A Practical Approach to Modeling Uncertainty in Intrusion Analysis

Xinming Ou
Kansas State University

Raj Rajagopalan
HP Labs

Sakthiyuvaraja Sakthivelmurugan
Kansas State University

Abstract

Uncertainty is an innate feature of intrusion analysis due to the limited views provided by system monitoring tools, including intrusion detection systems (IDS) and the numerous types of logs. Attackers are essentially invisible in cyber space and those monitoring tools can only observe the symptoms produced by malicious activities, mingled with the same effects produced by non-malicious activities. Thus the conclusions one can draw from these observations inevitably suffer from varying degrees of uncertainty, which is the major source of false positives/false negatives in intrusion analysis. This paper presents a practical approach to modeling such uncertainty so that the various security implications from those low-level observations are captured in a simple logical language augmented with certainty tags. We design an automated reasoning process so that the model can combine multiple sources of system monitoring data and identify highly-confident attack traces from the numerous possible interpretations of low-level observations. We develop our model formulation through studying a true intrusion that happened on a campus network, using a Datalog-like language to encode the model and a Prolog system to carry out the reasoning process. Our model and reasoning system can reach the same conclusions the human administrator did regarding which machines were certainly compromised. We then apply the developed model to the Treasure Hunt (TH) data set, which contains large amounts of system monitoring data collected during a live cyber attack exercise in a graduate course taught at University of California, Santa Barbara. Our results show that the reasoning model developed from the true intrusion is effective to the TH data set as well, and our reasoning system can identify high-confidence attack traces automatically. Such a model thus has the potential of codifying the seemingly ad-hoc human reasoning of uncertain events, and can yield useful tools for automated intrusion analysis.

1 Introduction

Intrusion detection is the last line of defense against cyber attacks. However building strong tools to detect intrusions in typical environments has been elusive. At the same time, forensic analysis has become important in the light of regulatory requirements as well as the appearance of sophisticated targeted attacks on enterprise networks. Due to the close relationship between the problems of intrusion detection and computer forensics, we use the term "intrusion analysis" to capture both, namely how to identify attack traces from large amounts of system monitoring data, either on the fly or off line. This problem in general is an inexact science that has to admit a range of uncertainty in output. First, quantifying the results of intrusion sensing in a robust manner has remained a hard problem for a variety of reasons [15]. Human administrators today use a combination of intuition, experience, and low-level tools to create and support positive or negative judgements on security events. Second, while the goal of intrusion analysis is detection of events at a high-level of abstraction (*e.g.*, a machine has been compromised and has been used to compromise others), tools today operate with any reasonable accuracy only at low levels of abstraction (*e.g.*, network packets, server logs, *etc.*). Frequently, sophisticated attacks combine multiple vulnerabilities to achieve their goals, as a result of which we have to combine the outputs of several disparate sensors manually to detect multi-step attacks that are found today. While the low-level observations all have potential implications for attack possibilities, few, if any of them can directly provide zero/one judgment at the high-level abstraction. Nevertheless, in many

network intrusions a relatively small number of such observations *altogether* are sufficient to show that an attack has certainly happened, as well as how it progressed. The difficulty is how to start from uncertain views of the potential problem (*e.g., IDS alerts*) and quickly search for a few log entries or events among millions so that the attacker's hand is clearly shown. System administrators are highly time-constrained. An automatic tool that can sift through the ocean of uncertainty to quickly and accurately locate the problem areas will be highly valuable in practice.

1.1 A true-life incident

Consider the following sequence of events that actually occurred recently at a university campus. The System Administrator (SA) noticed an abnormally large spike in campus-network traffic (*Observation 1*). SA took the netflow dump for that time period and ran a packet capture tool on it to search for known malicious IP addresses and identified that four Trend Micro (anti-malware) servers had initiated IRC connections to some known BotNet controllers (*Observation 2*). The SA suspected that the four Trend Micro servers had been compromised. He sat down at one of the server consoles and dumped the memory, from which he found what appeared to be malicious code modules (*Observation 3*). He also looked at the open TCP socket connections and noticed that the server had been connecting to some other Trend Micro servers on campus through the IRC channel (*Observation 4*). He concluded that all those servers were compromised with a zero-day vulnerability in the Trend Micro server. He did the same for the other identified servers and found even more compromised servers. Altogether he identified 12 compromised machines and took them off line.

When the administrator first noticed the spike in network traffic, the questions facing him was: is the network experiencing an attack and if so, which machines were compromised. However, none of the low-level observations alone can give a definitive answer to these high-level questions. Observation 1 (traffic spike) could mean many things, many of which benign[1]. Observation 2 (connections to BotNet controllers) has a higher chance of being malicious activity and hence a higher degree of likelihood that the identified servers are compromised. However, an IRC connection being made to a known BotNet controller does not necessarily mean that the machine has been compromised. The list of "known" BotNet controllers may contain false positives or it could also be that somebody was probing BotNet controllers for research purposes. (The interviewed SA suggested this false positive as he does this himself on a periodic basis.) Observation 3 (suspicious code in memory) is also a strong indication that the machine may have been controlled by an attacker. But it is not always easy to determine whether a suspicious module found in the memory dump is indeed malicious, especially with zero-day vulnerabilities. So this alert also contains some amount of false positive. Observation 4, like observation 2, cannot definitively prove that the machines observed are under the control of attackers because IRC channels are occasionally (rarely) used as a communication channel between servers. However, when we put all the four pieces of evidence together, it seems clear that an attack has certainly happened and succeeded and we can tell with almost certainty which machines have been compromised.

The main question we need to ask is: "are the security tools we have today good enough to detect these attacks?" If not, given that the SA spent many man-hours of critical down time trying to analyze the problem, can our tools at least help reduce the amount of time that the SA had to spend in the process? Unfortunately, it is the case that the SA had all the common security tools at his disposal and yet he had to spend an inordinate amount of time in discovering the compromised machines[2]. This true-life incident surfaces several challenges. First and foremost, there is a great deal of uncertainty surrounding the question of whether a machine has in fact been attacked, and if so, successfully breached and taken control of. It would be hard to definitively answer these questions, especially the latter, unless incontrovertible evidence

[1] To take an example, just some users started downloading movies through BitTorrent.

[2] The described campus network chose not to have any network IDS deployed, the reason being that there were too many false-positives and there was no resource for analyzing the generated alerts. In this case even a well maintained IDS probably would not have helped since the attack exploited a zero-day vulnerability.

is found in the logs in the form of explicit malicious activity emanating from this server, which is rare in practice. Thus, human analysts make their conclusions typically with shades of uncertainty. The SA in the case study did not have the right tools to handle the many hypotheses that were made and discarded about what might have happened in his network before reaching conclusions. While approaches for expressing the confidence levels have been proposed for IDS based on signal theory [1], we do not yet have a natural but robust language that human experts can use to codify their knowledge and works well in the IDS context [16]. Second, sophisticated attacks that succeed may not be directly detected using standard sensors and filters such as Snort but have to be inferred from the available data using models of systems and attacks and by combining data from a variety of sources. However, such models are never complete or totally accurate, and the data sources come with a variety of assumptions and caveats, thereby decreasing our level of confidence in the detection-and-inference process substantially. Typically IDS tools today that can detect sophisticated attacks cannot or do not distinguish between almost certain inferences and remote possibilities. We need tools that allow us to build these models and indicate the current level of confidence in them so that we can construct potentially complicated evidential paths along with the corresponding confidence levels in the conclusions. Finally, it has been difficult if not impossible to assign any probability structures to computer system events for intrusion analysis purposes. We do not have statistical data either for occurrence or detection rates for attacks analogous to those found in other areas such as failure rates in reliability analytics. Even for the sensors for which such characteristics as true/false positives/negative rates can be measured, the sensor characteristics are far from desirable from a signal theory point of view [1]. Recent theoretical advances [9] have shown how to combine the outputs of noisy low-level sensors in a way that optimizes the combined rate of fidelity for low-level conclusions. Yet it is not clear how to apply these in practice. Thus, while from a theoretical perspective it would be highly desirable to compute a rigorous quantitative confidence level for our final high-level conclusions, we are far from such a capability in practice.

