An Empirical Approach to Modeling Uncertainty in Intrusion Analysis

Xinming Ou
Kansas State University
Manhattan, KS, USA
xou@ksu.edu

Siva Raj Rajagopalan
HP Labs
Princeton, NJ, USA
raj.rajagopalan@hp.com

Sakthiyuvaraja Sakthivelmurugan
Kansas State University
Manhattan, KS, USA
sakthi@ksu.edu

***Abstract*—Uncertainty is an innate feature of intrusion analysis due to the limited views provided by system monitoring tools, intrusion detection systems (IDS), and various types of logs. Attackers are essentially invisible in cyber space and monitoring tools can only observe the symptoms or effects of malicious activities. When mingled with similar effects from normal or non-malicious activities they lead intrusion analysis to conclusions of varying confidence and high false positive/negative rates. This paper presents an empirical approach to the problem of uncertainty where the inferred security implications of low-level observations are captured in a simple logical language augmented with certainty tags. We have designed an automated reasoning process that enables us to combine multiple sources of system monitoring data and extract highly-confident attack traces from the numerous possible interpretations of low-level observations. We have developed our model empirically: the starting point was a true intrusion that happened on a campus network that we studied to capture the essence of the human reasoning process that led to conclusions about the attack. We then used a Datalog-like language to encode the model and a Prolog system to carry out the reasoning process. Our model and reasoning system reached the same conclusions as the human administrator on the question of which machines were certainly compromised. We then automatically generated the reasoning model needed for handling Snort alerts from the natural-language descriptions in the Snort rule repository, and developed a Snort add-on to analyze Snort alerts. Keeping the reasoning model unchanged, we applied our reasoning system to two third-party data sets and one production network. Our results showed that the reasoning model is effective on these data sets as well. We believe such an empirical approach has the potential of codifying the seemingly ad-hoc human reasoning of uncertain events, and can yield useful tools for automated intrusion analysis.**

***Keywords*-intrusion detection; uncertainty; logic;**

I. Introduction

Intrusion detection is the last line of defense against cyber attacks. However building robust tools to detect intrusions in practical environments has proved 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 possibly large amounts of system monitoring data, either on the fly or offline. Solving this problem in general has been an inexact science that has to admit a range of uncertainty in output. System administrators (SA) today use a combination of intuition, experience, and low-level tools to create and support positive or negative judgements on security events. However, the high volume of gathered data strains the intuitive capacity of analysts. Increasingly, sophisticated attacks combine multiple intermediate attack steps to achieve their goals, as a result of which we may have to combine the outputs of several disparate sensors to detect multi-step attacks that are found today. While the low-level observations (network packets, server logs, *etc.*) all have potential implications for attack possibilities, few, if any, of them can directly provide zero/one judgment at the high-level abstraction (*e.g.*, ”machine has been compromised”). Nevertheless, in many remote intrusions a relatively small number of critical observations *taken together* are sufficient to show that an attack has certainly happened as well as how it progressed. The difficulty is how to start from uncertain and voluminous views of the potential problem (*e.g., IDS alerts*) and search for a few pieces of data among millions so that the attacker’s hand is clearly and quickly shown. SAs are highly time-constrained – an automatic tool that can sift through the ocean of uncertainty to quickly and accurately locate the problem areas or reduce the search space will be invaluable in practice.

There are several technical challenges in handling uncertainty in automated situation awareness, among the hardest of which is quantifying the degree of certainty in various assertions regarding possible attack activities. However, through our interaction with highly capable SAs who face the challenge every day while protecting enterprise systems, we observed that humans do well without relying on any numerical measures. This is illustrated in the true-life story below.

A. A true-life incident

Consider the following sequence of events that actually occurred at a university campus (key observations are numbered in parentheses). The SA noticed an abnormally large spike in campus-network traffic (*Observation 1*). The SA took the netflow dump for that time period, searched it for known malicious external IP addresses, and identified that four Trend Micro (standard anti-malware) servers had initiated IRC connections to known BotNet controllers (*Observation 2*). The SA suspected that the four Trend Micro servers had been compromised. At the console of one of the servers he 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 some zero-day vulnerability in the Trend Micro server software. He did the same for the other identified servers and found even more compromised servers. Altogether he identified 12 compromised machines and took them offline.

B. Analysis methodology

When the administrator first noticed the spike in network traffic, the questions facing him were: is the network experiencing an attack, and if so, which machines were compromised. However, none of the low-level observations alone could give him a definitive answer to these high-level questions. Observation 1 (traffic spike) could indicate a variety of causes, most of which are benign. (For example, enterprise networks often see a spike in traffic on "Microsoft Patch Tuesdays.") Observation 2 (connections to BotNet controllers) has a higher likelihood of indicating malicious activity and hence a higher degree of likelihood that the identified hosts are compromised. However, an IRC connection being made to a known BotNet controller by itself is not incontrovertible evidence that a machine has been compromised. The list of "known" BotNet controllers may contain false positives or somebody could be probing BotNet controllers for research purposes. (The interviewed SA suggested this false positive as he did 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. Observation 4, like observation 2, cannot definitively prove that the machines observed are under the control of attackers because IRC is occasionally (but rarely) used as a communication channel between servers. However, when we put all the four pieces of evidence together within a small time period, it seems clear that an attack has certainly happened and succeeded and we can say that which machines have been almost certainly compromised.

