ABSTRACT

SERRANO, ALFREDO. Integrating Alerts From Multiple Homogeneous Intrusion Detection Systems. (Under the direction of Dr. Peng Ning.)

Intrusion Detection is a relatively young area of research, begun in the early 1980's. Currently most intrusion detection systems (IDSs) produce a large number of alerts based on low level attacks or anomalies. More distressing is that a large number of alerts are false positives. The false alert rate becomes even more important as networks become larger. Effectively monitoring a large network requires the deployment of multiple intrusion detection systems at key points on the network. Yet, this deployment increases the number of alerts that administrators must attend to. In addition, since most IDSs produce alerts based on low-level attacks, they give no indication about the relationship between alerts.

In this work, we describe a method for correlating intrusion alerts from low level alerts produced by multiple homogenous IDSs. Our technique extends the intrusion alert correlation technique developed at North Carolina State University, which uses an intrusion alert's *prerequisites* and *consequences* to construct high-level attack scenarios. The *prerequisite* of an alert specifies what must be true in order for the corresponding attack to be successful, and the *consequences* describe what can possibly be true if the attack succeeds. The extended technique relaxes the temporal constrains on alert from different IDSs to account for any possible timestamp inconsistencies (due to network delays, lack of system clock synchronization, host workload).

Our correlation method reduces alert volume, and improves performance with reduction in false positives compared to uncorrelated alerts. Our correlation of alerts from multiple intrusion systems provides for an automated method to show not only the relationship between alerts from one IDS, but also the relationships between alerts from different IDSs. Therefore, our method gives a more complete view of attack scenarios.

INTEGRATING ALERTS FROM MULTIPLE HOMOGENEOUS INTRUSION DETECTION SYSTEMS

BY

ALFREDO SERRANO

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF NORTH CAROLINA STATE UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTERS OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

RALEIGH

2003

APPROVED BY

_____

DR. PENG NING, CHAIRPERSON OF ADVISORY COMMITTEE

_____

DR. RUDRA DUTTA

_____

DR. DOUGLAS S. REEVES

DEDICATION

To my parents Lidia and Adolfo Serrano and my brothers and sisters: Miguel, Maria, Norberta, Carmen, Elia, Ramon, Edith, Procopio, Martha, and Elizabed.

BIOGRAPHY

Alfredo Serrano was born in Wilson, North Carolina. He graduated in 1995 from the University of North Carolina at Chapel Hill with a Bachelor of Arts in Psychology. In 1999, he received a Bachelor of Arts in Computer Science from the University of Texas at Austin. Alfredo worked at IBM in Austin before enrolling at North Carolina State University as a Master of Science student in computer science. During his time at North Carolina State University, he worked as a Teaching Assistant and interned at IBM in Austin during the summers.

ACKNOWLEDGMENTS

This work is partially supported by the U.S. Army Research Office under grant DAAD19-02-1-0219, and by the National science Foundation under grant CCR-0207297. We gratefully acknowledge this support.

I would like to sincerely thank all my colleagues in the Intrusion Detection Research Group: Pai Peng, DingBang Xu, Yan Zhai, and Donggang Liu for all the help and support.

I would like to express my sincere appreciation to my advisor, Dr. Peng Ning, for his guidance and constant support. His careful and critical comments significantly improved the content and presentation of this thesis. Thanks are also due to Dr. Rudra Dutta and Dr. Douglas S. Reeves for agreeing to be on my thesis research committee and providing valuable input.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

With the increased use of computers and computer networks, there has been an increased threat from hackers, competitors and others. Carnegie Mellon University's Computer Emergency Respond Team's (CERT) [2]Coordination Center reported 3,734 attacks in 1998, 52,658 attacks in 2001, and 82,094 in 2002. As a result, there has been an increase effort to protect network and computer resources. Stallings [50] identifies the goals of computer security as: confidentiality, authentication, integrity, non-repudiation, access control, and availability.

A number of methods have been identified to protect computers and networks from attack. Six general, non-exclusive anti-intrusion techniques are listed in [23].

1. Prevention: To preclude or severely handicap the likelihood of a particular intrusion's success. One can for instance elect to not be connected to the Internet if one is afraid of being attacked via it. It is an attempt to prevent attacks.

2. Preemption: To strike against the treat before it has had a chance to mount an attack. In a civilian setting, this is a dangerous (and probably illegal) approach.

3. Deterrence: Persuade an attacker to hold off his attack, or break off an on going attack. Typically accomplished by increasing the perceived risk of negative consequences for the attacker.

4. Deflection: Lure an intruder into thinking that he has succeeded when, in fact, he has been shunted off, or tricked away, from where he could do real damage.

5. Detection: This is where the subject under discussion fits into the greater scheme of things. Detection aims to find intrusion attempts, so that the proper response can be evoked.

6. Countermeasures: To actively and autonomously counter an intrusion as it is being attempted. This can be done without the need for detection, since the countermeasure does not have to discriminate between legitimate users that perform a mistake, and an intruder that sets off a predetermined response.

Current intrusion detection systems fall almost exclusively in the category of detection, even though recently, much interest has been given to the possibility of providing an automated response to detected intrusions. There are two general approaches to intrusion detection: anomaly detection and misuse detection. Anomaly detection deals with profiling user behavior. Misused detection deals with a priori prepared patterns or signatures.

The development of intrusion detection systems has normally focused on low-level attacks or anomalies. Alerts are generally raised independently, even though many of the events may be logically related. With the increase in network traffic and the number of intrusions, IDSs usually produce a large number of alerts, with a large portion of those alerts being false positives. The problem is farther exacerbated by the fact that to monitor a large network normally requires that more than one IDS sensor be used. This can lead to an unmanageable number of alerts for the network or security administrator. This has lead to a real need to develop techniques and tools to help improve the accuracy and quality of alerts.

Motivated by this, Ning, Cui, and Reeves have proposed a technique [36] [40] to correlate the alerts. Their technique correlates alerts generated by IDS, based on the *prerequisite*s and *consequences* of attacks. The *prerequisite* of an attack is the conditions that must be true in order for the attack to be successful, and the *consequence* of an attack specifies what is possibly true if the attack indeed succeeds. Logically linking attacks based on the

consequences of earlier attacks and the prerequisites of later attacks, allows for the correlation of attacks and the construction of attack scenarios.

In this thesis work, we expand on their correlation technique to allow for the correlation of alerts from multiple homogenous IDSs. We relax the temporal constraints on alert correlation to allow for timestamp differences that can occur between networked systems. The experiments in chapter 4 show the effectiveness of this technique to uncover the high-level attack scenarios from a set of alerts. The technique also shows a significant reduction in the false alert rate, after discarding uncorrelated alerts. The discarding of uncorrelated alerts also reduces the number of alerts that administrator must attend to. The method shows no loss in detection rate compared to correlating alerts from different IDSs using the original correlation technique.

1.1 Summary of Contribution

The original correlation technique showed potential in making alerts more manageable for the user, by showing the relationship between alerts to provide a high-level representation of attack scenarios and by filtering out most of the false alerts.

