in our deductions. This particular problem is not unique to us, however, and is well known among those who work in model-based reasoning. We can easily attach uncertainties to our inferences and manage them in a self-consistent way. However if we rely on this as our only measure of uncertainty, we will have the same problems encountered by many statistical intrusion detection schemes [Axelsson 99]. In our approach, suspicion is distinct from uncertainty in that suspicion evaluates the likelihood that a given event or pattern of events is evidence of malicious behavior, while uncertainty expresses the likelihood that deductions are correct and the likelihood the observed behavior is normal *under the assumption that no malicious behavior is present*. This avoids problems associated with the fact that the probability of an event being malicious is normally very low, but rises dramatically when an attack is underway.

The more subtle problem is that much of the uncertainty involved in forensic evaluations is not just quantitative in nature, but qualitative. Many of the concepts that are useful in expressing security knowledge and policies are by nature somewhat fuzzy and context-dependent. Much of this, we believe, is because security is a field that exists only because of a lack of complete information, and hence an ability to deal flexibly with uncertainty is often deeply embedded in the concepts used to describe security concerns. For example, security experts talk about trustworthiness instead of correctness. We have found that many of the concepts we have introduced in our models are best described as shades of gray instead of black and white concepts. For example, the concept of a "foreign executable" or an "untrusted executable" is important in many of our rules, since both the concepts of origin and executability are useful generalizations that cover a wide variety of scenarios. Each of the attributes can be evaluated and inferred from available evidence, but the degree to which a resource is untrusted, and even the degree to which a resource is executable can vary (is it directly executable? or does it merely contain potentially executable content? is it raw machine instructions, or some higher level language?). And these differences in degree can affect the degree to which an object is suspicious. Introducing such subjective evaluations will certainly require some tuning in order for them to be effective, but we feel that such a faithful representation of our evaluation of the evidence is crucial to building a system that can make sensible judgments about the relative importance of various bits of evidence.

3.2. Suspicion

There are a variety of nuances to the notion of suspicion, most of which we will not concern ourselves with here. The reason is because we are content to leave a suspicion as something where "we know it when we see it". This is not particularly inappropriate, as we are proposing to let experts define exactly what sorts of things raise their suspicions. And, after all, suspicions are necessarily imprecise, since many of them turn out to be wrong. However, consider the filename:

"C:\Prográm Files\Winzip\logfile.txt<binary 0><80 spaces>.exe"

One could take the position that this is a perfectly valid Windows filename, and without more information about the state of the file system or the context in which it occurs, nothing more than that can be said. It is likely, however, that most security experts would immediately point out that it is a very suspicious filename, and that several aspects of its construction suggest it is intended to seem like it is inside C:\Program Files, which it isn't, and that it is

a text file, when it is in fact an executable. In addition to alerting us to the fact that something may be going on here, and we should probably investigate further, the filename also sugge sts some plausible scenarios that should be investigated. Certainly, if we were to later see the same file name truncated at the binary zero, we might reasonably conclude that a string -handling bug probably had been exploited. It is this sort of knowledg e about what sorts of evidence causes one to become concerned, and what sorts of checks and further investigations one might do that we think has the potential to significantly improve the performance of intrusion detection systems.

Once we have described our expert knowledge of the system and relevant security concepts, we use suspicion to link together events into patterns of behavior. Events with certain characteristics are suspicious. Furthermore, certain patterns of inferences also cause us to become suspicious about what might be occurring. Also, suspicions may be linked through deduced relations between events or inferred actions, or through concrete objects: files, processes, and so on. Our overall suspicion is then a function of the degree to wh ich each resource, event, or inference is suspicious and the number of independent reasons we have for being suspicious of it. The inference rules specify how objects and events become suspicious. For example, if a file is added as a startup file, we bec ome suspicious of the process that added it, as well as the file that was added. In addition, we consider the possible alternatives that either the file the process was started from, or the process itself was somehow compromised. As we become suspicious of new objects, this guides our search for more evidence, since the rules concerning evidence pay special attention to suspicious objects and events in a prescribed deductive manner. For example, if we become suspicious of a file, we become suspicious of other events involving it as well as processes started from it. In this way, clues direct our search for more clues, as the clues we find become more significant in the context in which they appear. Once our search is complete, these clues and their rela tionships can be presented in an easily understood form for further evaluation.