In handling this incident, the SA noted that while each observation could have multiple interpretations, the unified interpretation linking the semantics of multiple observations is most likely to be the true one. For example, although neither observation 3 nor 4 alone can give us high enough confidence to say that the host is definitely compromised, linking them semantically allows to dramatically strengthen our confidence in the assertion, since both 3 and 4 point to the same interpretation. We observed a similar pattern in many other incidents we learned from interviewing system administrators. It appears that even without quantitative measures on uncertainty, the semantic links among possible evidence can dramatically increase one's confidence on whether an attack has actually happened and its consequences. As a result, humans can handle the uncertainty pretty well by "connecting the dots" among various pieces of evidence. However, manual analysis alone is not scalable and sustainable in the face of large-scale automated attacks we face today. In this incident, the human SA had all the common security tools at his disposal but none of the tools could provide the crucial capability of analysis and the manual analysis took a long time. As a first step, can we design tools that help reduce the amount of time that the SA had to spend in the process? This is the main question we aim to answer in this paper.

C. An empirical approach

We propose an empirical approach to automate reasoning with uncertainty in intrusion analysis. We design a reasoning model where human knowledge used to draw conclusions about uncertain events can be codified and applied mechanically to future incidents. Although reasoning about intrusions on different incidents can vary significantly with context, the basic principles are quite consistent. We design a reasoning model that captures the essence of the generic reasoning rationale, not specific features of any particular incident. The model provides a language whereby human experts can share knowledge useful for intrusion analysis in a machine readable format, and an automated reasoning engine can make use of the knowledge, significantly expanding a human's capability. Both the model and the reasoning engine are designed empirically through the study of real security incidents.

This is certainly not the first attempt at automating reasoning about intrusions. Past work has applied rule-based systems to correlate audit logs and detect attacks [1], [2]. There is also a great deal of work on IDS alert correlation [3], [4], [5], [6], [7], [8]. These previous approaches do not address the uncertainty problem explicitly, *i.e.*, the reasoning systems do not model when and how confidence levels on assertions can be strengthened in the correlation process.We believe an explicit model for uncertainty in reasoning is crucial to making alert correlation tools useful in practice. Zhai, *et al.* has pioneer work [9] in this area by combining alert-correlation, attack-graph, and Bayesian-Network techniques to reason about 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. Recent years have also seen the application of quantitative mathematical methods such as

Bayesian Network [10] and Dempster Shafer theory [11] in intrusion detection [12], [13], [14]. We have chosen not to start from those mathematical theories, because to utilize them we need *a priori* the logical structure among the various observations and hypotheses. In intrusion analysis, identifying the structure of attacks is a big challenge in and of itself, which is intermingled with the challenge caused by uncertainty of observation and interpretation. Instead of starting from these mathematical theories, we start from true-life experiences like the one described in Section I-A, and design a model that systematically "simulates" what a human does manually, formalizing empirical experience so that it can be applied mechanically to future incidents. We believe this empirical, bottom-up approach is an important first step in understanding the nature of reasoning about uncertainty in cyber security and gaining experience that may make truly quantitative approaches viable in the future.

D. Our contributions

An empirical model for uncertainty: First, 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. The model and inference process are based on how human administrators reason about attacks in real security incidents. We use qualitative rather than quantitative assessment to capture the uncertainty inherent in such assertions. The qualitative assessment reflects the imprecise nature of the certainty levels' semantics and it also makes it easier to understand by humans, enabling discussion/refinement of the reasoning model in an open community. Such a model can also 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 monitoring data.

Reasoning methodology: Second, we present a reasoning process that can utilize such a model and existing IDS tools to automatically identify high confidence 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. Our model of high-level conditions is generic predicates such as "compromised," "exploit sent," *etc.* that are independent of the scenario at hand. What can change from one scenario to another are the instantiation of the predicates and the certainty tags associated with them as the scenario events are processed and the paths that the reasoning process takes through these conditions. We believe that SAs have such small sets of "target" conditions in mind when they process intrusion data and there is value in capturing those target conditions directly in an automated reasoning process. We implemented a prototype reasoning engine 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.

A Snort add-on based on our model and method: Third, we automatically generated a significant part of our reasoning model from the `classtype`, `impact`, and `ease of exploit` fields associated with Snort rule descriptions and show that the knowledge base needed by our reasoning engine can be readily created if security monitoring tools use our model language (instead of natural language) to describe the potential meanings of various types of security alerts. Based on this automatically-derived knowledge base and the prototype implementation of our reasoning engine, we provide a Snort add-on, called SnIPS, that analyzes millions of Snort alerts reported from an enterprise network and only report those with high-confidence evidence associated with them.