The goal of this thesis is to improve on the original correlation framework to allow for effective correlation of alerts from multiple homogeneous intrusion detection systems. To accomplish this goal, we extend the original method by introducing the concept of an inter-IDS timestamp difference. The inter-IDS timestamp difference accounts for the possible delays that cause differences in alert timestamps between IDSs. We then relax the temporal constraints of the *prepares for* relationship using the maximum inter-IDS timestamp difference. This allows for correlation of alerts even when inter-IDS timestamp differences are present. We validate our technique by implementing the proposed extension and running experiments using the DARPA 2000 data sets. Our experiments showed that the extended method was effective at uncovering multi-stage attack scenarios. They also showed that our method was still able reduce the false alert rate by filtering out most false alerts. In addition, our experiments showed that our method was more effective at correlating alert from

different IDSs than the original method. The extended method showed no reduction in detection rates compared to the original method. The original method had a higher missed correlation and false correlation rate than the extend method, when stages of the attack scenario appeared to overlap due to the inter-IDS timestamp difference.

1.2 Organization of the Thesis

Intrusion detection is a relatively young and problematic area of research. Chapter 2 reviews the related research work being done in intrusion detection, with a brief overview of work being done using alert correlation. Chapter 3 discusses the alert correlation technique being developed at North Carolina State University. It provides a detailed overview of the alert correlator framework and reviews the results of initial experiments. Chapter 4 discusses an extension of the initial framework that can enable the system to correlate alerts from multiple homogenous intrusion detection systems. It also covers details of the setup and the results obtained from experiments. Chapter 5 presents our conclusions and ideas for future work.

Chapter 2

RELATED WORK

The Intrusion detection area of research is about two decades old, beginning with James P. Anderson's technical report [7]. In his report Anderson divided the possible attackers of computer systems into four groups: external penetrators, masqueraders, misfeasors, and clandestine users. His report did not suggest mechanisms for intrusion detection, but it did present the notion of automatic detection of anomalies in computer systems. Since then, a number of approaches have been used in the development of intrusion detection systems. This chapter gives a background on intrusion detection (ID). Section 2.1 gives a background on intrusion detection and the various works done on the development of intrusion detection systems (IDSs). There are many taxonomies of IDSs; he we closely follow the IDS taxonomy presented in [37]. Section 2.2 reviews work done in the use of alert correlation in intrusion detection systems.

2.1 Intrusion Detection

The work done in the area of intrusion detection, takes many approaches in [10], [21] and [55]. Although there are many proposed taxonomies, generally intrusion detection systems can be classified under two general approaches anomaly detection and misuse detection. Anomaly detection models deal with profiling user or system behavior. They define a model of what is normal user usage of a system and any deviation from this model is viewed as anomalous. Misuse detection concentrates on known attack patterns, also called signatures. Intrusions are detected by matching a pattern on audit data.

2.1.1 Anomaly Detection

The work on anomaly detection concentrates for the most part on how to represent normal behavior and how to differentiate abnormal behavior in computer systems.

2.1.1.1 Statistical models

The earliest methods for the detection of intrusions were statistical modeling. Statistical models are used to aggregate the user's behavior; then use this aggregation to distinguish an attacker from a normal user. Statistical models can also be applied to programs and user groups. This group of IDSs includes systems such as IDES, NIDES/STAT, and Haystack.

The Intrusion Detection Expert System (IDES)[33], an early IDS, monitors subjects (users, remote hosts, and target systems) and maintains track of various statistics. Some of the measurements included CPU usage, command usage, and network activity. The measurements are kept in a real valued vector as summarized statistics for the session. Other vectors maintain data, such as standard deviation, for a particular subject. So, the users profile could be represented through maintaining the vectors. Deviant actions could be flagged by comparing the users session profile, with his normal profile. IDES allowed profiles to be "aged," by updating the user behavior once a day.

NIDES/STAT (Next Generation Intrusion Detection Expert System) developed by Stanford Research Institute in the early 90's uses statistical modeling to detect intrusions [6] [9]. NIDES/STAT monitors behavior of subjects (users, groups) on the system and adaptively learns what is "normal" for each individual subject. The subject's normal behavior is used to construct a profile, which is then stored. The system generates an overall statistic T^2, which reflects the abnormality of the user. The value is updated periodically. In addition, the profile of a subject has to be updated frequently to incorporate new data. Another statistical modeling IDS is Haystack, developed for the Air Force [46]. Haystack uses a four-step process to analyze the activities of a user. First, it generates a session vector to represent the activities of a user for a particular session. Second, it generates a Bernoulli vector to represent the attributes that are out of range for a particular session, and a threshold vector is

used to specify the point at which values are out of range for the user. Third, a weighted intrusion score is generated using the Bernoulli vector and an intrusion vector. Last, a suspicion quotient is generated for the session and compared with other sessions for a particular intrusion type. Haystack was designed to detect six types of intrusions: attempted break-ins, masquerade attacks, penetration of the security control system, leakage, denial-of-service, and malicious use.

2.1.1.2 Machine learning

Machine learning techniques try to actually learn a user's normal behavior, so that events are compared to how close they resemble the user's normal behavior. Machine learning methods include TIM[52] (Time-Base Inductive Machine), Instance based learning (IBL)[29], and Neural Networks [22]. TIM learns the user's behavior by discovering sequential patterns in a sequence of events. Then, these patterns are represented with rules generated using inductive generalization. The main weakness with this approach was that it only considered the immediately following relationship between events. IBL creates an instance dictionary (a set of instances that exemplify a concept) and new instances are classified by their relationship to stored instances. It uses a notion of "distance" between instances, so that similar instances have shorter distance between them and are classified together.

Neural networks are based on the theories of how the brain works, specifically, the formation of neural pathways in the brain during learning. Fox, Henning, Reed, and Simmonian proposed using SOM (self-organizing map)[20], a type of unsupervised learning to identify intrusions. The more common approach is to use some form of supervised learning, like back-propagation. The main difference between these two approaches is that unsupervised learning does not require training data as input to the neural network model, while supervised learning models do. A main problem with neural network techniques is that it is difficult to decide what the input parameters for a neural network anomaly detector should be. For example, in their experiments Ghosh showed that different initial weights could lead to performance differences in anomaly detectors [22].

With the success of using data mining techniques in business areas such as marketing and biomedical research, there has been a recent move to try to apply these techniques to intrusion detection. Audit Data Analysis and Mining (ADAM)[12] is one method of intrusion detection through data mining. ADAM tries to discover abnormal patterns in large amounts of data, such as network audit data. ADAM uses three main techniques: association rules, domain-level miming, and classification.