4. Knowledge Representation

4.1. Models

The deduction rules are organized into a variety of different models, each concerned with analyzing a different aspect of the system, our understanding of intrusions, or suspicion itself. All the models operate based on a common set of concepts, and can reason based on each other's deductions as well as their own. In particular, the suspicion module, which contains a variety of generic r ules about how suspicion should be propagated, often interacts with other models, which contain rules about which particular events or inferred actions they find suspicious. Currently most rules are either part of the suspicion module, the system model, w hich is responsible for understanding and making inferences about what certain low level events imply (for example, that an attempt to connect to port 25 is likely an attempt to contact a mail server), or the trojan model, which encapsulates knowledge abou t actions which are commonly seen in worms and trojans.

4.2.1. Deductive Graph

As rules fire in response to observed events, there are two main data structures that are updated and maintained, the deductive graph and the suspicion graph. The deductive gra ph is a directed acyclic graph that includes as nodes all the observed events, as well as the actions that have been inferred from those events, and any other deductions that have been made. The directed edges indicate which nodes were deduced from which other nodes, so that the certainties can be updated as conclusions are confirmed or invalidated. Currently, we are using a very simple scheme for propagating and updating certainties (similar to what was used in EMYCIN [van Melle 84]), since our results s o far do not depend crucially on what sort of updating scheme is used. We plan to substitute a more sophisticated scheme in the future if one proves to be necessary.

4.2.2. Suspicion Graph

The suspicion graph, on the other hand, keeps track of the objects we are suspicious of and their interrelationships. We may be suspicious of individual events, inferred actions, or resources like files, processes and so on. Edges between individual nodes exist when two objects have been deductively linked together. I n the case where we have found no linkages, just a scattering of suspicious events, the graph will be completely disconnected. In most cases, it will have a variety of connected components of varying size. Each node also has a suspicion value which indic ates exactly how intrinsically suspicious the object is (that is, the suspiciousness absent independent confirmation). Currently, we are using a very simple model where objects are either suspicious (value = 1) or very suspicious (value = 5). We tend to pay less attention to small connected components with low overall values, and pay more attention to larger components. We are in the process of evaluating a variety of metrics for measuring the overall "suspiciousness" of the complexes as a whole. One op tion is that the suspiciousness is related to the number of directly observable suspicious events the node is linked to. We've also considered using the sum of the suspiciousness of the entire complex.

4.2.3. Interrelationship

Since the suspicions in the suspicion graph are produced by deductions in the deductive graph, the two structures are interrelated and updates to one can cause updates to the other as well. Newly deduced suspicions are added to the suspicion graph as they are made, but the suspicion graph also influences the deductive graph in the following way: as the suspicion of a node increases due to being linked with other events in a suspicion complex, we proportionally increase the certainty of the conclusion that the object was suspicious, a nd these changes propagate upward through the deductive graph and change the certainties of other inferences. In this way, confirmed suspicions modify our view of what is happening, while unconfirmed suspicions retain their original (usually low) certaint ies.

5. Current Implementation

The current implementation uses the SafeFamily wrapper [Balzer 00] as a sensor in order to watch processes as they execute. As

interesting events are observed, they are forwarded to the Cybersafe analyzer, which is loaded in side of the central control process that interacts with all the wrapped processes on the machine.

5.1. SafeFamily Wrapper

The SafeFamily wrapper is an existing access control mechanism that enforces application specific rules during execution of arbitrary Windows COTS applications and the wrappers already support the ability to simply gather information about resource accesses without enforcing any rules. The resources that can be monitored in this way include all files, registry keys, COM servers, spawning of new processes, and network communications. In addition, the wrappers have been augmented with some Cybersafe specific sensors to detect certain other events of interest.