1.2 Modeling uncertainty

We believe the key step in tackling the above-mentioned challenges is to develop a model that can link the low-level observations to the conditions under concern at the high level and at the same time allow us to track the confidence in the linkage. For example, an `abnormal high network traffic` (low-level observation) could mean that `an attacker is performing some network activity` (high-level condition). Similarly, the `netflow dump showing a host communicating with known BotNet controllers` (low-level observation) indicates that it is likely `an attacker has compromised the host` (high-level condition). All these assertions are associated with varying degrees of uncertainty. For example, compared with the anomalous high network traffic, a netflow filter that shows communication with known BotNet controllers is a more confident assertion on attacker activity. It is crucial that the model for linking the low-level events and high-level conditions are capable of expressing such differences, and it is desirable that this be done without relying on probability parameters which are difficult to obtain. Such a model should also express the logical relations between the various high-level conditions, so that such knowledge can be mapped to correlate low-level events. For example, the model should include knowledge like `after an attacker compromised a machine, he may possibly perform some network activity from the machine`. This knowledge can 'potentially reveal hidden correlations between low-level observations, (*e.g.*, high network traffic and netflow filtering result). Without other context to guide us, a traffic spike could be due to any of a number of things but in the context of a likely compromise the parameters of the traffic burst become important — if the traffic emanated from the compromised machine it can be assigned a different meaning than if it did not.

1.3 Our contributions

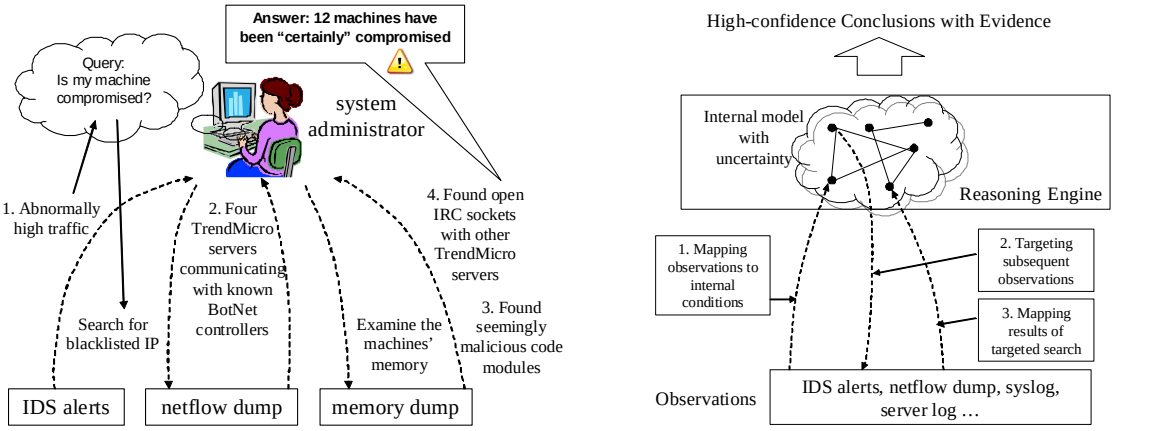
We present a model for capturing the meanings of low-level system observation data in terms of high-level conditions of interest to intrusion analysis. We use qualitative, rather than quantitative assessment to capture the uncertainty inherent in such meanings. The qualitative assessment makes it easier for our model to be linked to existing knowledge bases such as the Snort rule repository. Our model is capable of expressing logical connections among the high-level conditions (also with qualitative uncertainty assessment), so that it can reason about multi-host, multi-stage intrusions with traces spread across various types of system monitoring data. We present a reasoning process that can utilize such a model and existing IDS tools to automatically identify highly confident attack traces from large diverse sets of system monitoring data, not restricted to just IDS alerts. We also present within this model a method for *strengthening* the confidence in an assertion by combining different independent pieces of evidence of low or moderate confidence. The central part of our reasoning process is an "internal model" which represents the various internal conditions of interest applying to individual hosts or groups of hosts (networks, etc). This internal model contains generic conditions such as "under_attack," "compromised," "bot-netted," *etc.* that are independent of the scenario at hand. What can change from one scenario to another are the certainty tags associated with each condition as the scenario events are processed and the paths that the reasoning process takes through these conditions. We believe that human administrators similarly have a small set of "target" conditions in mind when they process intrusion data and there is value in capturing those target conditions directly in the automated reasoning process. We implemented a prototype of this tool using the true-life case study as a guide and showed that the tool's reasoning tracked the SA's reasoning process and achieved the same set of high-level conclusions with high confidence. To evaluate our tool, we applied it to the "Treasure Hunt" dataset [27] that contains traces of multi-stage attacks on operating systems services, web servers, and database servers, within about a hundred megabytes of diverse data sources including IDS sources such as TCP dump and Snare alerts but also general logs such as syslog and apache logs. To test the efficacy of our concepts, we kept our internal model unchanged and simply applied the tool to the modified data sources in the dataset. We found remarkably that our tool discovered most of the high-level attacks in the data (that we know of) and also discovered some subtle but minor gaps in our model in the process.

2 Informal description of the model and reasoning process

Before describing our model and reasoning process in formal details, we motivate our design using the examples in the case study to represent the analytic states the SA went through in the course of the investigation. Using the example of the case study, we identify the rationale behind the decisions at various points, and design a logic that captures this reasoning process.

2.1 Case study revisited

The scenario described in Section 1.1 is illustrated in Figure 1(1). Figure 1(2) presents our vision of automating reasoning about uncertainty in intrusion analysis. The reasoning framework consists of two layers: observations and an internal reasoning model. The observations are from system monitoring data such as IDS alerts, netflow dumps, syslog, *etc.* and are mapped into the internal reasoning model as conditions representing unobservable security status under interest. For example, `abnormal high traffic` is an observation, and `an attacker is performing some network activity` is an internal condition. Logical relations exist between observations and internal conditions with varying degrees of certainty. For example, we can say `abnormal high traffic indicates the possibility that an attacker is performing some network activity`. Another example: `netflow dump showing a host communicating with known BotNet controllers indicates that an attacker likely has compromised the host.` Likewise, there are logical relations among the various internal conditions and these relations contain varying degrees of certainty as well. For example, one can say `after an attacker compromised a machine,`



(1) Case scenario

(2) Vision of applying a logic with uncertainty for intrusion analysis

Figure 1: Case study and vision for automated intrusion analysis

`he may possibly perform some network activity from the machine`, and `after an attacker sends an exploit to a machine, he will likely compromise the machine`. These statements capture the rationale behind human reasoning when manually detecting intrusions from the logs. The internal model is analogous to human thinking — observations are reflected as beliefs with varying strengths, and the beliefs are related to one another with varying strengths as well. We design logical rules to capture both aspects and a formalized reasoning process to simulate human thinking such that an automated inference process can allow us to construct sophisticated attack conclusions along with a semi-quantitative measure of our confidence. This inference process is capable of deriving, from a large number of possibilities, high-confidence beliefs corroborated by a number of complementary evidences logically linked together. For example, even though the SA was not sure whether the abnormal high traffic really indicated an attack, he knew that this observation is logically linked to network activity, which can also be observed by netflow dump. When from the netflow dump the Trend Micro servers were shown to communicate with malicious IP addresses, and from the memory dump a potentially malicious code module was found, the two pieces of evidence both indicated that the server was likely compromised, thus strengthening the belief's confidence to almost certain.