Evaluation of the methodology: Finally, we applied the SnIPS tool to two third-party datasets created from real network activities as well as the production network in the computer science department of a university that was used to build the model. The core reasoning engine and model were kept unchanged in the evaluation. We found remarkably that even though our core reasoning model was developed from a very simple and completely different incident, our tool discovered interesting scenarios from these data sets. The application of SnIPS also resulted in dramatic reduction (99%) in the amount of data a system administrator would have to look at. The false positives from the analysis helped us identify imprecisions from the automatically generated Snort knowledge base as well as some subtle but minor gaps in the core model. This indicates that such an empirical approach could produce a shared knowledge base that can be iteratively refined among security practitioners to yield agile and accurate tools.

II. The reasoning model

We motivate our design using the true-life incident described in Section I-A. We study the analytic states that the SA went through in the course of the investigation, identify the rationale behind the decisions at various points, and design a logic that captures this reasoning process.

A. Modeling uncertainty

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 known accuracy only at low levels of abstraction (*e.g.*, network packets, server logs, *etc.*). Uncertainty arises from this semantic gap as well. For example, a packet pattern (a "signature") could be associated mostly with attacks but on occasion with legitimate use as

$$\begin{aligned}
A_1: &\quad \mathit{obs}(\mathit{anomalyHighTraffic}) \overset{p}{\longmapsto} \mathit{int}(\mathit{attackerNetActivity})\\
A_2: &\quad \mathit{obs}(\mathit{netflowBlackListFilter}(H, \mathit{BlackListedIP})) \overset{l}{\longmapsto}\\
&\qquad\qquad \mathit{int}(\mathit{attackerNetActivity})\\
A_3: &\quad \mathit{obs}(\mathit{netflowBlackListFilter}(H, \mathit{BlackListedIP})) \overset{l}{\longmapsto}\\
&\qquad\qquad \mathit{int}(\mathit{compromised}(H))\\
A_4: &\quad \mathit{obs}(\mathit{memoryDumpMaliciousCode(H)}) \overset{l}{\longmapsto}\\
&\qquad\qquad \mathit{int}(\mathit{compromised}(H))\\
A_5: &\quad \mathit{obs}(\mathit{memoryDumpIRCSocket}(H_1, H_2)) \overset{l}{\longmapsto}\\
&\qquad\qquad \mathit{int}(\mathit{exchangeCtlMessage}(H_1, H_2))
\end{aligned}$$

Figure 1. Observation correspondence

well. Furthermore, it may not tell us whether the attack succeeded. A key step in tackling the uncertainty challenge is to develop a model that can link multiple low-level observations to the conditions under concern at the high level and simultaneously allow us to specify our confidence in the assertions.

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 a continuous quantity ranging from completely unknown to completely certain, we found that, in practice, human SA's only deal with a few confidence levels that roughly correspond to the ones defined here.These words are also used routinely in natural-language description of security knowledge bases such as the Snort rule repository. We emphasize that these uncertainty levels are assigned by humans and apart from the obvious ordering ($p < l < c$) we are not ascribing a probability range to each level.

With this qualitative notion of uncertainty, we introduce the two types of logical assertions in our reasoning model: *observation correspondence* which maps low-level observations to high-level conditions, and *internal model* which captures relationships among high-level conditions (also called *internal conditions* hereafter) . Correspondingly, we use $\mathit{obs}(O)$ to denote a fact about observation O, and $\mathit{int}(F)$ to denote an internal condition F. For example, $\mathit{obs}(\mathit{netflowBlackListFilter}(\mathit{ip}_1, \mathit{ip}_2))$ is an observation from the netflow blacklist filter that "machine ip_1 is communicating with a known blacklisted (and hence likely malicious) host ip_2", whereas $\mathit{int}(\mathit{compromised}(\mathit{ip}_1))$ is an internal condition that "ip_1 is compromised."

B. Observation correspondence

Figure 1 shows the observation correspondence relation for the observations in the true-life incidents described in Section I-A. 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 are other equally possible interpretations for the same observation. A_2 and A_3 give the meaning to an alert identified in netflow analysis. There are a number of filtering tools that can search for potential malicious patterns in a netflow dump such as "capture daemon" and "flow-nfilter." These rules deal with one filter that identifies communication with known malicious IP addresses. Since any such activity is a strong indication of attacker activity and compromise of the machine involved, 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. It is legitimate to have multiple observation correspondence assertions for the same observation: they may represent different aspects or possibilities of an observation. 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 was used to exchange control messages between BotNet members.

We recognize that these observation correspondence assertions are subjective. Quantifying the results of intrusion sensing in a robust manner has remained a hard problem for a variety of reasons [15]. 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. Given the large base of similar deployed infrastructure, shared experiences from a large community can likely help tune the modes in those assertions. We envision a rule repository model like that for Snort, where a community of participants contributes and agrees 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 IV, we show how a small number of internal-model predicates can give meanings to the vast majority of Snort alerts and that the observation correspondence relation can actually be automatically generated from a Snort rule's classtype and the "impact" and "ease of attack" fields in the rule's natural-language description. If the Snort rule writers had a standard language for such information they would be able to readily provide the observation correspondence assertions for Snort alerts.