The SafeFamily wrapper operates entirely in user mode, within the monitored application itself, allowing highly efficient monitoring of all the application's resource requests. In addition, the monitoring mechanism has been hardened and is able to resist attempts to disable or modify the monitoring mechanism even if the application itself has been compromised. All resources are identified by the name given to them by the windows kernel itself, allowing reliable identification of resources even in the presence of alternate or short (DOS) path components or hard links. Sensor information produced by the wrapper is then forwarded to a central process that observes all relevant application behavior on the machine.

5.2. JESS-based analysis engine

The models and maintenance of the graphs are implemented using the Java Expert System Shell (JESS) [Friedman-Hill 03]. Events are translated into a form appropriate for the JESS implementation by the Cybersafe analyzer, which also includes a small component that allows the engine to make native system calls to query certain aspects of the underlying filesystem and operating system.

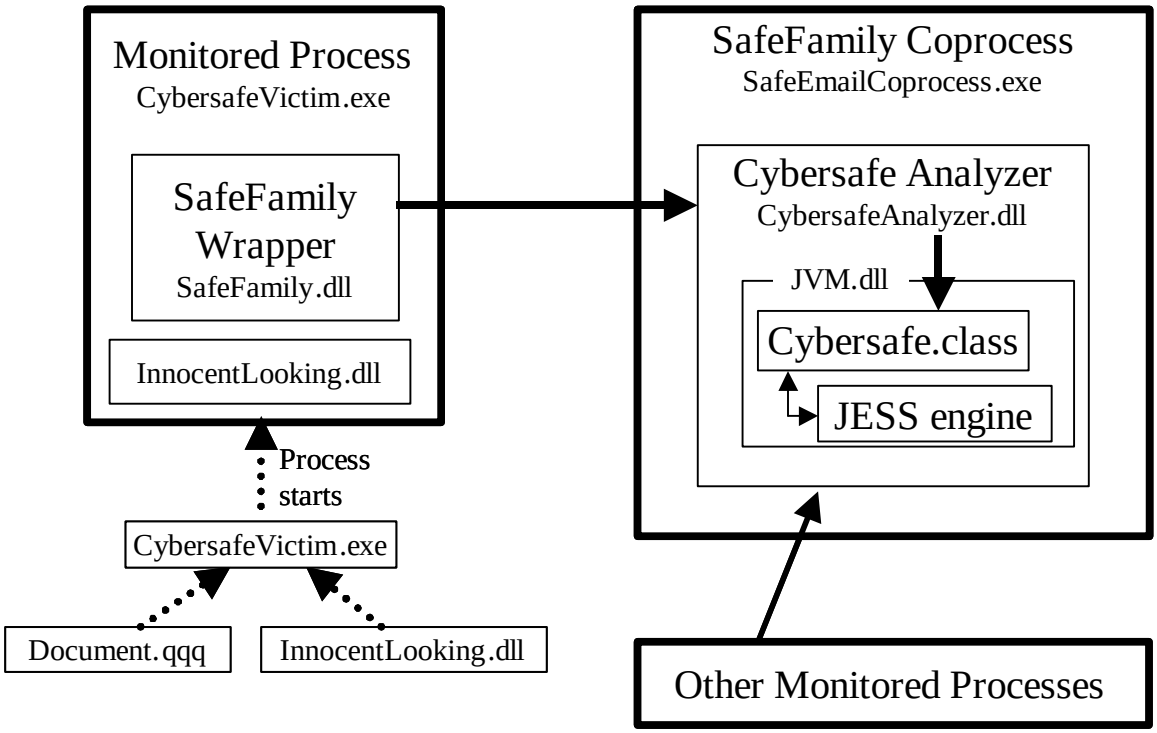

Figure 1: Cybersafe and SafeFamily architecture

6. Example

In order to test our prototype implementation, we built a small example that performs a number of activities of interest, to see how well the analysis engine can deduce their relationships and provide a coherent explanation of what has occurred.