A side product of this reasoning process is the downward arrow shown in Figure 1(2), which represents the "top-down" targeted search guided by the result of the current reasoning process. The volumes of data collection in enterprise networks have reached staggering proportions. A single site can collect Gigabytes of log information per day and searching there for a single attack event that may be poorly described in such enormous collections has been likened to "searching for a needle in a haystack of needles"[22]. A simple approach of scanning all the data first to build attack hypotheses at a low level is unlikely to scale — we need analytic methods that find all the attacks hidden in the data with reasonable confidence without having to trawl the entire data.

Reasoning framework Figure 2 presents the architecture of our reasoning system. The reasoning model consists of two modules — observation correspondence (described in Section 3.2) and internal model (described in Section 3.3). Roughly speaking, observation correspondence maps low-level observations into predicates in the internal model (the up arrows in Figure 1(2)), and the internal model encodes the logical connections among internal conditions. Both modules in the reasoning model are specified in Datalog [3], a simple logic-programming language that has efficient polynomial-time execution. The raw observations are pre-processed and the distilled results are converted to Datalog tuples as input to the reasoning system.

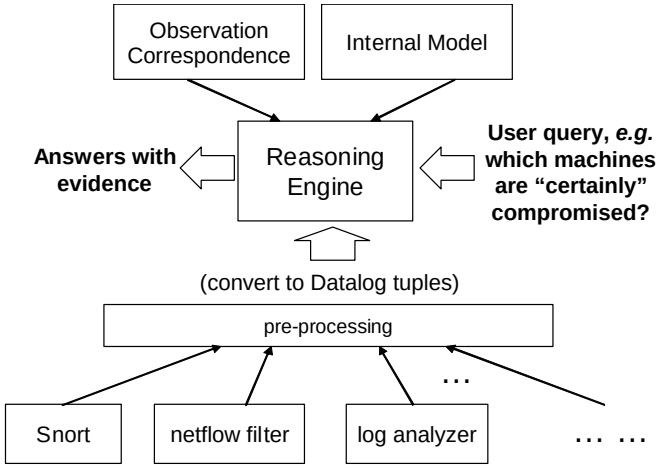


Figure 2: System architecture

The reasoning engine is implemented in Prolog and the reasoning rules are specified in Section 3.4. We use the XSB [23] Prolog system to run our reasoning engine. An important feature of our design is that every component of the system is specified declaratively, which has the useful property that once all specifications are loaded into the Prolog system, a simple Prolog query will automatically, and efficiently, search for true answers based on the logic specification. For example, a user can ask a question "which machines are certainly compromised?" in the form of a simple Prolog query. Our reasoning engine will then give the answer along with the evidence in the form of logical proofs.

Section 3 contains a formal description of our logic and reasoning framework. In Section 4 we describe our experience with applying our tool to the Treasure Hunt dataset.

3 Formal description of the logic

In this section, we formally describe our "logic with uncertainty" which is the core of our approach. As outlined in the previous section, we use the logic to convert sensed events into internal conditions which capture the semantics of the observations imbued with some certainty tags. A proof strengthening technique is introduced to derive high-confidence conclusions from such imperfect input, using an internal model that captures the logical relations (also with uncertainty) among the various internal conditions.

3.1 Notations

We use three modes p, l, c, standing for "possible, likely, certain" to express low, moderate, and high confidence levels. Even though one could think of certainty level as lying in a continuous spectrum between completely unknown to completely certain, we found that human SA's only deal with a few confidence levels in practice that roughly correspond to the ones defined here. It would become disproportionately harder to mentally keep track of a large number of certainty levels and these three levels seem to be adequate for most scenarios. We emphasize that these uncertainty levels are done by hand and apart from the obvious ordering we are not ascribing a probability range to each level.

We identify two types of logical assertions: *observation correspondence* to map external observations to internal conditions, and *internal model* to capture relationships between internal conditions. Correspondingly, we use *obs*(O) to denote a fact about observation O, and *int*(F) to denote an internal condition F. For example, *obs*(*netflowBlackListFilter*$(172.16.9.20, 129.7.10.5)$) is an observation from the netflow blacklist filter that `machine 172.16.9.20 is communicating with a known malicious IP 129.7.10.5`, whereas *int*(*compromised*$(172.16.9.20)$) is an internal condition that `172.16.9.20 has been compromised`. See

A_1 : *obs*(*anomalyHighTraffic*) $\overset{p}{\longmapsto}$ *int*(*attackerNetActivity*)

A_2 : *obs*(*netflowBlackListFilter*$(H, \textit{BlackListedIP})$) $\overset{l}{\longmapsto}$ *int*(*attackerNetActivity*)

A_3 : *obs*(*netflowBlackListFilter*$(H, \textit{BlackListedIP})$) $\overset{l}{\longmapsto}$ *int*(*compromised*(H))

A_4 : *obs*(*memoryDumpMaliciousCode(H)*) $\overset{l}{\longmapsto}$ *int*(*compromised*(H))

A_5 : *obs*(*memoryDumpIRCSocket*(H_1, H_2)) $\overset{l}{\longmapsto}$ *int*(*exchangeCtlMessage*(H_1, H_2))

Figure 3: Observation correspondence

I_1 : *int*(*compromised*(H_1)) $\xrightarrow{pc}$ *int*(*probeOtherMachine*(H_1, H_2))

I_2 : *int*(*compromised*(H_1)) $\xrightarrow{pc}$ *int*(*sendExploit*(H_1, H_2))

I_3 : *int*(*sendExploit*(H_1, H_2)) $\xrightarrow{lp}$ *int*(*compromised*(H_2))

I_4 : *int*(*compromised*(H_1)), *int*(*compromised*(H_2)) $\xrightarrow{pc}$ *int*(*exchangeCtlMessage*(H_1, H_2))

Figure 4: Internal model

Figures 3 and 4 for examples of these two types of predicates from the case study.

3.2 Observation correspondence

Observation correspondence gives a "meaning" to an observation, in the form of internal conditions. In A_1 an abnormal high network traffic *obs*(*anomalyHighTraffic*) is mapped to *int*(*attackerNetActivity*), meaning an attacker is performing some network activity. This is a low-confidence judgment thus the mode is p. Intuitively the p mode means there is other equally possible interpretations for the same observation. A_2 and A_3 give the meaning to an alert identified in a netflow analysis. There are a number of filtering tools available that can search for potential malicious patterns in a netflow dump, such as "capture daemon" and "flow-nfilter". This rule deals with one filter that identifies communication with known malicious IP addresses. Since any such activity is a strong indication of attacker activity and the fact that the machine involved has been compromised, the modality of the two rules is l. There are still other possibilities, e.g. the communication could be issued by a legitimate user who wants to find out something about the malicious IP address. But the likelihood of that is significantly lower than what is represented by the right-hand side of the two rules. A_4 says if memory dump on machine H identifies malicious code, then H is likely to be compromised. A_5 says if the memory dump identifies open IRC sockets between machine H_1 and H_2, then it is likely that the IRC channel is used to exchange control messages between BotNet members.

