ABSTRACT

ZHAI, YAN. Integrating Multiple Information Resources to Analyze Intrusion Alerts. (Under the direction of Associate Professor Peng Ning).

Intrusion detection systems (IDSs) are important components of network security. However, it is well known that current IDSs generate large amount of alerts, including both true and false alerts. Other than proposing new techniques to detect intrusions without such problems, this thesis presents some work we have done in improving the study of IDS alerts by incorporating other sources of relevant information. In particular, the work covers four issues.

The first issue is to integrate and reason about IDS alerts as well as reports by system monitoring or vulnerability scanning tools (discussed in Chapter 3). To facilitate the modeling of intrusion evidence, this approach classifies intrusion evidence into either *event-based evidence* or *state-based evidence*. Event-based evidence refers to observations (or detections) of intrusive *actions* (e.g., IDS alerts), while state-based evidence refers to observations of the *effects* of intrusions on system states. Based on the interdependency between event-based and state-based evidence, we developed techniques to automatically integrate complementary evidence into Bayesian networks, and reason about uncertain or unknown intrusion evidence based on verified evidence.

The second issue is the study of the robustness of the Bayesian analysis framework toward inaccuracies in the assignments of prior confidence with sensitivity analysis and qualitative analysis (discussed in Chapter 4). By performing sensitivity analysis and qualitative analysis on the Bayesian networks used to reason about intrusion evidence, we can measure or approximate individual evidence's influence on the reasoning results. Such study on the framework's robustness properties can provide guide line for evidence collection and analyses.

The third issue is to improve alert correlation by integrating alert correlation techniques with OS-level object dependency tracking (discussed in Chapter 5). With the support of more detailed and precise information from OS-level event logs, higher accuracy in alert correlation can be achieved. The chapter also discusses the application of such integration in making hypotheses about possibly missed attacks.

The fourth issue is to correlate intrusion alert and other security event information from multiple heterogeneous sources while protecting the privacy for each participating parties (discussed in Chapter 6). Based on a sanitization scheme utilizing both generalization and randomization, we proposed several techniques to flexibly balance between the privacy protection and the analysis

capability of the sanitized data. We also studied the various analyses supported by the sharing framework and its security against some different types of attacks.

Finally, the conclusion of my dissertation is provided and future work is pointed out.

Integrating Multiple Information Resources to Analyze Intrusion Alerts

by

Yan Zhai

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, NC

2006

Approved By:

Dr. S. Purushothaman Iyer	Dr. Ting Yu
Dr. Peng Ning Chair of Advisory Committee	Dr. Douglas S. Reeves

This paper is dedicated to my parents and my wife without whom none of this would have been even possible.

Biography

Yan Zhai received a Bachelor's degree on electrical engineering from Peking University and a Master's degree on computer engineering from North Carolina State University. He is a Ph.D. student in the Department of Computer Science at North Carolina State University from 2002 to 2006. His research interests are information and network security. In particular, he is interested in intrusion detection techniques, belief reasoning techniques on information security data, and privacy protection techniques.

Acknowledgements

With a deep sense of gratitude, I wish to express my sincere thanks to my Ph.D. advisor, Dr. Peng Ning, for the stimulating advice and patience that he had for me during my Ph.D. research. The training I received from him is priceless. I would also like to thank my committee members Dr. S. Purushothaman Iyer, Dr. Douglas S. Reeves and Dr. Ting Yu for their valuable comments, suggestions, and help.

My former colleagues in the Cyber Defense Laboratory supported me in my research work. I want to thank Yiquan Hu, Qinglin Jiang, Donggang Liu, Dingbang Xu, Pai Peng, Kun Sun, Pan Wang, Qing Zhang, Yi Zhang, and Qinghua Zhang for their help during my study in the lab.

In addition to the people in NCSU, I would like to thank many friends who have helped during this long journey. Special thanks to my friends Mr. Jim Bidzos and Mr. Scott Curry, who have given me invaluable help and advice during the past years.

Finally, I am deeply grateful to my parents Qibin Zhai and Xiaoning Shi, as well as my lovely wife Di Lu. This thesis work would not be possible without their continued support and encouragement during the past years in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation of This Work

With the fast development of computer technologies and the Internet, Our lives have never been more closely related with the computer networks. At the same time, network security has also been an increasingly serious problem for everyone in today's cyber-world. Intrusion detection systems (IDSs), which look into the activities performed in computer systems and networks for malicious behaviors, are usually considered as the last layer of defense in today's network security infrastructure.

In the last two decades, various techniques have been proposed and implemented to detect network intrusions. However, current intrusion detection techniques are still far from satisfactory. Firstly, the amount of alerts generated by an IDS are usually overwhelmingly large. Secondly, because misuse detection cannot detect unknown attacks, while anomaly detection can often be disguised by stealthy attacks, an IDS may miss many intrusions. This type of mistakes are often called *false negatives*. Thirdly, among the alerts generated by IDSs, many of them do not correspond to real malicious activities. This type of mistakes are often called *false positives*. These problems make the task of analyzing IDS alerts very difficult and headache-causing for thousands of security administrators all over the world. Thus, automatic alert post-processing tools are much needed to effectively make use of current IDSs.

Alert correlation is looked on as one effective technique to make the task of analyzing

IDS alerts easier. Many alert correlation techniques have been proposed to facilitate the analysis of intrusion alerts, including those based on the similarity between alert attributes [58, 51, 17, 13], previously known attack scenarios [19, 18], and prerequisites and consequences of known attacks [55, 39, 14].

However, most of these correlation methods focus on IDS alerts, overlooking other information sources of intrusion evidence such as system monitoring tools (e.g., anti-virus software), vulnerability scanning tools (e.g., Nessus [4], SATAN [22], Nmap [24]), and OS-level dependency tracking tools (e.g., Backtracker [31]). Since none of the above methods can perfectly construct attack scenarios due to the imperfection of the IDSs, it is desirable to include additional, complementary intrusion evidence to further improve the performance of intrusion analyses.

Also, because of increasingly large number of *distributed* attacks, including fast propagating malwares (worms) and attacks launched through botnets (zombie nets), sharing the information from multiple intrusion monitoring devices are necessary to facilitate the detection and analysis toward such attacks. Unavoidably, privacy becomes a big concern within the sharing and cooperation among multiple entities. Thus, it is also necessary to study the privacy issues in security information sharing and to develop sharing techniques that can protect privacy information.

1.2 Summary of Current Results

In this work, we will first present our techniques developed to integrate complementary intrusion evidence into IDS alert analysis, and then we will present our work on privacy-preserving alert sharing and analysis.

In general, our results in integrating complementary intrusion evidence into IDS alert analysis and privacy-preserving alert sharing include the following components:

1. **A Bayesian framework to automatically integrate and reason about intrusion alerts and local system state information.** The local system state information can be reports from system monitoring tools, vulnerability scanning tools, and human observations. This approach is based on the interdependency between attacks and system states. That is, an attack may need certain system states to be successful, and will modify the system states as a result. However, IDS alerts, which represent detected attacks, are uncertain due to the imperfection of current IDSs. To reason about uncertain IDS alerts, our approach automatically builds Bayesian networks that consist of variables representing IDS alerts and system states. With additional

complementary evidence about system states provided by system monitoring tools, vulnerability scanning tools, and human observations, we can then make further inference about uncertain IDS alerts. As a result, we can increase our confidence in alerts corresponding to successful attacks, and at the same time reduce the confidence in false alerts. Moreover, by combining system state evidence, we can also make reasonable hypotheses about attacks possibly missed by the IDSs.

2. **The robustness analysis on the Bayesian networks used to integrate multiple sources of intrusion evidence.** The robustness issue is important to the Bayesian framework because the prior confidence of intrusion evidence is sometimes hard to evaluate, as well as the verification of many attribute values are complicated and computational expensive. Thus, the robustness analysis results can be a guide line for evidence collection and analysis.

3. **A practical technique to improve alert correlation by integrating alert correlation techniques with OS-level object dependency tracking.** This approach focuses on using the OS-level object dependency to verify the soundness of the causal relationships identified in alert correlation, as well as to identify the causal relationships missed in alert correlation. As the OS-level object dependency tracking is much more reliable than IDS alerts and the correlation based on such alerts, the integration can dramatically increase the accuracy of the causal-relationship-based correlation results.

4. **A general framework to sanitize and share security event information for privacy-preserving analysis.** Sharing security event information is necessary to study the emerging attack trends and distributed security incidents, but the privacy issue with the shared security events is a big obstacle for people to do so. In this work, we first modeled the security event information to be shared and our analysis goals with those shared data. Then, using the sharing of IDS alerts as an example, we proposed a series of methods to sanitize the identity information in alert data and ways to perform various correlation analysis on the sanitized alert data.

1.3 Dissertation Organization

The rest of this dissertation is organized as follows: Chapter 2 introduces the background knowledge about our work including the related work in computer security and some mathematical tools used in our work. Chapter 3 presents the Bayesian framework for combining complementary

intrusion evidence. Chapter 4 provides our analysis on the robustness of the alert-attribute Bayesian networks used to reason about intrusion evidence. Chapter 5 discusses the technique to integrate OS-level event logging and dependency tracking to IDS alert correlation. Chapter 6 presents our sanitization and sharing framework for privacy-preserving security event data sharing. In the end, chapter 7 concludes.

Chapter 2

Related Work

Our techniques are closely related to intrusion detection and intrusion alert correlation techniques, as well as techniques in some other fields of research such as vulnerability analysis, privacy protection, and Bayesian inference. In the following, we will briefly introduce some related work in those research areas.

2.1 Intrusion Detection

Many techniques have been developed and applied to counter security threats to computer systems and networks, e.g., firewalls, anti-virus tools, and access control. However, every technique has its own limitations and will fail at some certain point. It is important to have the administrators alarmed when intrusions take place. An Intrusion Detection System (IDS) is such a security mechanism that monitors computer systems and networks to detect and alarm on those intrusions that pass through other protection mechanisms. Thus, upon receiving the alarms, administrators or automatic response systems can make quick response to those attacks to minimize their damages.

Based on the scope of security-relevant data monitored, intrusion detection systems can be categorized into host-based or network-based. Host-based IDSs reside at individual hosts and monitors incoming/outgoing traffic as well as operating systems and application audit trails, while network-based IDSs monitor activities over network connections and analyze the network traffic.

Based on the detection techniques, intrusion detection can also be classified into two main

categories: *misuse detection* and *anomaly detection*.

2.1.1 Misuse Detection

Misuse detection [52, 46, 27, 41] detects attacks by matching the audit data to pre-established signatures (patterns) of known attacks, which is similar to the way virus scanning tools work. For example, when an IDS such as Snort [46] sees the pattern of "$/..\%c1\%1c../$" in http requests, it will raise alert for web IIS unicode directory traversal attack. Misuse detection can be further divided into two categories: rule-based and state-based.

Rule-based detection detects intrusions by encoding intrusion activities as a set of rules and comparing the audit data with these rules. Alarms are raised when matchings between audit data and rules are found by the detection engine. Example IDSs falling into this category are MIDAS (Multics Intrusion Detection and Alerting System) [47], Snort [46], and Bro [41].

State-based detection defines attack patterns with state transition diagrams. In particular, in a state transition diagram, nodes represent system states and arcs represent actions causing state transitions. A state transition diagram that defines an intrusion scenario consists of an initial state, a number of legitimate states, and a number of compromised states. Starting from the initial state, the detection engine keeps the record of the current status of the state machines upon monitored transitions, and alarms the administrator when a compromised state is reached. Example IDSs falling into this category is USTAT (UNIX State Transition Analysis Tool) [26].

In general, misuse detection techniques are usually fast and accurate (compared with anomaly detection), but not effective in detecting novel attacks which do not correspond to any signatures in the signature knowledge bases.

2.1.2 Anomaly Detection

Anomaly detection [49, 25, 33, 8, 61] detects intrusions by monitoring the deviations of system and network activities from pre-established profiles of "normal" behaviors. Anomaly detection approaches build models to define the "normal" profiles and measure the deviations from such profiles. Such profiles can be statistical models based on the user's historical activities and system call models for the normal execution of programs. When the deviation from normal profiles is significant enough, it will be looked on as abnormal and alerts will be raised.

Examples of anomaly detection systems using user activity profiles include SRI's IDES

[36] and NIDES [8], and W&S [57]. In IDES [36] and NIDES [8], statistics are monitored on subjects such as CPU utilization, frequencies of specific system events, distribution of audit records, and etc. When the deviation exceeds the pre-defined statistical threshold, it is determined to be abnormal and alarm be raised. W&S [57] automatically creates detection rules based on the statistics of historical audit records, and use those rules to detect abnormal user activities.

Examples of those using program execution profiles include [23, 60, 61, 48]. Those approaches build their system call profile through either program training (monitoring normal program executions), or static analysis on programs' source codes and execution.

Anomaly detection techniques have the capability to detect novel attacks since they do not require prior knowledge about malicious activities at first place. However, there are several problems. Firstly, it is difficult to accurately build profiles for "normal" activities. Normal behaviors vary widely and tend to change with time, and it is also hard to assure the audit data to be 100% "clean" during training periods. Secondly, because normal behaviors vary widely, anomaly detection systems need to tolerate minor deviations from normal profiles. There for stealthy attacks which cause less-than-threshold deviations in the detection engine can pass through. It is also possible for attackers to "train" the anomaly detection system to accept intrusive activities as normal. Due to these problems, anomaly detection techniques are prone to missing lots of attack as well as generating a lot of false alarms.

It is obvious that misuse detection and anomaly detection are complementary to each other. Combining the two different detection techniques can improve the detection rate of intrusion detection systems. Many systems deploy both techniques to facilitate the intrusion detection. For example, NIDES [8] has both a statistical-based anomaly detection module and a rule-based misuse detection module.

2.2 Intrusion Alert Correlation

Given networks and systems with IDS sensors deployed, it is important for security administrators to effectively construct and analyze high-level aggregated attack information (e.g., attack clusters and scenarios) from the usually overwhelmingly large amounts of IDS alerts. Alert correlation, a process to analyze and correlate security alerts to provide an aggregated information on the networks and systems under protection, is looked on as an effective method to address this issue.

Several techniques of alert correlation and attack scenario analysis have been proposed in recent years, and can generally be classified into three categories: correlation based on similarities between the alert attributes, correlation based on pre-defined attack scenarios, and correlation based on the prerequisites and consequences of attacks.

Valdes and Skinner [58] proposed a probabilistic approach to correlate alerts by comparing the similarities of alert attributes. In particular, the similarity between an alert and "meta alerts" are evaluated by computing and normalizing the values of similarity functions of all the comparable alert attributes. Also based on the similarity between alerts, Cuppens [13] proposed to use clustering techniques to correlate similar alerts into clusters, and Julish et al. [30, 29] proposed to use hierarchical conceptual clustering technique to correlate alerts.

Attack scenarios are very useful information in forensic analysis. Instead of clustering alerts. Some researchers proposed to correlate alert by matching them against pre-defined attack scenarios, and developed attack specification languages such as LAMBDA [15] and STATL [21]. However, this category of correlation techniques cannot correlate novel attack strategies.

In order to correlate alerts into attack scenarios while not limited by pre-defined attack scenarios, some researchers proposed techniques to correlate alerts based on the causal relationships between alerts. Templeton and Levis [55], Cuppens and Miège [14], and Ning et al. [39] proposed correlation techniques based on the prerequisite and consequence of individual alerts. In particular, instead of pre-defined attack scenarios, each individual type of attacks' prerequisite (pre-conditions) and consequence (post-conditions) are specified before correlation. The correlation engine correlates alerts together by matching previous alerts' consequences to later alerts' prerequisites. Because our work in alert correlation is closely related with the correlation technique proposed in [39], we will introduce this technique in further detail later in this section. Qin and Lee [45] proposed a Bayesian method to reason about the causality between alerts, which requires less expert knowledge than assigning each individual attack's prerequisite and consequence, and can be complimentary to the correlation methods in [55, 39, 14]. Based on the correlation result in [39], Ning et al. [40] further extended the technique to make hypotheses about missed attack steps.

Most of these correlation methods have limited capabilities because they correlates alerts solely based on the IDS alerts, which are generically of low quality. Several researchers recently investigated ways to consider multiple information sources during intrusion analysis [38, 44]. A formal model named M2D2 was proposed to represent data relevant to alert correlation, including characteristics of monitored systems, properties of security tools, and observed events [38]. Though quite useful for alert correlation, M2D2 does not provide a specific mechanism to automatically

reason about information provided by multiple sources. Another mission-impact-based method [44] reasons about the relevance of alerts by fusing alerts with the targets' topology and vulnerabilities, and ranks alerts based on their relationships with critical resources and users' interests. Though the mission-impact based method can automate the analysis of intrusion alerts, the construction of a mission-impact based model requires substantial human intervention, and the constructed model is highly dependent on the monitored systems. Thus, it is desirable to seek other effective mechanisms that can handle complementary intrusion evidence automatically.

2.2.1 Alert Correlation Based on Prerequisites and Consequences of Attacks

Because we adopted the alert correlation method in [39] in our reasoning framework, here we give a brief overview of the technique. This method correlates intrusion alerts using the prerequisites and consequences of attacks. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful. For example, the existence of a vulnerable service is the prerequisite of a remote buffer overflow attack against the service. The consequence of an attack is the possible outcome of the attack. For example, gaining the root access from a remote machine may be the consequence of a ftp buffer overflow attack. In a series of attacks where earlier ones are launched to prepare for later ones, there are usually connections between the consequences of the earlier attacks and the prerequisites of the later ones. Accordingly, we identify the prerequisites (e.g., existence of vulnerable services) and the consequences (e.g., gain certain privilege) of attacks, and correlate the detected attacks (i.e., alerts) by matching the consequences of previous alerts to the prerequisites of later ones.

The correlation method uses logical formulas, which are logical combinations of predicates, to represent the prerequisites and consequences of attacks. The correlation model represents the attributes, prerequisites, and consequences of known attacks as alert types. The correlation process is to identify the *prepare-for* relations between alerts, which is done with the help of prerequisite sets and expanded consequence sets of alerts. Given an alert, its prerequisite set is the set of all predicates in the its prerequisite, and its expanded consequence set is the set of all predicates in or implied by its consequence. An earlier alert t_1 *prepares for* a later alert t_2 if the expanded consequence set of t_1 and the prerequisite set of t_2 share some common predicates. Intuitively, an earlier alert prepares for a later one if the consequence of the earlier one "contributes" to the prerequisite of the later one. An alert correlation graph is used to represent a set of correlated alerts. An alert correlation graph $CG = (N, E)$ is a connected directed acyclic graph, where N is a set of

alerts, and for each pair $n_1, n_2 \in N$, there is a directed edge from n_1 to n_2 in E if and only if n_1 prepares for n_2.

The advantage of this method is that the correlation result is easy to understand and directly reflects the possible attack scenarios. However, as the correlation is solely based on IDS alerts, the result highly depends on the quality of the IDS alerts. For example, the result may contain false correlations when there are false alerts.

2.2.2 Privacy-preserving Alert Sharing and Correlation

As large-scale distributed attacks such as worms, spams, DDoS, and botnets are becoming ever more popular, it has been well recognized that security systems such as intrusion detection systems (IDSs) need to be deployed at different locations over the Internet to collect and analyze security related data from multiple networks. However, data generated by security systems may include sensitive information (e.g., host addresses and packet payloads for some traffic) that data owners do not want to disclose or share with other parties. Because of such privacy concerns, although many security systems are already widely deployed over the Internet, most organizations and institutions owning those devices will not submit their security data without sanitizing the data first, which will affect the performance of current alert correlation approaches. Also, many institutions and organizations have the willingness to share their security data for government or third-party researchers to study if they can guarantee the privacy among those data is protected. Thus, it is desirable to have a sharing framework to sanitize, share, and correlate the security data from multiple organizations over the Internet.

Recently, some researchers and organizations have proposed several approaches to address this problem. DShield [56] collects and analyzes firewall logs from all over the network to discover new attack trends and better firewall rules. The client programs provided by DShield allows the user to perform partial or complete obfuscation (replacing the address with its /24 network address or a fixed value) over destination addresses of the dataset before submission. This approach can detect attack trends with high volume of traffic. However, it does not support higher level correlation between organizations due to its rough obfuscation scheme and low level data collected.

Lincoln et al. [35] proposed a log repository framework that enables community alert aggregation and correlation, while maintaining privacy for alert contributors. However, the anonymization scheme in the paper is partially based on hashing IP addresses, which is always vulnerable to dictionary attacks. Also, their approach does not support the correlation between repositories due

to separate re-keying mechanisms.

Xu and Ning [64] proposed to sanitize original values to more general values (e.g., IP addresses are replaced by network addresses), and proposed a series of techniques to correlate such generalized alerts. However, attackers can still extract useful information from generalized data when there's a large enough amount of data available. More over, in some cases, the uncertainty introduced by generalization can be excessive and cause problems in alert correlation.

Most recently, Xu and Ning [65] proposed to inject artificial alert data into the original data sets and use attribute perturbation to protect hosts' privacy. However, although some correlation methods such as clustering analysis will not be affected much by this approach, some other correlation methods, such as those trying to learn the attack scenarios, may be significantly affected by the injected artificial alerts and altered attributes. Also, the overhead introduced by injecting artificial alerts is considerably large.

2.3 OS-level Object Dependency Tracking

Dependencies among OS-level objects, such as files and processes, are established when they interact with each other during system operations. For example, a process writing to a file establishes the dependency from the process object to the file object, and a process reading from a file established the dependency from the file object to the process object. Because most attacks include OS-level operations. Tracking the dependencies among OS-level objects can also help us reason about the causal relationships between correlated alerts. Such OS-level dependencies can be tracked by monitoring the dependency causing system events such as the file read/write system calls.

Backtracker is an OS-level dependency tracking tool [31]. It monitors specific types of OS-level objects, i.e., processes and files. The objects are kept in a log with their properties such as the uid of the objects. It also monitors specific dependency-causing system calls like process forking, file reading, and memory sharing, which together are called "high-control events" in [31]. Given the information of a specific object such as the pid of a process or the $inode$ number of a file, Backtracker identifies the previous objects and system calls that could have potentially affected a target object, and displays chains of events in a dependency graph. In a Backtracker dependency graph, each node A represents an OS-level object, and each edge $A \rightarrow B$ represents that object B is dependent on object A. Moreover, an edge $A \leftrightarrow B$ is used to represent that objects A and B are

potentially dependent on each other. In its later version[32], the tool can also track dependencies between remote hosts by tracking the logged socket ID. As mentioned earlier, the major limitations of Backtracker are the complexity of its results and the inconvenience to use because of its dependency on the availability of "detection points".

Although Backtracker does not monitor all kinds of OS-level events, the process, file, and filename objects are among the most important elements in most attacks' prerequisites and consequences. The "high-control events" monitored by the Backtracker are the essential methods to modify the system attributes. Finally, although OS-level event logging can be disrupted by kernel-level attacks, kernel-level attacks are a lot more difficult and uncommon. Thus, it is reasonable to assume that most attacks will have their corresponding OS-level events logged by Backtracker.

2.4 Other Related Topics

2.4.1 Vulnerability Analysis

In our reasoning framework, some vulnerability/system scanning tools (e.g., Nessus [4], XScan [63] and Nmap [24]) are used to provide necessary evidence for the belief update in Bayesian reasoning.

Some approaches presented in this thesis are also related to the recent results on vulnerability analysis. In particular, the methods in [50, 7] also model system state as system attributes, and attacks as atomic transformations that establish post-condition given the attacks' pre-condition, which is the essential causal relationships that our Bayesian reasoning framework is built on.

2.4.2 Bayesian Belief Networks

In our reasoning framework, various sources of intrusion evidence are integrated together with Bayesian networks [28].

A Bayesian network is a graphical model that encodes a joint probability distribution over a set of random variables. It consists of a network structure S that represents the conditional independence among the variables, and a set of local *conditional probability distributions* (CPD) associated with each variable. S is a directed acyclic graph (DAG), where each node in S represents a random variable, and each edge in S represents immediate influence from the parent (the tail node

of the edge) to the child (the head node of the edge). The joint probability distribution of all the random variables in S complies with the Markov condition (each variable is conditional independent with all its nondescendents given its parents).

With the joint probability distribution encoded by a Bayesian network, we can do probabilistic inference to compute the probabilities of the variables that we are interested in. Several researchers have developed probabilistic inference algorithms for Bayesian networks (e.g.,[42, 34, 53]).

Because a Bayesian network has both causal and probabilistic semantics, it is ideal for combining prior knowledge and data to make predictions upon partial observations. During the last decade, Bayesian networks have been widely used in expert systems and artificial intelligence.

Chapter 3

Reasoning about Complementary Intrusion Evidence

In this chapter, we present techniques to automatically integrate and reason about complementary intrusion evidence, including IDS alerts, reports from system monitoring or vulnerability scanning tools, and human observations. Our approach is based on the interdependency between attacks and system states. That is, an attack may need certain system states to be successful, and will modify the system states as a result. However, IDS alerts, which represent detected attacks, are uncertain due to the imperfection of current IDSs. To reason about uncertain IDS alerts, our approach automatically builds Bayesian networks that consist of variables representing IDS alerts and system states. With additional, complementary evidence about system states provided by system monitoring tools, vulnerability scanning tools, and human observations, we can then make further inference about uncertain IDS alerts. As a result, we can increase our confidence in alerts corresponding to successful attacks, and at the same time reduce the confidence in false alerts. Moreover, by combining system state evidence, we can also make reasonable hypotheses about attacks possibly missed by the IDSs.

The main contribution of this work is a reasoning framework for complementary intrusion evidence. To our best knowledge, this was the first attempt to *automatically* integrate and reason about complementary intrusion evidence such as IDS alerts and vulnerability scanning reports. We

also performed a series of experiments to validate our approach and gain further insights into the problem. The experimental results demonstrate the potential of the proposed approach as well as the effectiveness of our techniques.

3.1 Reasoning Framework

In this section, we present our techniques to reason about complementary intrusion evidence, including IDS alerts and reports from system monitoring tools or vulnerability scanning tools. In the following, we first describe our representation of intrusion evidence, and then present the framework to reason about complementary intrusion evidence using Bayesian networks.

3.1.1 Modeling Intrusion Evidence

We classify intrusion evidence into two categories: *event-based evidence* and *state-based evidence*. Event-based evidence refers to observations (or detections) of attacks. For example, an IDS alert of a buffer overflow attack against `sshd` is event-based evidence. State-based evidence refers to observations of the *effect* of attacks on system states. For example, the existence of a rootkit[1] on a machine is state-based evidence indicating that the machine has been compromised.

System Attributes and State-Based Evidence

We follow [50, 7] to represent system states (e.g., vulnerabilities, user access privileges, and network connectivities) as *system attributes* (or simply *attributes*), each of which is a boolean variable representing the system's state. Notation-wise, we use a system attribute directly to represent that it is True, and use its negation to represent that it is False. There may be implication relationships between attributes. For example, `RootPrivilege` implies `FileTransferPrivilege`, which indicates that an attacker having the root privilege also has the privilege to transfer files from/to the system. Note that such a representation can be extended to include variables to provide more flexibility. For example, we may use `RootPrivilege(x)` to represent the attacker has acquired root privilege on host `x`. However, for simplicity, we do not do so in this paper.

[1] A rootkit is a collection of tools (programs) that a hacker uses to mask intrusion and obtain administrator-level access to a computer or computer network (`http://searchsecurity.techtarget.com`).

State-based evidence consists of observations on system attributes related to possible attacks. They may be collected by system scanning tools. We refer to the change of an attribute as an *attribute alteration*. Attribute alterations can be detected by system monitoring tools, comparing the system scanning reports, and human observations. The *timestamp* of an attribute alteration is the time when the alteration is detected or inferred. Such a timestamp can be stored together with each attribute alteration.

For convenience, we refer to the probability for a system attribute to be True as the *confidence* in the attribute. When a system attribute is in negation form, the confidence in the attribute is the probability that the negation form is True. Compared with IDS alerts, reports by scanning/monitoring tools are more reliable due to the verifiable nature of most system attributes. We can assume the confidence in a verifiable attribute is 1. However, some system attributes may not be verifiable because of the absence of an appropriate scanner. In addition, some system attributes are difficult to check due to the security policy on the target system or performance reasons. In such cases, unless we have any further knowledge or evidence about the attribute, we assume the confidence in such an attribute is 0.5. Intuitively, this represents the lack of information about the state of the attribute.