6.1. Description

In our scenario, an enticing looking document exists on a file server somewhere on a corporate intranet. We will refer to this file as *Document.qqq*, where "qqq" is an arbitrary file extension that is mapped to an application we have created. This application (*CybersafeVictim.exe*) will play the role of a COTS or custom application that is used widely within the organization. Unfortunately for any innocent user

who chooses to investigate the file, sitting in the same directory is *InnocentLooking.dll*, which happens to have the same name as a dynamic library *CybersafeVictim.exe* depends upon. Since *CybersafeVictim.exe* attempts to load this particular library using a relative path (a flaw that has been found in a wide variety of applications including Microsoft Word), and the current directory is part of the default library search path on Windows, the malicious version from the file server will be loaded instead of the intended library. Note that under the default settings of Windows, *InnocentLooking.dll* happens to be invisible to the user by virtue of its extension, and we assume it is marked as a hidden file in order to further reduce the chances of it b eing noticed. Once *InnocentLooking.dll* is loaded into *CybersafeVictim.exe* on the victim's machine, we assume it performs the following actions:

(1) creates a new network share on the victim's machine.
(2) copies InnocentLooking.dll and Document.qqq to the new directory.
(3) marks InnocentLooking.dll as hidden.
(4) contacts a remote host in order to inform it that the propagation has succeeded.

This sort of behavior is not atypical for a network trojan that is simply interested in propagating to as many hosts as possible.

6.2. Analysis

During the scenario above, the SafeFamily wrapper observes the events as they happen, and generates the following events that it forwards to the Cybersafe analyzer:

(1) file_library_load_relative
InnocentLooking.dll
\\Server\Share\InnocentLooking.dll
(2) create_network_share
VictimMachine
Share
C:\Share
(3) file_copy
\\Server\Share\InnocentLooking.dll
C:\Share\InnocentLooking.dll
(4) file_hide
C:\ Share\InnocentLooking.dll
(5) communications_access
12.34.56.78 port 80

The Cybersafe analyzer processes these events as they occur. In order to keep the example simple, we will simply be describing the deductions that occur, and will not discuss the certainties attached to the deductions.

In response to the first event, the follo wing rules fire:

(1a) *A file_library_load_relative event that tries to load a library that satisfies certain conditions is marked very suspicious. The file is also very suspicious.*
trojan_dll_1:
?e <- event file_library_load_relative
Abstract_File_Name File
trojan_dll_check(Abstract_File_Name, File, _)
=> suspicious event ?e very_suspicious
=> suspicious file File very_suspicious
(1b) *Files on network shares are foreign.*
file_foreign:
get_drive_type(File, network_share)
=> event foreign_file File
(1c) *Foreign executables are suspicious.*
ext_suspicious_2:
?e <- event foreign_file File
executable(extension(File), 4)
=> suspicious file File suspicious

In this particular case, we have chosen an attack that our sensors can see and that our rules (in partic ular, the rule that fired in 1a) can understand. However, when we evaluate what we have learned at the end, we will also compare our results with what our analysis would have shown if we had been unaware of this particular method of attack.

We also intent ionally chose an attack where the system technically behaves exactly as its designers was intended. Because of this, despite the fact that in this case what we have observed is "highly suspicious", there is no clear cut rule about under what circumstances loading a library via a relative path is strictly illegal; some Windows applications even rely on this behavior in order to function properly. So while this behavior is highly suspect, confirming evidence would be helpful before declaring it malicious. This allows the system to handle situations that are inherently ambiguous, and only become meaningful in context.

The remaining two rules merely encode suspicions about remote executables; even if we did not understand this particular attack vector, we sti ll would be suspicious of the file in question. Though our current rule set does not do so, the rules could be extended to understand the implications of loading suspicious dynamic libraries into a process.

In response to the second event, the following r ule fires:

(2a) *Creating a network share is suspicious.*
trojan_ns_2B:
?e <- event create_network_share NetworkShare
=> suspicious event ?e suspicious

Here, we notice the creation of a new network share, which we also find suspicious. As this is the pare nt directory for the files in several following events, it would be possible to notice a link between these events. However, the current set of rules does not, since the system model does not yet understand parent-child relationships between files and directories.