C. Internal model

The reasoning model should also express the logical relations among the various high-level conditions so that such knowledge can be mapped to correlate low-level events. For example, the model should include knowledge such as "after an attacker has compromised a machine, he may perform some network activity from the machine." This is a generic action common to many attack scenarios. This knowledge

$$
\begin{aligned}
I_{1f}: &\quad \mathit{int}(\mathit{compromised}(H_1)) \xrightarrow{f,p} \mathit{int}(\mathit{probeOtherMachine}(H_1, H_2))\\
I_{1b}: &\quad \mathit{int}(\mathit{probeOtherMachine}(H_1, H_2)) \xrightarrow{b,c} \mathit{int}(\mathit{compromised}(H_1))\\
I_{2f}: &\quad \mathit{int}(\mathit{compromised}(H_1)) \xrightarrow{f,p} \mathit{int}(\mathit{sendExploit}(H_1, H_2))\\
I_{2b}: &\quad \mathit{int}(\mathit{sendExploit}(H_1, H_2)) \xrightarrow{b,c} \mathit{int}(\mathit{compromised}(H_1))\\
I_{3f}: &\quad \mathit{int}(\mathit{sendExploit}(H_1, H_2)) \xrightarrow{f,l} \mathit{int}(\mathit{compromised}(H_2))\\
I_{3b}: &\quad \mathit{int}(\mathit{compromised}(H_2)) \xrightarrow{b,p} \mathit{int}(\mathit{sendExploit}(H_1, H_2))\\
I_{4f}: &\quad \mathit{int}(\mathit{compromised}(H_1)), \mathit{int}(\mathit{compromised}(H_2)) \xrightarrow{f,p}\\
&\qquad\qquad \mathit{int}(\mathit{exchangeCtlMessage}(H_1, H_2))\\
I_{4b_1}: &\quad \mathit{int}(\mathit{exchangeCtlMessage}(H_1, H_2)) \xrightarrow{b,c} \mathit{int}(\mathit{compromised}(H_1))\\
I_{4b_2}: &\quad \mathit{int}(\mathit{exchangeCtlMessage}(H_1, H_2)) \xrightarrow{b,c} \mathit{int}(\mathit{compromised}(H_2))
\end{aligned}
$$

Figure 2. Internal model

can reveal potential hidden correlations between low-level observations, (*e.g.*, high network traffic and netflow filtering result). Absent any 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 likely compromised machine it can be assigned a different meaning than if it did not.

Figure 2 shows the internal model we developed from studying the real-life incident. We use $C_l \xrightarrow{d,m} C_r$ to represent the inference rules for the internal conditions, namely condition C_l can infer condition C_r. There are two modality operators, d and m, associated with a rule. Like in observation correspondence, the m mode specifies the confidence in the inference and takes values from $\{p, l, c\}$. The d mode indicates the direction of the inference and could be either f (forward) or b (backward). In forward inference, C_r is caused by C_l, thus the arrow must be aligned with time, *i.e.* C_r shall happen after C_l. This can specify knowledge for reasoning what could happen after a known condition becomes true, *e.g.* `after an attacker sends an exploit to a machine, he will likely compromise the machine` (I_{3f}). In the backward inference, we reason what could have happened before to cause a known condition, and thus the direction of inference is opposite to time. Example: `if a malicious probe is sent from a machine, then an attacker must have certainly already compromised the machine` (I_{1b}). As another example, the forward inference rule I_{1f} specifies that "if an attacker has compromised machine H_1, he can perform a malicious probe from H_1 to another machine H_2." This inference has a low certainty: the attacker may or may not choose to probe another machine after compromising one. Thus the rule is qualified by the p mode. I_{4f} is the only rule in this model that has two facts on the left-hand side. As in typical logic-programming languages, the comma represents the AND logical relation.

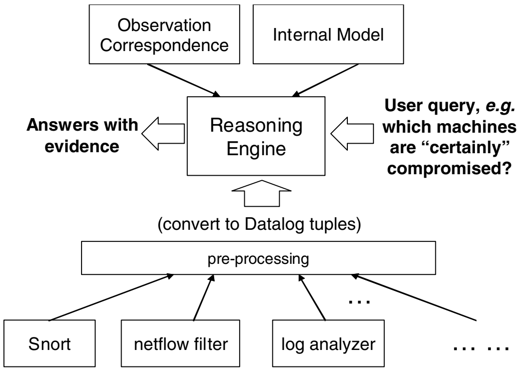
Observation Correspondence
Internal Model
Answers with evidence
Reasoning Engine
User query, *e.g.* which machines are "certainly" compromised?
(convert to Datalog tuples)
pre-processing
...
Snort
netflow filter
log analyzer
... ...