2.1.1.3 Computer Immunological Approach,

The computer immunological approach was inspired by the human immune system's ability to distinguish self from non-self [25]. The detectors in the immune system are lightweight autonomous agents (cells), and yet are able communicate with each other and identify elements in the body that are foreign. The computer immunological approach creates models in terms of sequences of system calls for a variety of conditions. The sequences can model normal behavior, error conditions, and attempted exploits. In [44], Sekar, Bender, Dhurjati and Bollineni use an automaton to represent a program's normal behavior. System calls made by a program are used as events to cause changes in states. One of the best-known computer immunology projects is that of the University of New Mexico (UNM). Researchers at UNM found that processes (such as sendmail, lpr, etc.) produce distinct sequences of calls to the system [11]. They produced a prototype system that uses simi-autonomous agents that monitor the system calls made by various processes and are able to distinguish a legitimate process (self) from an illegal one (non-self). The detectors monitor these calls, and if they vary from the known normal calls of the process, then an intrusion is occurring. They found that a window of only six system calls was sufficient to distinguish self from non-self.

2.1.2 Misuse Detection

Most commercial IDSs rely on misuse detection. The reason is that with today's technology, detection of known attack patterns is more efficient then trying to detect abnormal behavior of users and programs. Therefore the issue of how to represent the known attack pattern becomes very important. The more explicit and clear a representation of an attack pattern,

the easier it can be detected. There are number of ways to represent attacks, including rule-based languages, state transition analysis and color Petri automata.

2.1.2.1 Rule-Based Languages

The most common approach to represent attacks is through the use of a rule-based language. Attack patterns are represented as a set of rules, which a system uses to compare with intrusion data. Most often, the rules provide a sequential method to recognize an intrusion. One such rule-based language is RUSSEL. RUSSEL is flexible in describing sequential event patterns and corresponding actions. It provides for common control structures, like conditional, repetitive, and compound actions, through the use of primitive actions like assignment, external routine call, and rule triggering. RUSSEL programs analyze audit records one by one, executing all active rules for each record. The execution of a specific active rule may trigger new rules, raise alarms, write report messages, or alter global variables.

Another rule based language is P-BEST, developed for Multics Intrusion Detection and Alerting System, and later used by IDES, NIDES and EMERALD[17]. Like RUSSEL, P-BEST is a low-level language. It is a language preprocessor (i.e. Generates a precompiled expert system) and can invoke external C functions. In P-BEST, rules and facts are written in production rule specification language. The rules are translated into an expert system program in C language, which can be compiled into a standalone program or a set of library routines. The main problem with P-BEST is that specification of attack patterns is time consuming and the correctness of the rules can be difficult to verify.

2.1.2.2 State Transition Analysis

While rule-base languages are flexible and expressive in describing attack patterns, they can be difficult to use. To address this, State Transition Analysis Tool Kit (STAT) was introduced. STAT is based on state transition analysis to specify known attack patterns. State transition analysis models events though state transitional diagrams; attacks are viewed as a

network of states and transitions. Each state represents the status of the network, and each transition represents an event that occurs on the network. Therefore, an event (attack) can lead the system from a safe state to an unsafe state. With state transitional analysis, a finite state machine can be used to model an attack scenario. State Transition analysis allows complex intrusion patterns to be modeled in a simple way and can detect slow or distributed attacks. Implementations of STAT techniques include USTAT, NSTAT, and NetSTAT [54].

2.1.3 Intrusion Detection in Distributed Systems

Due to the rapid growth of the Internet and the development of large-scale systems, traditional IDSs are overwhelmed by the volume and complexity of data. Traditional IDSs were developed for small-networked systems and individual hosts, and lack the scalability to be applied to larger more complex network systems. The problem is that IDSs need to be more scalable to deal with the larger amounts of audit data from large distributed systems and should be able to understand heterogeneous information produced by other IDSs and component parts of the larger distributed systems. To address these issues intrusion detection research in distributed systems is mainly concentrated in three main areas: distributed IDS, network based IDSs and Interoperability between IDSs.

2.1.3.1 Distributed and Network-based Intrusion Detection Systems

Early distributed intrusion detection systems gathered information from multiple locations with sensors in multiple locations in the distributed system or network. Then the information was sent to a centralized location to be processed, for example DIDS [48] and ASAX [35]. These systems, although they did reduce some of the audit data, were still limited in how much they could be scaled before being overwhelmed by the amount of data that had to be processed. Recent distributed intrusion detection systems, such as EMERALD [31] and AFFID [49] attempt to deal with the scalability issue by reducing the amount of information that has to be analyzed at any one place. They often use a hierarchical structure so that lower level IDS components process the data they receive and send their results to higher-level

components. Each component deals with a limited amount of data, and intrusion information from different locations can be correlated together.

The organization of the hierarchy affects how effective and scalable the IDS can be. Ad hoc organizations are difficult to maintain and don't provide a general solution, while highly organized and preplanned hierarchies have trouble adapting to changes in attack patterns and locations. Generating the hierarchy dynamically is a more flexible and efficient method. For example, abstraction-based intrusion detection [38] generates a dynamic hierarchy by decomposing a specific signature into components (detection tasks) and then using a coordination mechanism to arrange the passing of messages between components so that the detection of attacks is distributed.

Another approach to deal with the scalability problem is the use of network-based intrusion detection systems. Instead of relying on host audit data, network IDS collects audit data from the network traffic. Such systems include NSM (Network Security Monitor)[9] and NetSTAT [54]. The advantages of using network based IDSs are: First, that standard structure of network protocols helps reduce the problems from heterogeneity in distributed systems. Second, network based IDSs normally run on a dedicated system, so do not burden the systems that they are protecting by using up system resources. Even so, with the increasing use of high-speed networks such as FIDDI (Fiber Distributed Data Interface) and giga-bit Ethernet, network based IDSs can have a difficult time keeping up with the amount and speed of traffic.

The main weakness of network-based intrusion detection systems is that they suffer from scalability issues as distributed IDSs. For example, although the collection of data may occur at different locations throughout the network, most network-based IDSs still analyze data at a centralized location. As mentioned earlier, network-based IDSs reduce some of the heterogeneity issues that distributed IDSs encounter, but they do not solve all the heterogeneity issues. In addition, it is difficult to keep up with the variety of network application protocols and systems employed.

We should note that the distinction between distributed IDS and network IDS, has recently become somewhat vague. This is because many network IDSs, like EMERALD now have distributed components, and many distributed IDSs such as DIDS have network detection components. Even pure network IDSs like Snort, allow for a distributed configuration [4]. Therefore, the distinction is more based on the fact that many distributed systems have host-monitoring components and network-based IDSs normally do not.

2.1.4 Intrusion Detection Problems

As mentioned, intrusion detection is a relatively new field in computer science. As a new field, it still suffers from a number of issues that although there is on going work to solve, currently no complete or absolute solutions exist. First, both anomaly detection and misuse detection have their limitations, and weaknesses for which there is no clear solution at this time. Second, increased network traffic can lead to IDS overloading, in which the performance of an IDS degrades, as it can no longer keep up with the amount of data it must process. Third, there is a lack of interoperability between IDSs. While different IDSs have their own strengths, the lack of interoperability limits how much advantage can be gained from using different IDSs to take advantage of this. Last, and perhaps the most pressing issue, IDSs produce a large volume of false alerts, which can overwhelm the network security administrator.