In response to the third event, the following rules fire:

(3a) *Any foreign file that is copied or moved is still foreign.*
trojan_fs_1:
event copy_file Source_File Destination_File
event foreign_file Source_File
=> event foreign_file Destination_ File

(3b) *Foreign executables are suspicious.*
ext_suspicious_2:
event foreign_file File
executable(extension(File), 4)
=> suspicious file File suspicious

(3c) *Writing an executable to a network share is suspicious. The file being written is also suspicious*
trojan_ns_4:
?e <- event copy_file Source_File
Destination_File
get_drive_type(Destination_File, network_share),
executable(Source_File)
=> suspicious event ?e suspicious
=> suspicious file Source_File

(3d) *Any suspicious file that is copied or moved is suspicious to the same degree.*
forward_propagate_suspicious_file _1:
?e <- event copy_file Source_File,
Destination_File
?s <- suspicious Source_File Suspicious
=> suspicious file Destination_File Suspicious

Two of these rules (3a and 3d) are simply housekeeping rules that propagate certain attributes of files as they are moved around the filesystem. In the case of suspicion, this also creates a link between them as their suspiciousness is causally related. The only other rule we haven't seen before not es that an executable is being written to a network share, which is noted as suspicious in and of itself.

In response to the fourth event, the following rules fire:

(4a) *Any file_hide event is suspicious*
file_hide_1:
?e <- event file_hide File
=> suspicious event ?e suspicious

This is yet another action that in many contexts is perfectly legitimate, but can also indicate that something is going on, especially in the absence of any legitimate reason to expect it. In this case, this gives us yet another ind ependent confirmation of our suspicions of this string of events related to this particular file.

In response to the fifth event, no rules fire since the action appears simply to be a connection to a web server of some sort; with the number of applications that make use of this sort of functionality these days, it would be impractical to warn about this sort of behavior except in very sensitive environments. We include it here because after seeing so many suspicious actions previously, one might be watchin g outgoing connections more carefully than what would normally be the case, and this sort of event might be detected where it would have otherwise gone unnoticed. At the very least, our suspicions about the process could be presented for evaluation before the process was allowed to contact an external machine.

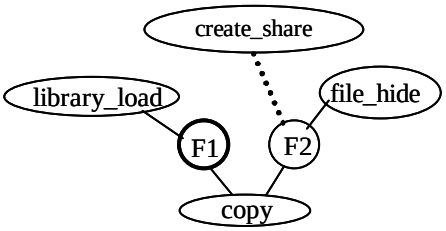

Figure 2: Suspicion Graph

The final suspicion graph is shown in Figure 2. F1 is the file \\Server\Share\InnocentLooking.dll, while F2 is the file C:\Share\InnocentLooking.dll. The remaining four n odes are events 1 through 4. The dotted line indicates a link that could be made, but is not made by the current implementation. The file F1 is considered very suspicious; the rest of the nodes are merely suspicious. The graph clearly shows the context of what has occurred, and the relationships between the various resources and events. Furthermore, the explicit relationships and the reasons for their existence can be clearly explained (see below). Even if the explicit attack had not been detected, the rest of the activity would still have been linked together; only the library_load node would be missing.

6.3. Explanations

The engine can generate an explanation of its results on request. Such a request can be made for any node in either the deduction g raph or the suspicion graph. Explanations are generated in terms of sequences of rule instances that lead to the specified node. For cases where a rule instance associated with a node in the deduction graph refers to a node in the suspicion graph or vice versa, a crossover from one graph to the other will be made. Rule instance sequences will be traced backward from the specified node until nodes representing directly observable events are reached.

Each rule has an explanation template that is used to generate a fragment of the overall explanation. An instance of each explanation template is generated for each instance of the corresponding rule that appears on one of the computed rule sequences. The template is instantiated with the relevant attribute v alues of the actual events that make up the rule instance.

For example, if we asked why F2 is suspicious in the example above, we would be told that "C:\Share\InnocentLooking.dll is suspicious because the process X attempted to hide it" and "C:\Share\InnocentLooking.dll is a copy of the suspicious file \\Server\\InnocentLooking.dll".