We recognize that these observation correspondence assertions are not objective. A common problem in intrusion detection is the lack of ground truth which makes it very hard to have objective classification of all events. Our goal is to create a flexible and lightweight framework wherein an SA can feed in these beliefs of certainty and see what consequences arise. For example, an SA may think the mode of A_4 ought to be c, which would be acceptable. One advantage of such a logic is that it facilitates discussion and sharing of security knowledge. Empirical experiences from a large community can help tune the modes in those assertions. We envision a rule repository model like that for Snort, where a community of participants contribute and agree upon a set of rules in an open language. Currently there are only coarse-grained classification and some natural-language explanations for the meanings behind each Snort alert. In Section 4, we show how a small number of internal-model predicates can give a meaning to the vast majority of Snort alerts. The Snort rule writers can use the observation correspondence assertions to encode the possible implications of Snort alerts with levels of certainty, which can then be reasoned about automatically in our logic.

3.3 Internal model

Once the observation correspondence has mapped external observations to internal conditions, we are now ready to relate these internal conditions to other internal conditions while also keeping track of the degree of certainty associated with them. To represent the logical relationship between internal conditions in our model, we use the $\stackrel{m_1 m_2}{\longrightarrow}$ operator, which represents the "leads to" relationship between internal conditions. The condition on the right-hand side of the operator is caused by the left-hand side and as a result the arrow must be aligned with time, *i.e.* it must happen no earlier than the left-hand side. However, the *reasoning* can go along both directions, and hence two modes (m_1, m_2) are associated with each rule. For example, I_1 says "if an attacker has compromised machine H_1, he can perform a malicious probe from H_1 to another machine H_2." The forward reasoning has a low certainty: the attacker may or may not choose to probe another machine after compromising one. Thus the forward reasoning is qualified by the p mode. Reasoning on the reverse direction however has the c mode: if an attacker performs a malicious probe from one machine to another, he must have already compromised the first machine.[3] Likewise, I_2 says "if an attacker has compromised machine H_1, he can send an exploit from H_1 to another machine H_2." For the same reason, the forward reasoning has the p mode and the backward reasoning has the c mode. In I_3, the fact that an attacker sends an exploit to a machine leads to the compromise of the machine. The forward reasoning is also not certain, since the exploit may fail to execute on the target host. We have used the confidence level l – attackers typically make sure their exploit will likely work before using it in an attack. In the other direction, sending in a remote exploit is just one of many possible ways to compromise a host. Thus the reverse reasoning for I_3 has the p mode. I_4 is the only assertion in this model that has two facts on the left-hand side. Like in typical logic-programming languages, the comma represents the AND logical relation. The forward direction has the p mode: if an attacker compromised two machines H_1, H_2, the two machines can possibly exchange BotNet control messages between them to coordinate further attacks. The backward direction has the c mode: if two machines are exchanging BotNet control messages, both of them must have been compromised.

3.4 Simple reasoning rules

As a result of combining the observation correspondence with the internal model we can derive many useful assertions with different levels of certainty. To describe this reasoning process we use $\mathit{int}(F, m) \Leftarrow \mathit{Pf}$ to represent that "the internal fact F is true with modality m", and *Pf* is the proof which shows the derivation steps in the logic arriving at the conclusion. We have designed declarative rules that simulate human reasoning with uncertainty. Many but not all of the rules are simple and straightforward. This section explains the simple rules and the next section explains the proof-strengthening rule which is the core of handling uncertainty.

From an observation one can derive an internal belief with some degree of certainty, based on the observation correspondence assertion. This is captured in the following rule called "obsMap".

$$\frac{\mathit{obs}(O) \quad \mathit{obs}(O) \stackrel{m}{\longmapsto} \mathit{int}(F)}{\mathit{int}(F, m) \Leftarrow \mathit{obsMap}(\mathit{obs}(O))} \text{ obsMap}$$

As an example of an application of this rule, the open IRCSocket identified through memory dump in the case study will be an input to our system: $\mathit{obs}(\mathit{memoryDumpIRCSocket(172.16.9.20,172.16.9.1)})$. Together with observation correspondence A_5, the above reasoning rule will derive:

$$\mathit{int}(\mathit{exchangeCtlMessage('172.16.9.20','172.16.9.1')}, l) \Leftarrow \mathit{obsMap}(\mathit{obs}(\mathit{memoryDumpIRCSocket('172.16.9.20','172.16.9.1')}))$$

[3] The predicate *probeOtherMachine* specifically means the malicious probing performed by an attacker and does not include benign probing from non-malicious parties.

The following two derivation rules capture the reasoning induced by each internal-model predicate.

$$\frac{\textit{int}(F, m) \Leftarrow \textit{Pf} \quad \textit{int}(F) \xrightarrow{m_1 m_2} \textit{int}(F') \quad m' = m \cup m_1}{\textit{int}(F', m') \Leftarrow \textit{intL}(\textit{int}(F, m) \Leftarrow \textit{Pf})} \text{ intL}$$

$$\frac{\textit{int}(F, m) \Leftarrow \textit{Pf} \quad \textit{int}(F') \xrightarrow{m_1 m_2} \textit{int}(F) \quad m' = m \cup m_2}{\textit{int}(F', m') \Leftarrow \textit{intR}(\textit{int}(F, m) \Leftarrow \textit{Pf})} \text{ intR}$$

We use $m_1 \cup m_2$ to denote the "join" relation between modes, which takes the lesser degree of certainty from the two. For example, $l \cup p = p$; $l \cup c = l$; and so on. The intuition is that the confidence level of a fact arising from a chain of reasoning is the confidence level of the weakest link in the chain. As an example, the internal fact *int*(*exchangeCtlMessage(172.16.9.20,172.16.9.1)*, l) derived above, together with the internal model predicate I_3, would yield the following derivation trace.

int(*compromised(172.16.9.20)*, l) $\Leftarrow$
int(*exchangeCtlMessage(172.16.9.20,172.16.9.1)*, l) $\Leftarrow$
obsMap(*obs*(*memoryDumpIRCSocket(172.16.9.20, 172.16.9.1)*))

Since the backward reasoning mode for I_3 is c, joined with the l mode in *int(exchangeCtlMessage(172.16.9.20, 172.16.9.1), l)*, we get $c \cup l = l$ as the mode for the resulting fact *int*(*compromised(172.16.9.20)*, l).