2.1.4.1 Limitation of Anomaly Detection

Anomaly detection has the potential to discover unknown patterns of attack. Yet, several issues limit the performance of anomaly detection systems, and may help explain why anomaly detection methods usually have high false-alarm rates. First, systems rely on being able to accurately develop a model of normal behavior. This model is often based on data collected over a period of time, in which there are assumed to be no intrusive activities occurring. Therefore, any intrusive behavior that does occur during period is recognized as normal behavior, and future occurrences will go undetected. Also, the possibility exist that a subject's normal behavior will change over time, most systems allow for gradual changes in

behavior and update the subject profile accordingly. Yet, this method can allow an intruder to gradually "train" the system into viewing intrusive actions as normal. The opposite occurs when the user quickly changes behavior, for example, when a user finishes one project and moves on to a new one. In this case, the new legitimate behavior can initially be classified as intrusive.

The building of anomaly detection models is also very difficult. Currently most models rely on domain experts (security experts) to determine the features that will be used as inputs. This technique can lead to missing features that can be critical to identifying anomalies. There is actually no correct way to select the input parameters, so each domain expert or group uses their own ad-hoc method. In addition, choosing features that are not important can introduce "noise" and affect the performance of the IDS.

2.1.4.2 Limitation of Misuse Detection

Misuse detection IDSs detect patterns of known attacks. While this leads to fewer false alarms than anomaly detection systems, this also means that misuse detection systems require that an attack pattern be identified. In other words, misuse detection systems are unable to detect new or novel attacks. Usually this requires human experts to understand attacks well and be able to generate a representation of the attacks. Generating attack patterns can be time consuming and error prone.

2.1.4.3 Overwhelming of IDS Due to High Traffic Volumes

Each IDS has a limit as to how much and how fast it can process traffic. Even with the use of specialized higher end hardware platforms that can capture traffic to ensure "no packet filter drops" [28] , an IDS can still be overloaded due to high volume of events or large number of "nuisance alarms." This overwhelming can be due to high spikes in traffic, or due to intentional overloading by attackers. A number of approaches have been proposed for tackling this issue. Kruegel et al. proposed partitioning the traffic on high-speed network links with the partitioning occurring in such a manner as to distribute traffic meaningfully to

a set of sensors with a limited set of detection rules. This approach relies on being able to divide the traffic into manageable subsets and requires that the partitioning occur in such way that each partition contains all of the evidence necessary to detect a specific attack. The limitation to this approach is that it is vulnerable to changes in traffic patterns that will overload a set of sensors, if the partitioning is static. In addition, if the partitioning is configured incorrectly, the sensors may not receive sufficient information to match their rules, which can lead to undetected intrusions. Another method is load shedding. Paxson suggested the use of load shedding to help real-time IDSs defend against overload attacks [41]. In load shedding, an IDS reduces the amount of traffic being analyzed. For example, if an IDS is being overloaded, it can stop analyzing http traffic to reduce its load. While, load shedding reduces the load of the stressed IDS, it may lead to undetected attacks as some traffic goes through unchecked. This is especially true, if the shedding is static, and the same protocol or service is shed every time the IDS becomes stressed. This allows for an attacker, to potentially take advantage of the "blindness," and in some cases if the attacker knows which service gets chosen, can plan to stress the IDS to mask his true attack.

Much effort has been placed on increasing the speed of the actual pattern matching in misuse detection systems to improve their performance. For example, recent improvements on SNORT (an open source network based IDS developed by Roesch)[4] [43] employ a fast pattern-matching algorithm to detect network misuse. This algorithm can match multiple signatures at a time, therefore leading to improved performance. Sekar, Guang and Shanbhag [45] have also developed a high-performance network IDS employing efficient pattern-matching algorithms, in which system performance is independent of the number of signatures.

2.1.4.4 Lack of Interoperability

Due to the increase in the number of commercial IDSs, there has been an increased need for the systems to communicate with each other. Research in this area currently under way has concentrated on providing a common way to represent data and exchange protocols. The US government's Defense Advanced Research Projects Agency (DARPA) was one of the first

groups to research the interoperability issues. The Common Intrusion Detection Framework (CIDF) began as a part of the DARPA Information Survivability program and was developed to allow different parts of the DARPA projects to interact with each other [42]. CIDF's main goal is to enable the interoperability and sharing of information and resources between different intrusion detection and response components. Using a communication framework and a common intrusion specification language (CILS), a Lisp-like language used to represent intrusion data; CIDF enables IDR (Intrusion Detection Resource) components to communicate via message passing. Under CIDF, IDR components are separated into four classes: event generator (E-boxes), event analyzers (A-boxes), event databases (D-boxes), and response units (R-boxes).

As part of Intrusion Detection Inter-component Adaptive Negotiation (IDIAN) project a CIDF negotiation protocol was developed [19]. The protocol allows information sharing by enabling IDS components to discover and negotiate the use of services between each other. The use of filters, a pattern of CIDF messages, helps negotiate services between CIDF components. CIDF components only send and receive messages if it matches one of the components filters.

Although the original CIDF project was ceased in 1999, it led to the creation of the Intrusion Detection Working Group (IDWG) within the Internet Engineering Task Force (IETF)[27] [3]. IDWG works to establish a standard for IDS interoperability. Among IDWG's proposals to the Internet Engineering Steering Group (IESG), is the Intrusion Detection Message Exchange Format (IDMEF). IDMEF is a data format that IDSs can use to report alerts. XML and an object oriented data model have been chosen to provide the base implementation. Another proposal by IDWG is the Intrusion Detection Exchange Protocol (IDXP). IDXP provides for the exchange of IDMEF messages, text and binary data. IDMEF is specified in the Blocks Extensible Exchange protocol (BEEP) profile. BEEP is an application protocol framework for exchanging messages between different intrusion detection systems and allows for the passing of information through firewalls. The main weakness with IDMEF is that it does not require uniformity of incident reporting. This allows vender specific

designation and a URL for incident reporting. This can lead to each IDS developer defining many unique attack descriptions.

2.1.4.5 High False Positives

It is not uncommon for an IDS to generate thousands of alerts per day. For the administrator it is very time consuming (sometimes impossible) to attend to every alert. This is made more difficult because large portions of the alerts turn out to be false positives. For this reason, there is a need to improve on IDSs to reduce the number of false alerts and to consolidate the amount of information presented to the user. The vulnerability of IDSs to high false positive rates has been well reported[32] [34]. Some even consider the need to reduce the high number of false positives, as a larger issue then improving detection rates. In his analysis in [8], Axelsson stated that " the limitation of IDS is not the ability to accurately detect misuse behavior but rather the ability to suppress false alarms." Presently, there is no clear method for reducing false positives.