Event-Based Evidence

Typical sources of event-based evidence include event logs, IDS alerts, network traffic logs, system call logs, etc. Different kinds of logs provide event-based evidence on the system in different granularities and toward different aspects of the system. In this paper, the only event-based evidence we consider is IDS alert, which is in a coarser granularity but more understandable by human compared with other types of system logs. We will use IDS alerts and event-based evidence interchangeably in the rest of the paper. Our representation of IDS alerts is closely related to our model of attacks. Thus, we first introduce our representation of attacks before discussing IDS alerts.

Similar to [7, 50], we model an attack as an atomic transformation that establishes a set of system attributes called *post-condition*, given a logical condition called *pre-condition* over system attributes. Intuitively, if the pre-condition of an attack is satisfied, the attack can then transform the system into the state specified by its post-condition. An IDS raises an *alert* when it detects an instance of an attack. IDS alerts are not exactly the attacks launched toward the target due to the imperfection of current IDSs. An IDS may report a false alert or miss an actual attack.

To facilitate the reasoning about IDS alerts, we use the prior confidence of each attack to

represent its quantitative property. The *prior confidence of an attack type T*, denoted *Pr(T)*, is the prior belief we have about the probability for a corresponding alert to represent an actual type *T* attack. Thus, given a type *T* alert e, the *prior confidence of* e is *Pr(e) = Pr(T)*. The prior confidence of each type of attack can be gathered by analyzing historical data. It represents our prior knowledge about IDS alerts based on our previous experience. One may observe that the probability for each attack type varies during different time period as they are dependent on not only the quality of the IDSs, but also the attack frequency and background activities in the network. However, in the later part of this paper, we will see that our reasoning approach is still useful despite the dynamic nature of the prior confidences, as it reduces the uncertainty of intrusion evidence when additional verified evidence is considered. In some sense, *Pr(T)* is the *belief* that a type *T* alert is a real instance of attack, and our reasoning framework is to increase or decrease our belief in alerts based on complementary intrusion evidence. Similar to the confidence in a system attribute, we refer to the probability that an IDS alert corresponds to a *successful attack* as the *confidence* in the alert.

We summarize our prior knowledge about IDS alerts and attacks below:

- An IDS alert *e* of attack type *T* has the probability *Pr(T)* to be a real attack;
- A real attack *E* has probability 1 to be successful when its pre-condition is satisfied by the system attributes before the attack happens;
- A real attack *E* has probability 0 to be successful if its pre-condition is not satisfied by the system attributes before the attack happens;
- The attributes in the post-condition of a successful attack *E* are True after the attack happens.

3.1.2 Basic Reasoning framework

In normal situations, a system should stay in a legitimate state. Starting from a legitimate system state, an attacker may launch a sequence of attacks to get the system into some intermediate states, and finally into the attacker's objective state. It is easy to see that there exist causal relationships among attacks and system attributes. Our approach is to use these causal relationships to reason about complementary IDS alerts and system attributes reported by scanning/monitoring tools. Specifically, we organize IDS alerts and system attributes into Bayesian networks [28] based on those causal relationships, and use these Bayesian networks to reason about complementary intrusion evidence.

Network Structure

To identify and represent these causal relationships, we integrate IDS alerts with system attributes based on the pre-condition and post-condition of attacks. Specifically, we place IDS alerts, available system attributes, and system attributes possibly modified by the corresponding attacks into a directed graph, which we call an *alert-attribute network*.

Each node in such a graph is a binary variable representing either an IDS alert or a system attribute. When a node represents a system attribute, it can denote either a piece of state-based evidence (e.g., scan report), or an inferred attribute alteration caused by an IDS alert. Each node is timestamped. The timestamp of an alert node is the time when the corresponding attack takes place, while the timestamp of an attribute node is the time when the attribute alteration is observed or inferred. For convenience, we reuse the alert or attribute name as the variable name.

All edges in the graph are directed. An edge from an alert node to an attribute node represents that the corresponding attack changes the system attribute into this new state. An edge from an attribute node to an alert node represents that the attribute is a part of the pre-condition of the corresponding attack. An edge from an attribute node to another attribute node represents that the first attribute implies the second attribute. There are no edges that connect two alert nodes together directly.

We construct such a graph starting with the initial system state, which is represented in the graph as a set of attribute nodes corresponding to the initial attributes. As time goes by, new IDS alerts and system monitoring reports are raised. When a new IDS alert is reported, a corresponding alert node is added into the graph only if the alert's pre-condition is evaluated to be True given the attributes presented in the graph by the time. Also, edges are added from the latest attribute nodes corresponding to the attributes in the alert's pre-condition to the newly generated alert node (to represent the causal relationships). To serve the same purpose, edges from the alert node to its post-condition attribute nodes are also established when they are created. For each attribute node in the alert's post-condition, if nodes related to the same attribute already exist in the graph, which could either be caused by some previous alerts or reported by system monitoring tools, an edge from the latest such node to the new node is added to represent the implication relationship. By doing so, each attribute node in the graph represents the accumulative effects on the attribute of all the prior related alerts. Note that the construction and analysis processes can be done off line following the time sequence of the evidence in IDS alert logs and scan reports.

Figure 3.1 shows an example alert-attribute network, which is constructed as discussed

above. The gray nodes represent initial or updated system attributes, and the white nodes represent IDS alerts. For simplicity, we do not show the initial system attributes that are not involved in the pre-condition or post-condition of the corresponding attacks. Alert `sshd_buffer_overflow` indicates an attempt to compromise the system through the vulnerable sshd. The pre-condition of `sshd_buffer_overflow` is `sshd_running` $\wedge$ `vulnerable_sshd`, and the post-condition is {$\neg$`sshd_running, root_access`}. Thus, this attempt can be successful since its pre-condition is satisfied in the system state. As a result, this attack introduces two attribute alterations: $\neg$`sshd_running` and `root_access`. In other words, the attacker stops the sshd daemon and gains root access to the system. As shown in Figure 3.1, the attacker then installs a mstream zombie program, changing the attribute `DDoS_daemon_installed` from False to True.

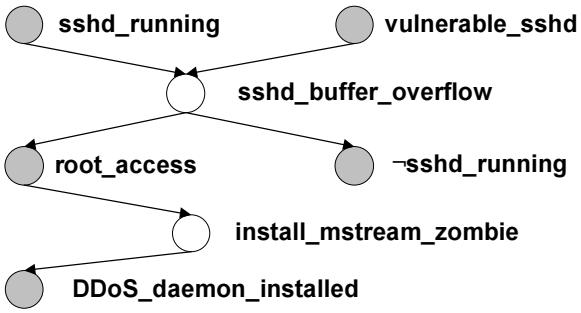


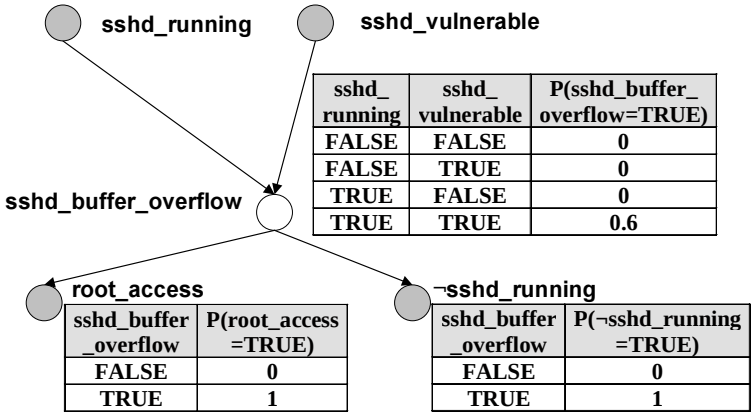
Figure 3.1: An alert-attribute network example

Conditional Probabilities

A Bayesian network is a directed acyclic graph (DAG), where each directed edge represents a direct influence between the two ends of the edge, and each node stores a conditional probability table describing the statistical relationships between the node and its parent nodes [28].

Based on the construction of the alert-attribute network, it is easy to see that a graph constructed in that way is acyclic. Indeed, all the edges are from previously existing nodes to newly added nodes, and thus will not result in any cycle. From our discussion above, the causal relationships among the nodes in an alert-attribute network are obvious. Now we discuss how to determine each node's conditional probability table so that the alert-attribute network becomes a Bayesian network.

When an IDS alert e is reported, the probability for the alert e to be a real attack is $Pr(e)$, the prior confidence of e. The variable e in the alert-attribute network being True represents that the corresponding attack is successful. We assume an attack will succeed if its pre-condition is satisfied. Thus, the probability of e being True is the prior confidence of the corresponding IDS alert when its pre-condition is satisfied, or 0 otherwise. Since the pre-condition of an attack is a logic formula of system attributes, the conditional probability of an alert node can be easily derived. The conditional probability table associated with node `sshd_buffer_overflow` in Figure 3.2 shows such an example, where we assume $Pr($`sshd_buffer_overflow`$)$ = 0.6. Note that the probability of an IDS alert variable being False under these pre-condition can be easily computed from the above probabilities. Thus, we do not include them here.



sshd_ running	sshd_ vulnerable	P(sshd_buffer_ overflow=TRUE)
FALSE	FALSE	0
FALSE	TRUE	0
TRUE	FALSE	0
TRUE	TRUE	0.6

sshd_buffer _overflow	P(root_access =TRUE)
FALSE	0
TRUE	1

sshd_buffer _overflow	P(¬sshd_running =TRUE)
FALSE	0
TRUE	1

Figure 3.2: Conditional probability tables

Conditional probability tables associated with system attributes are even simpler to compute. Indeed, if an IDS alert e represents a successful attack, all the system attributes in its post-condition should turn to True. Otherwise, the system attributes that are False before the IDS alert should remain False. If two attribute nodes of the same attribute are connected together with an edge representing implication relationship, and the earlier one is True, the latter one should also be True. Thus, the conditional probability of a system attribute a being True would be 1 if at least one of its parent variables (either alert nodes or attribute nodes) is True, and 0 if all its parent variables are False (unless it is reported by system scanning/monitoring tools). The tables associated with `root_access` and ¬`sshd_running` show examples of such conditional probabilities. Similar to the above example, we only show the probabilities for the attributes to be True, from which the

probabilities for the attributes to be False can be easily computed.

Reasoning about Intrusion Evidence

The Bayesian networks constructed in this way offer an excellent opportunity to reason about the uncertain intrusion evidence, particularly the IDS alerts. We call those attributes with a confidence value of 1 the verified attributes. The report of such verified attributes are observations of facts. When newly verified attributes are reported by system monitoring/scanning tools, we can use these observations to re-compute the confidence values in the related previous objects in the network with Bayesian inference. For each node in the Bayesian network, its final probability value is the combined result of all the evidence and knowledge. Take the Bayesian network shown in Figure 3.2 as an example. We may be uncertain about an IDS alert reporting a buffer overflow attack against `sshd`, since the IDS has reported the same type of alerts incorrectly in the past. However, if by scanning the system we find that `sshd` is not running properly after the IDS reports this alert, we can then update the confidence in $\neg sshd\_running$ to be 1. Thus, we are more certain about the alert, which caused the attribute alteration. Though human users would do the same reasoning, placing these evidence into Bayesian networks offers additional benefits, since such a reasoning process can then be performed automatically and systematically. Also such reasoning could become too difficult for human users when dealing with very complicated scenarios.

It is easy to see that the more verified state-based evidence we have, the better judgment we can make by reasoning about the uncertain IDS alerts and system states. This suggests that we should monitor the system closer and scan the system more frequently, as system monitoring tools and vulnerability scanning tools usually generate evidence with high confidence value. However, such monitoring and scanning are often expensive and may hurt the other applications by consuming resources. Thus, it is important to determine the right balance for system monitoring and scanning activities. Nevertheless, this problem is out of the scope of this paper. We leave it for future consideration.

Merging Attribute Nodes

As discussed earlier, there may be edges between attribute nodes corresponding to the same attribute, which represent implication relationships between them. We observe that in certain cases, such attribute nodes can be merged without affecting the reasoning about intrusion evidence

in alert-attribute networks. This observation is reflected by Lemma 3.1.1, which is presented next. For the sake of presentation, if two attribute nodes A and B are connected with edge (A, B), we refer to the action of removing node A with all its outgoing edges and redirecting all its incoming edges to node B as *merging* A *into* B.

Lemma 3.1.1 *Consider two attribute nodes* A *and* B *corresponding to the same attribute and connected by an edge* (A, B). *If either there is no other outgoing edge from node* A *or* A *is instantiated (verified), merging* A *into* B *does not change the probability of any other node when reasoning about intrusion evidence.*

Proof:

The proof is divided into two steps. The first step is to prove that merging the two nodes will not affect other nodes in the downward reasoning in the Bayesian network. The second step is to prove that such a merge will not affect the posterior probability values of other nodes in the upward reasoning (belief updating) in the Bayesian network.

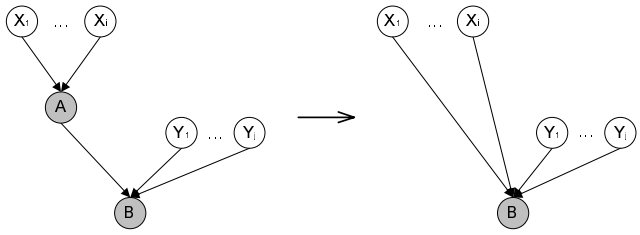


Figure 3.3: Merging two attribute nodes

As shown in Figure 3.3, we assume node A's parent nodes are $X_1, X_2, \ldots, X_i$, node B's parent nodes are $Y_1, Y_2, \ldots, Y_j$ and A, and (A,B) is the only outgoing edge from A.

Since A and B are both attribute nodes, A is True if any of X_1, X_2, ..., X_m is True, and B is True if any of A, Y_1, Y_2, ..., Y_n is True. Thus, B is True if any of A, X_1, X_2, ..., X_m, Y_1, Y_2, ..., Y_n is True. After merging A into B, B's parent nodes are X_1, X_2, ..., X_m, Y_1, Y_2,..., Y_n, and B is True if any of these nodes is True, which is exactly the same as before the merging. Thus, in the downward reasoning process, the probability value of any node other than A in the network remains the same as before merging A into B.

When computing the posterior probability value of a node M in the network after there is additional verified evidence E, the posterior probability can be computed as $P(M|E) = \frac{P(M,E)}{P(E)}$. $P(E)$ and $P(M, E)$ are derived from margining out all the other variables in the joint probability density function $Prob(S)$, where S is the set of all the nodes ($X_1, X_2, \cdots, Y_1, \cdots, M, \cdots$) in the Bayesian network.

We denote the parent nodes of A and B as X_i and Y_i. Because A and B's probability values solely depend on X_i and Y_j, given a set of input $(A, B, \{X_i\}, \{Y_i\}, \cdots)$, the probability value $Prob(A, B, \{X_i\}, \{Y_j\}, \cdots)$ either equals to 0 as A and B cannot be True given $(\{X_i\}, \{Y_j\})$, or equals to $Prob(\{X_i\}, \{Y_j\}, \cdots)$ as A and B are determined to be True given $(\{X_i\}, \{Y_j\})$. Thus, the result of margining out A and B from $Prob(S)$ before merging A into B is:

$$\sum_{B}\sum_{A} Prob(A, B, \{X_i\}, \{Y_j\}, \cdots) = \sum_{B=True} Prob(\{X_i\}, \{Y_j\}, \cdots). \tag{3.1}$$

Similarly, margining out B from the joint probability density function of all nodes after the merge can also be represented as

$$\sum_{B=True} Prob'(\{X_i\}, \{Y_j\}, \cdots), \tag{3.2}$$

where $Prob'(\{X_i\}, \{Y_j\}, \cdots)$ is the joint probability density function of the rest of the nodes in the merged network. Because A and B are solely dependent on $\{X_i\}$ and $\{Y_j\}$, and B's conditional probability table over $(\{X_i\}, \{Y_j\})$ does not change after the merge, formula 3.1 equals to formula 3.2. Thus, the posterior probability of any other node in the network remains the same as before merging A into B.

With Lemma 3.1.1, we can recursively merge attribute nodes that satisfy the condition specified in Lemma 3.1.1 to reduce the complexity of the network structure without affecting the reasoning result.

3.1.3 Alert Aggregation and Abstraction

In reality, IDSs often generate a large number of alerts for the same type of attack. Also, IDSs usually raise different alerts for similar attacks, or variations of the same attack. AS a result, many alert nodes share the same parent nodes and child nodes in the Bayesian network. This introduces two problems. First, the number of entries in conditional probability table of their children

nodes of those alerts is exponential to the number of alerts, which makes it difficult to take advantage of existing Bayesian network tools. Second, the effect of additional evidence will spread over these alerts as we do not know which of them indeed contributes to the modification of system attributes. In practice, we usually do not care about the subtle difference between the alert variations and which particular alert is the actual successful one, but whether at least one of them is successful. Thus, a natural approach to addressing the above problem is to abstract alert variations into one common alert and aggregate such alerts together into one single node, which represents "at least one of the component alerts corresponds to a successful attack".

The conditional probability table of an aggregated alert node can be computed similarly. However, we need to use aggregated prior confidence value P_0', which represents the probability that at least one of its component alerts corresponds to an actual attack. Given n alerts of an abstract attack type T merged into one aggregated alert, the aggregated prior confidence $P_0'(T)$ can be computed as $P_0'(T) = 1 - \prod_{i=1}^{n}(1 - P_0(Type_{a_i}))$, where $a_1, ..., a_n$ are the alerts to be aggregated, and $Type_{a_i}$ is the attack type of a_i.

3.1.4 Hypothesizing about Missed Attacks

When there are missed attacks, the effect of the attacks on the system will not be reflected in the alert-attribute network. As a result, some later alerts corresponding to successful attacks may be considered false. In other words, the current model only works when there are no missed attacks. (Note that this is a common problem shared by almost all alert correlation methods.)

We observe that when successful attacks are missed by IDSs, it is still possible for the system monitoring tools to catch the impact of the attacks on the system states. In other words, we may observe unexpected attribute alterations. Such a case essentially causes *inconsistency* in the alert-attribute networks, where a new attribute node is added without any node leading to it.

Inconsistencies are almost always caused by missed attacks: An "unexpected" attribute alteration causing the inconsistencies can either be directly caused by some successful attack missed by IDSs, or by a detected successful attack whose pre-condition is not satisfied in the network due to previously missed attacks. The only exception is that it could be caused by false alerts if the monotonicity property of attacks does not hold for some particular types of attacks. That is, a successful attack disables other attacks' pre-condition. According to [50], this kind of attacks are very rare. We can always recognize such attacks and pay additional attention in the investigation when they are involved. Thus, we propose to hypothesize about missed attacks based on the inconsistencies in

alert-attribute networks.

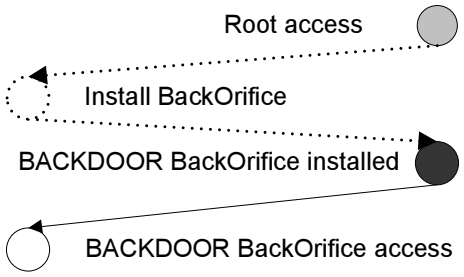


Figure 3.4: An example of hypothesized attack

Figure 3.4 shows an example to hypothesize about missed attacks to resolve an inconsistency. When the system monitoring tool reports the fact that a backdoor "`BackOrifice`" was found in the local system, node "`BACKDOOR BackOrifice installed`" is added to the graph immediately, which activates the pre-condition of the later alert "`BACKDOOR BackOrifice access`". However, there is no previous node possibly causing the "`BackOrifice installed`" attribute set to True. To fill in this gap, we look up the graph structure for established attributes and attacks, the knowledge base for possible attacks that can cause this attribute alteration, and the log of previously dropped alerts for possible related attacks. According to the above information, we make a hypothesis of a possibly missed attack "`Install BackOrifice`", linking the attribute nodes "`Root access`" and "`BackOrifice installed`". The hypothesized node and edges are presented with dotted lines in the figure.

A hypothesis of a possibly missed attack infers that (1) the attack has happened, (2) the attack has been missed by IDSs, and (3) the attack is successful. The probability of a hypothesized attack being a correct one can be estimated as $P_{hypothesis} = P_{happened} \cdot P_{missed} \cdot P_{successful}$, where $P_{happened}$ is the probability for the attack to have happened, P_{missed} is the prior probability for the IDS to miss the attack, and $P_{successful}$ is the probability for the attack to succeed if it happens. As P_{happen} fully depends on the attacker's knowledge and personal preference and is unpredictable, we only study the value $P_{missed} \cdot P_{successful}$, which represents our confidence in the hypothesis being `True` given the condition that it has actually happened. We call this confidence value the soundness of this hypothesis. From our previous discussion, the successfulness of an attack depends on whether its pre-condition is satisfied by the system attributes. Thus, the conditional probability table of a hypothesis node over the attributes in the attack's pre-condition is similar to a normal alert

node except that the non-zero values in the conditional probability table is P_{missed} instead of *Pr*. Accordingly, we refer to the probability computed from the Bayesian network with this conditional probability table the confidence in the hypothesized attack. Although this confidence value has a different meaning from those for normal alert nodes, it still shows which hypothesis is more possible given the available evidence.

We add the the hypothesized attacks with the corresponding conditional probability tables into the alert-attribute network. From the earlier discussion, we can see that such a hypothesis is made and placed into the alert-attribute network only if the attack is possible given the system state at the time. With the Bayesian network's belief update, we can always keep the hypotheses consistent with the newest observations. For example, we may find negative evidence against a hypothesis, and then the Bayesian inference process will update the probability of the hypothesized attack to 0, implying that the hypothesis cannot be a successful attack.

Validation is necessary for all hypotheses. From the above discussion about Bayesian inference about the hypotheses, we can see that the validation process is already embedded in the Bayesian inference process.

3.1.5 Scaling Up

The reader may have observed that as more IDS alerts are reported, the Bayesian network will grow larger and larger. Though by periodically scanning the system and gathering evidence about attacks, we may verify earlier alerts to be either successful or not, there will still be a number of unverifiable alerts. This has a severe impact on intrusion analysis. Indeed, both exact and approximate inferences in a Bayesian network upon partially observed evidence have been proven to be NP-hard [11, 16]. It is very expensive, and even infeasible, to make inferences upon new evidence if the Bayesian network is very large and complex.

One possible solution is to rebuild Bayesian networks when the previous ones grow too large. This can avoid intractable Bayesian networks. However, the effect of the evidence accumulated in the previous Bayesian networks will be lost, especially the system attributes that have been reasoned about using other evidence but not yet verified. As a result, information collected in an earlier Bayesian network cannot be carried over to the new one.

To make a trade-off between the accumulated information and the network size, we propose to use a sliding window to process and reduce the Bayesian networks. Specifically, we use a time window to decide what evidence to keep in the Bayesian network as well as what to remove.

When new alerts or scanning results are reported, we slide the window so that the front of the window advances to the most recent evidence. Some old evidence may move out of the window, and be removed from the Bayesian network. IDS alerts can be simply removed from the network. However, for system attributes, the last version before the end of the window will be used as the initial system state in the updated Bayesian network.

Note that the effect of the removed evidence is still kept in the Bayesian network. When a Bayesian network is first constructed, all the probabilities of the nodes are computed from the prior probabilities. As old nodes are removed, previously internal nodes become the root nodes of the updated Bayesian network. These new root nodes use the previously updated probabilities as their prior probabilities for later inferences. As a result, the effect of earlier evidence is retained by the updated Bayesian network.

One may point out that sliding windows give attackers an opportunity to defeat our technique. That is, an attacker may slow down his/her attacks so that the related attacks are not effectively considered since they do not appear in the same Bayesian network. However, even if an attacker slows down the attacks, the effect of each successful attack step is still captured by its post-condition in a Bayesian network, if the attack is detected. Thus, we can still reason about an individual alert if its post-condition is verified. Moreover, if an attacker has to slow down his/her actions to avoid being detected, our technique has already deterred attacks.

The size of the sliding window is critical to the effectiveness of the Bayesian networks. If the window size is too small (e.g., shorter than the time interval between two consecutive system scans), some IDS alerts may be discarded before we can use related evidence to reason about them. Certainly, such a Bayesian network cannot be too large due to the difficulty in computing with large Bayesian networks. Thus, we should balance the computational cost and the risk of losing information The computational cost of correlation and Bayesian inference is highly dependent on the amount of alerts and the amount of real attacks among those alerts. More frequent, deeper, wider system scans can decrease the size of the Bayesian network, while the computational cost of such scans also increases as its frequency, depth, and width increases. All those considerations make the problem even more complex. Those issues are already out of the scope of this paper, we will leave them to our future study.

3.2 Experimental Results

We have performed a series of experiments to evaluate the effectiveness of the proposed techniques. In our experiments, we connected three PCs through a hub in an isolated network. For convenience, we refer to them as *attacker*, *victim*, and *IDS*. We launched attacks from the attacker against the victim, while monitoring the attacks on the IDS.

3.2.1 System Setup

We use Snort version 1.9.1 [46] as the IDS sensor. We also use Nessus [4] and XScan [63] as the vulnerability scanning tools. We evaluate our techniques with five attack scenarios. The goals of these attack scenarios vary from modifying the target's web page to converting the target machine into a part of attacker's own distributed network. Some attack scenarios target MS Windows systems, while the others target Linux systems. Accordingly, the victim runs either Windows or Linux, depending on the attack scenarios. We run TripWire [6] (for MS-Windows) and Samhain [5] (for Linux) on the victim as the file system integrity monitoring tools. We also run Trojan horse scanning tools [54] (for MS-Windows) and chkrootkit 0.43 [2] (for Linux) on the victim as additional system scanning tools. We developed a program to automatically generate alert-attribute networks from the IDS alerts and the reports of these scanning tools, and then use JavaBayes [3] to make inference using these networks.

To simulate the realworld system administration, we configure the file system integrity monitoring tools (Tripwire and Samhain) to monitor important files and directories only, i.e., system configurations files, service configuration files, and the main webpage files.

To mimic an operational network, we also inject background traffic into the network during our experiments. We randomly select one of the training datasets (the training dataset on Monday in the third week) in the 1999 DARPA datasets [37] as the background traffic in the experiments, as it is attack free. This background traffic triggers 325 alerts in Snort, which are all false of course.

3.2.2 Attack Knowledge

The pre-condition and post-condition of the attacks used in our experiments are listed in table 3.1, as well as their prior confidence values are listed in table 3.2.

In the rest of this section, we will present 5 scenarios but only the Scenario 0 will be

Table 3.1: Pre-conditions and post-conditions of the attacks in our experiments

attack	pre-condition	post-condition
WEB-IIS cmd.exe access	IIS unicode vulnerability	{gain cmd.exe access}
WEB-IIS directory traversal attempt	IIS unicode vulnerability	{gain cmd.exe access}
FTP command overflow attempt	(Serv-U 5.0) ∧(anonymous access)	{cmd.exe root shell access}
Modify web page / shutdown Norton Antivirus	root access	{web page modified / ¬Norton Antivirus running}
Install BackOrifice	(user access ∧ ¬Norton Antivirus running)	{BackOrifice installed}
Install BackOrifice	(Norton Antivirus running)	{Virus BackOrifice quarantined}
FTP SITE EXEC format string attempt	(vulnerable wu-ftpd version ≤ 2.6.2)∧ (anonymous ftp access)	{Root shell access}
SNMP public access udp	SNMP public access	{gain public host information}
WEB-CGI redirect access	vulnerable ColdFusion / ClusterCATS	{gain account information}
ATTACK RESPONSE Invalid URL		{}

Table 3.2: Prior probabilities of the attack types in our experiments

attack	prior confidence	missing rate
WEB-IIS cmd.exe access	0.5	0.2
WEB-IIS directory traversal attempt	0.5	0.2
BACKDOOR BackOrifice access	0.6	0.5
FTP command overflow attempt	0.6	0.5
FTP SITE EXEC format string attempt	0.6	0.5
Remote control via cmd.exe root shell	N/A	1
Install BackOrifice	N/A	1
Modify web page via cmd.exe shell	N/A	1
Remote control via Trojan horse Glacier	N/A	1
Remote control via root shell	N/A	1
SNMP public access udp	0.15	0.5

analyzed in detail, and then summarize the results of all the five attack scenarios.

3.2.3 Scenario 0

Scenario Details: In this attack scenario, the attacker exploits the remote buffer overflow vulnerability in some old versions of Serv-U ftp server to get administrative access. The victim machine is a Windows box running a vulnerable Serv-U 5.0 ftp server with default public anonymous access. At the same time, the victim also runs Norton anti-virus with file system real-time protection. When the system attempts to access a file containing known virus or backdoor, the file system real-time protection will quarantine the file.

The attack scenario includes five steps:

1. remote buffer overflow attack against the Serv-U,

2. attempt to install BackOrifice on the victim, which was quarantined by the Norton anti-virus,

3. kill the Norton anti-virus process with system process tools through the remote administrative shell,

4. install the BackOrifice again (successful), and

5. changing the web page through BackOrifice.