7. Conclusion

Based on this example, we believe an explicit concept of suspicion shows promise in assisting a model-based intrusion detection system. In addition, deductive links between our suspicions both help us confirm that our suspicions are correct, and also clarify the relationships between events so that they can be clearly explained to a human analyst, or any other consumer of our conclusions. The separation of the deductive graph and the suspicion graph allows us to use certainties associated only with normal behavior in the deductive graph, and focus exclusively on relationships between potential indications of malicious behavior in the suspicion graph, updating our view of the world only once our suspicions have been confirmed.

References:

[Axelsson 99] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In 6th ACM Conference on computer and communications security, pages 1--7, Kent Ridge Digital Labs, Singapore, 1--4 November 1999.

[Anderson 95] Anderson, Debra, Teresa F. Lunt, Harold Javitz, Ann Tamaru, Alfonso Valdes, "Detecting unusual program behavior using the statistical component of the Nextgeneration Intrusion Detection Expert System (NIDES)", Computer Science Laboratory SRI-CSL 95-06 May 1995.

[Balzer 00] Robert Balzer and Neil Goldman: Mediating Connectors: A Non-ByPassable Process Wrapping Technology, DARPA DISCEX Conference 2000, Hilton Head SC, Jan 25-27, Vol II, pp 361-368.

[Barbara 01] D. Barbara, et al., ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection. SIGMOD Record 2001.

[Eckmann 00] Steve T. Eckmann, Giovanni Vigna, and Richard A. Kemmerer, 2000. "STATL: An Attack Language for State-based Intrusion Detection". Dept. of Computer Science, University of California, Santa Barbara.

[Farmer 90] Daniel Farmer and Eugene H. Spafford. The COPS security checker system. In Proceedings of the Summer Conference, pages 165--190, Berkeley, CA, 1990. Usenix Association.

[Forest 96] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 120--128, Los Alamitos, CA, 1996.

[Friedman-Hill 03] E. Friedman-Hill, "Jess in Action", Manning Publications Co, Greenwich, CT, July 2003.

[Garvey 91] T.D. Garvey and T.F. Lunt. Model-based intrusion detection. In Proceedings of the l4th National Computer Security Conference, October 1991.

[Hofmeyr 98] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. Journal of Computer Security, 6:151--180, 1998.

[Ilgun 95] K. Ilgun, R. Kemmerer, and P. Porras. State Transition Analysis: A RuleBased Intrusion Detection System. IEEE Transactions on Software Engineering, 21(3), Mar. 1995.

[Keppins 03] J. Keppins and J. Zeleznikow, "A Model-Based Reasoning Approach for Generating Plausible Crime Scenarios from Evidence", Proceedings of the Ninth International Conference on Artificial Intelligence and the Law, June 2003, 51-59.

[Kumar 94] Kumar, S. and Spafford, E., "A Pattern Matching Model for Misuse Intrusion Detection," Proceedings of the Seventeenth National Computer Security Conference, pp. 11--21 (Oct. 1994).

[Lee 98] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.

[Lunt 88] T. F. Lunt. Automated audit trail analysis and intrusion detection: A survey. In Proceedings of the 11th National Computer Security Conference, October 1988.

[Nelson 04] N. Murilo and K. Steding-Jessen, chkrootkit. http://www.chkrootkit.org/.

[Porras 97] P.A. Porras and P.G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In Proceedings of the Nineteenth National Computer Security Conference, pages 353--365, Baltimore, Maryland, 22-25 October 1997. NIST/NCSC.

[Ryan 98] Ryan, J. , Lin,M., and Miikkulainen, R. (1998). "Intrusion Detection with Neural Networks". In Advances in Neural Information Processing Systems, vol. 10, MIT Press. 1998.

[van Melle 84] W. van Melle, E. H. Shortliffe and B. G. Buchanan, "EMYCIN: A Knowledge engineer's Tool for Constructing Rule-Based Expert Systems," in Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project," B.G. Buchanan and E.H. Shortliffe (eds), 1984: Addison-Wesley.

[Wagner 02] D. Wagner and P. Soto. Mimicry attacks on host based intrusion detection systems. In Proc. Ninth ACM Conference on Computer and Communications Security, 2002.