Figure 3. System architecture

III. Reasoning methodology

The reasoning model described in section I-A is analogous to human thinking – observations are reflected as beliefs with varying strengths and the beliefs are related to one another with varying strengths. This section will introduce the reasoning process to use such a model to "simulate" human thinking such that an automated inference process can allow us to combine observations 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 evidence pieces logically linked together.

Reasoning framework: Figure 3 presents the architecture of our reasoning system. The two modules of the reasoning model — observation correspondence (described in Section II-B) and internal model (described in Section II-C) are input to the reasoning engine. Both modules are specified in Datalog [16], a simple logic-programming language. The raw observations are pre-processed and the distilled results are converted to Datalog tuples as input to the reasoning system. The reasoning engine is itself implemented in Prolog. 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.

A. Pre-processing

The pre-processing step is performed to compact the information entering the reasoning engine. We apply a data abstraction technique by grouping a set of similar "internal conditions" into a single "summarized internal condition". The summarization is done on both the time stamps and IP addresses. For timestamps, if a set of internal conditions differ only by timestamp we merge them into a single summarized internal condition with a time range between the earliest and latest timestamp in the set. For Snort alerts, we also abstract over external IP addresses so that alerts that differ only on the external source or destination addresses are merged and the corresponding IP address is abstracted as "external".

B. Application of inference rules

All the rules in the observation correspondence and internal model can be viewed as inference rules. The reasoning engine applies those rules on the input Datalog tuples to derive assertions with various levels of certainty. The certainty of the derived fact is the lowest certainty of the facts and rules used to derive it.

Handling time: Timestamps associated with security monitoring events are important in tracking and diagnosing root causes of problems. Time stamps are used in the reasoning process but for presentation simplicity we do not show time stamps associated with the observations and internal conditions. When an observation correspondence rule is used, the internal condition on the righthand side simply inherits the time field of the observation on the left.When an internal model is used, the time stamps associated with the derived assertion will be derived from the timestamp on the lefthand side and the direction of the rule in a straightforward manner (*e.g.*, for a forward rule the righthand-side shall happen after the lefthand side). Latency in detection and clock skews can also make timestamps imprecise and less useful, which needs to be addressed through techniques like time windows which we leave as future work.

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 that shows the logical derivation steps arriving at the conclusion. From an observation one can derive an internal belief with some degree of certainty, based on the observation correspondence relation. As an example, the open IRC-Socket identified through memory dump in the incident described in section I-A will be an input to our system: $\mathit{obs}(\textit{memoryDumpIRCSocket(172.16.9.20,172.16.9.1)})$. Together with observation correspondence A_5, the rule will derive:

$$\begin{aligned}&\mathit{int}(\textit{exchangeCtlMessage('172.16.9.20','172.16.9.1')}, l) \Leftarrow\\&\qquad \mathit{obs}(\textit{memoryDumpIRCSocket('172.16.9.20','172.16.9.1')})\end{aligned}$$

The fact $\mathit{int}(\textit{exchangeCtlMessage(172.16.9.20,172.16.9.1)}, l)$ derived above, together with the internal-model rule I_{4b_1}, would yield the following derivation trace.

$$\begin{aligned}&\mathit{int}(\textit{compromised(172.16.9.20)}, l) \Leftarrow\\&\quad \mathit{int}(\textit{exchangeCtlMessage(172.16.9.20,172.16.9.1)}, l) \Leftarrow\\&\qquad \mathit{obs}(\textit{memoryDumpIRCSocket(172.16.9.20, 172.16.9.1)})\end{aligned}$$

Since the certainty mode for I_{4b_1} is c, joined with the l mode in *int(exchangeCtlMessage(172.16.9.20, 172.16.9.1), l)*, we get l as the mode for the resulting fact $\mathit{int}(\textit{compromised(172.16.9.20)}, l)$.

One could argue that the certainty of the derived fact should be lower than that of the weakest fact in the derivation chain, especially when the derivation chain is long. However, given the observation from SAs that most enterprise network intrusions are carried out in just a few steps, we do not expect the derivation chains to be long in practice. Since the certainty modes only represent a rough guess, accounting for certainty level decay along short derivation paths is unlikely to be significantly valuable. By taking the upper-bound of the certainty level (weakest link in the chain), we err on the side of false positives, which can be ruled out at a later stage after analyzing the attached proof trace.

C. Proof strengthening

The key purpose of reasoning about uncertainty is to derive high-confidence facts from low-confidence ones. In the true-life incident, the SA strengthened his belief that the Trend Micro server was compromised by combining three different 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.

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

The $\parallel$ relation indicates that two proofs are independent, meaning they are 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 can be strengthened. The *strengthen* function is defined below.

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

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.