The initial system attributes include Serv-U 5.0 running on port 21, anonymous ftp access, and Norton Anti-virus running with file system real-time protection.

During the attack process, Snort reported 2 alerts:

- 1 `FTP command overflow attempt` alert
- 1 `BACKDOOR BackOrifice access` alert

Norton also logged that BackOrifice was found in the file system and quarantined successfully during the attack period. In the end, Tripwire logged and reported the modification to the web page file and the system logged that Norton anti-virus was shut down.

Reasoning: Our alert-attribute network generation tool generated the network shown in Figure 3.5 based on the above information, as well as the prior probabilities and attack type information listed in table 3.2 and table 3.1.

To distinguish between different types of nodes in a Bayesian network, we use white nodes to denote IDS alerts, gray nodes to denote unverified system attributes, and black nodes to denote verified system attributes. The relative vertical position of nodes in the graph represents the relative time order among nodes.

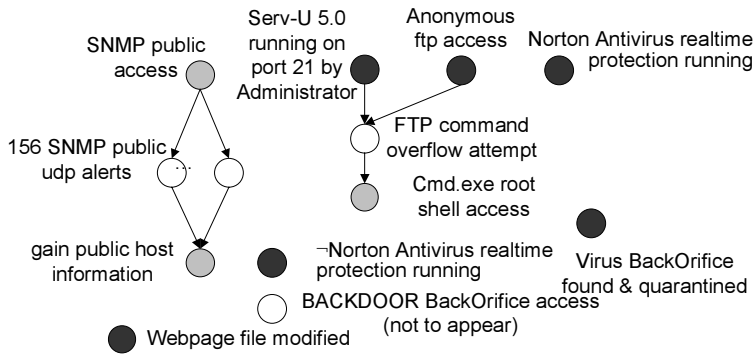


Figure 3.5: Initial alert-attribute network

Note that Figure 3.5 includes 156 "`SNMP public access udp`" alerts, which results in 2^{156} entries in the conditional probability table of `gain public information`. Computing with such a conditional probability table is out of JavaBayes' handling capacity. However, after alert aggregation, the 156 nodes are aggregated into a single node and thus can be handled easily by JavaBayes.

Now let us look at possible missed attacks. There are several obvious inconsistencies in Figure 3.5. There are no detected alerts causing the verified attributes "`Norton Anti-virus not running`", "`Virus BackOrifice found & quarantined`", and "`Webpage file modified`". Based on our knowledge about attacks, "`Shut down Norton Anti-virus via cmd.exe shell`" and "`Install BackOrifice`" are the only possible hypotheses that can fill in the first two gaps. For the attribute "`Webpage file modified`", it could be done through remote control via either cmd.exe shell or BackOrifice access. The first option implies hypothesized remote control via cmd.exe, while the second one implies hypothesized installation of BackOrifice after Norton was shut down. These hypotheses lead to a new alert-attribute network in Figure 3.6. In Figure 3.6, the dotted nodes and edges denote hypothesized attacks and corresponding causal relationships. Conditional probability table of each node can be generated automatically given the network structure and prior probability values. Then JavaBayes generates updated confidence values of each node in this Bayesian network. The confidence values of the related alerts before and after reasoning are shown in Figure 3.7. We can see significant increases in the confidence values of successful attacks; however, all the false alerts have either decreased or unchanged confidence.

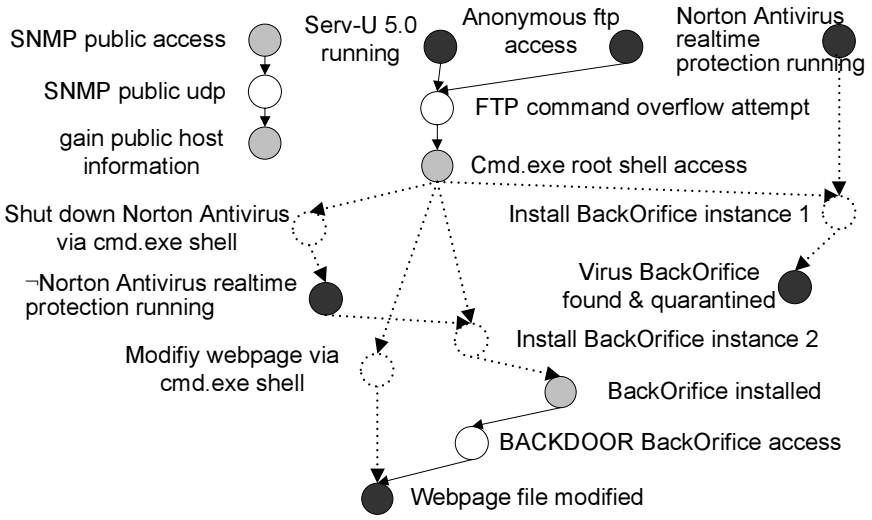


Figure 3.6: Updated alert-attribute network

ALERT NAME	BEFORE	AFTER	INCREASE
FTP command overflow	0.3	1	233.30%
BACKDOOR BackOrifice access	0.3	0.6	100%
Individual SNMP public access udp	0.075	0.075	0%
aggregated SNMP public access udp	0.5	0.5	0%
other 169 alerts	0.25	0	-100%
Shut down Norton Antivirus (Hypothesized)	N/A	1	N/A
Install BackOrifice Instance 1 (Hypothesized	N/A	0	N/A
Install BackOrifice Instance 2 (Hypothesized	N/A	1	N/A
Modify web page (Hypothesized)	N/A	1	N/A

Figure 3.7: Changes in confidence

We also find some interesting observations in the table shown in figure 3.7. The confidence values in three of the hypothesized nodes turned into 1, and two of them are the two options to resolve the same inconsistency. As we have discussed in Section 3.1.2, unless a hypothesis is the only option to solve the inconsistency, a confidence value of 1 for a hypothesized attack does not mean that the attack must have happened. Instead, it implies that *if* that attack has happened, it must be successful. Thus, although the confidence values for the two hypothesized nodes are both 1, it does not mean that both attacks must have happened. However, comparing the probability of the path from the initial verified attributes to the later verified attribute (by multiplying the probabilities of all the intermediate nodes along the path), we find that the one through "`Modify web page`

`via cmd.exe`" has a greater probability than the other one. Although it is not what exactly happened in our experiment, it shows that both methods can achieve the goal of modifying web page without being detected, and modifying through established remote cmd.exe shell is simpler and easier compared to the other option, which requires several extra attack steps. Also, the probability of a hypothesized node being 0 means either it is not missed by the IDS, or it is a failed attack attempt.

Using Confidence for Intrusion Detection: With the reasoning framework for intrusion evidence, we are able to associate a quantitative measure (i.e., confidence) with each IDS alert.

In our experiments, we used a confidence threshold to determine whether an IDS alert is a successful attack or not. Specifically, if the confidence in an alert is greater than or equal to the threshold, we accept the alert. Otherwise, we simply drop it. We change the threshold value between 0 and 1, and collect the detection rates and false alert rates. To compare the results in different situations, we repeated the above process in two cases: (1) without alert aggregation and abstraction, (2) with alert aggregation and abstraction. The performance graphs for the five attack scenarios are very similar.

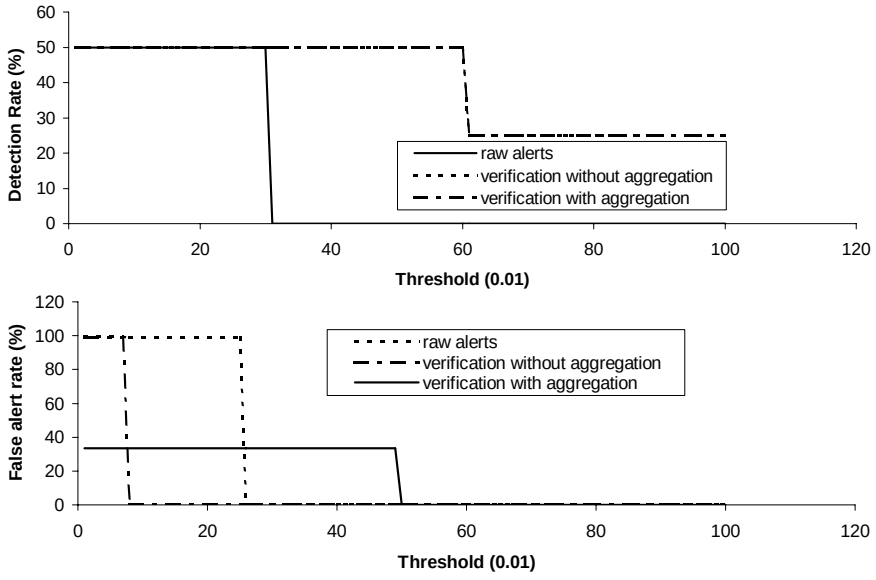


Figure 3.8: Detection rate and false alert rate

In our evaluation, we abuse the notions of detection rate and false alert rate to represent the *detection rate of successful attacks* and *false alert and failed attack rate*, respectively. Figure 3.8 shows the detection rate and false alert rate w.r.t. different thresholds in all cases for one of our

scenarios. (Since the meaning of the confidence in a hypothesized attack is different from that in an IDS alert, we do not consider hypothesized attacks in this evaluation.) This figure shows that the Bayesian reasoning with verified evidence can significantly increase the detection rate and decrease the false alert rate with appropriate threshold values.

3.2.4 Scenario 1

This scenario is fairly simple. We simulated a common scriptkid's activity, which exploits a common vulnerability to get certain privilege, and modify the remote server's web page. In this particular scenario, we exploited the format string vulnerability of wu-ftpd 2.6.0 on a RedHat linux 6.2 server to get remote root access. The attack scenario includes two steps:

1. A remote format string attack toward the wu-ftpd, and
2. replacing the remote server's web page with a "Gotcha" web page via the remote root shell access gained after the previous attack.

Snort raised 1 "FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux" alert. The file system monitoring tool (Samhain) generated alert for the web page modification since we configured the threshold on the times of modifications on those files to be 1.

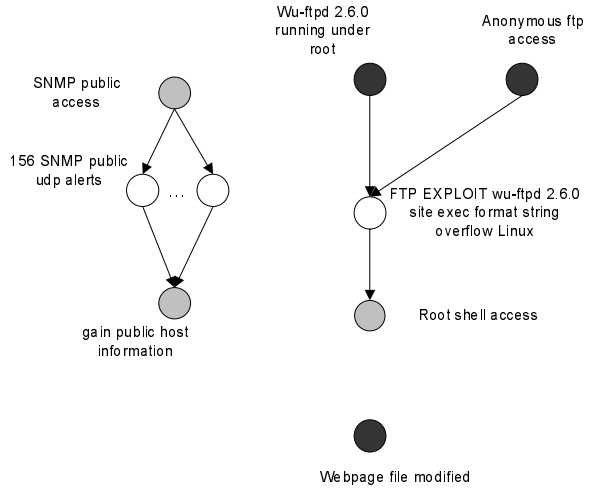


Figure 3.9: Initial alert-attribute network

Initial vulnerability scan showed that the system was running a vulnerable wu-ftpd 2.6.0 on port 21 with anonymous access open. However, we are not sure about whether the SNMP public access is turned off because we did not check the SNMP options in Nessus. The result alert-attribute network before making hypotheses is as shown in Figure 3.9.

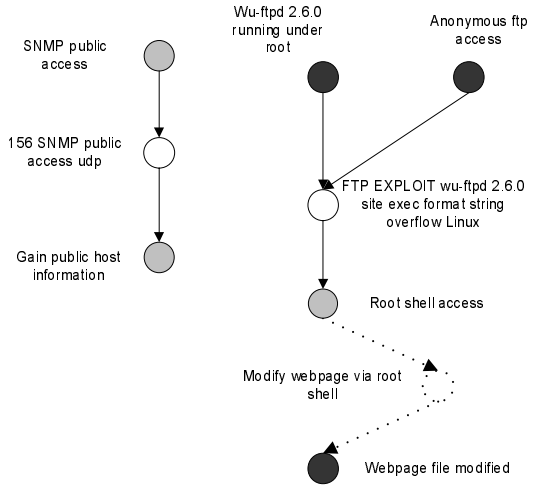


Figure 3.10: Alert-attribute network after hypotheses

Based on the observation of the inconsistency in the alert-attribute network shown in Figure /reffig:wuattack0, together with the attack type knowledge, the only possible hypothesis to fill in the inconsistency is that some remote control via the root shell caused the web page modification. Thus, the complete alert-attribute network is shown as in Figure 3.10.

We use dotted nodes and edges to denote hypothesized nodes and relationships in this new figure of alert-attribute network.

With the prior probability values and attack type information, our program generated the Bayesian network from the evidence log automatically and the reasoning result using JavaBayes is shown in table 3.3. Table 3.3 also shows the relative increase of the confidence values of the alerts.

The confidence in the hypothesized attack "`Remote control via root shell`" turned to 1 after the inference, which indicates that it would be successful and missed by the snort if it happened.

When using probability threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 3.11 and Figure 3.12.

Table 3.3: Confidence values before and after the reasoning

alert name	before	after	increase
156 SNMP public access udp(156)	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
FTP SITE EXEC format string attempt linux	0.3	1.0	333.33%
Remote control via root shell (hypothesized)	N/A	1.0	N/A
Other alerts	0.25	0	−100%

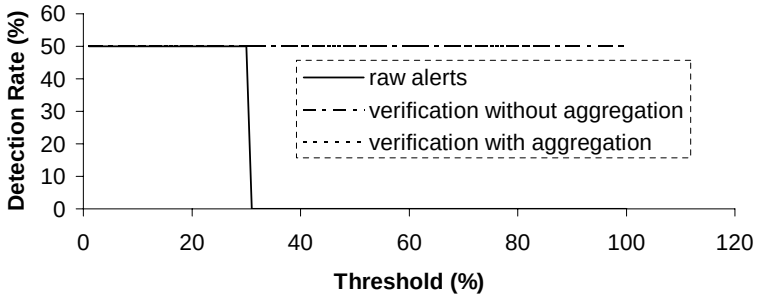


Figure 3.11: Detection rate VS. threshold (Scenario 1)

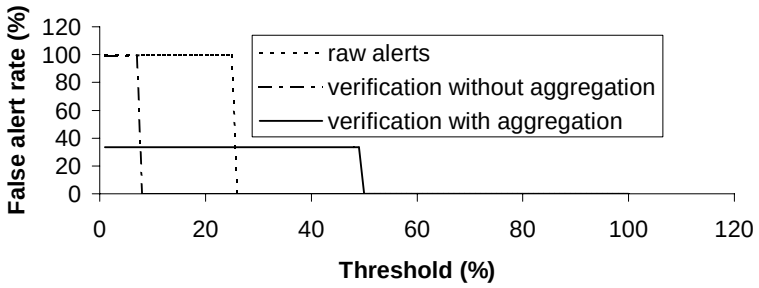


Figure 3.12: False alert rate VS. threshold (Scenario 1)

3.2.5 Scenario 2

This attack scenario exploits the unicode vulnerability of MS IIS 5.0. The victim machine was a windows box configured to be running a vulnerable IIS 5.0. The initial system vulnerability scan showed that there existed `IIS 5.0 unicode` vulnerability on the victim. The attacks scenario includes two steps:

1. Exploiting the unicode vulnerability to download and install a Trojan horse named `Glacier` to the victim.

2. Monitoring/controlling the victim file system remotely through the Trojan horse. The activities include replacing the web page.

When the above attacks were launched without background traffic, Snort generated the following 6 alerts:

- 2 WEB-IIS unicode directory traversal attempt alerts, and
- 4 WEB-IIS cmd.exe access alerts.

Both the WEB-IIS unicode directory traversal and the WEB-IIS cmd.exe access alerts indicate web attacks exploiting IIS 5.0's unicode vulnerability, thorough which the attacker can execute commands remotely thorough local host's cmd.exe program and cause various system modifications. They can actually be taken as Snort's reports on the same type of attacks. Thus, in the later analysis we aggregate them together into a single node. The post-condition of such attacks are highly dependent on the detailed content of the message sent by the attacker. As snort does not detect and distinguish those details, we define for this type of attacks a general post-condition "`Various system modifications done through cmd.exe`", which could imply any system modifications possibly done through cmd.exe.

Snort failed to detect the attempts of installing the Glacier backdoor and the remote control accesses through the Glacier backdoor. However, on the local system side, Tauscan did report in real time that Trojan horse Glacier was found in the system immediately after the IDS reported WEB-IIS alerts. Then Tripwire logged the modifications to the files in the IIS's web page directories a few minutes later after that.

Again, before aggregation and making any hypotheses about missed attacks, we have an alert-attribute network with inconsistencies as shown in Figure 3.13.

For the Tauscan's report of Trojan horse Glacier found, this attribute may only be implied by the "`Various system modification done`" caused by the IIS unicode exploits, thus we can have a hypothesized implication relationship between them, which is denoted by a dotted empty arrow in the figure. For the modifications in web page files, according to the system state and knowledge base (attack type information) we have, there are two options to achieve it on the victim: It could either be caused by some missed WEB-IIS unicode exploit, or by the remote control via

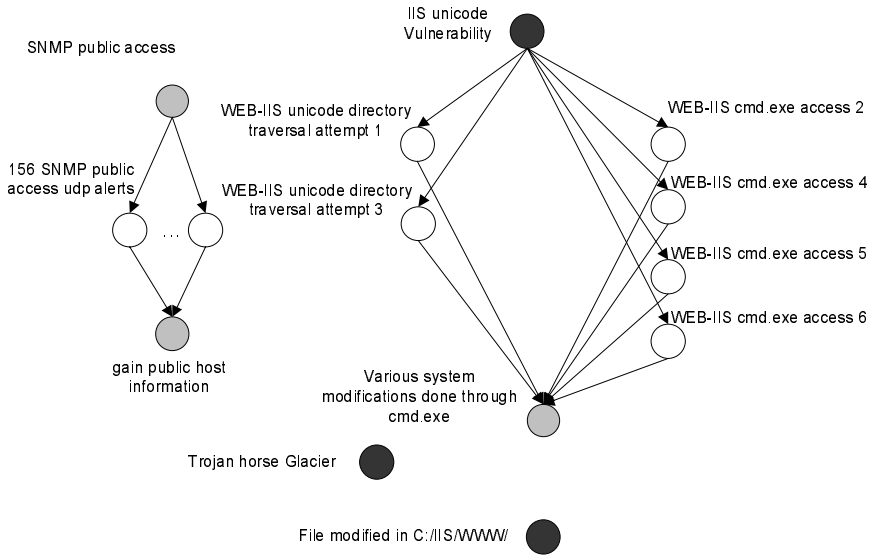


Figure 3.13: Initial alert-attribute network

the Glacier Trojan horse. Due to the first option, a hypothesized IIS unicode exploit alert node is added to the network to link the initial "`IIS unicode vulnerability`" node and the web page file modified node. Due to the second option, a hypothesized alert node "`Remote control via Glacier Trojan horse`" is added to link the "`Trojan horse Glacier found`" and the "`File modified in c:/IIS/WWW/`" node. With all these hypotheses and necessary alert aggregations/abstractions the complete network is shown in Figure 3.14.

The probabilistic confidence values of alerts before and after the analysis are shown in table 3.4.

Table 3.4: Confidence values before and after the reasoning

alert name	before	after	increase
2 individual WEB-IIS directory traversal attempts	0.25	0.50394	101.58%
4 individual WEB-IIS cmd.exe access	0.25	0.50394	101.58%
Aggregated WEB-IIS unicode exploit	0.822	1	21.6%
Remote control via Trojan horse Glacier (hypothesized)	N/A	0.8	N/A
IIS unicode exploit (hypothesized)	N/A	0.4	N/A
156 individual SNMP public access udp	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
Other 169 alerts	0.25	0	−100%

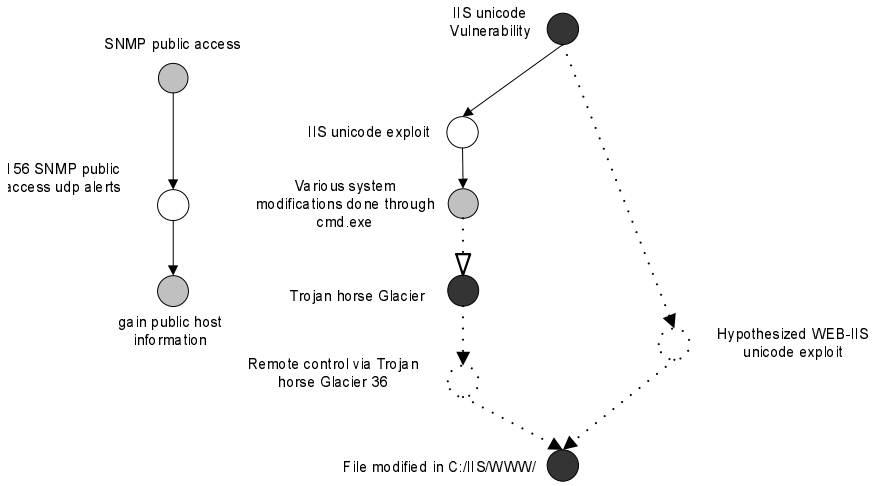


Figure 3.14: Updated alert-attribute network

In the reasoning, we assumed the remote control via Trojan horse Glacier to have a much larger probability to be missed by snort, compared with the IIS unicode exploit. This results in a higher confidence that the remote control via Glacier actually succeeded and was missed by snort, compared with the other hypothesis.

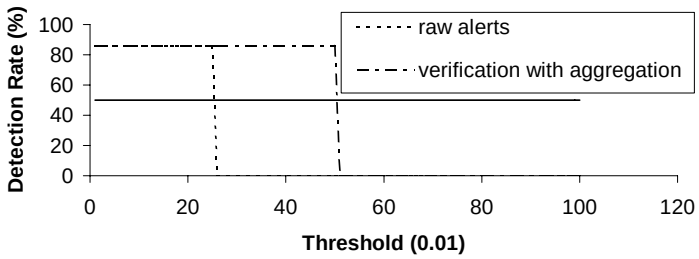


Figure 3.15: Detection rate VS. threshold (Scenario 2)

When using probability threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 3.15 and Figure 3.16.

Note that the reason for the rate after alert aggregation and abstraction being smaller than before aggregation is that the true alerts are aggregated into one single alert. Thus, the total numbers

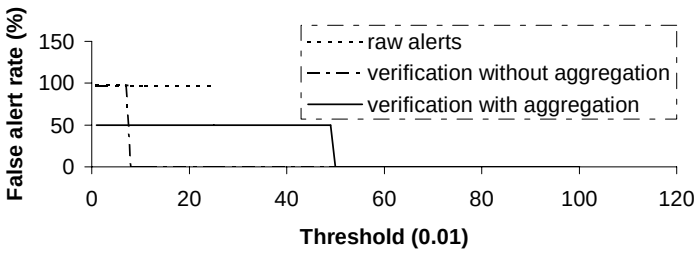


Figure 3.16: False alert rate VS. threshold (Scenario 2)

of alerts are different for the two sets of results.

3.2.6 Scenario 3

This attack scenario was a popular one on the Internet in the fall 2002, cause we have read a number of victim reports on the Internet and one of the machines in our lab was unfortunately one of them. The attack exploits the remote buffer overflow vulnerability of several older versions of Serv-U ftp server to get administrative access, and installs IRC DCC bot on the victim server to make it part of the attacker's public distribution network.

We configured the victim machine to be a Windows box running a vulnerable Serv-U 5.0 ftp server with default public anonymous access. At the same time, the victim was also running snort to log intrusion activities, and Tripwire to monitor the local file system. The attack scenario includes four steps:

1. Two remote buffer overflow attack attempts toward the Serv-U 5.0 server, one of which failed while the other succeeded.

2. Downloading and installing the IRC DCC bot on the victim through the remote root shell.

3. Cleaning the attack trace logged by snort after noticing the existence of snort in the system process list..

4. Starting another ftp server on port 28021.

The initial system scan reported finding vulnerable Serv-U 5.0 ftp server running on port 21 with public anonymous access.

When the attacks were launched without background traffic, snort reported two "`FTP command overflow attempt`" alerts and the system monitoring tools reported the following two observations:

- File modifications found on "`c:/snort/log/alerts.ids`" and "`c:/servu/ServUDaemon.ini`" reported by Tripwire.
- IRC DCC bot running on port 6666 and Serv-U 5.0 ftp server running on port 28021 reported by later system port scan.

Thus, when combined with the false alerts generated by the background traffic, we have the alert-attribute network shown as in Figure 3.17.

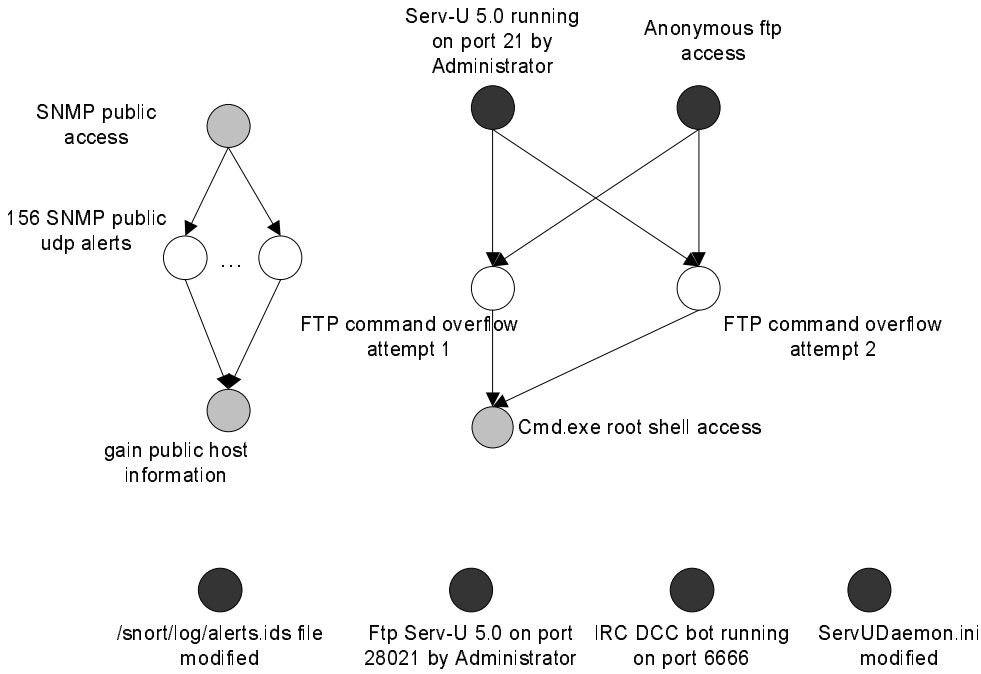


Figure 3.17: Initial alert-attribute network

After aggregation and making hypotheses, the final alert-attribute network is shown in Figure 3.18, and the confidence values of the related alerts before and after reasoning is shown in table 3.5.

When using probability threshold to decide whether an alert denotes a successful attack,

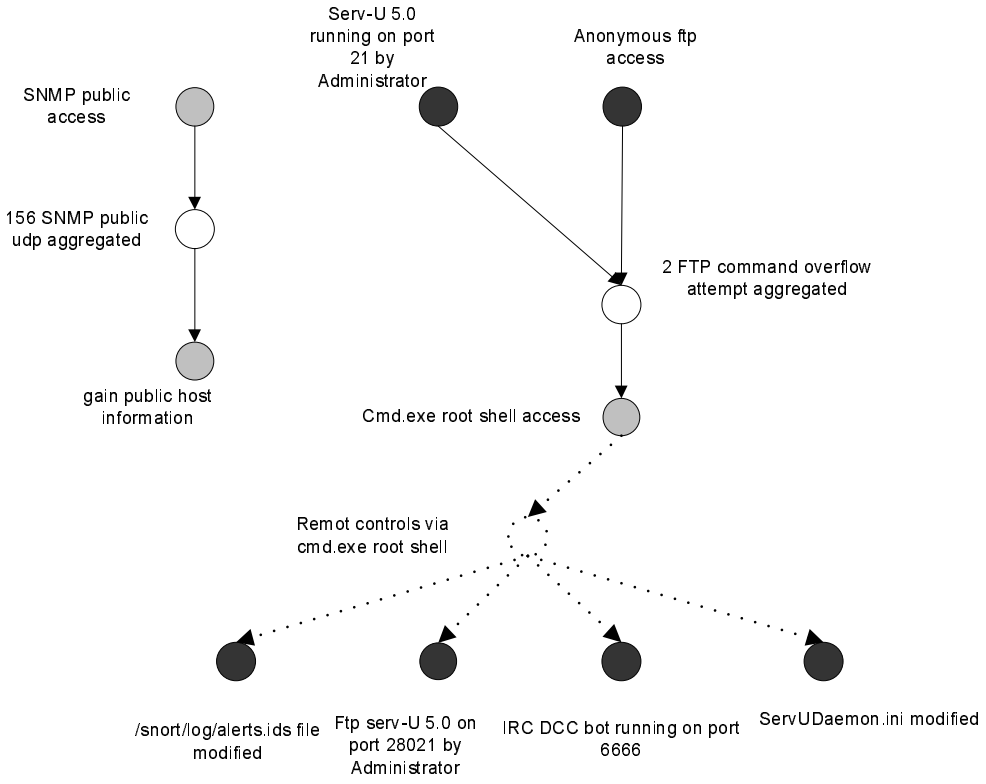


Figure 3.18: Updated alert-attribute network

Table 3.5: Confidence values before and after the reasoning

alert name	before	after	increase
2 individual FTP command overflow attempts	0.3	0.714	40.06%
Aggregated FTP command overflow attack	0.51	1	96.08%
Remote control via cmd.exe root shell (hypothesized)	N/A	1	N/A
156 individual SNMP public access udp	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
other 169 alerts	0.25	0	−100%

the experiment yields the detection rate and false alert rate curves as shown in Figure 3.19 and Figure 3.20.