2.2 Alert Correlation for Intrusion Detection

The high false alert rate (the high number of false positives) is a major issue for intrusion detection. Since every declared intrusion requires time to investigate, having a large number of false positives can mean that true positives go unattended. Most IDSs focus on low-level events, and therefore are unable to capture the logical steps or strategies involved in the attacks. To address these problems, alert correlation has been proposed by a number of researchers [15] [51][53]. Alert correlation, works on the presumption that intrusion events are not isolated, but that the intrusive events are linked; therefore the events should be correlated. Alert correlation methods fall into three classes: correlation based on similarities between alert attributes, correlation based on known attack scenarios, and correlation based on preconditions and consequences of individual attacks.

2.2.1 Correlation based on Feature Similarities

Correlation based on feature similarities simply compares the degree to which alerts have similar features (source address, destination address, ports, etc), and alerts with a high degree of feature similarity can be correlated. One such system is Probabilistic Alert Correlation proposed by Alfonso Valdes and Keith Skinner [53]. Their approach uses a mathematical framework for correlating alerts that are matched closely. The features considered by the system include sensor identification, alert thread, incident class, source and target IP address and TCP/UDP ports. For each feature, they define a similarity function, which determines the closeness of values. An overall similarity value is used to determine whether two alerts should be correlated.

While, correlation based on feature similarities can be used to correlate alerts and reveal multiple attack steps, it cannot fully discover the causal relationships between alerts. Data mining methods, which fall into this category, can cluster alarms based on root causes[26]. Although, feature based correlation can reduce the false positive rate, it cannot provide an attack scenario.

2.2.2 Correlation based on known attack Scenarios

An intrusion often requires several actions to take place in order to reach the intrusion objective. For example, if the goal is to carry out a DoS (denial of service) attack on the DNS server, the attacker could first do an nslookup, ping, and scan port 139 then a winnuke (sends out-of-Band data to an IP address of a windows machine). The earlier steps may be required in order to carry out the winnuke attack. An intrusion scenario is the corresponding steps required to reach the intrusion objective. For example, an illegal NFS mount scenario can be defined as follows:

1. Rpcinfo to know if portmapper is running on the target machine.

2. Showmount command, allows the intruder to see exportable partitions.

3. Mount enables intruder to mount one of these partitions.

4. By modifying the .rhost file of this partition, intruder gets root access on the target machine.

5. Last the intruder can rlogin to access the target system with root privileges.

Each step can be viewed as a stage of the overall attack.

Low-level alerts from IDS(s) are compared against the predefine scenario before the alerts can be correlated. Correlation based on this approach is proposed in [18].

Correlation based on known attack scenarios can reduce false positives, yet it does have weak points. First, it requires that the scenarios be specified by human users or learned though training data. This procedure can be time consuming since it requires human users to develop the scenarios, or come up with appropriate training data. Second, the amount of correlation that can be achieved by this approach is limited to known attack scenarios. Therefore, even if the underlying IDS can produce alerts that represent a multistage attack, the system would be unable to correlate the alerts unless the attack scenario was predefined.

2.2.3 Correlation based on causal relationships between attacks

Correlation based on preconditions and consequences works at a higher level then correlation based on feature similarities, but at a lower level then correlation based on known scenarios. For each attack, certain conditions must exist for the attack to occur; these are the preconditions of the attack. Similarly, certain conditions may exist after a specific attack has occurred; these are the consequences of the attack.

JIGSAW[51] is an approach that tries to find attack scenarios using specifications of individual attacks. The weaknesses of JIGSAW are that it remains a language, and there are no specific mechanisms for alert correlation. In addition, JIGSAW requires that all capabilities (preconditions) to be satisfied. What this leads to is failure to correlate alerts

when IDS fails to detect all events related to the attacks, or when an attack does not fully prepare for later attacks.

New research by Cuppens uses an approach that is similar to Ning's, in both method and functions. MIRADOR [15] correlates alerts using partial match of prerequisites and consequences of attacks. Since, our work is an expansion of Ning's correlation method, we will discuss the NCSU correlator in detail in the next chapter. The Framework for the MIRADOR consists of the following modules:

1. Alert based management: receives alerts from multiple IDSs and stores them into a relational database;

2. Alert clustering: accesses the data and generates alerts clusters that correspond to the same occurrence of an attack;

3. Alert merging: correlates global alerts in order to create candidate plans that correspond to the intrusion plan that is executed by an intruder;

4. Intention recognition: extrapolates the candidate plans to anticipate the intruder's intentions.

MIRADOR reduces the number of alerts early to generate global and "synthetic alerts". MIRADOR treats alert aggregation as an individual stage before alert correlation. This reduction in the number of alerts could potentially lead to loss of information. That could be used to detect a correlation.

Cuppens, Autrel, Miege, and Benferhat have recently introduced an extension to their work in [15]. In [16], Cuppens proposed extending attack correlation based on preconditions and consequences to include correlated attacks with intrusion objectives. Intuitively, they suggest that attacks can be correlated not only with alerts they prepare for, but also to intrusion objectives. An attack that can lead (directly or indirectly) to the fulfillment of an intrusion objective can be correlated with that objective. For example, get_file (Agent, File) can be correlated with illegal_file_access (file).

They also introduce the concept of anti-correlation. Anti-correlation, is the idea that if the occurrence of one event interferes that the occurrence of another then the two cannot be correlated (as true alerts). For example, the removal of a file A means that the file A cannot be illegally read. By recognizing that a sequence of correlated alerts can no longer lead to an intrusion objective, it can be used to reduce the number of alerts that the administrator needs to worry about. Cuppens claims that this can lead to the elimination of a category of false alerts because these alerts can no longer lead to an intrusion objective.

2.2.4 Attack and Scenario representation

Various languages have been proposed to represent attack scenarios. These include STAT [54], CPA (Colored-Petri Automata), LAMBDA [17] and MuSig[30]. For example, the MIRADOR project uses LAMBDA, a logic-based methods to specify the preconditions and post-conditions of attacks scenarios. However, currently most languages are limited in that they can only specify known attack scenarios. This limitation does not allow for the discovery of unknown attack scenarios. The NCSU correlation method describes *prerequisites* and *consequences* of individual attacks and then correlates the detected attacks based on the relationship between *prerequisites* and *consequences*. Therefore, it can potentially correlate alerts from unknown scenarios.

2.2.5 Multiple IDS Alert Correlation

To take advantage of the fact that different IDSs have different strengths, it would be beneficial to correlate alerts from multiple sensors. SPADE [24] presents an approach, but it is limited to portscans. Most of the work on alert correlation assumes that the alerts are produced by multiple sensors on the network[15] [53]. Yet, they don't present a method for handling timing issues or a method to convert heterogeneous alerts to a single representation. In Cuppens work, there is an assumption that all alerts are compliant with the Intrusion Detection Message Exchange Format (IDMEF). Therefore, they make no attempt to provide a method to get alerts from various IDSs in the same format.