We emphasize that the strengthening rules are defined through our empirical study on real-life security incidents and these strengthening conditions do reflect the mental process a human SA goes through when catching real

```
| ?- show_trace(int(compromised(H),c)).
int(compromised('172.16.9.20'),c) strengthen
  int(compromised('172.16.9.20'),l) I_4b1
    int(exchangeCtlMessage('172.16.9.20',
                     '172.16.9.1'),l) A_5
      obs(memoryDumpIRCSocket('172.16.9.20',
                             '172.16.9.1'))
  int(compromised('172.16.9.20'),l) A_4
    obs(memoryDumpMaliciousCode('172.16.9.20'))
  int(compromised('172.16.9.20'),l) A_3
    obs(netflowBlackListFilter('172.16.9.20',
                             '129.7.10.5'))
```

Figure 4. Result of applying the reasoning system on the case study

attacks. The rules presented above are by no means final products and will need to be further refined through more empirical study.

D. Implementation

We use the XSB [17] system to evaluate the Prolog reasoning engine. We also implemented a simple proof-generator so that whenever a fact is derived, the proof trace for that fact can also be obtained simultaneously. We applied the reasoning system and the model described in II-B and II-C on the input for our case study. The result is shown in Figure 4 (IP addresses are sanitized.). The user enters the query `show_trace(int(compromised(H),c))`, to find 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 a derivation trace for `172.16.9.20`, the IP address for the compromised Trend Micro server first identified by the SA. It is clear that the derivation trace exactly matches the reasoning process the human SA used to identify the compromised server — the confidence level is strengthened from concordant evidence emanated from netflow dump and memory dump.

IV. Automating model building for Snort

Snort (http://www.snort.org/) is a popular open-source network intrusion detection system utilizing a rule-driven language for specifying alert conditions. We developed a Snort add-on, called SnIPS, with the purpose of helping the Snort user community create an empirical model for reasoning about Snort alerts using the techniques discussed in this paper. The add-on is based on the system architecture in Figure 3, with only Snort alerts as input from the bottom. Each Snort alert is converted into a tuple like the one below.

```
obs(snort('1:1140', '172.16.9.18,
      '192.168.0.20', '2008-06-09 21:05:14')).
```

The first parameter `1:1140` is the unique SID associated with the Snort rule that generated this alert. The second and third parameters are the source and destination IP address of the captured network packet. The last parameter is the timestamp (our system actually represents the time stamp as an integer internally). We use the *same* internal model and reasoning engine developed from the real-life incident to analyze Snort alerts. But we still need to create the observation correspondence relations for Snort alerts. In the Snort rule repository each rule is given a "classtype" along with some natural-language description. We find that these pieces of information can be used to automatically infer the internal predicate and mode to be assigned to an alert. For example, the Snort rule `1:1140` 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, which is an application with known vulnerabilities. The classtype of the rule is "attempted-recon", indicating that these packets are generally considered reconnaissance activities. Based on this we can generate an observation correspondence assertion as follows.

```
obsMap(obsRuleId_3614,
   obs(snort('1:1140', FromHost, ToHost)),
   int(probeOtherMachine(FromHost, ToHost)),l).
```

There are 30 classtypes a Snort rule writer can use to classify the nature of the potential security implications. We try to map the classtypes to the internal predicates in our model. But classtypes alone do not always convey enough information. Thus we also make use of two fields in the natural-language description of a rule: "impact" and "ease of attack". These two fields for the above Snort rule are:

```
Impact:  Information gathering and system integrity
         compromise. Possible unauthorized
         administrative access to the server.
         Possible execution of arbitrary code of
         the attackers choosing in some cases.

Ease of Attack:  Simple. Exploits exist.
```

Since there is a keyword "exploits exist" in the "Ease of Attack" field, our automated program infers that the mode of the observation correspondence assertion is "likely", as shown above. In general, we found that these two fields are often composed of a set of fixed keywords. Examples are "possible unauthorized administrative access", "possible execution of arbitrary code", "exploits exist", and so on. These phrases often indicate what internal predicate and certainty mode can be assigned to the alert. Our automated program searches for those keywords in the Snort rule repository. Based on the combination of keywords contained in a Snort rule description, a simple heuristic algorithm infers the internal predicate and the certainty mode for alerts generated by the Snort rule. In this example, we can output another observation correspondence assertion for the alert:

```
obsMap(obsRuleId_3615,
  obs(snort('1:1140', FromHost, ToHost)),
  int(compromised(ToHost)), p).
```

Using this simple method, we were able to automatically generate the observation correspondence relations for 60%

of all the approximately 9000 Snort rules. This is certainly just a rough baseline, since our automated program has to make an "educated guess" from imprecise and incomplete information currently in the Snort rule repository. But we believe such an initial model, derived from existing knowledge would prove to be helpful for Snort users to benefit from our empirically developed technique. If our reasoning system proves to be useful by a significant number of users, there will be incentives for more people to help "fine-tune" the observation correspondence. As for future Snort rules, the security expert who writes a Snort rule (or any such specification) has the best knowledge on what the observation means and is best qualified to provide the information that enables our processing and reasoning. This should not incur much additional burden since much of this information is already being maintained in an unformatted and ad-hoc manner.