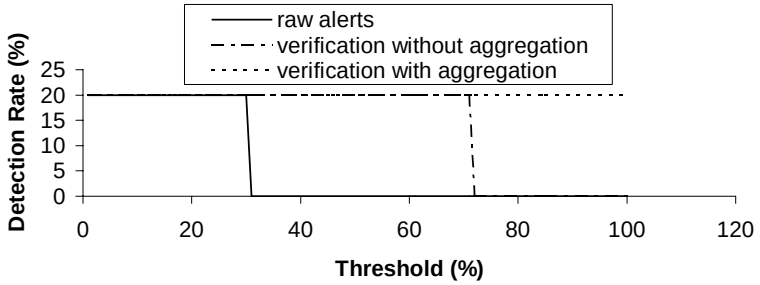


Figure 3.19: Detection rate VS. threshold (Scenario 3)

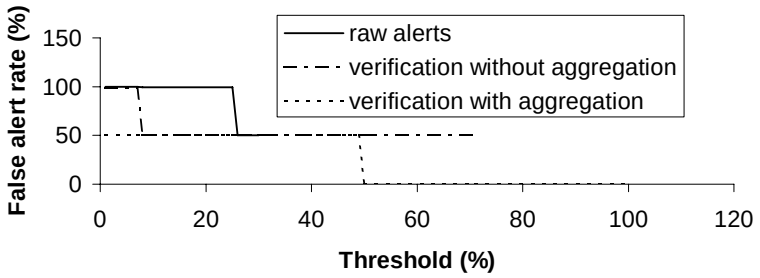


Figure 3.20: False alert rate VS. threshold (Scenario 3)

3.2.7 Scenario 4

This attack scenario studies the attacks on a target with multiple vulnerabilities. The victim machine was configured to be running both a vulnerable Serv-U ftp service and a vulnerable IIS 5.0. Initial system scan showed that vulnerable Serv-U 5.0 is running on port 21 with public anonymous access, and IIS is vulnerable to unicode attacks.

We exploited the ftp vulnerability to attack the victim machine. The attack scenario includes 2 steps:

1. Remote buffer overflow attack to the Serv-U ftp and get remote root shell.

2. Modifying the web page file on the remote system.

During the attack, snort reported one "`FTP command overflow attempt`" alert, while Tripwire logged and reported the modification of web page file.

Thus, when combined with the false alerts generated by the background traffic, we have the alert-attribute network shown as in Figure 3.21. After aggregation and making hypotheses, the final alert-attribute network is shown in Figure 3.22.

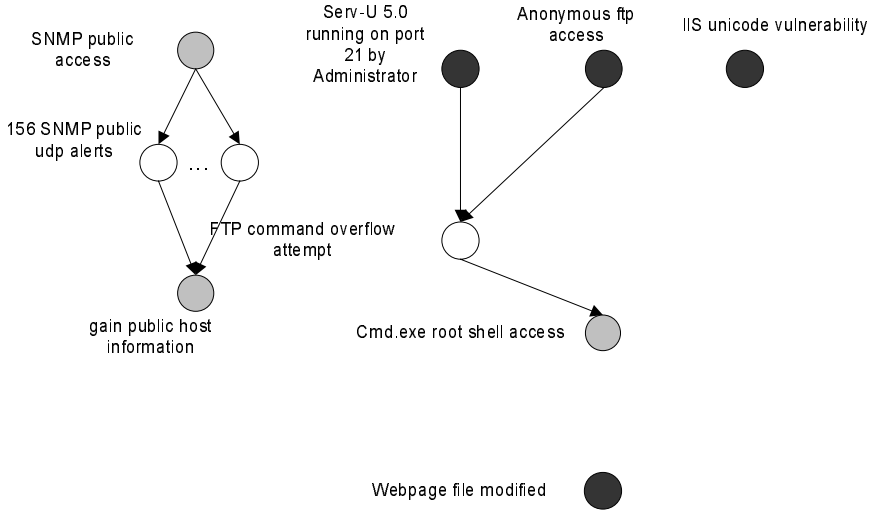


Figure 3.21: Initial alert-attribute network

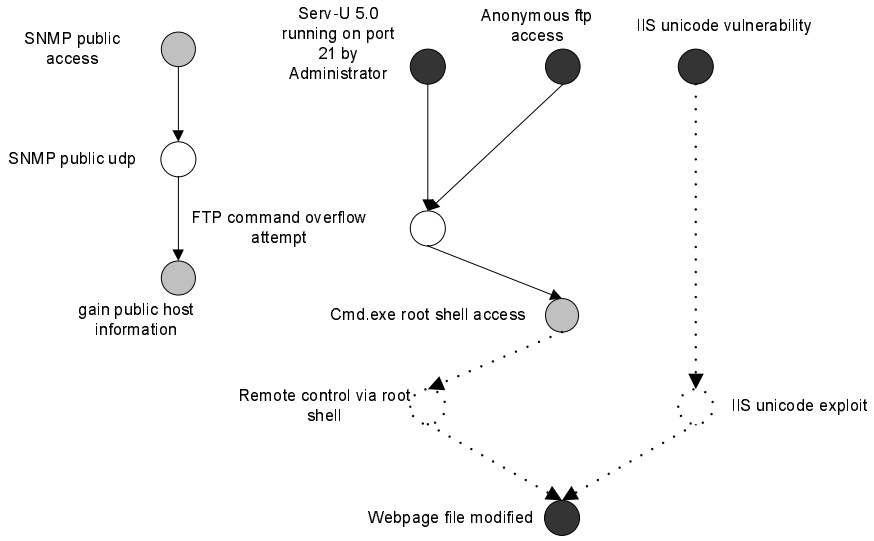


Figure 3.22: Updated alert-attribute network

The confidence values of the related alerts before and after reasoning are shown in table 3.6.

Table 3.6: Confidence values before and after the reasoning

alert name	before reasoning	after reasoning	relative increase
FTP command overflow attempts	0.3	0.88235	194.12%
Remote control via cmd.exe root shell (hypothesized)	N/A	0.88235	N/A
IIS unicode exploit	N/A	0.29412	N/A
156 individual SNMP public access udp	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
other 169 alerts	0.25	0	−100%

From the comparison in table 3.6, we can see that although multiple vulnerabilities introduce multiple choices when making hypotheses and we cannot have definite confidence in the hypotheses. The comparison result of the hypothesized attacks shows that when both pre-condition are satisfied, the attack with a higher missing rate gains a higher confidence from the reasoning, which means that they are more expectable in reality.

When using probability threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 3.23 and Figure 3.24.

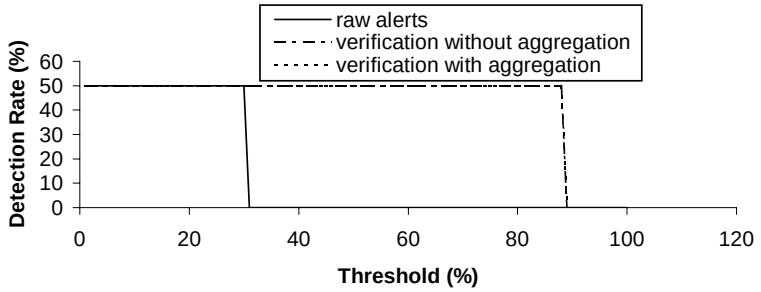


Figure 3.23: Detection rate VS. threshold (Scenario 4)

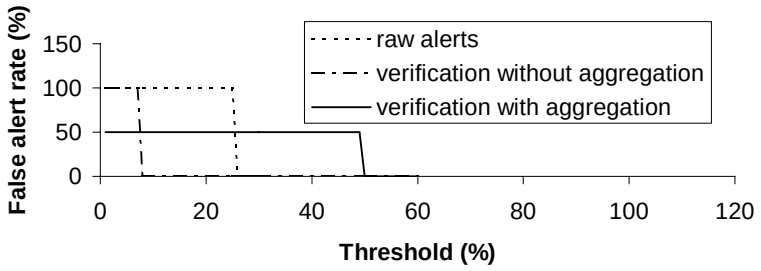


Figure 3.24: False alert rate VS. threshold (Scenario 4)

3.2.8 Summary of Results

In the following, we summarize the results obtained from all the five attack scenarios. We first discuss the impact of the proposed techniques on alerts, and then describe the results about hypothesized attacks.

We use a simple metric named confidence ratio to examine the usefulness of the proposed techniques. Specifically, a *confidence ratio* is the ratio between the average confidence of alerts corresponding to successful attacks and the average confidence of the other alerts (i.e., false alerts and alerts corresponding to failed attack attempts).

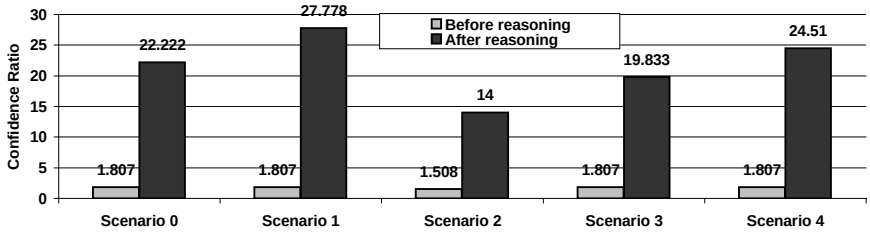


Figure 3.25: Confidence ratio changes

Figure 3.25 shows the confidence ratios in all five attack scenarios before and after using the proposed techniques. (We have discussed one scenario in the previous subsection; details of the other scenarios can be found in the full paper [66].) These results indicate that with the proposed techniques, the average confidence in alerts of successful attacks are greatly increased compared with the average confidence in the other alerts (false alerts and alerts for failed attack attempts). In fact, the average confidence in the other alerts either remain the same or decrease.

Table 3.7: Accuracy of hypotheses

Scenario	0	1	2	3	4
Accuracy	75%	100%	50%	100%	50%

We made ten hypotheses during the analysis of the five attack scenarios. Table 3.7 shows the accuracy of these hypotheses. In the experiments, six out of the ten hypothesized attacks are actual successful attacks missed by Snort, and one out of the other 4 hypotheses is a failed attack attempt. Among the seven real attacks, we have definite confidence that four of them must have happened from the alert-attribute networks. The result shows that with sufficient local system evidence, our model is efficient and effective in discovering some missed attacks.

3.3 Summary

In this chapter, we developed a method to integrate and reason about complementary intrusion evidence, including IDS alerts, reports of system monitoring or vulnerability scanning tools, and even human observations. By using the interdependency between attacks and system states, we combine various kinds of intrusion evidence into Bayesian networks, and infer about uncertain IDS alerts based on additional observations of system states. We proposed to refine these Bayesian networks through alert aggregation and abstraction, so that we can focus on the reasoning about existences of successful attacks and use complementary intrusion evidence more effectively. Our initial experimental results have demonstrated the potential of the proposed techniques.

A limitation of our approach is that it reasons about successful attacks, but cannot handle attack attempts in the same way. In other words, with additional evidence such as a verified attribute, our approach will increase the confidence in alerts corresponding to successful attacks, but decrease the confidence in those representing failed attack attempts. This feature certainly restricts the applicability of our approach. Another limitation is that our model cannot reason about attacks which has no effect on the local system, i.e., probes and scans. Indeed, most attackers need to gather certain information via network to launch attacks. For example, a probe to some specific ports may be necessary for attackers to gain related information to launch some corresponding exploits. However, the effect of such information gathering activities is on the remote attackers' side, which cannot be predicted and be used as pre-condition of attacks. Information can be gathered in

multiple ways other than network scans, e.g., chatting with a careless administrator or wiretapping the telephone. Thus, such attacks will not appear in the alert-attribute Bayesian network so that the reasoning will not affect and be affected by such alerts. Information of such alerts is usually useful for human administrators in analyzing the attacker's intentions and strategies in realworld. For example, people may have a higher belief on alerts of follow-up attacks after monitored probes on some special ports. However, modeling this observation brings risk of being distracted by forged traffic from attackers.

Chapter 4

Robustness Analysis of The Bayesian Reasoning Framework

In this chapter, we discuss our study in the robustness of the alert-attribute Bayesian networks we just presented in the previous chapter.

As part of the initial set up of the Bayesian networks, the prior probability assignments of related alerts will certainly affect the final reasoning results. In practice, the uncertainty in a lot of intrusion evidence can be eliminated by gathering more detailed evidence and going through more complicated analysis. For example, manually going through logged individual packets to find out whether those packets represent a real attack. However, the expense of collecting and analyzing such details, as well as the often large amount of evidence make it infeasible to carry out such procedures on every piece of evidence that we are not certain about. When using a Bayesian network to inference about one of its nodes' posterior probability, we call the inference result the output of the Bayesian network on this node, and call the node of interest the output variable in this case. In a Bayesian network, different nodes usually have different influences on the output result depending on the network topology and prior probability assignments. Some nodes may have much larger influences than others. Thus, it will be more economical to spend the limited time and resource in analyzing those "sensitive" evidence to get better results. Also, sometimes people are interested in the qualitative relations between intrusion evidence and attacks. By studying the

qualitative properties of alert-attribute Bayesian networks, we can derive such relationships even if we do not have good knowledge about the prior probabilities. Below we discuss the processes and results of such analysis.

4.1 Sensitivity Analysis

In an alert-attribute Bayesian network, the output result may be affected by the inaccuracies involved in human experts' assessments on the prior confidence of alerts. Although many alerts can be verified to be either True or False by carefully analyzing the detailed system/network logs and evidence, such evidence gathering and analysis process can usually be too expensive to perform on every alert we encounter in the Bayesian network. Thus, it is desirable to find out which variables have the most significant influence on the output variable, and focus our resource and efforts on analyzing such variables, so as to improve the quality of the results at the lowest cost. For example, if we know that one particular alert's prior confidence affects the output of an alert-attribute network the most, we can focus more on the investigation in this particular alert if we cannot investigate on all of them. Such robustness properties of the alert-attribute Bayesian network can be evaluated by subjecting the network to a sensitivity analysis. A sensitivity analysis amounts to varying the assessments for one or more of its numerical parameters and investigating the effects on the output of the Bayesian network. The result of the sensitivity analysis can help us to estimate the effects of the inaccuracies in the parameters of the network.

Let $x = P(b_i|\pi)$ denote the prior conditional probability value for node B to take any value b_i that B can take given any assignment π of its parents. The output value of the Bayesian network can be presented as a function over x, which is denoted as $f(x)$. We call this function the *sensitivity function* of the output variable over parameter $P(b_i|\pi)$ in this Bayesian network. It has been proved that a sensitivity function is a quotient of two functions that are linear in the parameter x [9, 12]. That is, the function takes the form of

$$f(x) = \frac{x + c_1}{c_2 \cdot x + c_3},$$

where c_1, c_2,and c_3 are constants derived from the other parameters in the Bayesian network. Thus, we can find the actual sensitivity function from either a symbolic inference in the Bayesian network, or assigning multiple values to x and solve the corresponding coefficients. An important value associated with the sensitivity function is the sensitivity value $\frac{df(x)}{dx}$, which represents how sensitive the output changes upon minor changes on variable x.

Here we use one alert-attribute Bayesian network from one of our experiments to demonstrate the sensitivity analysis in these networks. The network is as shown in Figure 4.1. For the simplicity of presentation, we assume all the hypothesized alert nodes are also actual alerts reported by the IDS.

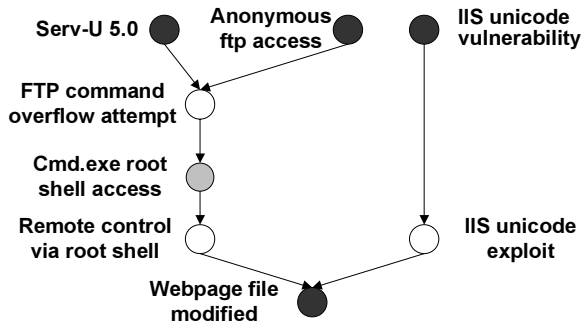


Figure 4.1: An alert-attribute Bayesian network

Assume that node "$FTPcommandoverflowattempt$" is picked as the output variable and the prior confidence values of all the attack types in the network are assigned to be 0.5. Now we study the sensitivity function of the output $P(\text{“}FTP\ command\ overflow\ attempt\text{”}= True|\{\text{“}Webpage\ file\ modified\text{”}= True)\}$ over parameter $x = P_0(\text{“}IIS\ unicode\ exploit\text{”})$. By assigning different values to x and compute the function value, we have:

$$\begin{cases} \dfrac{0.5 + c_1}{c_2 \cdot 0.5 + c_3} = 0.6 \\ \dfrac{0.4 + c_1}{c_2 \cdot 0.4 + c_3} = 0.5833 \\ \dfrac{0.6 + c_1}{c_2 \cdot 0.6 + c_3} = 0.61538 \end{cases}$$

By solving the above equations, we have $f(x) = \frac{x+1}{x+2}$. Similarly, we can compute the sensitivity functions as well as the sensitivity values for different combinations of output variables and x given node "`Web page file modified`" instantiated to be True and False. The result is shown in table 4.1.

In table 4.1, $f_{true}(x)$ and $f'_{true}(x)$ represent the sensitivity function and sensitivity value when node "`Web page file modified`" is instantiated to be True; $f_{false}(x)$ and $f'_{false}(x)$ represent the sensitivity function and sensitivity value when the node is instantiated to be False. From the sensitivity values shown in table 4.1, we can see that the output variable is

Table 4.1: Sensitivity functions

output variable	x	$f_{true}(x)$	$\lvert f'_{true}(x)\rvert$	$f_{false}(x)$	$\lvert f'_{false}(x)\rvert$
FTP command overflow attempt	FTP command overflow attempt	$\frac{3x}{x+2}$	0.96	$\frac{x}{2-x}$	0.889
FTP command overflow attempt	Remote control via root shell	$\frac{x+1}{x+2}$	0.16	$\frac{1-x}{2-x}$	0.444
FTP command overflow attempt	IIS unicode exploit	$\frac{x+1}{3x+1}$	0.32	$\frac{1}{3}$	0
Remote control via root shell	FTP command overflow attempt	$\frac{2x}{x+2}$	0.64	0	0
Remote control via root shell	Remote control via root shell	$\frac{2x}{x+2}$	0.64	0	0
Remote control via root shell	IIS unicode exploit	$\frac{1}{3x+1}$	0.48	0	0
IIS unicode exploit	FTP command overflow attempt	$\frac{2}{x+2}$	0.32	0	0
IIS unicode exploit	Remote control via root shell	$\frac{2}{x+2}$	0.32	0	0
IIS unicode exploit	IIS unicode exploit	$\frac{4x}{3x+1}$	0.64	0	0

more sensitive to its own prior confidence than to any other attacks' prior confidences; the posterior probability of alert "`Remote control via root shell`" is as sensitive to node "`FTP command overflow attempt`" as to node "`Remote control via root shell`".

In practice, decisions about alert being successful or not are usually made based on threshold values. That is, given a threshold $0 < t \leq 1$ on the confidence of an alert, if the alert's (posterior) confidence value is less than t, the alert is treated as a false alert or a failed attack attempt. Otherwise, the alert is treated as a successful attack. Given these sensitivity functions, we can also compute the corresponding ranges for the prior confidences that will or will not affect the decision. For example, from table 4.1, the sensitivity function of the output $P(\text{“}FTP\ command\ overflow\ attempt\text{”} = True | \{\text{“}Webpage\ file\ modified\text{”} = True)\}$ over parameter

$x = P_0(\text{“}Remotecontrolviarootshell''})$ is $\frac{x+1}{x+2}$. If we set the threshold as 0.6, by solving the inequation $\frac{x+1}{x+2} \geq 0.6$, we know that alert `FTP command overflow attempt` will always be determined to be False when `Remote control via root shell`'s prior confidence is within $(0, 0.5)$

The statistical information about certain attack types may not always be available. However, we are still interested in the change of the probability of alerts upon observed system evidence. In such cases, we still need to assign prior confidence values for those attacks in order to make the Bayesian network sound and complete. From a conservative point of view, we do not want such arbitrarily assigned values to affect the reasoning too much. According to [10], the sensitivity value, which is the differential coefficient value of the corresponding sensitivity function, of a parameter for a binary variable has an upper bound:

$$|f'(x_0)| \leq \frac{p_0 \cdot (1 - p_0)}{x_0 \cdot (1 - x_0)},$$

where x_0 is the value specified for x, and p_0 is the corresponding value of the probability of interest. Thus, this inequation reveals that the large sensitivities are expected only for the more extreme values of x_0, and the probability of interest will be quite insensitive to small shifts in x with an original value of 0.5. So, for the attack types which are lack of statistical information, we use 0.5 for their prior confidence in the reasoning.

4.2 Qualitative Analysis

As we have discussed earlier, the analysis on the quantitative properties of the alert-attribute Bayesian networks (Bayesian inference, sensitivity analysis) are computationally costly because Bayesian inference is NP-hard. However, in some instances, we may be only interested in some qualitative properties of the networks such as the increase or decrease on some variables in networks. As we will see later, the qualitative properties of the networks are relatively easier to determine, especially given some special conditional probability distributions associated with variables in alert-attribute networks. The qualitative property we are interested in an alert-attribute Bayesian network is how the probabilities of alerts change upon observed evidence. If we can decide the general direction of how a piece of evidence would affect our confidence in a specific alert, it will save the expensive computations during Bayesian inference. For example, when the confidence of an alert already exceeds certain threshold, if we can decide that the new evidence will not change

this situation without performing the inference, it will save a lot of computation cost. Also, the prior conditional distributions associated with alerts may contain inaccuracy, which will affect the quantitative inference. Thus, it is desirable to have some qualitative properties that is independent of the prior distribution values. This is the motivation of our qualitative analysis on the alert-attribute Bayesian networks. In particular, we want to study the qualitative properties of the alert-attribute Bayesian networks, so as to know that under which conditions a piece of verified evidence will indirectly verify other nodes in the alert-attribute Bayesian network, and whether the confidence in the other nodes in the network will increase or decrease upon such additional evidence.

Indirect verifications

Because of the way the alert-attribute Bayesian networks are constructed, an instantiated node may also verify some other related nodes in the network. For example, a system attribute node verified to be False may also cause its parent nodes to be verified as False. Because this type of verifications are not directly observed evidence from scanning reports or system logs, we call them *indirect verifications*. Below we will study the conditions for such indirect verifications.

Among the verified system attribute *nodes*, some of them are verified to be True while others are verified to be False. Except for the initial system attribute nodes, an attribute node's value being True indicates the corresponding attribute being altered by previous activities, which will increase the confidence in its previously related alert nodes. Thus, we call the first type of verifications the *positive instantiations* because they have positive impacts on identifying real attacks. Similarly, we call the second type of verifications *negative instantiations*.

Lemma 4.2.1 *Given an alert-attribute Bayesian network $\mathcal{B}$ and a negatively instantiated attribute node V, a node $C \in \mathcal{B}$ will be indirectly verified to be False by V if C is one of V's direct parents.*

The proof is obvious: If C is True, because V is C's post-condition, it should also be True. Thus, C must be False if V is True.

We call an alert-attribute Bayesian network *complete* if it represents a legitimate attack scenario. That is, in this network, each alert node has its pre-condition satisfied in the Bayesian network by the time it takes place, and each attribute node is either an initial system attribute or the child node of a previous alert node.

Lemma 4.2.2 *Given an alert-attribute Bayesian network $\mathcal{B}$ and a positively instantiated node x, a node $C \in \mathcal{B}$ will be indirectly verified to be True by x if all the complete subnets of $\mathcal{B}$ containing x also contain the node C.*

This lemma is also obvious given the way alert-attribute Bayesian networks are constructed. Indeed, if there is a complete subnet that contains x but does not contain C, which means x can be caused by a scenario without C, C is not guaranteed to be True. On the other hand, if there is not such a subnet, which means that all legitimate attack scenarios depends on Cs success, C must be True given x positively instantiated.

By checking whether the conditions indicated by lemma 4.2.1 and 4.2.2 are satisfied, we can identify indirectly verified nodes in alert-attribute Bayesian networks without performing the usually costly Bayesian inference.

Monotonicity properties

Now we study the "monotonicity" properties of the alert-attribute Bayesian networks. That is, how the output of alert-attribute Bayesian networks change toward positive and negative instantiations.

Before we start the analysis, we first introduce some related definitions, lemmas, and theorems in the research field of Bayesian networks and qualitative probabilistic networks.

Definition 1 *Given a Bayesian network $\mathcal{B} = (G, \Gamma)$, where G is a DAG and Γ is a set of conditional probability distribution, let C be the output variable of the network and let $X(G)$ be the observable variables. The* distribution output function *for $\mathcal{B}$ is a function $f_{P_r}(x) = P_r(C|x)$ for all the joint value assignments x over $X(G)$. [59]*

When there exists a total ordering on the discrete values of the variables and set of variables in a Bayesian network, based on the posterior probability distribution of given variables, we can discuss the monotonicity of the distribution output function we just defined and the qualitative influence in the network. The following definition describes the idea of monotonicity that we will study in this section.

Definition 2 *A Bayesian network $\mathcal{B} = (G, \Gamma)$ is said to be* isotone in distribution *for the observable variables $X(G)$ if*

$$x \leq x' \Rightarrow f_{P_r}(x) \leq f_{P_r}(x')$$

holds for all value assignments x, x' of variables $X(G)$. [59]

To study the monotonicity of Bayesian networks, researchers have designed the concept of qualitative influence, and studied the properties of such influence in Bayesian networks. Below we will quote the definition and some related lemmas from some existing literatures in Bayesian networks research. Later we will use the definition and lemmas to study the monotonicity in the alert-attribute Bayesian networks.

Definition 3 *A variable V has a positive qualitative influence on variable W if, for all values w of W and all values v, v' of V, with $v \leq v'$, inequation*

$$F_{P_r}(w|v's) \leq F_{P_r}(w|vs)$$

holds for any joint value assignment s to the context set $S = \pi(W)$

$\{V\}$ of parents of W other than V, where the $F_{P_r}(w|vs)$ is the cumulative conditional distribution function which equals to $P_r(W < w|vs)$. We use sign '+' to denote this qualitative influence. Analogously, by replacing the $\leq$ in the above equation with $\geq$ and $=$, we can define the negative and zero qualitative influence, and we use signs '-' and '0' respectively to denote the qualitative influences. Otherwise, if the influence changes upon different value assignments to S, we call it an uncertain qualitative influence and use the question mark '?' to denote it. [59]

Lemma 4.2.3 *The set of all qualitative influences between the variables of a Bayesian network exhibits the property of symmetry. That is, if the network includes an influence of sign δ of V on W, $\delta \in \{+, -, 0, ?\}$, then it also includes an influence of W on V of sign δ. [62]*

Theorem 4.2.4 *Given a Bayesian network $\mathcal{B} = (G, \Gamma)$ with the set $X(G)$ of observable variables and the output variable C. If, for all variable $X_i \in X(G)$, X_i has a positive qualitative influence on C, then $\mathcal{B}$ is isotone in distribution for X(G). [59]*

Now we study the monotonicity of the alert-attribute Bayesian networks. Without losing the generality, we define a complete ordering of $True > False$ on the values for all the binary variables in networks. Thus, we can have the following lemma:

Lemma 4.2.5 *Given two uninstantiated (neither directly verified nor indirectly verified) variables V and W in an alert-attribute Bayesian network, if they are directly connected by an edge, or they are alert nodes and share the same parent attribute node, V and W have positive qualitative influences on each other.*

Proof:

There are 4 types of connections between two neighboring variables in an alert-attribute Bayesian network.