The EMERALD group at SRI [53] uses probabilistic techniques to aggregate diverse sensor reports according to feature similarity functions. They present an approach that can handle some incomplete, inconsistent, and inaccurate alert reports. In their experiments they ran Snort and EMERALD sensors, and converted the Snort alerts to EMERALD reports by running Snort with IDMEF XML plug-in from Silicon Defense[4]. Then they implemented a translator to convert the XML alerts to EMERALD binary messages. Yet, their approach is limited to Snort and EMERALD messages, and therefore not easily extendable. In addition, they do not explain, how they deal with timings (network, system delays) differences that might be present when dealing with multiple sensors on a network.

Chapter 3

Intrusion Alert Correlator

This chapter describes the Alert Correlation framework developed at North Carolina State University (NCSU) and proposed in [39] and [36]. It also gives an overview of the implementation and results done in [40].

3.1 Framework

The intrusion alert correlation framework is based on the observation that intrusion events are usually not isolated attacks. They are usually related, in that an attack may be part of a much larger attack scenario, with earlier events preparing for later ones. For example, attackers hoping to carry out a buffer overflow attack against a Sadmind service will likely first carry out port scans to find out which port to attack. In this case the port scan prepares for the buffer overflow, so a Sadmind Ping alert (generated by the detection of the port scan) can be correlated to a Sadmind_Amslverify_Overflow alert (generated by the detection of the actual buffer overflow). The correlation occurs by identifying *prerequisites* (e.g., existence of vulnerable services) and *consequences* (e. g., the discovery of vulnerabilities) of attack types. The *prerequisites* are the necessary conditions for the attack to be successful, and the *consequences* are the possible outcomes of the attack. Correlation occurs by matching the *consequences* of some earlier alert and *prerequisites* of a later alert.

The basic unit of correlation is the individual attack. The alerts for a series of attacks can be correlated based on the *prerequisites* and *consequences* of each attack. The existence of *prerequisites* of an attack does not mean that a preparing attack exists. That is, a preparing attack does not have to occur in order to carry out a particular attack. For example, an

attacker need not launch an attack to find out if a service is running in order to launch a buffer overflow attack against it. He may just blindly assume that the service is running and guess the port. Yet, if an attacker does launch a preparing attack, then the preparing attack should be correlated with the prepared-for attack.

In the following sections, the formal approach to the NCSU alert correlation method is described.

3.2 Prerequisites and Consequences of Attacks

As mentioned earlier *prerequisites* and *consequences* are the preconditions and results of an attack, but in order to use these concepts there must be a logical representation that can be translated into code. This correlation method uses predicates to represent *prerequisites* and *consequences*. For example consider a buffer overflow attack against the UDP services (assuming that the vulnerability exist), which can lead to an attacker gaining root access by modifying the .rhost file. The *prerequisite* and *consequence* can be represented as:

Prerequisite:

UDPVulnerableToBOF(VictimIP, VictimPort) □
UDPAccessibleViaFireWall(VictimIP, VictmPort)

Consequence:

{GainRootAccess(IP), rhostModified(IP)}

Logical operators □(conjunction) and (disjunction) are used to express prerequisites and consequences with multiple conditions.

As shown by the example above, multiple consequences can be related to an attack. This is represented as a set of predicates. The set of predicates is essentially a conjunction of predicates and can be represented using a single logical formula. Therefore the consequence:

{GainRootAccess(IP), rhostModified(IP)}, can be represented using the logical formula: *GainRootAccess(IP) ⇒ rhostModified(IP)*.

The *consequences* of an attack are the possible results of the attack, not necessarily the actual outcome of the attack. Therefore, this represents the worst-case consequences. For example, port scan does not always reveal the port a particular service is running on.

3.3 Hyper-alert Type and Hyper-alert

The basic constructs of *prerequisite* and *consequence* of each type of alert are used to define the notion of a *hyper-alert type. Hyper-alert type* encodes the knowledge about a specific attack. It can be formally defined as:

Hyper-alert Type: A Hyper-alert type T is a triple (*fact*, *prerequisite*, *consequence*), where:

1. *fact* is a set of attribute names, each with an associated domain of values,
2. *prerequisite* is a logical combination of predicates, whose free variables are all in *fact*
3. *consequence*: is a set of logical formulas such that all the free variables in consequence are in *fact*.

Fact gives what kind of information is reported along with the alert. *Prerequisite* specifies the conditions that must be true in order for the attack to be successful. *Consequence* specifies what can be true if the attack succeeds. An example of a hyper-alert type could be *SadmindBufferOverflow*, and could be defined as the triple({ VictimIP, VictimPort }, ExistHost(VictimIP) ⇒ VulnerableSadmind(VictimIP), { GainRootAccess(VictimIP)}). Intuitively, the hyper-alert type provides the IP address of the victim host (VictimIP), and what conditions should exist for the attack to be successful. In this case, there should be a host at IP address VictimIP and the Sadmind service should be running on port VictimPort and be vulnerable to a buffer overflow attack. The *consequence* is that the attacker may gain root access on the given host.

The hyper-alert type is just that, a type of hyper-alert. In other words, it defines a hyper-alert, but is not an instance of a hyper-alert. That means, that it cannot be linked to a specific attack detected by an IDS. We define a hyper-alert instance as follows:

Hyper-alert: A hyper-alert (instance) h of T, where hyper-alert type T=(*fact*, *prerequisite*, *consequence*), is a finite set of tuples on *fact*, where each tuple is associated with an interval-based timestamp[begin_time, end_time]. The hyper-alert h implies that *prerequisite* must evaluate to True, and all the logical formulas in *consequence* might evaluate to True for each of the tuples.

An instance of hyper-alert of type SadmindBufferOverflow could be {(VictimIP=152.1.19.5, VictimPort = 1235), (VictimIP = 152.1.19.7, VictimPort = 1235)}. We can refer to the instance as $h_{\text{SadmindBufferOverflow}}$. This implies that for the attack to be successful the logical formulas ExistHost(152.1.19.5) □ VulnerableSadmind(152.1.19.5) and ExistHost(152.1.19.7) □ VulnerableSadmind(152.1.19.7) must be True. It also implies that the possible *consequences* GainRootAccess(152.1.19.5) and GainRootAccess(152.1.19.7) may be True after the attack. Intuitively, the hyper-alert says that there are buffer overflow attacks against Sadmind at the IP addresses 152.1.19.5 and 152.1.19.7 and can result in the attacker gaining root access on those hosts. Here we treated the corresponding timestamps implicitly. Technically hyper-alert $h_{\text{SadmindBufferOverflow}}$ has a start and end time. For simplicity, we will omit timestamps if they are not explicitly necessary in our discussion.

The *fact* component of the hyper-alert can on one level be seen as relational schema, and a hyper-alert as a relation instance. The *prerequisite* and *consequence* are instantiated by replacing the free variables in *prerequisite* and *consequence* with specific values. If *fact* consists of multiple tuples then *prerequisite* and *consequence* can be instantiated multiple times. For example during an IPSweep attack, several IP addresses may be involved, then *prerequisite* and *consequence* of the corresponding hyper-alert type will be instantiated for each of the addresses involved.