3.5 Proof strengthening

As can be easily seen, the simple reasoning rules above can only decrease the confidence level of a fact. The key purpose of reasoning about uncertainty is to derive high-confidence fact from low-confidence ones. In the case study, the system administrator strengthened his belief that the Trend Micro server was compromised by combining three pieces of evidence: netflow filter result showing communication with a blacklisted IP address, memory dump result showing likely malicious code modules, and memory dump result showing open IRC sockets with other Trend Micro servers. These three pieces of evidence are *independent* — they are rooted on observations at different aspects of the system, and yet they are *logically connected* — all of them indicate that the Trend Micro server is likely compromised. Thus they altogether can strengthen our belief in the fact that the server is compromised. We generalize this reasoning process in the following proof strengthening rule.

$$\frac{\textit{int}(F, m_1) \Leftarrow \textit{Pf}_1 \quad \textit{int}(F, m_2) \Leftarrow \textit{Pf}_2 \quad \textit{Pf}_1 \parallel \textit{Pf}_2}{\textit{int}(F, \textit{strengthen}(m_1, m_2)) \Leftarrow \textit{strengthenedPf}(\textit{Pf}_1, \textit{Pf}_2)} \text{ strengthenedPf}$$

The $\parallel$ relation indicates that two proofs are independent — based on disjoint sets of observations and internal conditions. This deduction rule states that if we have two reasoning paths to a fact with some confidence levels, and if the two paths are based on independent observations and deductions, then the confidence level of the fact will be strengthened. The *strengthen* function is defined below.

$$\textit{strengthen}(l, l) = c \quad \textit{strengthen}(l, p) = c \quad \textit{strengthen}(p, l) = c$$

Simply put, two independent proofs can strengthen to "certain" if at least one of them can yield a "likely" mode. There is no definition for *strengthen* when both parameters are p or at least one of them is c. Since the p mode represents very low confidence we do not allow strengthening from just possible facts. There is no need to strengthen a fact if it is already proved to be certain.

3.6 Implementation

All the above reasoning rules can be straightforwardly specified in Prolog. We also implemented a simple proof-generator so that when a fact is derived, the proof trace can also be displayed. We applied the reasoning system and the model described in 3.2 and 3.3 on the input for our case study. The result is shown in

```
| ?- show_trace(int(compromised(H),c)).
int(compromised('172.16.9.20'),c) strengthenedPf
    int(compromised('172.16.9.20'),l) intRule_4
        int(exchangeCtlMessage('172.16.9.20','172.16.9.1'),l) obsRule_5
            obs(memoryDumpIRCSocket('172.16.9.20','172.16.9.1'))
    int(compromised('172.16.9.20'),l) obsRule_4
        obs(memoryDumpMaliciousCode('172.16.9.20'))
    int(compromised('172.16.9.20'),l) obsRule_3
        obs(netflowBlackListFilter('172.16.9.20','129.7.10.5'))

int(compromised('172.16.9.1'),c) strengthenedPf
(1)    int(compromised('172.16.9.1'),l) intRule_4
          int(exchangeCtlMessage('172.16.9.20','172.16.9.1'),l) obsRule_5
              obs(memoryDumpIRCSocket('172.16.9.20','172.16.9.1'))
(2)    int(compromised('172.16.9.1'),p) intRule_2
          int(sendExploit('172.16.9.1','172.16.9.20'),p) intRule_3
              int(compromised('172.16.9.20'),l) obsRule_4
                  obs(memoryDumpMaliciousCode('172.16.9.20'))
(3)    int(compromised('172.16.9.1'),p) intRule_2
          int(sendExploit('172.16.9.1','172.16.9.20'),p) intRule_3
              int(compromised('172.16.9.20'),l) obsRule_3
                  obs(netflowBlackListFilter('172.16.9.20','129.7.10.5'))
(4)    int(compromised('172.16.9.1'),p) intRule_3
          int(sendExploit('172.16.9.20','172.16.9.1'),p) intRule_2
              int(compromised('172.16.9.20'),l) obsRule_4
                  obs(memoryDumpMaliciousCode('172.16.9.20'))
(5)    int(compromised('172.16.9.1'),p) intRule_3
          int(sendExploit('172.16.9.20','172.16.9.1'),p) intRule_2
              int(compromised('172.16.9.20'),l) obsRule_3
                  obs(netflowBlackListFilter('172.16.9.20','129.7.10.5'))
```

Figure 5: Result of applying the reasoning system on the case study

Figure 5 (we annotated the second derivation trace with some numbers for presentation purposes). The user enters the query `show_trace(int(compromised(H),c))`, to find out all the provable facts in the form of `compromised(H)` with "certain" mode. This is essentially asking the question "which machines are certainly compromised". The reasoning engine prints out two derivation traces. The first one is for `172.16.9.20`, the IP address (fake) for the compromised Trend Micro server first identified by the SA. The second one is for `172.16.9.1`, another Trend Micro server with which the first one maintains an open IRC socket. It is clear that the first derivation trace exactly matches the reasoning process the human SA did to identify the compromised server — the confidence level is strengthened from concordant evidence emanated from netflow dump and memory dump. The second derivation trace is more interesting. The fact's "certain" mode is also strengthened from a number of evidence traces. The trace marked by (1) is based on the fact that the machine maintains an IRC connection with another machine. This by itself does not provide high-certainty evidence that the host has been compromised. But the other machine (`172.16.9.20`) is also likely compromised (third line on each of (2) – (5)) from other independent observations. (2) and (4) are based on the memory dump result; whereas (3) and (5) are based on the netflow dump. (2) and (3) reflects the possibility that `172.16.9.1` is used as a stepping stone to compromise `172.16.9.20`; whereas (4) and (5) reflects the possibility that `172.16.9.20` is a stepping stone to compromise `172.16.9.1`. Each can only derive a "possible" mode for the fact that `172.16.9.1` is compromised (first line on each of (2) – (5)). However, combined with the evidence in (1) the fact's mode is strengthened to "certain". This exactly matches the rationale behind the human reasoning.

This case study also demonstrates that our internal model is compact yet powerful enough to handle complicated reasoning in intrusion analysis. We also observe that all the simple reasoning rules in our logic

can be encoded as Datalog programs, and the proof-strengthening rule can be easily implemented with a Datalog proof generator. Datalog has polynomial time complexity and can be efficiently executed in the XSB Prolog system. Indeed, it has been shown that an attack-graph generator based on Datalog evaluation can scale to enterprise networks with thousands of machines [20, 21].

4 Experiments and Results

Evaluation of intrusion detection systems has always been difficult due to the lack of data from a large variety of scenarios and settings. Performance result on a single data set in the form of false positive/negative numbers can hardly be persuasive, since one can always "fit" a tool to work well for a particular data set. Indeed, it has been pointed out that by just looking at the TTL field of the packets in the MIT Lincoln Lab data set [11] one can distinguish between malicious packets and non-malicious packets, due to the way the data was generated [2, 14]. In this section we describe our effort of experimenting our reasoning model from a different perspective. Instead of trying to obtain the false positive and false negative numbers as is typical of IDS research, we would like to discover whether our reasoning model developed from studying the true-life incident can be applied to a completely different data set and find interesting attack traces. We use the Treasure Hunt (TH) data set [27] (`http://www.cs.ucsb.edu/~vigna/treasurehunt/`), which was created as a part of the competition organized in a graduate security course at University of California, Santa Barbara. Our motivation to use this particular data set was the large amounts of diverse system monitoring data in the forms of TCPdump, Syslogs, Snare logs, Apache server logs, kernel audit logs, *etc*. Moreover, the data set provides the valuable "meta data" such as the back story (competition task details) and network topology which can help us understand the result (The network topology and task details are provided for reference in Appendix 7). We would like to see how easy it is to apply our unmodified reasoning system to this data set.