1. a direct link from an alert node V to its post-condition node W,
2. a direct link from a pre-condition node W to alert node V,
3. a direct link from an attribute node V to an implicated attribute node W, and
4. (inter-causal link) two alert node V and W are both linked to an instantiated post-condition node X.

For the first type of link, because $P(W = 0|V = 1) = 0$ and $P(W = 0|V = 0) \geq 0$, we have $F_{P_r}(w|v = 1, s) \leq F_{P_r}(w|v = 0, s)$ holds for all s. Thus, link $V \rightarrow W$ is associated with a positive influence sign. For the second type, because $P(W = 0|V = 1) = 0$ and $P(W = 0|V = 0) \geq 0$, we can also conclude that the link is associated with a positive influence sign. For the third type, because $P(W = 0|V = 1) = 0$ and $P(W = 0|V = 0) \geq 0$, it is also a positive influence sign. For the forth type, when node X is instantiated to be False, obviously both V and W must be False, and the qualitative influence is positive; when node X is instantiated to be True, from Bayes'

theorem, we can have $P(W = 0|V = 0, X = 1) \geq P(W = 0|V = 1, X = 1)$, which also makes it a positive influence. From lemma 4.2.3, all the direct links in an alert-attribute Bayesian network exhibit positive qualitative influences.

Following [62], we can use the sign '+','-', '0', and '?' to denote positive, negative, zero, and uncertain qualitative influences respectively. The joint qualitative influence on a variable can be computed using sign multiplication and addition operators [62, 20]. According to [20], the sign of a trail between two nodes is the product of the signs of all direct and inter-causal [20] links in the trail; the sign of a node to another node is the sum of the signs of all evidential trails from the first node to the other one. Table 4.2 [62] defines those operators.

Table 4.2: The $\otimes$ (multiplication) and $\oplus$ (addition) operators for combining signs.

$\otimes$	+	-	0	?
+	+	-	0	?
-	-	+	0	?
0	0	0	0	0
?	?	?	0	?

$\oplus$	+	-	0	?
+	+	?	+	?
-	?	-	-	?
0	+	-	0	?
?	?	?	?	?

With lemma 4.2.5 and the sign operators, we can have our conclusion about the monotonicity of the alert-attribute Bayesian networks:

Lemma 4.2.6 *Any alert-attribute Bayesian network is isotone in distribution for any set of observable variables in the Bayesian network.*

Proof:

Because all the direct links in an alert-attribute Bayesian network exhibit positive qualitative influences 4.2.5, with the operators shown in table 4.2, it is obvious that there can only be positive qualitative influence between any two variables in the network. Thus, according to Theorem 4.2.4, the Bayesian network is isotone in distribution.

With lemma 4.2.6, we can decide whether the output values will be certain to increase or decrease upon new instantiations. By applying it in some special cases, we have a corollary:

Corollary 4.2.7 *(Positive Update) In an alert-attribute Bayesian network, if there are only positive (negative) instantiations on attribute nodes, the posterior confidence values of all the alert nodes in*

the network are no less (more) than their confidence values before the Bayesian inference upon the instantiations.

When both positive and negative instantiations exist, the joint effect of multiple observations on a variable of interest can be computed as the sum of the effects of the separate observations on this variable's probability distribution.

As we can see, with the results in qualitative influences and indirect verifications, we can learn a lot of information about how the evidence would affect our assessments to the variables qualitatively in alert-attribute Bayesian networks even without performing the quantitative Bayesian inferences. This is complementary to quantitative analysis when computation resource is limited.

4.3 Summary

In this chapter, we studied the robustness of the alert-attribute Bayesian networks which are used to integrate and reason about complementary intrusion evidence. The contribution of this robustness study is embodied in the three issues. Firstly, by performing sensitivity analysis on the alert-attribute Bayesian networks and finding out the sensitivity functions, together with the decision threshold values, we can compute the ranges of specific alerts' prior confidence values that will affect the decision. Secondly, by performing sensitivity analysis and studying the upper bound of sensitivity values, we point out that when the prior distribution of an variable is unknown, appointing it to be 0.5 will most likely to make the rest of the network insensitive to this variable. Thirdly, by performing qualitative analysis on the alert-attribute Bayesian networks, we prove that such Bayesian networks are monotonic toward variable instantiations. Thus, given an instantiation on a variable in the network, we can decide how the posterior confidences of interested alerts will change toward observed evidence.

Chapter 5

Alert Correlation Using OS-level Object Dependency

Intrusion alert correlation techniques correlate alerts into meaningful groups or attack scenarios for the ease to understand by human analysts. However, the performance of correlation is undermined by the imperfectness of intrusion detection techniques. Falsely correlated alerts can be misleading to analysis. In this chapter, we propose to harness OS-level event logging and dependency tracking to improve the accuracy of alert correlation. OS-level dependency tracking is a recently developed technique to analyze the system operation history toward a given object. The dependency tracking technique used in this paper is *Backtracker* [31], which to our best knowledge is the only such tool available. It tracks dependency-causing events such as process forking and file operations in the system event log, and spans up a tree of system objects connected by these events from the target object. Though a very useful tool for forensics applications, backtracker has two limitations. Firstly, because it is system call oriented, the complexity of tracking and tracking results can be very high. For example, during a normal system run, the resulting dependency graph of such a tracking (using *Backtracker*[31]) can contain up to tens of thousands of objects and hundreds of thousands of edges. Such complexity with tracking is obviously time and resource consuming, while the tracking results are also hard to understand. Secondly, the tracking is highly dependent on the availability of so-called “detection points”, which are significant evidence of system being attacked.

However, such “detection points” are not usually available, making it inconvenient to use in security administration. Integrating event logging and dependency tracking tools with alert correlation can potentially address the above limitations, and at the same time improve the performance of alert correlation.

Our integration method is based on the following observations: Firstly, most attacks have corresponding operations on specific OS-level objects. Secondly, other than a few exceptions, if one attack prepares for another, the later attack’s corresponding operations would be dependent on the earlier one’s. Because logging these system calls is more straightforward than detecting attacks using rules and signatures, such information is considerably more accurate and trustworthy than the IDS alerts. Utilizing such information will improve the performance of alert correlation.

Given the alert correlation results and an OS-level dependency tracking tool, the integration is done in two phases. The first phase is to identify the system objects corresponding to the IDS alerts based on their semantics. The second phase is to verify the relationships demonstrated in the correlation result or discover the missed relationships among the alerts by tracking the dependencies among their corresponding system objects using a dependency tracking tool such as Backtracker.

The contribution of this work is the development of a framework to integrate the information from OS-level event logging and dependency tracking into IDS alert correlation. With the support of OS-level event logs, we can achieve better accuracy in the final result than the original alert correlation method. We also discuss how such integration can facilitate the hypotheses about possibly missed attacks. Finally, we evaluate the effectiveness of this scheme by performing a series of experiments. Our experiment results show that the integration can greatly improve the correctness of correlation and help making hypotheses about possibly missed attacks. For example, in our experiment, our approach can totally remove the false correlations in all three attack scenarios.

5.1 Integrating Alert Correlation and OS-Level Dependency Tracking

Our integration is to identify the relevancy between the relationships among IDS alerts and the dependencies among OS-level objects, and then use the OS-level dependencies to verify or discover the relationships among IDS alerts. To identify such relationships, we first look into attacks’ OS-level behaviors.

From the operating system’s point of view, an attack is a set of OS-level events that access or modify a set of system objects. The OS-level objects and operations corresponding to an attack

can be derived from the semantics of the attack. In our model, such semantics consist of two parts: one is the prerequisites and consequences of attacks, and the other is the correspondence between the predicates in attacks prerequisites or consequences and the OS-level objects on the host. With such information, we can identify the OS-level objects corresponding to the attacks on the host. For example, given an attack that exploits a vulnerable service as its prerequisite and yields a shell as its consequence, we can identify the corresponding service process and shell process for this attack.

Accordingly, the OS-level objects corresponding to an attack can be divided into two sets: the *prerequisite object set*, which are the objects derived from the attack's prerequisite, and the *consequence object set*, which are the objects derived from the attack's consequence. These two sets may overlap, because some attacks' consequences may affect their prerequisite objects. By backtracking among the OS-level objects, we can also find dependencies among those objects at the OS level. Though different from the *prepare-for* relation used in alert correlation, such OS-level dependencies can be utilized to verify or discover the *prepare-for* relations among the alerts.

In our framework, we first extract necessary information from the alerts to identify the corresponding OS-level objects. We then verify the dependencies among alerts by using the OS-level dependencies among their corresponding objects, and thus improve the alert correlation based on the causal relationship. Moreover, by identifying the OS-level objects corresponding to the specific evidence indicating possibly missed attacks, and generating a forest of dependent objects by tracking back from these objects, we can improve the performance of existing methods [40, 67] for hypothesizing about possibly missed attacks.

An attack has to have impacts on the local system in order to be observable in the OS-level log. Since some unsuccessful attacks do not have the same impact on system objects as successful ones, our method only guarantees improvement of alert correlation for the alerts of successful attacks, though it may provide positive results for some failed attack attempts.

5.1.1 Identifying OS-Level Objects Corresponding to Intrusion Alerts

Now we discuss how to find the OS-level objects accessed by the attacks which trigger IDS alerts. We call this process the mapping of IDS alerts to OS-level objects.

We summarize the semantics carried by an alert that can be used to identify the corresponding OS-level objects. Firstly, an IDS alert comes with a timestamp, which indicates when the attack happens. Secondly, given an alert, we have the knowledge about how the attack works and how the system should behave in response to it. For example, given a Snort alert "FTP EX-

PLOIT wu-ftpd 2.6.0", we know that the corresponding attack exploits a vulnerable wu-ftpd server and forks a root shell. Finally, given local system's configuration, we can identify which OS-level objects correspond to each predicate in attacks' prerequisites and consequences. For example, a predicate "Samba server" may correspond to "/usr/sbin/smbd" process on a given computer. Below we discuss how each type of knowledge is used to map the alerts.

Though the number of logged events and objects is large in system logs, the timestamp of each alert can be used to easily narrow down the potentially relevant system objects. In Backtracker's log, each OS-level object or event is associated with a time period, which is the lifetime of the object or event. (The original Backtracker toolkit does not provide exact timestamp information. We slightly modified Backtracker's source code and added this functionality.) Given a fixed time period $T = [t_1, t_2]$, an object o can be accessed during T if $o's$ lifetime $[t_s, t_e]$ overlaps with T. Given the timestamp of an alert, we can estimate an approximate time window during which all the relevant OS-level activities occur, and then narrow down the scope of OS-level objects that need to be examined. Such a time window has to be relaxed to accommodate delays in OS-level operations and the clock discrepancy between the IDS sensor and the OS.

Given the name of an alert, we have the corresponding attack's prerequisite and consequence from experts' knowledge. According to [39], the prerequisite of an attack is a logical combination of predicates, and the consequence of an attack is a set of predicates. Each of those predicates is associated with some OS-level objects such as services, processes, and files. Thus, for each predicate in attacks' prerequisites and consequences, we can identify the corresponding file or process on the host computer, and represent them as ($predicate$, $OS\text{-}level\ object$) pairs in the knowledge base. For example, given a pair ($Samba\_service(host\_IP)$, "$/usr/sbin/smbd$") in the knowledge base, whenever there is predicate of $Samba\_service(host\_IP)$, we can locate its corresponding process of "$/user/sbin/smbd$". Thus, after identifying the predicates in an attack's prerequisite and consequence, we can identify the OS-level objects corresponding to those predicates on the host computer. Based on whether the corresponding predicate belongs to attack's prerequisite or consequence, those objects can be divided into prerequisite objects and consequence objects.

There are additional constraints for the mapped objects. Firstly, some predicate implies constraints on the properties of its corresponding OS-level objects. For example, the predicate $Root\_shell(host\_IP)$ implies the privilege of its corresponding OS-level object is root, which is represented by $uid = 0$ for the corresponding object in Backtracker's log. We represent such a constraint along with the object mapping information in the knowledge base in the form of ($predicate$,

$OS\text{-}level\ object, constraint$). In the above example, we may have a triple ($Root\_shell(host\_IP)$, $/usr/bin/sh, uid = 0$) to indicate that the predicate $Root\_shell$ is mapped to a process $/usr/bin/sh$, and the process's uid should be 0.

Secondly, if we focus on successful attacks, we expect to see the prerequisite and consequence of a successful attack at the OS level. In other words, for each predicate p in an attack's prerequisite and consequence, there must be at least one object o in the mapped object set corresponding to p, unless the logging tool does not monitor objects corresponding to such predicates. For example, assume the prerequisite and the consequence of a Samba buffer overflow attack are $Vulnerable\_Samba\_service(dstIP)$ and $Root\_shell\ (dstIP)$, respectively. If this attack is successful, there must be OS-level objects corresponding to these predicates in the object sets. Thus, if such OS-level objects do not exist, the alert must represent a false alert or a failed attack.

Finally, the system activities corresponding to an attack should all be related in Backtracker dependency graph. Moreover, the consequence of an attack should be dependent on the prerequisite of the attack at the OS-level. Thus, for each consequence object corresponding to an attack, it should be a prerequisite object, or dependent on some prerequisite objects of the attack. Thus, we have the dependency constraint: Given alert A and its mapped prerequisite object set P_A and consequence object set C_A, for each consequence object o in C_A, if P_A is not empty, there should exist paths from objects in P_A to o in Backtracker's dependency graph unless o is also in P_A. If such paths do not exist, the corresponding consequence object o should not be associated with the given alert. For example, given an alert with prerequisite object set $\{service\_process\}$ and consequence object set $\{shell\_process, file\_foo\}$, if $service\_process$ is not connected to $shell\_process$ in the Backtracker dependency graph, the consequence object $shell\_process$ should not be included in the consequence object set.

After mapping IDS alerts to OS-level objects and with OS-level dependency tracking tools, the number of false correlations can be potentially reduced by verifying the dependencies between the corresponding objects. Below we discuss in detail how to identify the OS-level dependencies among alerts with OS-level dependency tracking, and how to use such dependencies to achieve better accuracy in alert correlation.

5.1.2 OS-Level Dependencies among IDS Alerts

After IDS alerts are mapped to groups of objects within particular time periods in the Backtracker dependency graph, some groups are connected with each other through objects and

events in the dependency graph while others are not. It is not difficult to see that an later object is dependent on an earlier object if there exists a path in the dependency graph from the earlier object to the later object. Such paths among these object groups reveal the dependencies between their corresponding alerts. However, such dependencies can be not only malicious attack behaviors but also normal system activities, which makes it different from the prepare-for relations used in alert correlation. To find out the security-relevant dependencies interested by alert correlation, below we discuss in further detail about these dependencies.

As we have mentioned, the OS-level objects corresponding to an alert can be divided into two subsets: the prerequisite object set O_P, which are derived from the attack's prerequisite, and the consequence object set O_C, which are derived from the attack's consequence. As discussed earlier, the consequence objects in O_C should be dependent on the prerequisite objects in O_P. Moreover, two alerts being correlated with each other means the earlier attack's consequence "contributes" to the later attack's prerequisite. Thus, at the OS level, such a prepare-for relation should be reflected by the paths from the earlier attack's consequence object set to the later attack's prerequisite object set (i.e., the later attack's prerequisite objects are dependent on the earlier attack's consequence objects). To distinguish such dependencies from other dependencies, we say alert A is *strongly connected* with alert B if, in the Backtracker dependency graph, there exists a path from one of A's consequence objects to one of B's prerequisite objects. Thus, if alert A prepares for alert B, A should be strongly connected with B.

5.1.3 Verifying the Dependencies among Correlated IDS Alerts

As discussed in [31], there are several types of attacks that can evade Backtracker. For example, attacks exploiting modified guest kernels and attacks utilizing hidden channels. Such attacks cannot be accommodated by the techniques discussed in this paper. However, other than a few exceptions, Backtracker is capable of tracking OS-level dependencies among most other types of attacks. In other words, in normal cases, if two alerts are not found strongly connected with each other, there should not be prepare-for relations between them.

Given an alert correlation graph, we can map the alerts to OS-level objects and check whether the correlated alerts are strongly connected in OS-level dependency graph. If two correlated alerts are not found strongly connected, the correlation between the two alerts are considered a false correlation. For example, assume $A \rightarrow B$ are a pair of correlated alerts. To verify this correlation, we first map the two alerts into OS-level object sets. If the mappings are successful, there will be

corresponding prerequisite and consequence object sets: P_A and C_A of alert A, as well as P_B and C_B of alert B. By tracking back from the objects in P_B with Backtracker, we can verify whether the two alerts are strongly connected. If there does not exist a path between objects in C_A and objects in P_B, we consider the correlation between A and B false.

Note that two alerts being strongly connected in the dependency graph does not guarantee that the earlier one prepares for the other. This is because OS-level dependencies can be operations of benign programs. That being said, being strongly connected on OS-level dependency graph indicates that the involved attacks have higher possibility to be causally related. Thus, we can use this information to discover attacks missed by IDSs, which lead to missing correlations among alerts.

5.1.4 Facilitating Hypotheses of Missed Attacks

Integrating IDS alert correlation and OS-level event logging can also help in making hypotheses about possibly missed attacks. Several approaches [40, 67] have been proposed in making hypotheses about possibly missed attacks. Given evidence of attacks being missed, these methods search among known attack types of attacks to fill in the gaps between their correlation graphs and the evidence based on attacks' prerequisites and consequences. However, such search processes can be computationally expensive considering the size of the attack type knowledge base and the number of steps that could have been missed.

Integrating IDS alert correlation with OS-level dependency tracking can facilitate hypothesizing of missed attacks. Since the evidence studied as sign of missing attacks can be either IDS alerts or system objects, such evidence can also be mapped to groups of system objects. Assume evidence E is mapped to a set of OS-level objects. By tracking backward from these objects in the OS-level log, these objects can span a forest of system objects connected via various events. For any missed attack, unless it is one of the attacks that can evade the OS-level dependency tracking tool, part of its mapped objects must be in this spanned forest. Using the information of the correspondence between predicates in attacks' prerequisites/consequences and OS-level objects, this forest of objects can be converted to predicates. Then, the searching space for possibly missed attacks can be reduced to the set of attacks related to these predicates. Also, for each hypothesis candidate, we can validate it by trying to map it to OS-level objects. A hypothesis failing to be mapped is considered as invalid.

For example, an attacker attacks a host in the following steps: (1) Attacker successfully

launches a buffer overflow attack toward a vulnerable Samba server, which yields a root shell. (2) The attacker deletes the web page files via the shell. Now assume all those activities are missed by the IDS while the file deletion is detected by some file system integrity monitoring tool, which is taken as evidence indicating previous attacks being missed. By tracking back from the deleted file, the file is found dependent on the following objects in the OS-level event log: a smbd process and a shell process forked by the smbd process. Thus, when searching for possibly missed attacks, we can limit the search within attacks related to Samba and shell. Since only Samba is an initial system service, we hypothesize there is a Samba-related attack missed. By trying to map each candidate attack to OS-level objects, we can eliminate the majority of invalid hypotheses. The uncertainty within the remaining results is affected by the knowledge we have about the local system (e.g., the version of Samba) and the attacks (e.g., the number of attack types in the knowledge base).

5.2 Experimental Results

We have performed a series of experiments to validate the effectiveness of our method. Because our method requires that Backtracker monitor the victim system involved in attacks, we were not able to use the data sets available for IDS evaluation (e.g., DARPA's Grand Challenge Problem (GCP) datasets), which only include either tcpdump of attack traffic or simulated IDS alerts. To facilitate the evaluation, we developed three attack scenarios in our lab, in which an attacker launches a sequence of attacks against a computer monitored by Backtracker and Snort.

5.2.1 Experiment Setup

Our target machine is a linux server with a modified 2.4.20 kernel to run Backtracker. The Backtracker was slightly modified to add timestamps of system calls to its log. The server is configured to run two vulnerable services: Samba 2.2.8 and icecast 1.3.11. Snort 2.40 [46] was installed on the server to monitor the network traffic as an IDS sensor. To detect more attacks, we used the "Bleeding Snort Rulesets" [1] with Snort. Because both Snort and Backtracker are running on the same computer, clock drifting is not considered in our experiments. We also injected background traffic during the experiments to mimic an operational network. The background traffic was collected on the target machine when it was connected to our campus network, and was manually verified to contain no attacks toward the target machine. We also injected some failed attempts of

wu-ftpd buffer overflow attacks into the background traffic.

Table 5.1 gives the prerequisite and consequence of the attacks in our experiments, and table 5.2 shows the object mappings for the predicates involved in the experiments. Table 5.3 presents the implication between predicates related to our experiments.

Table 5.1: Prerequisites and consequences of alerts (i.e., detected attacks) in our experiments. (All alerts have two attributes: source IP address (srcIP) and destination IP address (dstIP).)

Attack Name	Prerequisite	Consequence
NETBIOS SMB trans2open buffer overflow Attempt	Vulnerable_Samba_service (dstIP)	{Root_shell(dstIP)}
FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow linux	Vulnerable_wu-ftpd_service (dstIP)	{Root_shell(dstIP)}
SHELLCODE x86 NOOP	N/A	{Shell(dstIP)}
ATTACK-RESPONSE id check returned root	Root_shell(srcIP) $\wedge$ id_command(srcIP)	N/A
P2P iroffer IRC Bot offered files advertisement	iroffer_service(srcIP)	N/A
DDOS tfn2k icmp possible communication	{tfn2k_server_daemon (dstIP)}	tfn2k_functions (dstIP)
SCAN nmap TCP	N/A	{Gain_host_info(dstIP)}
SNMP public access udp	N/A	{Gain_host_info(dstIP)}

Table 5.2: OS-Level objects corresponding to the predicates appeared in our experiments

Predicate	OS-level Object	Constraint
Vulnerable_Samba_service (hostIP)	"/usr/sbin/smbd"	N/A
Root_shell(hostIP)	"/usr/bin/sh"	uid=0
Root_shell(hostIP)	"/usr/bin/bash"	uid=0
Shell(hostIP)	"/usr/bin/sh"	N/A
Shell(hostIP)	"/usr/bin/bash"	N/A
id_command(hostIP)	"/usr/bin/id"	N/A
iroffer_service(hostIP)	"/home/attacker/iroffer"	N/A
tfn2k_server_daemon	"/home/attacker/td"	N/A

Now we present our experiment results. Due to the limit in space, we only discuss the reasoning process on Scenario 1 in detail, and only briefly present the result of the other two scenarios.

Table 5.3: Implications between Predicates

Predicate	Implied Predicates
Root_shell(hostIP)	{Shell(hostIP), tfn2k_server_daemon(hostIP)}

5.2.2 Scenario 1 Detail

Our first attack scenario exploits the vulnerable Samba service on the target server. It includes the following 4 steps:

1. We launched 2 remote buffer overflow attacks exploiting the vulnerable Samba server. A remote root shell session was created for each of the attacks.
2. Through one of the root shell sessions, we transmitted a pre-compiled server (daemon) file for the DDoS tool TFN (Tribe Flood Network) to the target host.
3. We then started the TFN server on the target machine through the same root shell.
4. We used the TFN's client program to communicate with the TFN server on the target server, and directed the target to start SYN flood and UDP flood attacks against another computer.

The above attacks took about 5 minutes. During this period, Backtracker logged 81,613 events. Moreover, the Snort sensor raised 26 alerts about these attacks:

- 9 "NETBIOS SMB trans2open buffer overflow attempt" alerts (No. 1–8 and No.10) for the buffer overflow attacks toward the Samba server,
- 15 "DDOS tfn2k icmp possible communication" alerts (No. 12–26) for the control messages sent to the TFN2K server daemon,
- 2 "ATTACK-RESPONSES id check returned root" alerts (No.9 and No.11) for the server's responses to the "id" commands, and
- 2 "ATTACK-RESPONSES id check returned userid" alerts for the server's responses to the "id" commands.

The background traffic triggered 32 alerts related to the target server:

- 8 “SCAN nmap TCP” alerts,
- 23 “SNMP public access udp” alerts, and
- 1 “FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux” alert.

Among the above 3 types of alerts, the third one is triggered by the failed attempt of wu-ftpd buffer overflow attack injected into the background traffic.

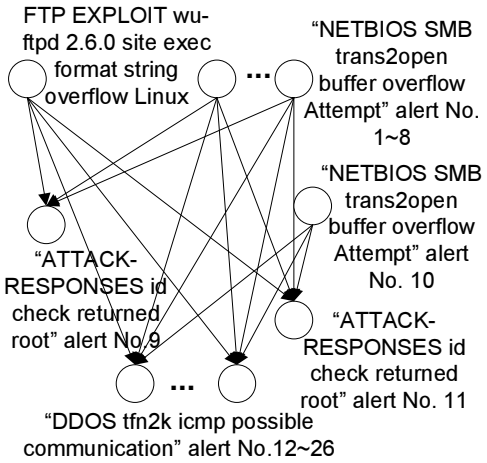


Figure 5.1: Original correlation graph

Table 5.4: OS-level objects corresponding to the alerts in scenario 1

Alert	Prerequisite Objects	Consequence Objects
“NETBIOS SMB trans2open buffer overflow Attempt” No.8	{smbd_2717}	{sh_2720}
“NETBIOS SMB trans2open buffer overflow Attempt” No.10	{smbd_2717}	{sh_2725}
“ATTACK-RESPONSE id check returned root” No. 9	{sh_2722, /usr/bin/id_324551}	Null
“ATTACK-RESPONSE id check returned root” No. 11	{sh_2727, /usr/bin/id_324551}	Null
“DDOS tfn2k icmp possible communication” No.12 26	{td_2737}	Null

Using the alert correlation method proposed in [39], we generated the correlation graph shown in Fig. 5.1. Obviously, it contains many false correlations due to the false positives within the reported alerts. Using the Backtracker's log and the semantics of these alerts, we mapped these alerts to a number of OS-level objects, as listed in Table 5.4.

For each alert prepared by other alerts in Fig. 5.1, we generated Backtracker dependency graphs (Fig. 5.2, 5.3, and 5.4) by tracking back from their prerequisite objects. Because we have discussed the first scenario in detail in the main text of the paper, we are only showing the full dependency graphs for verifying relationships between alerts in scenario 1.