An instance of a hyper-alert may refer to one or more alerts. If an alert has all the information necessary to instantiate a hyper-alert, then a hyper-alert can be generated for that alert.

Sometimes, an IDS reports multiple alerts or a series of alerts for one attack (for example, RealSecure reports three alerts for an FTP [5] and EMERALD may report multiple alerts for an attack that spreads over a period of time). In this case, to instantiate a hyper-alert may require the information from one or all the alerts. A hyper-alert can also represent an aggregation of all related alerts. If several alerts for the same type of attack are reported within a short period of time, then they can be aggregated into one hyper-alert. This allows for some flexibility in reasoning about alerts.

To correlate hyper-alerts, we check if an earlier hyper-alert contributes to the *prerequisite* of a later one. Specifically, we decompose the *prerequisite* of a hyper-alert into parts of predicates and test whether the *consequence* of an earlier hyper-alert makes some parts of the *prerequisite* True (i.e., makes the prerequisite easier to satisfy). If the results are positive, then we correlate the hyper-alerts.

Prerequisite set: The *prerequisite set* of h, denoted as P(h), is the set of all predicates (with the preceding negation if there is one) that appear in *prerequisite*.

Consequence set: The *consequence set* of h, denoted as C(h) is the set of all predicates (with preceding negation if there is one) that appear in *consequence*.

A given hyper-alert h, which is an instance of type T=(*fact*, *prerequisite*, *consequence*), the *prerequisite set* of h, denoted as P(h), is the set of all such predicates (with the preceding negation if there is one) that appear in *prerequisite* and whose arguments are replaced with the corresponding attribute values of a tuple in h. Each element in P(h) is associated with the timestamp of the corresponding attribute values of a tuple in h. Similarly, the *consequence set* of h (denoted C(h)) is the set of all such logical formulas in *consequence* whose arguments are replaced with the corresponding attribute values of a tuple in h. Each element in C(h) is associated with the timestamp of the corresponding tuple in h.

An example of *prerequisite sets* and *consequence sets* can be seen as follows: Consider the Sadmind Ping attack with which an attacker discovers possibly vulnerable Sadmind services. The corresponding hyper-alert type can be represented by SadmindPing = ({ VictimIP,

VictimPort }, ExistHost(VictimIP),{ VulnerableSadmind(VictimIP) }). It is easy see that P(SadmindPing) = { ExistHost(VictimIP) }, and C(SadmindPing) = { VulnerableSadmind (VictimIP)}.

Correlation is based on *the prepares* for relation. That is we can say that hyper-alert A is correlated to B if A prepares for B. We formally define the prepare for relation as:

Prepares for: A hyper-alert h_1 *prepares for* hyper-alert h_2, if there exist $p \in P(h_2)$ and $C \subseteq C(h_1)$ such that for all $c \in C$, c.end_time < p.begin_time and the conjunction of all the logical formulas in C implies p.

To show an example of the *prepares for* relation lets consider two alerts, $h_{SadmindPing}$ and $h_{SadmindBOF}$. Hyper-alert $h_{SadmindPing}$ of type SadmindPing has the following tuples: {(VictimIP = 152.141.129.5, VictimPort = 1235)}. Then the *prerequisite set* of $h_{SadmindPing}$ is $P(h_{SadmindPing})$ = (ExistHost(152.141.129.5)} and the *consequence set* is $C(h_{SadmindPing})$ = {VulnerableSadmind (152.141.129.5)}. Hyper-alert $h_{SadmindBOF}$ of type SadmindBOF with tuples ={(VictimIP=152.1.19.5, VictimPort = 1235), (VictimIP = 152.1.19.37, VictimPort = 1235) }. Similar to $h_{SadmindPing}$, we can easily get $P(h_{SadmindBOF})$ = {ExistHost(152.141.129.5), ExistHost(152.141.129.37), VulnerableSadmind (152,141.129,5), VulnerableSadmind (152.141.129.37)}, and $C(h_{SadmindBOF})$ = {GainRootAccess (152.141.129.5), GainRootAccess (152.141.129.37)}.

Assuming that all tuples in $h_{SadmindPing}$ have timestamps earlier than every tuple in $h_{SadmindBOF}$; by comparing the contents of $C(h_{SadmindPing})$ and $P(h_{SadmindBOF})$, it is clear that the element VulnerableSadmind(152.141.129.5) in $P(h_{SadmindBOF})$ (among others) is also in $C(h_{SadmindPing})$. Thus, $h_{SadmindPing}$ *prepares for*, and should be correlated with $h_{SadmindBOF}$.

3.4 Temporal constraints for Hyper-alerts

As mentioned earlier, multiple alerts can be aggregated into one hyper-alert. The definition of hyper-alert in the previous section is overly flexible. It allows for multiple alerts to be aggregated as a single hyper-alert, regardless of how far apart in time they actually occur.

This can affect the effectiveness of alert correlation. While an attacker can distribute their attack over a period of time, it can be misleading to aggregate alerts that occur at arbitrary points in time.

In order to limit alert aggregation to meaningful ones, time constraints are introduced.

Duration Constraint: Hyper-alert h satisfies *duration constraint* of D (time duration) if Max {t.end_time|□t□h} – Min{t.begin_time|□t□h} <D

Interval Constraint A hyper-alert h satisfies *interval constraint* of I(time interval) if(1) h has only one tuple, or (2) for all t in h, there exist t.begin_time < T < t.end_time, t'.begin_time < T'< t'.end_time, and |T-T'|<I.

Using temporal constraints restricts aggregated hyper-alerts to meaningful ones, but does not imply that alerts have to be aggregated. And in the experiments done in [14] [40], each individual alert is treated as a hyper-alert.

3.5 Hyper-alert Correlation Graph

The correlation of hyper-alerts using the *prepares for* relation, lends itself as a natural way to represent causal relations between correlated alerts. The relationship is visibly represented using a *Hyper-Alert Correlation graph*. We can formally define a hyper-alert correlation graph as:

Hyper-Alert Correlation graph: A hyper-Alert correlation graph CG=(N, E) is a connected graph, where N is a set of hyper-alerts and for each pair n_1, n_2 □ N, there is a directed edge from n_1 to n_2 in E, if and only if n_1 prepares for n_2.

An example of a hyper-alert correlation graph is given in figure 3-1a. Hyper-alerts can represent the high-level strategy or logical steps behind a sequence of attacks.

In addition to providing a visual representation of attack scenarios, the hyper-alert correlation graph allows for a visual representation of other hyper-alert relationships. First, it provides an

intuitive representation of the correlated hyper-alerts, figure 3-1d. It also provides a method to discover hyper-alerts that are directly or indirectly correlated to a particular hyper-alert, figures 3-1 b, c and d. Using hyper-alert correlation graphs, we can define three main operations on hyper-alerts: *precedent*, *subsequent,* and *correlated*.