V. Experiments and Results

In this section we describe our effort on conducting experimental evaluation of our reasoning system. Due to the empirical nature of our approach, the evaluation is focused on whether a reasoning model developed empirically from studying one incident can be generalized to find interesting attack traces in others. We use the SnIPS tool described in section IV for our evaluation. The reasoning engine and internal model of SnIPS were developed based on the true-life incident we studied and the observation correspondence relations were automatically generated from the Snort rule repository. We then applied SnIPS to two third-party data sets as well as a production network. Note that the data we use to test SnIPS has nothing to do with the true-life incident based on which we developed SnIPS.

A. Experiment on the Treasure Hunt data set

The first set of experiments was performed on the Treasure Hunt (TH) data set [18]. This data set was created during a cyber-attack competition organized in a graduate security course at University of California, Santa Barbara. Our motivation to use this particular data set was that the data set provides valuable "meta data" such as the back story (competition task details) and network topology which can help us understand the result. We only used the TCPdump portion of the dataset to generate Snort alerts as input to SnIPS.

The first task in the TH competition was to gain access to the web server. This scenario was identified by our model and the high-confident output is shown in Figure 5 (the parenthesized numbers are added for explanation purpose). The web server, 192.168.10.90, was certainly compromised (1) based on two independent proofs: (2) and (4), which capture the first step: an exploit being sent from an external host to the web server (3), and the second step: reconnaissance by the attacker from the web server to learn about the internal network (5). The two pieces of evidence both point to the compromise of the web server, strengthening our confidence level to "certain". In total, there were 18 such proofs verifying the two web servers were compromised. Table 6 shows the reduction in amount of data that was presented by our tool to the system administrator for further analysis. The raw alerts belonging to the summarized internal condition can be identified using the mapping variable `obslist(Var)`.

```
(1)int(compromised('192.168.10.90'),c) strengthen
(2)   int(compromised('192.168.10.90'),p) I_3f
(3)       int(sendExploit(external,'192.168.10.90'),
                                                p)
           summarizedFact obslist(273)
(4)   int(compromised('192.168.10.90'),l) I_1b
(5)       int(probeOtherMachine('192.168.10.90',
                               '128.111.48.35'),l)
           summarizedFact obslist(257)
```

Figure 5. Partial output trace from the reasoning system

We manually validated raw alerts of the 18 proofs generated by the reasoning engine for false positives. From our analysis we perceive all of them are plausible. 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). Thus it was impossible to identify if the reasoning engine missed any true attack traces (false negatives).

B. Experiment on data collected on a Honeypot

We conducted our second set of experiments on a data set collected from a honeypot deployed at Purdue University. This data set was created for a unrelated project whose main intention was to collect spam relayed using open proxies. The network traffic in a single machine running misconfigured squid proxy was captured as TCPdump over a period of two months. (The total size of the zipped TCPdump files was about 68GB.) The reasoning engine had enough information to conclude the honeypot host was compromised. With knowledge about operating system and services running in the honeypot host, we validated the traces manually.

C. Experiments on a production system

The last two experiments also helped us find a few inaccuracies in the observation correspondence relations created by the automated model building process, which were subsequently fixed. In the last experiment, SnIPS with the updated knowledge base was applied on an university network. We installed Snort in our departmental network having 300 workstations along with dedicated web servers, file servers, databases server *etc*. We were only able to analyze the alerts captured over a period of three days. Snort reported about 1.1 Million alerts with 150 different alert types and 15 class types. Figure 6 shows the number of alerts generated by Snort and the number of high-confidence proofs presented to SA. The 17 proofs pointed that 4 hosts has a higher chance of being compromised. To verify this

Data set	Snort alerts	Summarized alerts	High-confidence proofs
TH	4,849,937	278	18
Honeypot	637,564	30	8
Department	1,138,572	6634	17

Figure 6. Reduction of alerts to high-confidence proofs

result would require further analysis of other log data which we did not have access to. We analyzed the output traces with the corresponding low-level alerts. It appeared that a couple traces were worthy of further investigation and we forwarded those to the system administrator. The others were likely to be false positives. Our tool helped reducing the search space and time spent on intrusion analysis.

VI. Related Work

Uncertainty in data, even specifically in the context of security analysis, is a vast and fertile topic and space constraints require us to only give the highlights of related work here. Probabilistic reasoning appears to be a natural candidate for such problems and there have been several attempts along this direction [12], [14], [9]. As explained earlier, a fundamental challenge in applying these techniques is how to obtain the logical structure needed for combining the probability measures. Moreover, these techniques require as inputs statistical parameters in terms of probability distributions of related events, conditional probability tables, *etc.* that have proven very hard to estimate or learn from real-life data because of overwhelming background noise (see, *e.g.*, [19] and [20] for two different viewpoints, estimation theory and learning theory). For security analysis, it is nearly impossible to obtain the ground truth in real traces and public data sets and it is hard if not impossible to realistically simulate attack scenarios. Consequently calibrating analysis techniques with metrics such as false positive/negative ratios is a huge challenge (see [8] for definitions) and intrusion analysis remains a manual and intuitive art, which has 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 exists in intrusion analysis of real data. We hope that by creating an option to specify the confidence level explicitly and by providing general practical tools to manipulate these uncertain pieces of knowledge, we can bypass some of these fundamental problems and gain experience and insight that may make some statistical approaches viable in the future.