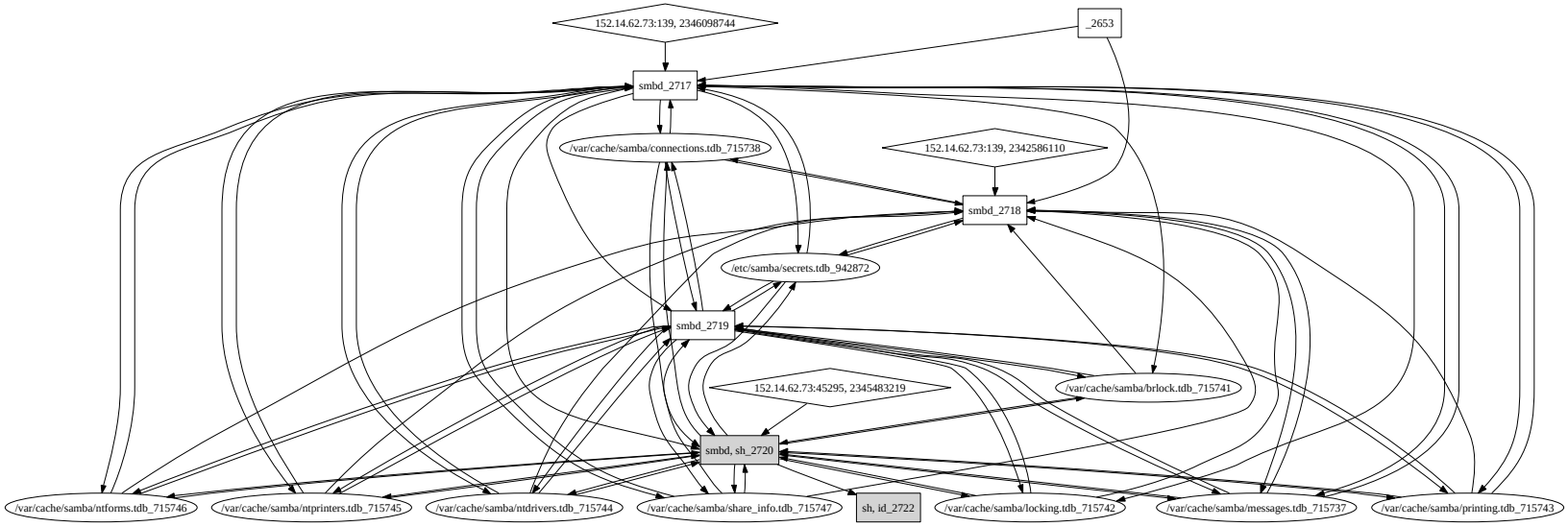


Figure 5.2: Whole graph between alert $No.8 \rightarrow No.9$ in scenario 1

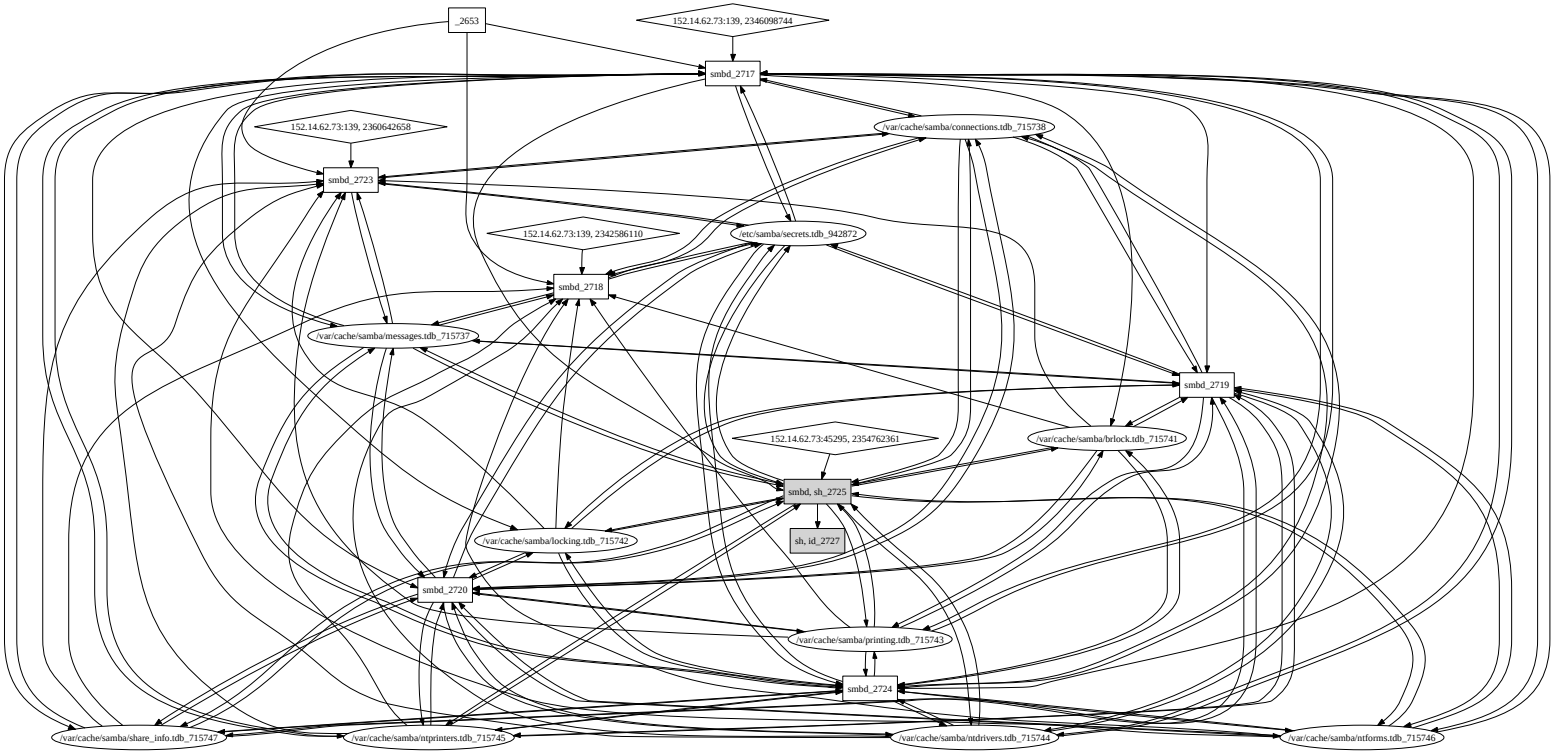


Figure 5.3: Whole graph between alert $No.8 \rightarrow No.11$ and $No.10 \rightarrow No.11$ in scenario 1

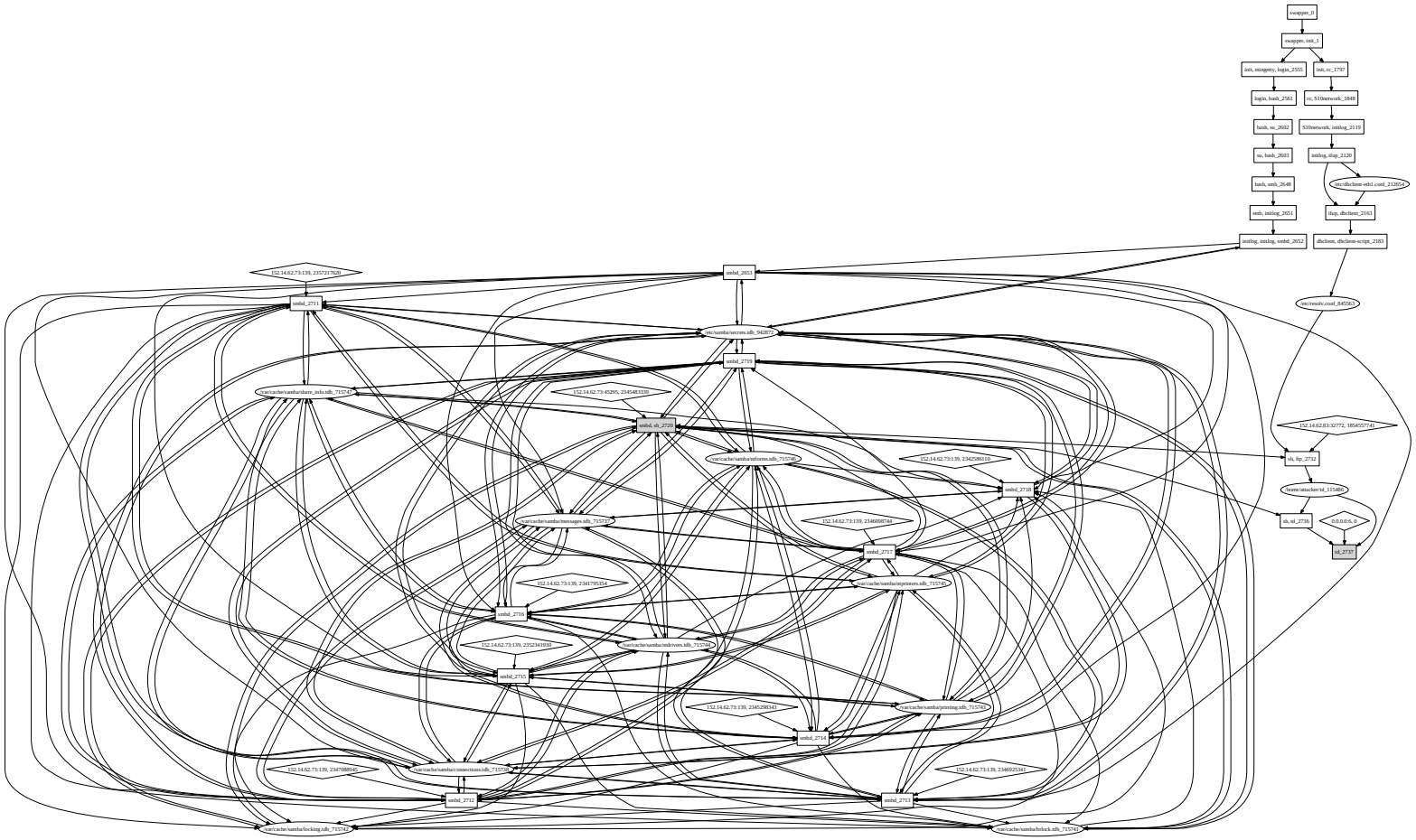
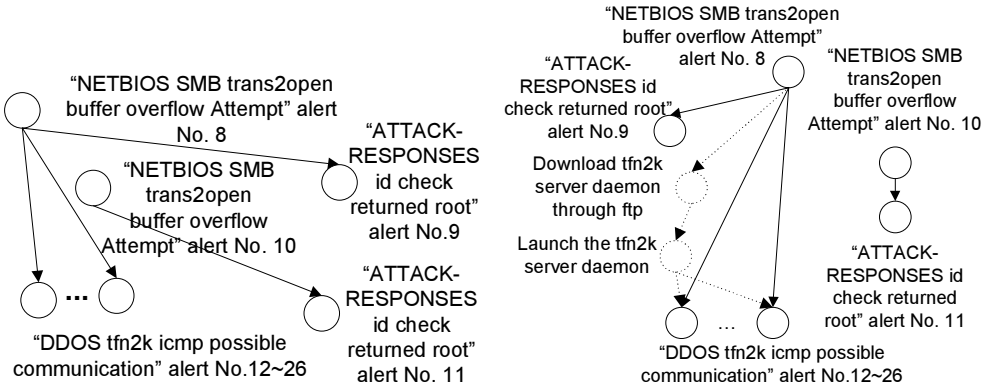


Figure 5.4: Whole graph between $No.12 - 26$ and others in scenario 1

In other words, we want to find in the corresponding Backtracker dependency graph paths from an earlier alerts' consequence objects to an later alert's prerequisite objects if the former prepares for the later. An example of such paths found in our experiments are shown in Fig.5.6. According to our previous discussion, when there exists such a path, the corresponding alerts are strongly connected and thus the correlations between them are verified at the OS level. Otherwise, the correlation would be considered false. In this way, we can verify each of the correlations in the original correlation graph, remove those that are verified to be false, and finally come up with a new correlation graph. The new correlation graph for scenario 1 is shown in Fig. 5.5(a). We can see it is the correct correlation graph of the reported Snort alerts based on the actual attack scenario.



(a) Without Hypothesized Attacks (b) With Hypothesized Attacks

Figure 5.5: New correlation graphs

In the correlation graph, the "NETBIOS SMB trans2open buffer overflow attempt" alerts prepare for the "DDOS tfn2k icmp possible communication" alerts, because the consequence of the former (i.e., Root_shell (dstIP)) implies the prerequisite of the later (i.e., tfn2k_server_daemon ()). However, this implication is based on the assumption that an attacker may install the TFN daemon if he/she can create a root shell on a victim computer. This indicates possible attacks have been missed between these two alerts. By backtracking from the prerequisite object set $\{td\_2737\}$ of the "DDOS tfn2k icmp possible communication" alert, a tree of OS-level objects are spanned. Because the consequence object set $\{sh\_2720\}$ of "NETBIOS SMB trans2open buffer overflow attempt" alert is among them, we focus on the paths linking the two object sets. Along the path connecting them as shown in Fig. 5.6(c), we found the following OS-level objects are involved: process object

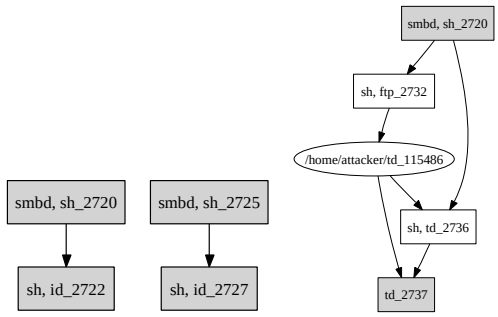


Figure 5.6: Paths between alerts' corresponding objects

"ftp_2732", file object "$/home/attacker/td\_115486$", and process object "td_2736". Thus, instead of guessing all possible ways to install a TFN daemon, which are numerous, we can limit the searching within the activities related to the ftp and tfn2k server program. It is easy to hypothesize that the attacker downloaded the tfn2k server program via ftp and launched it via the shell. These hypotheses are shown in dotted circles and lines in Fig. 5.5(b).

5.2.3 Scenario 2 & 3 Results

Scenario 2

The attack scenario is as below:

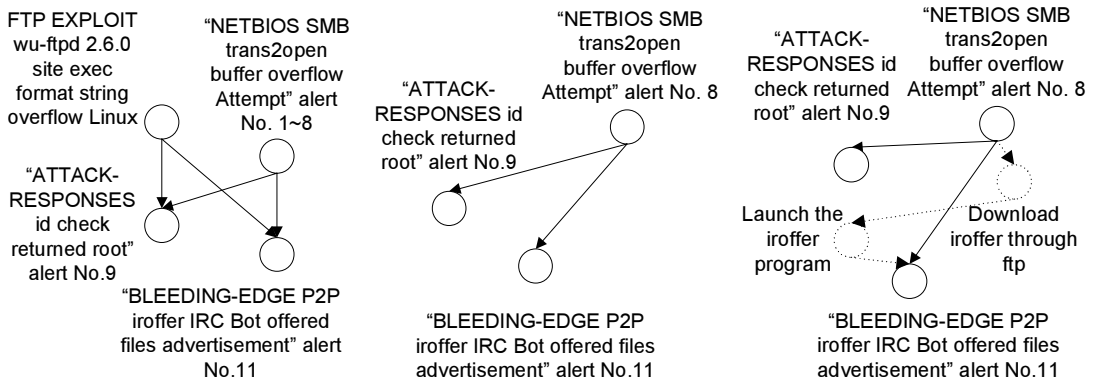
1. Gain root shell from samba trans2open overflow exploit.
2. Download the pre-compiled iroffer to the target server.
3. Launch the iroffer program and turn the target server into part of the attack's own P2P file sharing network.
4. Transferring files via this P2P network.

The Snort raised the following alerts toward the attack:

- 8 "NETBIOS SMB trans2open buffer overflow attempt" alerts, and

- 1 “ATTACK-RESPONSES id check returned root” alert, and
- 1 “ATTACK-RESPONSES id check returned userid” alert, and
- 1 “BLEEDING-EDGE P2P iroffer IRC Bot offered files advertisement” alert.

Using the same background traffic, and the original correlation graph is as shown in Fig.5.7(a). Similar to the analysis in Scenario 1, by mapping the alerts to the Backtracker log and verifying the dependencies between object $iroffer(pid = 3057)$ and $sh(pid = 2848)$, we have the new correlation graph as shown in Fig.5.7(b). The graph with hypotheses are shown in Fig.5.7(c), within which the hypothesized attacks and correlations are represented in dotted line.



(a) Original Correlation Graph (b) New Correlation Graph (c) New Correlation Graph With Hypotheses

Figure 5.7: Correlation graphs in scenario 2

Scenario 3

The attack scenario is as below:

1. Gain root shell by exploiting the buffer overflow vulnerability in icecast 1.3.11. (With 1 failed attempt.)
2. Through the root shell, add a new user.
3. Change the user group to root.

4. SSH to the target server using the newly generated user account.
5. Download the gzipped source code of iroffer to the server.
6. Extract the source code and compile it.
7. Launch the iroffer service.
8. Transfer files from the target via the iroffer service.

The Snort raised the following alerts toward the attack:

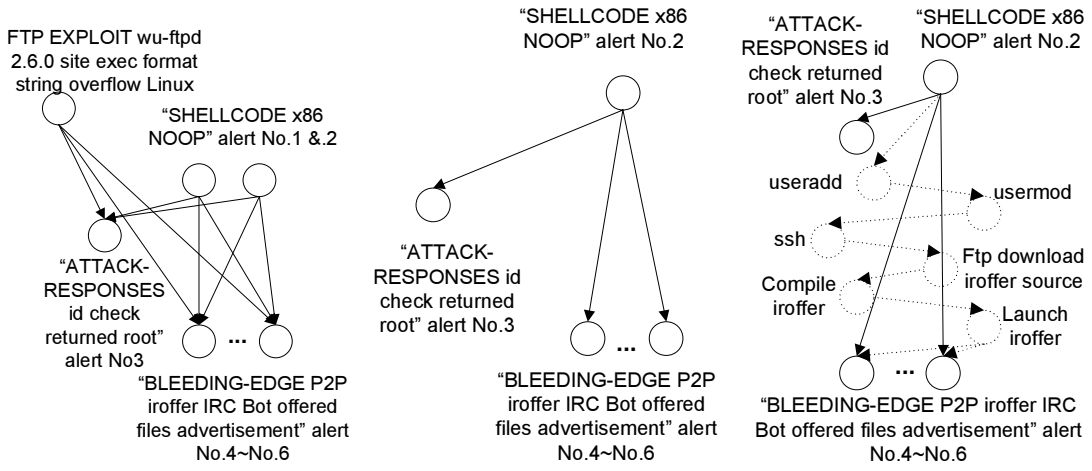
- 2 “SHELLCODE x86 NOOP” alerts, and
- 1 “ATTACK-RESPONSES id check returned root” alert, and
- 1 “ATTACK-RESPONSES id check returned userid” alert, and
- 3 “BLEEDING-EDGE P2P iroffer IRC Bot offered files advertisement” alert.

The original correlation graph is as shown in Fig.5.8(a). Similar to the analysis in Scenario 1, we mapped the alerts to Backtracker’s log. By verifying the dependencies between the shellcode alert ($icecast(pid = 2983) \rightarrow sh(pid = 2996)$ and iroffer alert $iroffer(pid = 11671) \rightarrow socket : /5143$, we have the new correlation graph as shown in Fig.5.8(b). The graph with hypotheses are shown in Fig.5.8(c), within which the hypothesized attacks and correlations are represented in dotted line.

5.2.4 Overall Evaluation

Assume an alert correlation method outputs that an alert A prepares for another alert B. If both alerts are detections of actual attacks, and the attack corresponding to alert A is indeed used to prepare for the later attack corresponding to alert B, we consider this correlation as a *true correlation*. Otherwise, it is considered a *false correlation*. Moreover, if one attack is used to prepare for another attack, but there is no correlation corresponding to these attacks (due to missing detection or incorrect correlation), we say there is a *missing correlation*. In our experiments, since we know the details of the attack scenarios, we can easily identify true, false, and missing correlations.

We use two metrics, false correlation rate and missing correlation rate, to evaluate the overall performance of alert correlation before and after integrating Backtracker’s results. Given a



(a) Original Correlation Graph (b) New Correlation Graph (c) New Correlation Graph With Hypotheses

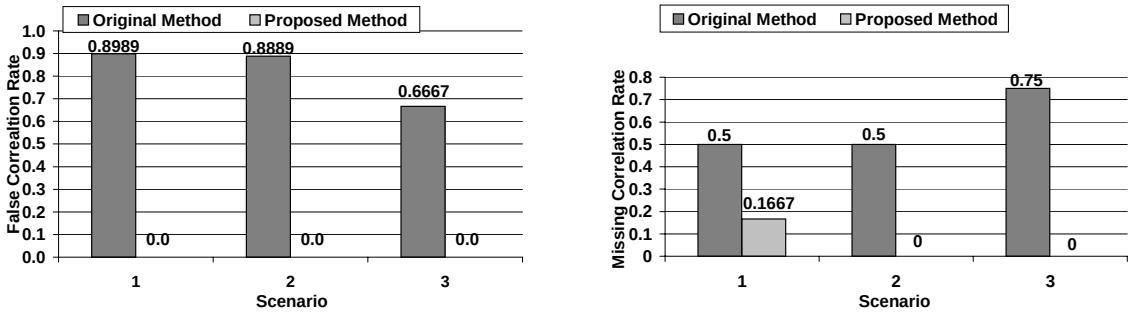
Figure 5.8: Correlation graphs in scenario 3

set of correlated alerts, the *false correlation rate* is the ratio between the number of false correlations over the total number of correlations generated by alert correlation. The *missing correlation rate* is the ratio between the number of missing correlations over the total number of correlations between *attacks*. Intuitively, false correlation rate shows how correct the identified correlations are, while missing correlation rate demonstrates how complete we can identify the correlations between attacks. Obviously, the smaller these two metrics are, the better performance alert correlation has.

Fig. 5.9(a) shows the false correlation rates in all three attack scenarios. As we can see, our proposed method reduces the false correlation rate significantly in all three scenarios. Indeed, false correlations are completely removed in all scenarios. This is not surprising, because OS-level dependency provides another way to properly verify the correlation between alerts through trustworthy information kept in OS-level logs.

Fig. 5.9(b) shows the missing correlation rates in the three attack scenarios. We can see significant reduction in missing correlation rate in all three scenarios. While the missing correlation rate is reduced to 0 in scenarios 2 and 3, the missing correlation rate of the first scenario is still non-zero after making hypotheses. This is because the DDoS attack (via the tfn server) is neither detected by Snort, nor hypothesized by our approach.

Our experiments also showed that OS-level object dependency graphs can often be too



(a) False Correlation Rate

(b) Missing Correlation Rate

Figure 5.9: Comparison between the original correlation method [39] and the Proposed method

complicated to understand in reality. For example, during the 30 minutes' attacks in the third attack scenario, Backtracker logged more than $410,000$ events, and the resulting dependency graph contains more than $4,000$ nodes. Analyzing such a complicated graph requires a lot of human experts' time and detailed knowledge about attacks' OS-level behaviors. However, instead of analyzing the Backtracker dependency graph directly, our method uses it as complementary evidence for IDS alert correlation. Because the method *verifies* alert correlations instead of *detects attacks*, only moderate information about attacks' OS-level behaviors is required, and the verification of whether there exists strong connections between correlated alerts can be automatically done by computer programs instead of human experts.

The experiment results also show how the proposed method can help make hypotheses about possibly missed attacks. In the attack scenarios used in our experiments, there is one type of attack that is hard to detect by all types of IDS, which are the attackers' "legitimate" activities after the break-in. Without the OS-level dependency information provided by Backtracker, it would be quite difficult to guess about such activities that prepare for the later attacks.

5.3 Summary

In this chapter, we developed a series of techniques to integrate the alert correlation method (based on prerequisites and consequences of attacks) and OS-level object dependency tracking. A critical step in this integration is to map IDS alerts to OS-level objects, so that connections between alert correlation and OS-level objects can be established. We also identified a number of

constraints that the OS-level objects should satisfy if they are relevant to the IDS alerts (or attacks) that are correlated. By using these constraints, we can verify the IDS alerts as well as the correlation between IDS alerts, and filter out false correlations. Moreover, the dependency between OS-level objects can also facilitate the hypotheses of attacks possibly missed by the IDSs by tracking OS-level objects. Our experimental evaluation gave favorable results, showing that OS-level dependency tracking can significantly reduce false correlations when integrated with the alert correlation method.

The proposed method has several limitations. First, it depends on experts' knowledge about attacks' OS-level behavior. Second, it is not as effective against kernel level attacks, which may corrupt the logs kept by, for example, Backtracker. Finally, to use OS-level information, we have to focus on attacks that have impacts on OS-level objects. Thus, it does not provide any performance guarantee for the correlation of failed attack attempts.

Chapter 6

A Flexible Privacy-preserving Framework for Security Event Data Sharing

In recent years, large scale distributed attacks such as worms and zombie networks have become more and more significant threats to Internet security. Such distributed attacks can be very destructive and very hard to stop from a single victim's end. This trend of distributed attacks poses an immediate necessity for the sharing of the information of security incidents to facilitate collaborated detection and analysis. Collaborated intrusion detection has been widely recognized as a necessary step forward to mitigate the growing threat on Internet security. However, the privacy issue becomes a big concern for organizations to participate in such sharing schemes. To protect the participants' privacy, it is necessary to sanitize the sensitive data before sharing it with the others.

Recently, some organizations and researchers proposed several approaches to address this problem [56, 35, 64, 65] with various data sanitization techniques. However, as we have pointed out in section 2.2.2, they all have certain problems in either privacy protection or correlation capabilities. One common problem with those aforementioned methods is the inflexibility to adjust the degree of privacy protection to accommodate the correlation needs. Because of the dynamic nature of the

security data, the sanitization's effect on the performance of various correlation analyses changes all the time. The schemes based on fixed configurations cannot provide appropriate trade-off between data usability and privacy. Balancing between data's privacy protection and correlation capability has become a critical issue in making the sharing schemes practical. Another problem with many of the existing approaches is the lack of systematic study on sanitized data's usability. Many previous works only discussed the usability on some particular type of analysis, which is not enough for a practical sharing scheme. In this chapter, we study a variety of analysis goals and threats for such sharing schemes, and propose a framework combining several existing and new sanitization techniques to achieve the goals. To solve the balance issue, we also propose a series of techniques to evaluate and gradually adjust the sanitization's effect on data's correlation capability to achieve flexible trade-off in the sharing model.

In the following parts of this chapter. We first define the analysis goal, the threat model, and some other issues related to security event data sharing in section 6.1, then we introduce our general sanitization and sharing scheme in section 6.2. After we present the sharing scheme, we discuss about the analyses supported by this sharing scheme in section 6.3. Afterward, section 6.4 presents our approaches to gradually release additional sanitized information to flexibly balance the usability and privacy. Then, section 6.5 discusses the framework's security against various types of attacks. Finally, section 6.6 concludes.

6.1 Sharing the Security Event Data

In this section, we model the information carried by security events, as well as outline the goals people want to achieve in analysis and the problems to be solved in achieving those goals.

There are many types of information that are security-related and can be useful in analyses if shared. For example, firewall logs, system call logs, program crash dump history, IDS alert reports, and etc. Different types of security information require different analysis skills and may contain different types of privacy concerns. As the starting point of this research, we will focus on the reports of security-related events such as IDS alerts and firewall alerts. In particular, we will use the intrusion alerts as an example to study the usability and privacy concerns of sharing the security event information. The techniques discussed in this paper can also be extended to the sharing of other security-related information.

6.1.1 Security Event Information

A security event can be the detection of an intrusion attempt (intrusion alert), the discovery of a virus (virus report), an activity taken by the firewall (firewall log), or etc. Different tools and devices may raise different formats of reports with different amount of information. However, as event information, all such reports should contain the following four categories of basic information that define an event:

Type The type information indicates what kind of event the report is about. It can contain a lot of semantics with the event based on the type definitions. For example, an IDS alert's alert name gives the type of intrusion the IDS has observed; the virus name in a virus scanning log gives which virus was found in the host.

Time The time information indicates when the event happens. In security event reports, such information usually resides in timestamps or sequence numbers of event entries.

Location The location information indicates where the event takes place. For security events, such information resides in the identity of the reporters and involved parties. For example, in an IDS report, the sensor's identity reveals in which part of the network has the events taken place.

Identity The identity information indicates who are the involved parties in the event. The identity can be the IP addresses of involved hosts in an IP network, or phone numbers in a telephone network, or node IDs in a peer-to-peer network.

There may be some other types of information describing an event, e.g., details about how the event takes place and some other parameters. However, as the starting point of this research, we will not discuss about those detailed information for the following reasons: Firstly, part of such information can be derived from the type information; Secondly, those details are not as relevant in correlation analysis as the type/time/identity/location information; Finally, details such as packet payloads and process dumps may introduce very complex security and privacy concerns if made public.

6.1.2 Privacy Concerns

There may be various privacy concerns during the sharing of security information, depending on the policies each participant may have and the different kinds of information to be

shared. However, those concerns can be roughly classified in terms of the information fields within the security event reports: type, identity, time, location, and etc. As we have mentioned earlier, we only study the the privacy concerns with the aforementioned 4 types of information in this paper.

Type Sharing a security event's type information can cause privacy concerns. They include: 1. Certain type information may reveal the service and security weaknesses within the systems in the network. For example, from worm alerts or botnet alerts, an outsider can learn that there exists compromised hosts or certain vulnerable services in the network. 2. The type information can be utilized by an active attacker to compose special patterns for later recognition, so as to recognize the identity of the event source. (We will discuss more details about this issue in section 6.1.5.)

Time There are privacy concerns with sharing the time information of security events because time can serve as an indicator for various correlation analysis. One example is that an attacker can selectively launch a series of attacks toward some certain hosts, and try to identify later whether the attacks have been detected by looking up the event log corresponding to the time period. This threat also falls into the probe-response attack. We will discuss more about it in section 6.1.5.

Identity Identity information is the most important privacy-concerning information. If attackers cannot identify the identities of the parties involved in the events, no matter what kind of information they can conclude from the shared event logs, the privacy of related parties is still not infringed. On the other hand, without identity protection, attackers can differentiate the events related to the parties of interest from all the other events, and thus to perform further analysis to gain more information about the involved parties.

Location Location information can cause privacy concerns because, in a structured network, the location information can help attackers reduce their search spaces for the actual identities of involved parties in the events. Due to the structure of network addresses, such information is usually derived from the identity information.

At this time, we will focus on the protection on the identity privacy because it plays a key role in linking the shared information with the involved parties. Because we can always encode the identity information into code words with limited lengths, the sanitization techniques applied to the source/destination IP addresses shown in the example can be also applied to any other types of identity information.