4.1 Data pre-processing

We applied Snort to the TCPdump to generate IDS alerts, which were sent to the *Prelude Manager* (a program for collecting, managing and storing alerts from IDS sensors). By using *Prelude-Correlator* the alerts were clustered based on Snort rule ID's as well as source and destination IP addresses. We then hand-coded the clustered alerts into Datalog format understood by our reasoning system, some samples shown below. (The Time field is not used since our logic currently does not handle time stamps)

```
/* obs(snort(ID, FromHost, ToHost, Time)). */
/* Web-Misc guesbook.pl access from external IP to WWW server */
  obs(snort('1:1140', '128.111.49.46', '192.168.10.90', _)).
  obs(snort('1:1140', '128.111.49.137', '192.168.10.90', _)).

/* WEB-MISC /etc/passwd access from external IP to WWW server */
  obs(snort('1:1122',  '128.111.49.47', '192.168.10.90', _)).
```

Observation correspondence We already have the internal model and reasoning engine developed from the case study, but we do not yet have the observation correspondence assertions for the Snort alerts generated from the TH data set. As a rationale for writing the assertions, we made use of the Snort rule and the natural-language description available at the Snort rule repository as well information that can be found at Bugtraq, *etc*. For example, the Snort rule for "WEB-MISC guestbook.pl access" alert is

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC guestbook.pl access"; flow:to_server,established;
uricontent:"/guestbook.pl"; nocase; metadata:service http;
classtype:attempted-recon; sid:1140;  rev:12;)
```

This rule alerts for all the web request containing "/guestbook.pl" in the URI. The observation correspondence for the alert is

```
/* Web-misc guestbook.pl access */
obsMap(obsRule_th1, obs(snort('1:1140', FromHost, ToHost, _Time)),
                    int(sendExploit(FromHost, ToHost)), p).
```

The assertion is mapped to `sendExploit(FromHost, ToHost)` because the vulnerable "guestbook.pl" script can be used for *remote code execution* and has a *possible* mode because the rule alerts for legitimate access also. Similarly, for the Snort alert "WEB-MISC Apache Chunked-Encoding transfer attempt" (`http://www.snort.org/pub-bin/sigs.cgi?sid=1:1807`) the description and observation correspondence are given as

> This event is generated when an attempt is made to exploit a known vulnerability on a web server or a web application resident on a web server.

```
/* Apache chunked encoding transfer attempt */
obsMap(obsRule_th5, obs(snort('1:1807', FromHost, ToHost, _Time)),
                    int(sendExploit(FromHost, ToHost)), c).
```

The observation correspondence has a "certain" mode because as an exploit rather than a feature it is used only by an attacker.

4.2 Applying the reasoning engine

It is important to note that we use the same internal model and reasoning system developed from the case study. As in the case study, the main goal of the analysis is achieved by issuing a Prolog query to find out the hosts that are *certainly* compromised. Figure 6 shows the partial results of running this query on the (prepared) TH data. The numbers in parentheses have been added by hand to aid in discussion of the output. (1) shows that host `192.168.10.90` (WWW server) was *certainly* compromised. The output also lists the facts that were used by the reasoning process to strengthen this top-level goal to certain mode. (2) is the "TCP portscan" alert for probing activity from WWW server to the file server, which happened after WWW server was attacked. (3) is the "Apache Chunked-Encoding transfer attempt" exploit sent to the WWW server. The *guestbook.pl* access which can give the attacker unauthorized access and possibly escalated privileges by remote code execution on the WWW server is shown in (4) tagged as *possible* attack. This set of proofs is sufficient for the reasoning system to strengthen the internal model `compromised(192.168.10.90)` (WWW server) to *certain* as indicated by the strengthening rule in section 3.5.

The reasoning system also used heuristic functions to aid in searching for evidences of attack. One of the heuristic function used is given in Figure 7(a). This heuristic was used to search for *guestbook.pl* pattern

```
| ?- show_trace(int(compromised(H), c)).

(1) int(compromised('192.168.10.90'),c) strengthenedPf
(2)        int(compromised('192.168.10.90'),l) intRule_1
                int(probeOtherMachine('192.168.10.90','192.168.70.49'),l) obsRule_th6
                        obs(snort('122:1','192.168.10.90','192.168.70.49',_h272))
           ...
(3)        int(compromised('192.168.10.90'),l) intRule_3
                int(sendExploit('128.111.49.46','192.168.10.90'),c) obsRule_th5
                        obs(snort('1:1807','128.111.49.46','192.168.10.90',_h336))
           ...
(4)        int(compromised('192.168.10.90'),p) intRule_3
                int(sendExploit('128.111.49.137','192.168.10.90'),p) obsRule_th1
                        obs(snort('1:1140','128.111.49.137','192.168.10.90',_h588))
           ...
```

Figure 6: Partial output trace from the reasoning system

```
/* heuristic function for searching guestbook.pl pattern*/
heuristics(heuristics_th1,
   obs(snort('1:1140', _FromHost, ToHost, Time)),
   target(ToHost, log_pattern('apache_access', Time, 'guestbook.pl'))).
```

(a): An example heuristic function

```
128.111.49.47 - - [06/Dec/2002:12:31:47 -0800] "GET
/cgi-bin/guestbook.pl?/etc/passwd HTTP/1.1" 200 1924 "-" "Mozilla/5.0
(X11; U; Linux i686; en-US; rv:1.0.1) Gecko/20021003"