A closely related work to ours, BotHunter [21], is an application for identifying Bot machines by correlating Snort alerts with a number of other system-monitoring events. The notions 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 but more general model for intrusion analysis.

Hollebeek and Waltzman proposed a notion of "suspicion" in modeling uncertainty in intrusion analysis [22]. It appears that the approach does not further differentiate the various meanings that could be associated with a "suspicious" event. The system relies on a "deductive graph" and a "suspicion graph" to propagate certainty levels within the context of the system under concern. But with just a single notion of "suspicion" for each event, it is not clear how to build or interprete the meanings of such graphs in a systematic and consistent manner.

The literature on intrusion alert correlation is vast and the insights into logical causality relations among intrusion alerts have informed our work; [1], [3], [4], [5], [6], [8] are only a few examples. Most of these works model IDS-specific states using pre- and post-conditions that drive a correlation model and rely on the existence of a sparse (nearly deterministic) mapping from alerts to their pre-/post-conditions. It appears to be difficult to model in this manner ubiquitous alerts such as "abnormally high network traffic" that could be indicative of any of a wide variety of possible conditions. 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 semantics. Such flexibility is important when the evidence is tenuous and subject to multiple interpretations.

Our pre-processing step includes data reduction based on clustering and simple correlation of local observations. Much previous work in IDS has addressed this important problem [5], [8] (including the "layered approach" of Martignoni *et al.* [23]). We intend to use all applicable tools and approaches from these works.

VII. Conclusion

We presented an empirical approach to modeling uncertainty in intrusion analysis 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 completely different data sets 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.

Acknowledgment

This work was partially supported by the U.S. National Science Foundation under Grant No. 0716665. Any opinions, findings and conclusions or recomendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would like to thank Abhinav Pathak from Purdue who provided us the Honeypot data for evaluation.

References

[1] A. Mounji, "Languages and tools for rule-based distributed intrusion detections," Ph.D. dissertation, Purdue University, September 1997.

[2] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C. lin Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur, "Dids (distributed intrusion detection system) - motivation, architecture, and an early prototype," in *In Proceedings of the 14th National Computer Security Conference*, 1991, pp. 167–176.

[3] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling multistep cyber attacks for scenario recognition," in *DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C., 2003, pp. 284–292. [Online]. Available: http://www.sdl.sri.com/papers/cheung-lindqvist-fong-discex3-cr/

[4] F. Cuppens and A. Miège, "Alert correlation in a cooperative intrusion detection framework," in *IEEE Symposium on Security and Privacy*, 2002.

[5] P. Ning, Y. Cui, D. Reeves, and D. Xu, "Tools and techniques for analyzing intrusion alerts," *ACM Transactions on Information and System Security*, vol. 7, no. 2, pp. 273–318, May 2004.

[6] S. Noel, E. Robertson, and S. Jajodia, "Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances," in *20th Annual Computer Security Applications Conference (ACSAC 2004)*, 2004, pp. 350– 359.

[7] F. Valeur, "Real-Time Intrusion Detection Alert Correlation," Ph.D. dissertation, University of California, Santa Barbara, May 2006.

[8] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "A Comprehensive Approach to Intrusion Detection Alert Correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 146–169, 2004.

[9] Y. Zhai, P. Ning, P. Iyer, and D. S. Reeves, "Reasoning about complementary intrusion evidence," in *Proceedings of 20th Annual Computer Security Applications Conference (ACSAC)*, December 2004, pp. 39–48.

[10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1999.

[11] G. Shafer, *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[12] M. Almgren, U. Lindqvist, and E. 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.

[13] T. M. Chen and V. Venkataramanan, "Dempster-shafer theory for intrusion detection in ad hoc networks," *IEEE Internet Computing*, 2005.

[14] G. Modelo-Howard, S. Bagchi, and G. 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.

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

[16] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about Datalog (and never dared to ask)," *IEEE Transactions Knowledge and Data Engineering*, vol. 1, no. 1, pp. 146–166, 1989.

[17] P. Rao, K. F. Sagonas, T. Swift, D. S. Warren, and J. 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)*. Dagstuhl, Germany: Springer Verlag, July 1997, pp. 2–17.

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

[19] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 3, pp. 186–205, 2000.

[20] C. Drummond and R. Holte, "Severe class imbalance: Why better algorithms aren't the answer," in *Machine Learning: ECML 2005*, ser. Lecture Notes in Computer Science. Springer US, 2005, vol. 3720, pp. 539 – 546.

[21] 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.

[22] T. Hollebeek and R. Waltzman, "The role of suspicion in model-based intrusion detection," in *Proceedings of the 2004 workshop on New security paradigms*, 2004.

[23] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. Mitchell, "A layered architecture for detecting malicious behaviors," in *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*. RAID, September 2008.