6.1.3 Intrusion Alerts

Intrusion alerts are usually reports from intrusion detection systems (IDSs) toward suspicious network/system activities that are considered as possible intrusions. Among various kinds of security events, intrusion alerts are especially important in sharing for the following reasons:

1. Many of today's attacks take place across multiple networks, which makes it very hard to analyze such attacks based on the data collected from one single network.

2. Current intrusion detection technology still cannot provide satisfying accuracy in detection. There will be attacks missed by IDSs and false alerts. Thus, reconfirming the detection results can help analysts to get a better idea about what has happened.

3. Distributed attacks and worm attacks are very popular in today's networks. Sharing the alert information can help administrators to detect the trends of such attacks and be prepared early.

Normally, an intrusion alert contains at least 7 fields of information: the source and destination host addresses, the source and destination port numbers, the name of alert (attack type), the timestamp, and the reporting sensor's ID. Among the 7 fields, the alert name and port numbers correspond to the type information of the event; the timestamp corresponds to the time information of the event; the source and destination host addresses, as well as the sensor IDs, correspond to the identity information of the event. As the identity of sensors can be easily protected by simply removing this information field from the alerts, we will focus on the privacy issues with the source and destination addresses within IDS alerts. To avoid confusions in this work, without losing the generality, we always use word "source address" to exclusively represent the attacker's address, and use word "destination address" to exclusively represent the victim's address. Given host A and host B, an alert $A \rightarrow B$ represents an alert of host A attacking host B.

6.1.4 Analysis Capabilities

The goal of sharing security event information is to make it available for various analyses. Some analyses are desirable for security administration. Those include:

Realtime Classification Analysis: Due to large numbers of false positives within IDS alerts, it is highly desirable that, in case a local IDS detects an intrusion incident, the local administrator can get supporting evidence from other collaborating sensors to support his belief in this

alert being true or false. One way to do so is to perform classification analysis based on host addresses and attack types and see if similar incidents have been reported. If so, the administrator can correlate those alerts with the local detected incidents and re-evaluate his belief in the particular alert. Since many alerts require administrators' immediate response, it is also desirable if this analysis can be performed in realtime.

Realtime Clustering Analysis: As we have mentioned earlier in the beginning of this chapter, large scale distributed attacks have become a very large threat to today's network security. One important goal of sharing security information is to seek opportunities to detect or prevent such attacks at their early stages. To analyze new emerging trends of attacks using shared intrusion alerts, we can cluster newly reported alerts based on the involved port numbers, alert type, and host addresses. For example, if suddenly a large number of alerts toward a specific port number are reported, it is possible that some vulnerability on the particular port is being exploited heavily, thus administrators can take necessary preventive actions such as tightening up the firewall rules on this port to better protect their networks. Similar to the previous discussion, the realtime property is also important for such applications.

Attack Scenario Analysis: Correlating alerts into attack scenarios can help the administrators to better understand the security flaws and fix them, as well as help the security researchers to learn the attacker's strategies and attack trends.

Further Information Sharing Between Participants: In case when attacks take place between hosts residing in different participating networks, it is possible for the involved parties to perform further sharing of more detailed information related to the attack incidents. The sharing scheme should support such communications and sharing.

After we present the details about our sharing framework, we will discuss the framework's support to these 4 types of analyses.

6.1.5 Threat Model

There are also some possible threats that can potentially infringe the identity privacy or disrupt the sharing and correlation analysis. Those include:

Diction Attacks: For any sanitization technique that utilizes public hash functions, dictionary attack is an immediate threat. Because IP address space is limited (32 bits), attacker can pre-

compute all the possible hash values within the space of possible addresses (the whole Internet, the whole subnet, etc.), and try matching them with the address fields in the shared alerts.

Probe-response Attacks: The attacker can intentionally launch a series of attacks to a known host with a carefully designed pattern composed with time sequence and type information. Then the attacker will try to identify the alerts corresponding to the same host within the sanitized alerts by recognizing the pattern and identifying the corresponding sanitized values. Such attacks are extremely hard to prevent in alert sharing because the type and time sequence information is critical in most analysis and cannot be modified much, and the time sequence and type information are all that this attack needs. In fact, all the existing alert sharing schemes, including [56, 35, 64, 65], are vulnerable to this kind of attacks to some extend. In our approach, although we also cannot fully prevent this type of attacks, we can mitigate the consequences of such attacks by limiting attacker's capability to launch such attacks, introducing extra uncertainties into sanitized addresses, and introducing extra uncertainties into the information used by attackers to compose patterns (e.g., using type abstraction to generalize the type information, and using time round-up to generalize the time information).

Insider Attacks: Participating parties may be malicious, sensors may be compromised. Thus, we need to consider the threat of having insider attacks in the sharing scheme. An insider may be capable of breaking the encrypted information with shared secrets, provide fake information during sharing, and even disrupt the sharing mechanism with denial of service attacks.

After we present our sharing scheme, we will study the security of the scheme against these attacks.

6.2 Privacy-Preserving Sharing Scheme

6.2.1 General Sharing Model

The sources of the alerts are a number of network intrusion detection sensors deployed in various organizations' networks. To provide the owners of the sensors the capability to perform clustering and classification analyses in realtime, it is necessary that each sensor sanitizes and reports the alerts as soon as they are detected, and the reported alerts should be accessible by all the

participating parties. Thus, the sharing should take place in an instant message sharing channel such as an IRC server or a mailing list, where only the authorized subscribers (IDS sensors and administration parties) can post and receive messages in the channel instantly. Each sensor should first sanitize the detected alerts before sending them to the sharing channels. After a certain period of time (e.g., a day or a week), the alert log of the channel will be made available to the public for research purposes. To make probe-response attack more difficult to execute and easier to detect, the alerts' time and type information can be further generalized before the data is released to the public.

6.2.2 Sanitization Scheme

From our previous discussion on the analysis goals and privacy concerns, given an alert $(SrcIP, SrcPort, DstIP, DstPort, AlertName, Timestamp)$, we have a few basic requirements for sanitization:

1. The port number and alert name should remain because these fields of information are essential for trend analysis and scenario analysis;
2. The source/destination address information can and should be sanitized because these fields can infringe the involved hosts' identity privacy;
3. The timestamp information cannot be greatly modified since it also plays an important role in many analyses.

Because organizations usually have higher privacy standards for their local hosts than for their remote hosts, it is necessary for individual sensors to classify host addresses into local addresses (the ones reside within the network monitored by the sensor) and remote addresses (the ones outside of the network monitored), and enforce double standards on them during sanitization. Because attacks between two remote hosts should not be detected by a local sensor, if a sensor does detect such an attack, it is very likely that a local host launched the attack with spoofed IP. The sensor can decide based on its own privacy policy on whether to report the alert with "spoofed IP" flagged (e.g., setting the source address to a special fixed value designated for spoofed IP).

There are two basic ways to protect the privacy of addresses: randomization [35, 65], and generalization [64]. In this framework we choose a combined approach utilizing both techniques to sanitize the addresses. More specifically, each address value is first converted to more general network addresses (e.g., by removing the last 8 bits, address $192.128.1.134$ is converted to

192.128.1.0). Then, we use keyed hash (e.g., HMAC) to encrypt the addresses in IDS alerts. Because organizations usually have different privacy standards for local and remote addresses, sensors may have different degrees of generalization for local and remote addresses, and each sensor uses two keys to encrypt the addresses: For remote addresses, it uses a secret session key K_s shared among all the participating sensors ($ADDR'_{remote} = HMAC(K_s, ADDR_{remote})$), so that everyone can verify the identity of the remote hosts attacking the participating networks; For local addresses, it uses a private secret key K_p to encrypt the addresses ($ADDR'_{local} = HMAC(K_p, ADDR_{local})$), so that only the local sensor can decrypt them.

6.3 Security Analyses Based on the Sharing Model

Now we discuss the various security analyses we can perform over the sharing model presented earlier.

6.3.1 Clustering Analysis

Clustering analysis is widely used in alert correlation and other kinds of security analysis. Based on certain attributes of the incidents, it clusters related events together into clusters, so that analysts can correlate these similar events together and take a more general view at the events. In our sharing model, the clustering technique can be applied to all the event data posted to the sharing channel. It is useful because: identifying similar security events taking place in multiple locations can help administrators to discover potential large scale or multi-hop attacks, and thus to launch further collaborated investigation between the involved networks. The realtime clustering analysis can also give people the opportunity to discover on-going attack trends from the reported events. To study the trend of attacks, analysts can perform realtime clustering analysis toward the reported events within a flying time window. The goal of this analysis is to identify groups of similar or related events within the latest past time period, and those groups of security incidents may correspond to some emerging large scale attacks.

However, because the identity information (source and destination addresses in alerts) is generalized during the sanitization process, it will cause increased similarity between alerts if addresses have been chosen as a feature in clustering. For more effective clustering, the priority of such features should be decreased in the clustering analysis.

6.3.2 Classification Analysis

Upon receiving the reported events with sanitized identity information in the sharing channel, analysts can perform classification analysis on the shared data. The classification can be based on various attributes of the events such as the alert names, port numbers, and addresses of alerts. Due to the sanitization, the capability of classification on addresses is limited and the results will contain uncertainty correspondingly. That is, when two sanitized address values match, it demonstrates that the two actual hosts share the same generalized network address. Given the degree of generalization being u bits, we can have the conclusion that the probability of the two addresses corresponding to the same host is $\frac{1}{2^u}$.

Classification analysis on distributed event data can provide very useful knowledge to analysts, which can be quite hard to find from local data.

Firstly, classifying the events based on their time properties (i.e., timestamps) and type properties (i.e., alert names and port numbers) can give a more general view of security incidents over the networks, which can help local administrators or analysts to make better judgments on locally detected security incidents. For example, given a specific alert, analysts may be unsure about the its truthfulness. Thus, the analysts can classify all the reported alerts during the same time period based on their alert names, so as to see whether there are similar incidents taking place during the time period. Large number of similar incidents may indicate an outbreak of such attacks, which increases analysts' confidence in the particular alert being true. Also, if many alerts have been reported against a particular host/network, that host/network is more likely to have been compromised thus the alert is also more likely to be true.

Secondly, although we have sanitized the source and destination addresses in alerts to protect hosts' identity privacy, because all the remote addresses are sanitized using hash functions, we can still get useful results from classifying the alerts based on the addresses. The output of the classification are categorized in (hashed) network addresses instead of individual hosts. Thus, we can still find out which networks correspond to the source of certain attacks and can use such information in worm/spam protection. For example, an administrator may tighten up the firewall rules for traffics from a network that has been active in attacking other hosts recently.

Thirdly, by performing realtime classification analysis to the alerts based on various alert attributes such as port numbers, attack types, and addresses, analysts can get the information about how the numbers of particular types of incidents change upon time. For example, analysts can compute the following statistics within sanitized alerts:

1. identifying the occurrence frequency of a particular type of alerts,
2. identifying the occurrence frequency of security events on specific ports, and
3. identifying the occurrence frequency of particular (hashed) network addresses in the reported alerts.

The above statistics are important in studying the emerging trend of attacks in the network. They can provide the type, port, and location information about recent popular attacks.

6.3.3 Attack scenario analysis

There are basically two categories of correlation methods that focus on intruders' attack scenarios: the correlation based on matching alerts with pre-defined attack scenarios, and the correlation based on matching individual alert's pre-conditions (prerequisites) and post-conditions (consequences). In general, both techniques can be break into two procedures:

- utilizing the type information to either match alerts into the positions in predefined scenarios or combine together the alerts with potential causal relationships, and
- applying a series of constraints, such as time and host address constraints, to filter out inappropriately correlated alerts.

Note that the sequence of the two procedures is interchangeable and can interlace with each other.

Because the identification information of events is sanitized, the host address constraints need to be adjusted in these correlation methods. Generally, a host address constraint is a set of equations on alerts' address values. For example, given a correlation between two alerts A and B, an address constraint can be $Destination(A) = Source(B)$, which indicates the destination address of alert A should be the same as the source address of alert B. After sanitization, uncertainty is introduced into the address values of alerts during the generalization process. Thus, the evaluation of host constraints needs to be modified correspondingly. Given two address values $Addr_A$ generalized by x bits and $Addr_B$ generalized by y bits (assume $x \geq y$), we say the two addresses match if the probability for the two addresses to represent the same host is larger than 0, which means $Addr_B$ can be further generalized into $Addr_A$. If $Addr_A$ matches $Addr_B$, the probability for the two addresses to be the same host is 2^{-x}. Such probabilities of constraints can be combined together to generate

the probability for the correlations between individual alerts, and can be used to evaluate and sort the correlation results.

6.3.4 Additional Information for Collaborated Analysis

The cross-network sharing of intrusion alerts provides opportunities for different sensors to collaborate in alert analysis. For example, when a sensor detects an attack from a remote host that is under monitoring of another sensor, it is possible for the two sensors to exchange information about the incident and thus to improve the detection performance.

Before we discuss in detail about such collaborated analysis, we need to define the situations when it is necessary and feasible for involved parties to further share the forensic information to facilitate collaborated alert analysis.

From a participating IDS sensor's point of view, we classify the reported alerts based on 2 criteria:

1. Where the involved hosts are located in (local network, remote participating network, or outside of participating networks), and

2. Which role (attacker or target) does the host play in an alert.

We call a host/address an internal host/address if it resides in one of the participating networks, and use symbol "$\mathcal{I}$" to denote an internal host/address. Correspondingly, we call a host/address an external host/address if it does not reside in any of the participating networks, and use symbol "$\mathcal{E}$" to denote an external host/address. Thus, based on the role of involved hosts, alerts can be classified into four categories: $\mathcal{I} \rightarrow \mathcal{I}$, $\mathcal{I} \rightarrow \mathcal{E}$, $\mathcal{E} \rightarrow \mathcal{I}$, and $\mathcal{E} \rightarrow \mathcal{E}$. Note that from an individual sensor's point of view, an internal host/address can be further classified into local and remote internal host/address.

We assume that, given an internal host address, each sensor knows the host's corresponding sensor. Thus, in case of a sensor finds that one of its local hosts is attacked by a remote internal host, since the other network is also participating in this sharing scheme, we would expect sensors to perform advanced sharing to gather more information from each other and perform further analysis based on that piece of information. This advanced sharing should be carried out in a request-response fashion. That is, the sensor on the target host's side would initiate a request for additional information of the attacker host from the sensor on the attacker's side, and the attacker-side sen-

sor will make the decision on whether to release the information based on the request and its local privacy policy.

When a sensor A detects an alert $I_1 \rightarrow I_2$, where I_1 is a remote internal address and I_2 is a local address to this sensor, to request for more information from I_1's sensor, it needs to prove to that sensor that this alert is real. Thus, the target-side sensor A computes a "ticket": $T = HMAC(K_s, I_1|I_2)$, attach this value to the sanitized alert, and send this message to the attacker-side sensor B via some private channel between the two sensors (e.g., a separate IRC channel, email, or some instant message tool). When sensor B receives the request, given the sanitized alert information and sensor A's identity, it can immediately identify the corresponding alert and verify the "ticket". Then, B can decide whether to release the previous records about the attacking host based on its own privacy policies. In case we have keyed hash chain $UIDC$s implemented in the sharing (see section 6.4.3 for details about $UIDC$), B can release the hash chain information of host I_1 to A, so that A can easily identify the previous alert history of host I_1 and thus to better analyze this attack incident.

6.4 Balancing the Sanitization Between Privacy And Usability

The uncertainty introduced into alert addresses during the generalization can cause problems in alert correlation. For example, in scenario-based analyses, when address constraints cannot be effectively applied, separate attack scenarios can be wrongfully correlated into one single attack scenario, which can make the result very confusing. To avoid such problems, sensor owners (organizations, institutions, etc.) may be willing to make limited compromise in privacy to improve the correlation performance. Hence, here we will first propose several measurements to evaluate the privacy protection and correlation capability on the sanitized data, then we will propose three schemes for making such trade-offs to support flexible sanitization control.

6.4.1 Evaluation of the Protection on Privacy

Privacy policies usually contain qualitative criteria such as "host A's address should not be revealed". However, to better study the privacy protection and to have more flexible controls, it is necessary to have quantitative measurements on privacy. Intuitively, the identity privacy can be defined as the capability for the other parties to identify the actual identities given the sanitized

data. Such identification capability is based on the information carried by (sanitized) data. Thus, following the approach in [64], we can use information entropy to measure how sanitization has affected such capability.

Given a sanitized identity attribute A, if each of its possible value a_i has a probability of p_i to be the actual original value from a third party's point of view, its entropy can be computed as

$$H(A) = \sum_i (-p_i log(p_i)).$$

We call this entropy value, which is computed using the general probability distributions of possible attribute values, the *local entropy* [64] of the data. Obviously, the higher this local entropy value is, the more uncertainty we have within the sanitized data, which corresponds to better protection on the sensitive data. Thus, we use the local entropy as the measurement to evaluate the protection of privacy.

6.4.2 Evaluation of the Correlation Capability of Sanitized Data

As we have mentioned earlier, too much uncertainty within sanitized alert information can cause confusions in alert correlation. Those problems are caused by the removal of too much diversity (variety) among the alerts during the generalization. A typical example of such problems is multiple separate attack scenarios being correlated together due to the ineffective host constraints. Because the network owners cannot make accurate assumptions about the analysts' correlation methods, it is impossible for the owners to evaluate how exactly the correlation is affected by the sanitization. Thus, we propose an entropy-based measurement on the change of attribute value distribution to serve as a *reference* to evaluate the effect of generalization on correlation analyses.

Let us consider a general sanitization scheme: Within a number of original alerts, there are n different values on a particular attribute (e.g., local addresses) of the alerts, and the total number of such attribute entries within these alerts is S. Assume those values are $N_1, N_2, \ldots, N_n$, and for each N_i ($1 \leq i \leq n$), there are n_i attribute entries being N_i ($\sum_i n_i = S$). After sanitization, the number of different values on this attribute is reduced to m ($m \leq N$). Assume those values are $M_1, M_2, \cdots, M_m$, and for each M_i ($1 \leq i \leq m$), there are m_i attribute entries being M_i. Thus, if we consider the total number of different attribute values as the total set of attribute value space, we can compute the entropy of these attribute values based on their probabilistic distributions before and after the sanitization:

Before sanitization: $H = -\sum_{i=1}^{n} \left(\sum_{j=1}^{n_i} (\frac{1}{S} log \frac{n_i}{S}) \right)$

After sanitization: $H' = -\sum_{i=1}^{m} \left(\sum_{j=1}^{m_i} (\frac{1}{S} log \frac{m_i}{S} \right)$

Note that the entropy values are computed based on the specific distribution of the attributes within the reported alerts instead of the general probabilistic distribution of attributes. To distinguish such entropy values from the *local entropy* values we just discussed earlier, we call this type of entropy the *global entropy*.

It is obvious that $H' \leq H$ ($H' = H$ only if $m = N$). Thus, given a fixed number of alerts, we can compute the *difference* between the global entropy values of the original alerts and the sanitized alerts, and use this *difference* value to measure how much variety the sanitization has removed from the original alerts, which, in many cases, will affect the performance of correlation.

In practice, based on the daily average volume of alerts raised by sensors, network administrators can select a fixed number S for the evaluation. For example, if we pick S to be 1000, we will be computing the global entropy value for every 1000 sanitized attributes. Then, the administrators can assign a threshold for the global entropy change according to historical data. Thus, by computing the global entropy differences and comparing with the threshold value, the administrators can decide whether the sanitization has caused too much confusion for analysis and it is necessary to release additional identity information about the alerts. For example, let us assume we have a threshold of 3.0 for 100 alerts with 100 different local addresses. The original global entropy is 6.64. When the local addresses are generalized into 20 different values, each sanitized value corresponds to 5 local addresses, the global entropy value would be 4.32. Thus, the change of global entropy is $6.64 - 4.32 = 2.32$, which is lower than the threshold. When the local addresses in those alerts are generalized into 10 different values and each value corresponds to 10 local addresses, the global entropy value for the sanitized alerts is 3.32. Thus, the change of global entropy is $6.64 - 3.32 = 3.32$, which is higher than the threshold. Thus, in the second case, it may be necessary to release additional identity information to improve the correlation capability of the sanitized alerts.

In the following we propose three schemes to release additional identity information. The first approach tries to increase the variety by marking out the attribute entries with the same original values; The second approach tries to release extra generalized bits to increase the attribute value variety; The third approach tries to partition the sanitized attribute entries based on the variety of their original values, so as to increase the variety of the sanitized values.

6.4.3 Information Release Scheme 1: Unique Identification Code

One way to increase the variety within the sanitized values is to mark out the sanitized attributes of the same original values. By doing so, analysts can distinguish the marked ones from the others even though they may have the same sanitized attribute value. For example, a sensor owner can find out all the alerts associated with one particular original address, and mark out the corresponding sanitized alerts. As a result, analysts can differentiate these alerts from the others with the same sanitized address value.

Because the time information and sequence numbers may be distorted in case of time generalization, these two fields cannot be used as accurate identifiers for marking the alerts. Thus, we have designed a unique identification code ($UIDC$) to be attached to each sanitized attribute.

In our specific alert sharing framework, because the local addresses are usually sanitized to a higher degree of generalization and cause more problems in correlation, our scheme can support releasing additional information on sanitized local addresses. The method of generating the identification codes in our address sanitization scheme makes use of *keyed hash chains*. Basically, we generate a keyed hash chain for each local IP address in the local sensor's alert log, and attach the hash values to those alerts. Following the time sequence in a local sensor's alert log, when a local host's IP address has its first appearance in an alert in the log, we will compute the initial value $U_0 = HMAC(K_0, SrcIP)$ of its corresponding hash chain, where K_0 is a private secret key only known by the sensor. When there is a new alert with the same local address reported later, we hash the previous value U_i ($i \geq 0$) in the hash chain with a secondary key K' to generate a new $UIDC$: $U_{i+1} = HMAC(K', U_i)$ for the new alert. Please note that K_0 is a universal value for all hash chains computed by the specific sensor, while K' is unique for each individual original addresses. When the number of alerts has reached the limit S, all the existing hash chains will end and new hash chains will be started from $U'_0 = HMAC(K_0, U_0)$ with a new $K'' = HMAC(K_0, K')$. By doing so, the alerts are divided into separate groups, and releasing information in one group will not affect the others.

When the sensor owner decides to release additional identity information on a local host, he can release the corresponding secondary key K' together with the initial value (U_1) in the hash chain to the analysts. With the two values, an analyst would be able to reconstruct the hash chain and identify all the sanitized local addresses marked with this hash chain. Thus, the variety within the sanitized alerts is increased and the analyst can apply the host address constraints more effectively in correlation. On the other hand, because key K' is unique for each individual hash chains, analysts

cannot find out any extra information about the other hash chains. Thus, the anonymity of the specific local host and the privacy of the other hosts are still protected against the public.

In case when the owner of the sensor has decided to release additional identification information when the global entropy change exceeds the pre-determined threshold, the owner needs to decide which local addresses' hash chains should be released. There are several rules for the selection of keys:

Firstly, because the privacy policy usually rules out the usability, no release will be performed for those events or hosts that are specified against such operations in the privacy policies.

Secondly, when there are multiple choices of keys to release, because we want to improve the usability with as few keys released as possible, we want to release the ones that can effectively increase the variety. For this particular reason, we call the amount of increase on the global entropy value caused by releasing a key the *efficiency* of the key or the original address, or simply the *efficiency* of the original address. Although the owners of the sensors do not necessarily need to always release the most "efficient" keys, it would be helpful if we can have a method to quickly find out the ones with high efficiency.

One immediate solution to the problem of finding the most "efficient" keys is to compute and compare the amounts of increase on the global entropy by releasing each individual keys. However, due to the amount of computation overhead involved, it is obviously not an efficient method.

Let a_i be the set of all the addresses with their original value being N_i, and b_i be the set of all the addresses with their sanitized value being M_i. Because addresses with the same original values also have the same sanitized values, and multiple original values may be sanitized to the same value due to the generalization, the sanitization process can be modeled as a mapping $\{a_1, \ldots, a_n\} \rightarrow \{b_1, \ldots, b_m\}$, and each member in $\{b_1, b_2, \ldots, b_m\}$ is a member or the union of several members of set $\{a_1, a_2, \ldots, a_n\}$. Thus, this mapping can also be looked on as partitioning the set $\{a_1, a_2, \ldots, a_n\}$ into m portions.

The global entropy before the sanitization is:

$$H = -\sum_{i=1}^{n} \frac{n_i}{S} log \frac{n_i}{S}.$$

The global entropy after the sanitization is:

$$H' = -\sum_{i=1}^{m} \frac{m_i}{S} log \frac{m_i}{S}.$$

Assume the key of set a_p ($1 \leq p \leq n$) is released, and $a_p \subseteq b_q$ ($1 \leq q \leq m$). Thus, the new global

entropy after releasing the key is:

$$H'' = -\sum_{i=1}^{m} \frac{m_i}{S} log \frac{m_i}{S} - (\frac{n_p}{S} log \frac{n_p}{S} + \frac{m_q - n_p}{S} log \frac{m_q - n_p}{S} - \frac{m_q}{S} log \frac{m_q}{S}).$$

Let $x = \frac{n_p}{m_q}$, and $k = \frac{m_q}{S}$, the above equation becomes:

$$H'' = -\sum_{i=1}^{m} \frac{m_i}{S} log \frac{m_i}{S} - (kxlog(kx) + k(1-x)log(k(1-x)) - klogk).$$

Take the second part of the right side of the above equation and let it be a function $f(x)$ of x, we have:

$$f(x) = kxlog(kx) + k(1-x)log(k(1-x)) - klogk,$$

where $0 < x < 1$. By computing the differential coefficient $f'(x) = k(logx - log(1-x))$, we know that the function $f(x)$ reaches its minimum when $x = \frac{1}{2}$, and $f(x)$ is symmetric to $x = \frac{1}{2}$. Thus, when k is fixed, the new global entropy has its maximum value when x equals to $\frac{1}{2}$, and the further x is different from $\frac{1}{2}$, the smaller the global entropy value will be. Similarly, we can prove that when multiple hash chains are to be released for the same sanitized value, the e

Similarly, consider the function of k:

$$f(k) = kxlog(kx) + k(1-x)log(k(1-x)) - klogk.$$

After simplifying the above equation, we have $f(k) = (xlogx + (1-x)log(1-x))k$, which is linear to k if x is fixed. Because $xlogx + (1-x)log(1-x) \leq 0$ if $0 < x < 1$, the bigger the k goes, the higher the global entropy value will be.

With the previous discussion, given two sets $\{x_1, k_1\}$ and $\{x_2, k_2\}$, where $|x_1 - 0.5| \leq |x_2 - 0.5|$ and $k_1 \geq k_2$, we know that releasing the key corresponding to $\{x_1, k_1\}$ will result a higher global entropy than releasing the key corresponding to $\{x_2, k_2\}$. So for a sensor owner to find the most efficient key to release, he can quickly identify the most efficient keys for each different sanitized value, then he can compare them to find the most efficient ones.

After keys are released to increase the variety within the sanitized alerts to the acceptable range (the decrease on global entropy value is below the pre-determined threshold), it is necessary to evaluate how the protection of privacy has been affected. Recall our discussion on using the local entropy values for the evaluation of privacy protection, we now compute the local entropy values for each sanitized address under different conditions.

When the analysts have no other information but the published sanitized alerts and a released key, the probability for a sanitized address to correspond to an original value still does not

change after its corresponding hash chain is released. Thus, the local entropy values on the sanitized addresses do not change after releasing keys.

If, by some means (e.g., a malicious sensor participating the sharing), the (malicious) analysts have gained the knowledge of the generalized address values before they are encrypted with keyed hash, given the degree of generalization being g bits, the local entropy on a sanitized address will be g bits. Releasing keys does not affect the local entropy because the probability for a sanitized address value being any value within the range of generalization is still $\frac{1}{2^g}$.

However, when the malicious analyst has gained not only the knowledge about the generalized address values in plain text, but also the exact original address value through probe-response attacks or some other means, releasing hash chains may affect the local entropy values. If attacker knows the exact value of an address and the key corresponding to its $UIDC$ is released, all the sanitized entries with the same original value will be identified. In other words, the privacy of all the probed hosts that have their $UIDC$ released is compromised. Assume t addresses are compromised by the release of $UIDC$, the local entropy value for the rest of the addresses would be $H = log(2^g - t)$. When t is small, the affect on other addresses can be ignored.

The risk of having the privacy of some addresses compromised in case of probe-response attacks is certainly not desirable. Thus, we propose a second scheme to reduce that risk with preserved uncertainty.