128.111.49.47 - - [06/Dec/2002:12:43:12 -0800] "GET
/cgi-bin/guestbook.pl?guestbook.txt;useradd%20-G%20root%20-p%20foobar%20-r%20bill
HTTP/1.1"
200 632 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.1)
Gecko/20021003"
```

(b): Partial Output from the targeted search

Figure 7: Targeted search using heuristic function

in the Apache logs on the target host. With information provided by the heuristic, we narrowed down our search in the Apache logs. Partial output of manual log search is shown in Figure 7(b). The log indicates a `useradd` command was injected via web request to gain access to the host. Such concrete evidence accompanying the output of the reasoning engine helped us confirm how the attacker compromised the system.

4.3 Result validation and analysis

An attack targeted on a host followed by a probing activity from that host allows us to conclude that the host was compromised. Referring back to Figure 6, our reasoning system made use of the two stages of attack: attempts to compromise the WWW server ((3), (4)) and attacker activity like probing on the file server (2) to strengthen the proof associated with `compromised(WWW)` to certain.

The published TH data set did not include a truth file (a file containing information on how the actual attacks were carried out and to what extent they were successful) to verify the correctness of our experimental results. We performed manual analysis of the logs and compared it with the output. In the process, we made a couple of observations that are interesting not just to our study but intrusion analysis in general. First, we found that the reasoning engine had used the alert "Apache Chunk Encoding transfer attempt" to strengthen the confidence of a proof. But from our manual analysis, the success of this alert was inconclusive because the Apache access log had a "Bad Request" (Apache response code: 400) as response to all such requests made to the Apache server. While it is possible that all the attempts failed, the absence of evidence is not a conclusive proof that none of the attempts succeeded. This is because the behavior of a program (in this case, Apache) after a successful exploit is not fully predictable. An alert verification process like that in [26] would have removed the alert from entering the reasoning engine by using probing techniques to find if the service is running (The data set had no such details to aid in removing the alert).

Our reasoning system was able to identify the second stage of attacks on the File Server. However, there is yet another stage of attack. The third stage of attack on the SQL server was not captured. Here, the attacker gained access to the SQL server by creating a new fraudulent account on the file server which provides authentication information to the SQL Server. Such scenarios are not currently handled in the internal model because the nature of logged information about the attack makes it far from conclusive. This aspect needs further research.

5 Related Work

Uncertainty in data, specifically in the context of security analysis, has been handled in previous works using various statistical tools, especially the Bayesian Networks (BN) [8, 13, 28]. Zhai et al [28] use BN to correlate complementary intrusion evidence from both IDS alerts and system monitoring logs so that high-confidence traces can be distinguished from ones that are less certain. While statistical tools like BN is theoretically rigorous and elegant, having been proved effective in other areas of research, they have an unaddressed gap, namely, how to set statistical parameters in terms of hard probability distributions: in the specific case of BN, the conditional probability table (CPT) for each node in a BN. In practice, it has proven very hard not only to estimate the probabilities necessary but also hard to learn them from large real-life data because of overwhelming volume of background data that has to be eliminated from the "signal" of intrusion data. [1] and [6] converge on this difficulty from two different viewpoints, estimation theory and learning theory, respectively. For security analysis, it is nearly impossible to obtain the ground truth in real traces and it is hard if not impossible to simulate attacks. On the other hand, in practice SAs have been managing to detect attacks in logs and real-time alerts without the benefit of such theoretical models. This inspired us to formulate a logic that approximates human reasoning that works with a qualitative assessment on a few confidence levels that are relatively easy to understand. We acknowledge that this formulation not only hides the lack of knowledge of base probabilities but also reflects the great deal of ambiguity that genuinely exists in intrusion analysis of real data. We hope that by creating an option to specify the confidence level explicitly and by providing practical tools to manipulate these uncertain pieces of knowledge, we can bypass some of these fundamental problems and gain experience that may make some statistical approaches viable in the future.

BotHunter [10] is an application for identifying Bot machines through correlating Snort alerts with a number of other system-monitoring events. The notion of "confidence score" and "evidence threshold" are introduced to capture the uncertainty in the correlation process, and specific processes are designed for the purpose of Bot detection. The goal of our work is to provide a simple model for the general problem of intrusion analysis, not specifically targeted at a special type of attack such as Bot infection.

There is more literature on intrusion alert correlation [4, 5, 18, 19, 25, 26] than can be done justice to in a short section. These vital works have provided important insights into logical causality relations in IDS alerts that have informed this work. Most of these works model around IDS alerts with pre- and postconditions, which drives an internal reasoning based on graphs. However, we have found that sometimes it is difficult to come up with a compact pre- and post-condition model for ubiquitous observations that can be symptomatic of a wide variety of seen and unseen conditions. For example, in our study there were too many possibilities for the abnormally high network traffic event. Our observation correspondence model assigns a *direct* meaning to an observation and our internal model allows such meanings to be flexibly linked together based on their inherent semantics. We believe that such flexibility is important in intrusion analysis, especially in cases where the evidence is tenuous. Another general model for incorporating all possible types of data in security analysis is M2D2 [17]. Apart from the fact that M2D2 does not deal with uncertainty in modeling, our model is much easier to understand for a non-expert – rather than classifying the incoming information into various categories with mathematical notations, we represent knowledge in our model as simple "statements"that can be easily translated into natural language.

One step in our reasoning process is data pre-processing which involves data reduction based on clustering and simple correlation of local observations. Much previous work in IDS has addressed this problem [18, 26] and we intend to use applicable tools and approaches from the prior work to reduce the number of observations that need to be entered into our reasoning system. Recent work by Martignoni *et al.* [12] proposed a "layered approach" for detecting malicious behaviors, which could be combined with the approach presented in this paper to link low-level monitoring traces such as system calls to high-level meanings with uncertainty for intrusion analysis.

There is a strong literature on creating logics for dealing with uncertainty. While many of them may prove to be useful theoretical underpinnings for our system, we have not investigated this direction. In particular Friedman and Halpern introduced a new approach to modeling uncertainty based on "plausibility measures" [7]. Our notion of the uncertain modes is motivated by similar considerations but we do not require this precise definition because the meaning of negation is not always clear in our context. We note that our logic with *mode operators* is inspired by modal logic. A formal description of our logic as a modal logic is the subject of future research. Recent years even see revolutionary works that integrate logic programming and statistical modeling, most notably the PRISM system [24]. Whether our logic can be expressed in a formal language like PRISM will be our future research.

6 Conclusions and Future Work

We presented a practical approach to modeling uncertainty in intrusion analysis. Our goal is to help the system administrator in reaching conclusions quickly about possible intrusions, when multiple pieces of uncertain data have to be integrated. The model language we designed has two components: observation correspondence and internal model. The observation correspondence gives a direct meaning to low-level system monitoring data with explicit uncertainty tags, and can be derived from natural-language description that already exists in some IDS knowledge bases, *e.g.* the Snort rule repository. The internal model is concise and captures general multi-stage attack conditions in an enterprise network. We developed a reasoning system that is easy to understand, handles the uncertainty existent in both observation correspondence and the internal model, and finds high-confidence attack traces from many possible interpretations of the low-level monitoring data. Our prototype and experiments show that the model developed from studying one set of data is effective for analyzing a completely different data set with very little effort. This is a strong indication that the modeling approach can codify the seemingly ad-hoc reasoning process found in intrusion analysis and yield practical tools for enterprise-network environments.

As this is the first time that uncertainty has been dealt with in this explicit but qualitative manner, much work remains to be done, some of which we have already alluded to earlier. Specifically, our modality is a very crude operator – we do not distinguish the various forms of uncertainty, such as lack of accuracy vs lack of precision. On a practical level adding too much complexity to the modality itself may be counter-productive. But with some maturity of modeling and experience, we hope to be separate in our model these two sources of uncertainty as well. As in any modeling tool, there is a natural question of how granular our models have to be to achieve best results. Because the uncertainty of knowledge increases after a certain granularity, we expect that there is an optimal point that can only be discovered with experience. Our modeling and reasoning are monotonic and we do not deal with negation in our models. Although we did not need it in the two datasets that we analyzed, it is plausible that a new observation can *reduce* the modality of an internal condition, e.g. from likely to possible. This is a subject for future research.

References

[1] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, 2000.

[2] S.T. Brugger and J. Chow. An assessment of the DARPA IDS evaluation dataset using snort. Technical Report CSE-2007-1, University of California, Davis, Department of Computer Science, 2007.

[3] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions Knowledge and Data Engineering*, 1(1):146–166, 1989.

[4] Steven Cheung, Ulf Lindqvist, and Martin W Fong. Modeling multistep cyber attacks for scenario recognition. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 284–292, Washington, D.C., 2003.

[5] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, 2002.

[6] C. Drummond and R.C. Holte. Severe class imbalance: Why better algorithms aren't the answer. In *Machine Learning: ECML 2005*, volume 3720 of *Lecture Notes in Computer Science*, pages 539 – 546. Springer US, 2005.

[7] N. Friedman and J.Y. Halpern. Plausibility measures and default reasoning. *J. ACM*, 48(4):648–685, 2001.

[8] Saurabh Bagchi Gaspar Modelo-Howard and Guy Lebanon. Determining placement of intrusion detectors for a distributed application through bayesian network modeling. In *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*. RAID, September 2008.

[9] G. Gu, A.A. Cárdenas, and W. Lee. Principled reasoning and practical applications of alert fusion in intrusion detection systems. In *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 136–147, 2008.

[10] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[11] Joshua W Haines, Richard P Lippmann, David J Fried, Eushiuan Tran, Steve Boswell, and Marc A Zissman. 1999 DARPA intrusion detection system evaluation: Design and procedures. Technical Report TR-1062, MIT Lincoln Laboratory, 2001.

[12] Matt Fredrikson Lorenzo Martignoni, Elizabeth Stinson and Somesh Jha John Mitchell. A layered architecture for detecting malicious behaviors. In *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*. RAID, September 2008.

[13] Ulf Lindqvist Magnus Almgren and Erland Jonsson. A multi-sensor model to improve automated attack detection. In *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*. RAID, September 2008.

[14] M.V. Mahoney and P.K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, 2003.

[15] J. McHugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1:14 – 35, 2001.

[16] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 3(4):262–294, 2000.

[17] Benjamin Morin, Hervé, and Mireille Ducassé. M2D2: A formal data model for IDS alert correlation. In *5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.

[18] Peng Ning, Yun Cui, Douglas Reeves, and Dingbang Xu. Tools and techniques for analyzing intrusion alerts. *ACM Transactions on Information and System Security*, 7(2):273–318, May 2004.

[19] Steven Noel, Eric Robertson, and Sushil Jajodia. Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances. In *20th Annual Computer Security Applications Conference (ACSAC 2004)*, pages 350– 359, 2004.

[20] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 336–345, 2006.

[21] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[22] S. Peisert, M. Bishop, S. Karin, and K. Marzullo. Analysis of computer intrusions using sequences of function calls. In *IEEE Transactions on Dependable and Secure Computing (TDSC*, pages 137 – 150, 2006.

[23] Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David S. Warren, and Juliana Freire. XSB: A system for efficiently computing well-founded semantics. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.

[24] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[25] F. Valeur. *Real-Time Intrusion Detection Alert Correlation*. PhD thesis, University of California, Santa Barbara, May 2006.

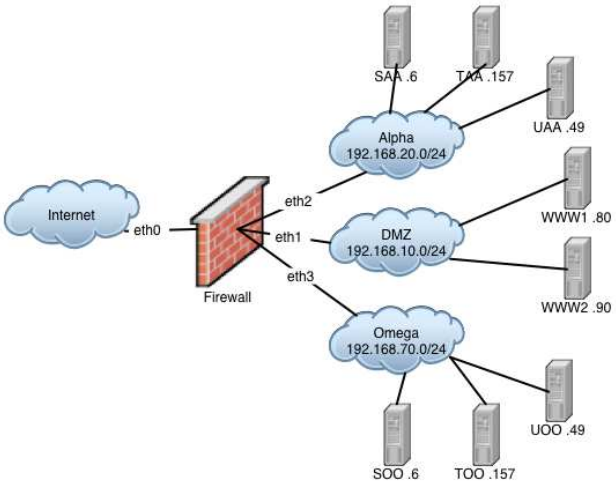
[26] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, 2004.

[27] G. Vigna. Teaching Network Security Through Live Exercises. In C. Irvine and H. Armstrong, editors, *Proceedings of the Third Annual World Conference on Information Security Education (WISE 3)*, pages 3–18, Monterey, CA, June 2003. Kluwer Academic Publishers.

[28] Yan Zhai, Peng Ning, Purush Iyer, and Douglas S. Reeves. Reasoning about complementary intrusion evidence. In *Proceedings of 20th Annual Computer Security Applications Conference (ACSAC)*, pages 39–48, December 2004.

7 Appendix

7.1 Network Topology used for TreasureHunt experiment



Note: The figure is taken from the TH website `http://www.cs.ucsb.edu/~vigna/treasurehunt/`

7.2 List of tasks used during the treasure hunt exercise

Task	Description	Max Duration
1	Determine the active hosts in subnet X.Y.Z. Also determine each host's OS and the services/applications that are remotely accessible. Scanning techniques that will evade detection by the Snort system will receive additional bonus points.	20 minutes
2	Get interactive access to the web server host by exploiting a web-based vulnerability. You must be able to login into the host as a user account other than root.	30 minutes
3	Get root privileges on the web server host.	30 minutes
4	Determine the hosts that are located in the specified internal subnet. Also determine their OSs and the services/applications that are remotely accessible. Scanning techniques that will evade detection by the Snort system will receive additional bonus points.	20 minutes
5	Access the MySQL database on host SQL and obtain the content of the table Employees.	20 minutes
6	Get interactive access to the MySQL server host. You have to be able to login with an account that is not root	20 minutes
7	Get root access to the MySQL server host.	20 minutes
8	Modify the database table Employees, setting the account number of each employee to an account number of your choice.	10 minutes
9	Obtain access to the transaction service on host TRN. Schedule a paycheck payment that will transfer the employee paychecks to your account.	30 minutes

7.3 Observation correspondence and internal model used in the experiment