6.4.4 Information Release Scheme 2: Reveal Extra Bits

Another technique to increase the variety in the sanitized alerts is to carefully reverse the generalization process. That is, because we generalize the attributes by cutting off the last a few bits of their values, we can release some of the removed bits when it is necessary to increase the variety. In our case of address sanitization and sharing, due to the hierarchy of IP addresses, we release the extra bits in the sequence from high to low. For example, on a sensor enforcing generalization over the last 7 bits of the addresses, both address $10.1.1.194$ and $10.1.1.160$ will be converted to $10.1.1.128$, and an analyst will not be able to tell the difference between them. However, if the 26th bit on both addresses are released, which are different on the two addresses, they will make the difference and increase the variety within the sanitized values. Please note that it is still necessary for each sanitized alert to be published with some certain identifier. These identifiers are not necessarily as complex as the $UIDC$ we have discussed in the previous section, but they must be unique with each alert and not affected by the second-stage sanitization.

In order to protect the identity privacy, we must guarantee that the additional bits can only be used to distinguish different hosts and cannot give the analysts any more information about the original attribute values. To achieve this goal, the bit values cannot be released directly (otherwise, part of the original value is leaked). Instead, the sensor uses a random per-session key (which is used on all the attribute entries within the release session) or a series of per-attribute keys (which are used on corresponding sanitized attribute values within the release session) to hash the sanitized attribute values combined with the extra bits to generate the final values to be released. For example, assume two attribute entries a_1 and a_2 are both sanitized to b_1, as well as a_3 and a_4 are both sanitized to b_2. Let the first extra bit to be released be $1, 0, 1, 0$ for $a_1, \ldots, a_4$ respectively. Thus, we can either randomly pick a per-session key sk for all the 4 entries, or two per-attribute keys ak_1 for a_1, a_2 and ak_2 for a_3, a_4, to compute the keyed hash on $b_1|1$,$b_1|0$,$b_2|1$, and $b_2|0$. With the additional information, analysts are able to distinguish between a_1 and a_2, as well as a_3 and a_4. Using the per-attribute keys gives more security to the scheme: one key being compromised does not affect the others. Also, instead of releasing extra bits on all the sanitized attribute values, extra bits can be released for individual sanitized attribute values to improve the data's usability. It is easy for the sensor owners to compute the amounts of increase on the global entropy caused by releasing extra bits on each sanitized attribute value, and make their decision based on those amounts of increase.

Now let us discuss how the usability and privacy protection are affected after extra bits are released on all or some of the attributes. For an individual sanitized attribute value that has extra bits released, both per-session approach and per-attribute approach have the same effect on the involved attributes if we ignore the possibility of keys getting compromised. Thus, if we only release extra bits on one sanitized attribute value, assuming the attribute entries are partitioned into k groups containing the original values $a_1, a_2, \ldots, a_k$ respectively, and the total number of attribute entries is S, the increase on the global entropy values will be

$$-\sum_{i}^{k}(\frac{a_i}{S}log\frac{a_i}{S}) + \frac{\sum_i^k a_i}{S}log\frac{\sum_i^k a_i}{S},$$

which is always non-negative. Obviously, when the number of released bits is fixed, the increase on the global entropy is determined by the distribution of the bits for the particular sanitized value.

Because the bit release is performed with keyed hash, although the released bits can partition the sanitized attributes, without additional information, analysts still do not know what exact value the released bits are, and how many bits are actually released in order to form the partition. Thus, the local entropy of the sanitized attributes is not affected in this case.

When an attacker actually knows the exact original value for some sanitized attributes through probe-response attacks or some other means, the release of extra bits will help him to identify the other entries sanitized to the same value. That is, if he knows exactly how many bits are released, e.g., t bits, given another attribute entry in the same partition of the one probed, the probability for them to be the same becomes 2^{t-g} instead of the previous 2^{-g}. The local entropy for these sanitized attributes becomes $g - t$.

From the above discussion, we can see that, because there is still uncertainty kept within each group of sanitized attributes, even an attacker can identify some of them through probe-response attacks, he cannot be certain about whether the other attribute entries have the same sanitized value.

6.4.5 Information Release Scheme 3: Partitioning The Sanitized Attributes

Generically, the useful information provided by releasing extra bits is to partition the sanitized attributes into separate groups which corresponds to different original values. However, though it is convenient to partition with bits, providing such partitioning results also releases information about the differences of attribute values on the particular bits, which is unnecessary for ordinary correlation analyses but helpful for attackers to figure out the original values. Such unnecessary information leak can be avoided at the price of a little bit more complex partitioning method. Instead of partitioning the attributes based on the values of some particular bits, we can simply partition the sanitized attributes based on their original attribute values. That is, we partition a group of sanitized attributes G into a number of sub-groups so that, for any unique original attribute value, all its corresponding sanitized attribute entries in G are partitioned into the same sub-group. It is obvious that such partitioning on attributes will cause more variety on attribute values' distribution and thus to increase the global entropy.

Given a sanitized value s, assume that there are n different original attribute values corresponding to it: $o_1, \ldots, o_n$. Each o_i ($1 \leq i \leq n$) corresponds to a_i sanitized attribute entries, and we use A_i to denote the set of these entries. A partition on the sanitized value is to split all those corresponding attribute entries into multiple disjoint groups: $P_1, \ldots, P_m$ ($2 \leq m \leq n$), that each group is the union of one or more A_is.

Let the total number of attribute entries be S, the number of attribute entries with the same particular sanitized value be h, and the number of entries in each partition be k_i. After the

partitioning information is released, the increase on the global entropy is:

$$\frac{h}{S}log\frac{h}{S} - \sum_i (\frac{k_i}{S}log\frac{k_i}{S}).$$

Let $\frac{h}{S}log\frac{h}{S}$ be a function over $\frac{h}{S}$, and we can see it is monotonic and reaches its maximum value when $\frac{h}{S} = 1$. Also, let $\frac{k_i}{S}$ be x_i and the above global entropy be a function $f(x_1, \ldots, x_m)$. It is easy to prove using mathematical induction that function f reaches it maximum value when the partition is even. Thus, we have the following two conclusion:

1. When the number of partitions is fixed, the evener the sanitized attribute entries are partitioned, the higher increase we can get on the global entropy.

2. When the probabilistic distribution of the partition is fixed, the larger h goes, the larger the global entropy will increase.

Similar to the second scheme, partitioning does not affect the local entropy of partitioned attributes when there is no prior knowledge about particular attributes' actual original values. In addition, because the partitioning is not based on any specific bits of the original values, knowing the original value of an attribute in a partitioned group can only provide the attacker information on two issues:

1. The attribute entries partitioned into the other groups must have different original values than the one he knows. However, when the number of partitioned groups and the number of different original attributes identified by attackers are relatively small compared to the degree of generalization, the effect of such information can be ignored.

2. Correspondingly, the probability for the attribute entries in the same group to have the same original value will increase, and the amount of increase is determined by both the prior distribution of attribute values and the way how the sensor owner partitions the attribute entries. However, if we assume uniform distribution on attribute values and the partitioning is decided randomly, when the number of partitions is relatively small compared to the degree of generalization, the increase will also be small enough to be ignored.

Assume that n different original values generalized into one single sanitized value. Before partition, each attribute entry has $\frac{1}{n}$ probability to be a specific original value. If a group of S attributes are partitioned into m sub-groups with s_i members in each sub-group, and an attacker

has got the $(original, sanitized)$ correspondence information through probe-response attacks on k different original values. Thus, assuming uniform distribution, each sub-group may have $\frac{k \cdot s_i}{S}$ probed attribute values. Thus, for each un-probed attribute entry in the ith group, it cannot take the probed values in other sub-groups, and its probability to correspond to any of the rest original values is:

$$p = \frac{1}{n - k\frac{S - s_i}{S}}.$$

From the above equation, we can see that the smaller the s_i is, the larger probability we will have, which will result in lower local entropy. It is easy to see that even partition yields larger local entropy than uneven ones. In an ideal situation of even partition, the above equation becomes:

$$p = \frac{1}{n - k\frac{m-1}{m}}.$$

Thus, the smaller m and k are, the smaller probability values we have, which result in larger local entropy values. When m and k are both small compared with n, the effect on local entropy values can be minor. If the sensor owner has a lower bound on the uncertainty with sanitized values and can estimate the number of probed hosts, he will be able to estimate the upper bound of the partition numbers to maintain the uncertainty over its lower bound. For example, if the sensor owner decides that it is acceptable to have the uncertainty of each sanitized value decreased to $\frac{15}{16}$ of the original uncertainty, given $n = 2^8$ and estimated number of probed hosts being less than 20, the number of partitions should be less than $\frac{20}{20 - \frac{2^8}{16}} = 5$.

From the above discussion, we can see that, to effectively lower the global entropy with little compromise on local entropy values, sensor owners should:

1. partition those sanitized values with large number of sanitized attribute entries,
2. try to partition the attributes evenly, and
3. avoid partitioning a group of attribute entries into large number of sub-groups.

In practice, when partition is necessary, the sensor owner can just pick the sanitized value with the largest population, and try to partition it as evenly as possible (which is a knapsack problem [43] with the capacity of the sack being half of the total population). For details about various knapsack algorithms, please refer to [43].

Among the three information release schemes, we can see that scheme 3 is the more generalized form of the previous 2 schemes, scheme 1 and scheme 2 can both be looked on as

special cases of scheme 3. Because they both enforce strict rules in the partitioning (partitioning based on one specific original value, and partitioning based on the values of specific bits), they both provide chances for attackers to gain further knowledge with certain prior knowledge.

6.5 The Protection of Privacy in the Sharing Model

In this section, we discuss how the privacy of individual hosts are preserved through out the sharing scheme Our sharing and sanitization scheme provide pretty good privacy for alert sources. Because all the addresses are hashed using secret keys, both the source and the destination addresses are unreadable by human. Below we will study how secure the scheme is against some potential attacks.

6.5.1 Dictionary attacks

Because we use keyed hash for scrambling IP addresses and generating $UIDC$, it is not possible for attackers to perform any sort of dictionary attacks without knowing the keys used in hash functions, so our scheme is safe against dictionary attacks.

In case of attackers having the session key K_s used for keyed hash, the only result of dictionary attacks is the generalized network addresses, which still protect the actual hosts' identity by some uncertainty (e.g., 2^16).

Without actively launching any attacks toward a host, the only way an attacker can extract out the specific host's identity is by cracking the $UIDC$. Since it is also protected by keyed hash, it is computationally secure against brutal force attacks.

6.5.2 Probe-response analysis

Our method cannot achieve perfect protection of identity privacy against probe-response analysis. Attackers can always launch a series of carefully selected attacks toward a designated host with specific time intervals between them. Thus, the attackers can look into published alert reports for the special type and sequence/frequency patterns in their attacks, and thus to identify the corresponding sanitized addresses of the hosts probed. In this way, an attacker can identify the target networks' corresponding hash values without knowing the shared secret key K_s. Techniques such as time generalization and attack type abstraction [35] can only make it a little more difficult

for attackers to perform the probe-response attacks stealthily, but cannot fully deter it. For example, attacker can increase the amount of attacks to defeat time round-ups; attacker can pick a larger variety of attacks to form his attack pattern in order to defeat the type abstraction. However, the the uncertainties we introduce into the host IP addresses can prevent the attackers from further identifying the specific host in the alert log, as they can only identify the generalized addresses and cannot isolate the specific host's alerts from other alerts in this network.

One thing we want to point out is that it is extremely difficult to guard against probe-response analysis. Both [35, 64] cannot guard against such attacks. Actually, probe-response analysis can very easily identify the specific host's alert in [35]'s approach.

6.5.3 Insider attacks

If one of the participating sensors is compromised, it gives the attacker the following additional capabilities:

1. The attacker now has the shared key K_s used in hashing host addresses.
2. The attacker can report fake alerts to the shared channel.
3. The attacker can request to other sensors for advanced sharing and investigation, or to provide fake information when other sensors request for advanced sharing and investigation.

If an attacker has the shared key K_s, it can recover all the network addresses of *external* hosts in the alert reports. It can also get some internal network addresses in case some $\mathcal{I} \rightarrow \mathcal{I}$ alerts are reported. However, for the internal addresses in $\mathcal{E} \rightarrow \mathcal{I}$ and $\mathcal{I} \rightarrow \mathcal{E}$ alerts, because the keys used to hash those addresses are not shared among sensors, they are still protected. Also, because all the host addresses are generalized, the attacker still cannot get the exact host addresses involved in an alert unless the alert is monitored by the compromised sensor itself.

The attacker can fake alert reports sent to the sharing channel. By doing this, he/she can either hide the security incidents within the local network from the others, or send out a lot of fake reports regarding a particular network/host that he/she wants to defame. Too many fake alerts involving some particular remote internal hosts can be detected by the remote hosts' own corresponding sensor and possibly reported to the administrative parties of the sharing framework. Fake alerts involving external hosts will not be detected by the other sensors in the scheme. However, because the compromised sensor still needs to generalize the addresses in alerts in order to appear

normal to others, the attacker cannot defame a specific host at this stage until he/she gets some sensor into advanced investigation, which will be discussed later in this subsection.

The request for advanced sharing can only be approved by the other receiving sensor if it also detects the same alert-triggering event and it is on the attacker host's side of the event. Thus, the attacker owning the compromised sensor has to be able to generate such an event that is observable by the remote sensor in order to start advanced sharing, which is quite difficult since he/she needs hosts inside that network and spoof IP. However, if the attacker could send out spoofed attack attempts from a host inside the remote network to a host monitored by the compromised sensor, he would be able to pull out the exact alert history of the spoofed host through advanced sharing if the privacy policy in that network allows. Because sensors are usually well protected, and it is possible to detect IP-spoofing inside a local network in various devices such as gateways, routers, and firewalls, we consider the chance of having this threat to be quite rare.

When other sensors request for advanced sharing to a compromised sensor, the attacker can either simply deny the requests or approve the requests and release keys to the other sensors. Denying valid requests can only affect the effectiveness of collaborated analysis but does not infringe the remote parties' privacy. However, approving the requests and releasing fake information can cause the remote sensor to get more detailed information from attacker's previous fake reports, and can affect the corresponding administrators' assessment on particular hosts/networks' security status. For example, to defame a particular network/host, an attacker can inject a lot of fake alerts toward that network/host, and actively launch some attack attempts from a local address to remote internal hosts. When the remote networks' sensors detect the attempts and request for advanced sharing and investigation, the compromised sensor approves such requests and releases corresponding information about the fake reports. In this way, the attacker can get over the generalization and achieve his/her goal of defaming a particular party. Again, this threat is valid to our model but no party's privacy is infringed except for the network monitored by the compromised sensor.

6.6 Summary

In this chapter, we studied the privacy issues with security event information sharing. To solve those problems, we also proposed and analyzed several sanitization techniques as well as corresponding correlation analyses supported by those sharing and sanitization techniques. To study the privacy problems in sharing the security events, we first modeled the different information fields

within a security event; then we introduced the desirable analyzing capability on the shared security event data; afterward, we modeled possible threats to compromise the privacy in the information. With the help of all these studies, we proposed our framework of security event information sharing and corresponding sanitization techniques as well as some other techniques in privacy control and analysis.

The sanitization technique adopted in this sharing framework is a combination of generalization and randomization. The reason why generalization is necessary is that the uncertainty introduced by the generalization can help protect the privacy when randomization is compromised by various attacks (e.g., insider attacks and probe-response attacks). Compared with the approaches in [35], our technique provides much stronger cross-network correlation analysis capability and is more secure against the probe-response attacks. Compared with the approaches in [64, 65], our framework employs randomization technique to hide the information that is not necessary for third party analysts, and provide more analysis capability and control over data generalization.

The uncertainty introduced into the sanitized attributes not only improves the protection on data privacy, but also affects the performance of various correlation analyses. Thus, we proposed references to evaluate the privacy protection and the usability of sanitized data, together with two methods to release additional information to balance the privacy and usability.

Our framework also have its limitations. For example, as we have discussed in section 6.5.2, like all the other sanitization techniques, our techniques also cannot fully prevent the probe-response attacks. This is generic for all the sanitization techniques as long as the time sequence information and alert name information is preserved for future analysis during the sanitization period. However, by introducing uncertainty into the sanitized data, even if an attacker can launch such attacks, he still cannot get precise information about the identity of the event data. Another limitation is the communication overhead while transferring the event data to all the subscribers. Peer to peer may be a solution to this problem, I plan to further study into this issue and improve the performance of this framework in my future study.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

As computer systems and services are becoming ever more complex, various new exploits and attacks are reported every day. Securing the systems and networks cannot solely depend on prevention systems such as firewalls and passwords. It is important to deploy intrusion detection systems (IDSs) to monitor the systems and networks to notify the administrators on possible intrusive activities. However, due to the current limitations in the intrusion detection technology, today's IDSs suffers from three common problems: excessive alert amount, large number of false positives, and false negatives. Alert correlation analysis becomes a necessary step for people to extract the useful information from IDS reports and better understand the reported security incidents.

The problems with current IDSs also place difficult challenges in front of the analysts to perform effective and accurate correlation analyses – the results can be faulty and misleading if those issues (large volume, false positives and negatives) are not well handled.

My dissertation work focuses on improving the performance of correlation analysis of intrusion alerts by integrating multiple security information sources into the correlation analysis. In particular, I have studied the following problems.

Combining Multiple Types of Information in Alert Correlation. Observing the fact that the information from many other security tools such as vulnerability scanners, anti-virus softwares, and file system integrity monitors are more accurate and reliable than IDS alerts, as well

as the statistical nature of the experience knowledge from human experts, we build a correlation framework based on Bayesian statistics. By classifying various security evidences into state-based and event-based, and modeling the intrusion attacks as transformations between prerequisites and consequences, the causal relationships between intrusions and system states are established. Human experts' knowledge on the attack types are encoded into the topology of Bayesian networks, while their experience on the likelihood of intrusion alerts are encoded into the conditional probabilistic distributions associated with each nodes in the network. Thus, by performing Bayesian inference within the established Bayesian networks, we combine the various types of security evidence together into the alert correlation, so as to improve the correlation performance. Our experiments show that, after combining the more reliable evidence from other security devices, the confidence values in the real attacks are increased through the inference, while those of false alerts remain the same or get decreased. With this statistical framework, we can also hypothesize about possibly missed attacks and make reasonable assessments on the likelihood of the hypotheses.

The Bayesian Reasoning Framework's Robustness. One problem with the Bayesian reasoning framework is the ineluctable errors in human experts' assessments of prior confidence in individual attack types. It is necessary to perform a theoretical analysis on such Bayesian networks' robustness. We study this issue by analyzing the robustness issue from both quantitative measurements and qualitative measurements. We also discuss on how to use the results from sensitivity analysis and qualitative analysis to direct the evidence investigation more efficiently.

Combining OS-level Dependency Tracking with Alert Correlation. Through looking into the generic common features of attack scenarios, we observe that the attacks within the same attack scenario usually have dependencies between related OS-level objects. We study individual attack's OS-level behavior, and the types of OS-level objects and events tracked by the dependency tracking tools. By mapping individual attack's pre-conditions and post-conditions into OS-level objects and events, we establish the correspondences between intrusion alerts and OS-level objects/events. Then, we present the method to use the dependency tracking results over OS-level objects to verify the correlation results over IDS alerts, and the method to use the dependency tracking result to make reasonable hypotheses about missed attack steps within the attack scenarios.

Privacy Preserving Sharing and Correlation of Security Event Data. In spite of correlating alerts with other types of security evidence, we can also improve the correlation performance by combining the detected results from multiple sensors in the network. Such sharing and correlation analysis are especially helpful in detecting large scale network attacks. However, organizations' privacy concerns for the security alert data require that sensitive data be sanitized, which obviously

will affect the performance of correlation analysis. We first discuss the information fields within security event data, the analysis goals, and the privacy threat models in this problem scope. Then we focus on the privacy concerns on the identities within event data, and present our scheme of sanitization and sharing. The sanitization method employed in this scheme combines both generalization and randomization techniques to achieve better protection on the privacy. To better balance the usability and privacy of shared data, we also discuss on using entropy to evaluate the usability and privacy of the data, and propose three schemes to release additional identity information in case the usability is affected by the sanitization too much.

7.2 Future Work

There are still many problems in the field of alert correlation that are not fully addressed. In the following, I list two directions that I plan for further investigate.

Correlation analysis in detecting botnets A botnet is one of the most powerful weapon owned by malicious attackers. A botnet is a group of compromised hosts on the Internet, that communicates with the attacker and perform designated actions upon attacker's commands. Botnets are widely used in the business of spamming, DDoS attacks, identity theft, and many other attacks. Modern botnets are extremely hard to detect because various technologies such as p2p communication, polymorph, self-update are widely used to make the botnets stealthy. Existing techniques usually rely on detecting well known outdated botnet patterns such as special IRC commands or sequences of IRC operations, which is not applicable for more recent botnets. On the other hand, it is possible to detect the botnets by correlating all the abnormal patterns that can be associated with botnets and thus to direct further evidence gathering. For example, similarities among individual anomalies may be correlated together to identify possible coordinated attack; the patterns in time sequences and locations of security incidents may be correlated and identified to detect possible botnets. These are all challenging problems that need to be solved.

Trend analysis in distributed intrusion detection The outbreaks of large scale distributed attacks, such as worms and DDoS attacks, have become serious threat to Internet security in recent years. To control the damage of such attacks, it is desirable to detect the trend of outbreaks early, and to fix the vulnerability exploited fast. Obviously, these require distributed detec-

tion, effective analysis, and prompt communication. The problem is very challenging due to the large volume of data, privacy concerns in sharing the data, and critical requirements in processing speed. In chapter 6, we have discussed about the privacy issues and correlation analyses over shared alert data. However, there are still many more problems to be solved in this direction.

Bibliography

[1] Bleedingsnort. www.bleedingsnort.com.

[2] checkrootkit. http://www.checkrootkit.org. Accessed on Feb. 4, 2004.

[3] Javabayes. http://www-2.cs.cmu.edu/~javabayes/Home/. Accessed on Oct 10, 2003.

[4] Nessus. http://www.nessus.org. Accessed on Feb. 4, 2004.

[5] Samhain. http://la-samhna.de/samhain/. Accessed on April 4, 2004.

[6] Tripwire. http://www.tripwire.com. Accessed on Feb. 4, 2004.

[7] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, November 2002.

[8] D. Anderson, T. Frivold, A. Tamaru, and A. Valdes. NIDES software design, product specification, and version description documentation. Technical report, SRI International, July 1994.

[9] E. Castillo, J.M. Guitiérrez, and A.S. Hadi. Sensitivity analysis in discrete bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 27:412–423, 1997.

[10] Hei Chan and Adnan Darwiche. A distance measure for bounding probabilistic belief change. In *AAAI/IAAI*, pages 539–545, 2002.

[11] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[12] Veerle M. H. Coupé and Linda C. van der Gaag. Properties of sensitivity analysis of bayesian belief networks. *Annals of Mathematics and Artificial Intelligence*, 36:323–356, 2002.

[13] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001.

[14] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[15] F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Proc. of Recent Advances in Intrusion Detection (RAID 2000)*, pages 197–216, September 2000.

[16] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[17] O. Dain and R.K. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, pages 231–235, June 2001.

[18] O. Dain and R.K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pages 1–13, November 2001.

[19] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, LNCS 2212, pages 85 – 103, 2001.

[20] Marek J. Druzdzel and Max Henrion. Efficient reasoning in qualitative probabilistic networks. In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI-93)*, pages 548–553, Menlo Park, CA, 1993. AAAI Press/The MIT Press.

[21] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

[22] D. Farmer and W. Venema. SATAN: Security administrator tool for analyzing networks. `http://142.3.223.54/~short/SECURITY/satan.html`.

[23] S. Forrest, S. A. Hofmeyr, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings of 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, May 1996.

[24] Fyodor. Nmap free security scanner. `http://www.insecure.org/nmap`, 2003.

[25] A. K. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 14th Annual Computer Security Applications Conference*, pages 259–267, December 1998.

[26] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transaction on Software Engineering*, 21(3):181–199, 1995.

[27] Internet Security Systems. RealSecure intrusion detection system. `http://www.iss.net`.

[28] F.V. Jensen. *Bayesian Networks and Decision Graphs*. Statistics for Engineering and Information Science. Springer, 2001.

[29] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 12–21, December 2001.

[30] K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *The 8th ACM International Conference on Knowledge Discovery and Data Mining*, July 2002.

[31] S.T. King and P.M. Chen. Backtracking intrusions. In *Proceedings of the 2003 Symposium on Operating Systems Principles (SOSP)*, October 2003.

[32] S.T. King, Z.M. Mao, D.G. Lucchetti, and P.M. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of the 2005 Network and Distributed System Security Symposium (NDSS)*, February 2005.

[33] C. Ko, M. Ruschitzka, and K. Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of 1997 IEEE Symposium on Security and Privacy*, pages 175–187, Oakland, CA, May 1997.

[34] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications with probabilities on graphical structures and their applica-

tions to expert systems. In *Proceedings of the Royal Statistical Society*, B., pages 154–227, 1988.

[35] P. Lincoln, P. Porras, and V. Shmatikov. Privacy-preserving sharing and correlation of security alerts. In *Proceedings of 13th USENIX Security Symposium*, August 2004.

[36] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion detection expert system. In *Proceedings of 1988 IEEE Symposium on Security and Privacy*, May 1988.

[37] MIT Lincoln Lab. 1999 DARPA intrusion detection scenario specific datasets. `http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html`, 1999.

[38] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.

[39] P. Ning, Y. Cui, and D. S Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002.

[40] P. Ning, D. Xu, C. Healey, and R. St. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, pages 97–111, February 2004.

[41] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[42] J. Pearl. Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[43] D. Pisinger. *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen, January 1995.

[44] P.A. Porras, M.W. Fong, and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 95–114, 2002.

[45] X. Qin and W. Lee. Statistical causality analysis of infosec alert data. In *Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, September 2003.

[46] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA conference*, 1999.

[47] M.M. Sebring, E. Shellhouse, M.E. Hanna, and R.A. Whitehurst. Expert systems in intrusion detection. In *Proceedings of the 11th National Computer Security Conference*, pages 74–81, 1988.

[48] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of 2001 IEEE Symposium on Security and Privacy*, pages 144–155, Oakland, CA, May 2001.

[49] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: A new approach for detecting network intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 03)*, pages 265–274, November 2002.

[50] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2002.

[51] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[52] S. Staniford-Chen and L. Heberlein. Holding intruders accountable on the internet. In *Proceedings of 1995 IEEE Symposium on Security and Privacy*, pages 39–49, Oakland, May 1995.

[53] H.J. Suermondt and G.F. Cooper. Probabilistic inference in multiply connected belief networks using loop cutsets. *International Journal of Approximate Inference*, 4:283–306, 1990.

[54] Tauscan. `http://www.agnitum.com/products/tauscan/`.

[55] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*, pages 31 – 38. ACM Press, September 2000.

[56] J. Ullrich. DShield - distributed intrusion detection system. `http://www.dshield.org`.

[57] H. Vaccaro and G. Liepins. Detection of anomalous computer session activity. In *Proceedings of 1989 IEEE Symposium on Security and Privacy*, pages 280–289, Oakland, CA, May 1989.

[58] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 54–68, 2001.

[59] van der Gaag Linda, Hans Bodlaender, and Feelders Ad. Monotonicity in bayesian networks. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 569–576, Arlington, Virginia, 2004. AUAI Press.

[60] D. Wagner and D. Dean. Intrusion detection via static analysis. In *Proceedings of 2001 IEEE Symposium on Security and Privacy*, pages 156–168, Oakland, CA, May 2001.

[61] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of 1999 IEEE Symposium on Security and Privacy*, pages 133–145, Oakland, CA, May 1999.

[62] M. P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artif. Intell.*, 44(3):257–303, 1990.

[63] X-scan. `http://www.xfocus.org`.

[64] D. Xu and P. Ning. Privacy-preserving alert correlation: A concept hierarchy based approach. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC '05)*, December 2005.

[65] D. Xu and P. Ning. A flexible approach to intrusion alert anonymization and correlation. In *Proceedings of 2nd IEEE Communications Society/CreateNet International Conference on Security and Privacy in Communication Networks (SecureComm 2006)*, August 2006.

[66] Y. Zhai, P. Ning, P. Iyer, and D.S. Reeves. Reasoning about complementary intrusion evidence. Technical Report TR-2004-25, Department of Computer Science, North Carolina State University, 2004.

[67] Y. Zhai, P. Ning, P. Iyer, and D.S. Reeves. Reasoning about complementary intrusion evidence. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04)*, December 2004.