```
/* model_th.P - predicates that describes the models Observation
```

```
Correspondence and Internal Conditions */


/*
-- obs: Observations like Snort alerts, syslog etc
-- int: Internal condition
-- target: hint on how to target the search
*/

/***** Explanation of observation predicates used in the model *****
obs(snort(ID, FromHost, ToHost, Time)):
  Snort alert generated at time Time by Snort rule id ID;
**********************************************************************/

/******* Explanation of internal predicates used in the model ********
int(compromised(H)): Host H is compromised.
int(sendExploit(FromHost, ToHost)): FromHost sent an exploit to ToHost
int(port_open(H,P)): Port P is open on host H.

**********************************************************************/

/******* Explanation of targeting predicates used in the model ********
target(H, log_pattern(Log, Time, Pattern)): target the Log on host H
                  around time Time and look for Pattern.
target(H, open_port(P)): check whether port P is open on host H
**********************************************************************/

/******** Observation correspondence *************/

/* attempt to exploit the guestbook.pl vulnerability */
obsMap(obsRule_th1,
obs(snort('1:1140', FromHost, ToHost, _Time)),
int(sendExploit(FromHost, ToHost)), p).

/* oversize chunk encoding */
obsMap(obsRule_th3,
obs(snort('119:16', FromHost, ToHost, _Time)),
int(sendExploit(FromHost, ToHost)), p).

/* Apache chunked encodin transfer attempt */
obsMap(obsRule_th4,
obs(snort('1:1807', FromHost, ToHost, _Time)),
int(sendExploit(FromHost, ToHost)), c).

/* Apache chunked encoding worm attack */
obsMap(obsRule_th5,
obs(snort('1:1809', FromHost, ToHost, _Time)),
int(sendExploit(FromHost, ToHost)), c).


/* ICMP PING NMAP */
obsMap(obsRule_th6,
obs(snort('1:469', FromHost, ToHost, _Time)),
int(probeOtherMachine(FromHost, ToHost)), l).

/* TCP Portscan */
obsMap(obsRule_th6,
obs(snort('122:1', FromHost, ToHost, _Time)),
int(probeOtherMachine(FromHost, ToHost)), l).
```

```
/****** Internal model *********/
intRule(intRule_1,
   int(compromised(FromHost)),
   int(probeOtherMachine(FromHost, _ToHost)),
p, c).

intRule(intRule_2,
   int(compromised(H1)),
   int(sendExploit(H1, _H2)),
   p, c).

intRule(intRule_3,
   int(sendExploit(_H1, H2)),
   int(compromised(H2)),
   l, p).

intRule(intRule_4,
   int(compromised(H1)), int(compromised(H2)),
   int(exchangeCtlMessage(H1, H2)),
   p, c).

/******* Heuristics on targeting information *******/
heuristics(heuristics_th1,
obs(snort('1:1140', _FromHost, ToHost, Time)),
target(ToHost, log_pattern('apache_access', Time, 'guestbook.pl'))).

heuristics(heuristics_th2,
int(port_open(H, P)),
target(H, open_port(P))).
```