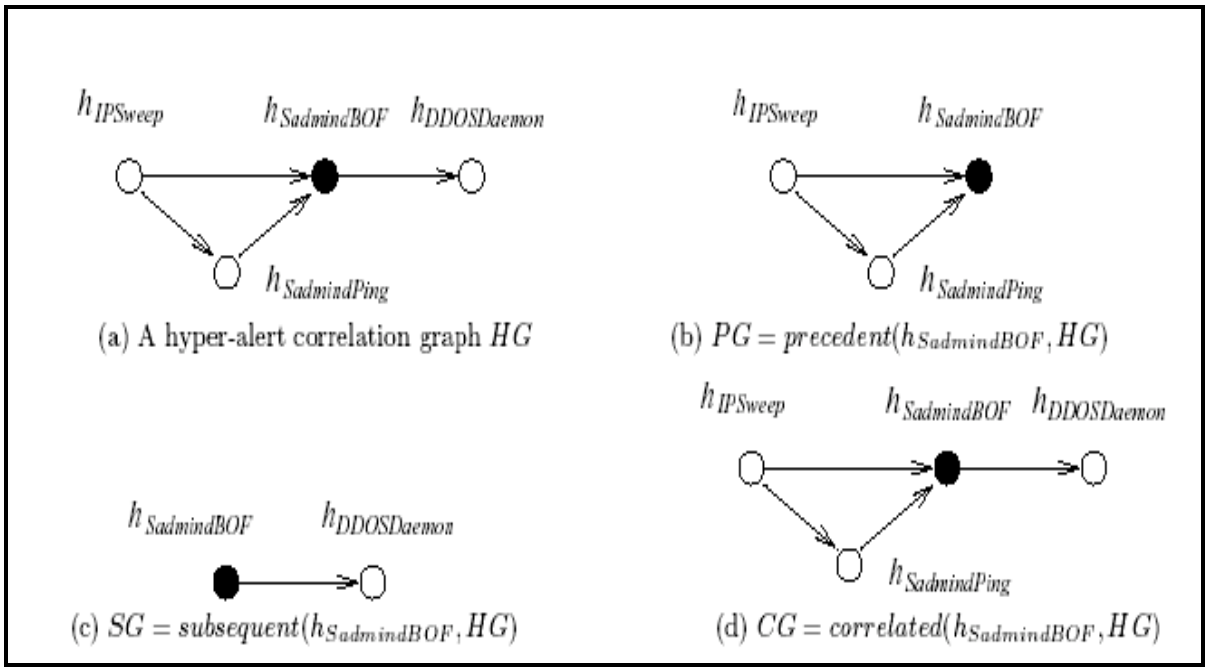

Figure 3-1: Hyper-alert correlation graph concepts [14]

Precedent with respect to a hyper-alert graph is all the alerts in a hyper-alert graph (HG) that prepare for n (a particular hyper-alert). This includes alerts that prepare for n directly or indirectly. We formally define precedent as:

Precedent: Given a hyper-alert correlation graph HG=(N,E) and a hyper-alert n in N, Precedent (n, HG) is an operation that returns the maximal sub-graph PG=(N',E') of HG that satisfies the following conditions: (1) n ∈ N', (2) for each n' ∈ N' other than n, there is a directed path from n' to n, and (3) each edge e ∈ E' is in a path from a node n' in N' to n. The resulting graph PG is called the precedent graph of n w.r.t. HG.

Figure 3-1b shows a precedent graph PG w.r.t HG, where $h_{IPSweep}$ and $h_{SadmindPing}$ prepare for $h_{SadmindBOF}$.

Subsequent hyper-alerts with respect to n, are all hyper-alerts for which n prepares. Subsequent graphs with respect to a hyper-alert graph, represent all the hyper-alerts in HG, for which n prepares. The subsequent operation is formally defined as:

Subsequent: Given a hyper–alert correlation graph HG= (N,E) and a hyper-alert n in N, subsequent (n, HG) is an operation that returns the maximum sub-graph SG=(N'E') of HG that satisfies the following conditions: (1) n ∈ N', (2) for each n' ∈N' other than n, there is a directed path from n to n', and (3) each edge e ∈ E' is a path from n to a node n' in N'. The resulting graph SG is called the subsequent graph of n w.r.t. HG.

In addition to subsequent and precedent operations, we can define a correlated operation that includes related hyper-alerts with respect to a particular HG and hyper-alert n. So, given $h_{SadmindBOF}$, in figure 3.1 d, CG=(HG, $h_{SadmindBOF}$) includes the precedents $h_{SadmindPing}$ and $h_{IPSweep}$ and subsequence $h_{DDOSDaemon.}$ Formally, we define correlated operation as:

Correlated: Given a hyper-alert correlation graph HG= (N,E) and a hyper-alert n in N, correlated (n, HG) is an operation that returns the maximal sub-graph CG = (N', E') of HG that satisfies the following conditions: (1) n ∈ N', (2) for each n'∈ N' other than n , there is either a path from n to n', or a path from n' to n, and (3) each edge e ∈ E' is either in a path from node in N' to n, or in a path from n to a node in N'. The resulting graph CG is called the correlated graph of n w.r.t. HG.

3.6 Utilities

Although not part of the original framework, the need for methods to manage large sets of correlated alerts became apparent. Large data sets can produce correlation sets that would be difficult to understand by human users. To facilitate the interactive analysis of large or complex sets, three utilities were created: adjustable graph reduction, focused analysis, and graph decomposition.

Adjustable graph reduction: Reduce the complexity (i.e., the number of nodes and edges) of hyper-alert correlation graphs while keeping the structure of sequences of attacks.

Focused analysis: Focus analysis on the hyper-alerts of interest according to user's specification. This may generate hyper-alert correlation graphs much smaller and more comprehensible than the original ones.

Graph decomposition: Cluster the hyper-alerts in a hyper-alert correlation graph based on the common features shared by the hyper-alerts, and decompose the graph into smaller graphs according to the clusters.

The utilities reduce the number of hyper-alerts and/or edges in a hyper-alert correlation graph to simplify it, making the graph more readable. The utilities can also be used to filter out alerts, leaving only alerts of interest. For example you may only be interested in hyper-alerts with a particular destination IP address.

3.7 Implementation Architecture and Evaluation

An implementation of an off-line intrusion alert correlator based on the framework discussed in the previous section was implemented by Yun Cui[14]. In this section, we give an overview of the implementation and review the results of early experiments.

3.7.1 Implementation Architecture

Here we give a short overview of the implementation architecture. For a more detailed information on the implementation refer to [14]. The Intrusion Alert Correlator was designed as a tool to correlate alerts reported by commercial Intrusions Detection Systems (IDSs). The current version is written is Java and interacts with a database management system (DBMS). JDBC is used to communicate with the database.

The Intrusion Alert Correlator was designed and implemented with a number of assumptions. First, it assumes that all the alerts generated by an IDS and the knowledge base are stored in a relational database. Although, sensor information is maintained for each alert, a sensor ID is associated with each alert; the implementation ignores this, and process correlation with the assumption that all alerts come from one sensor.

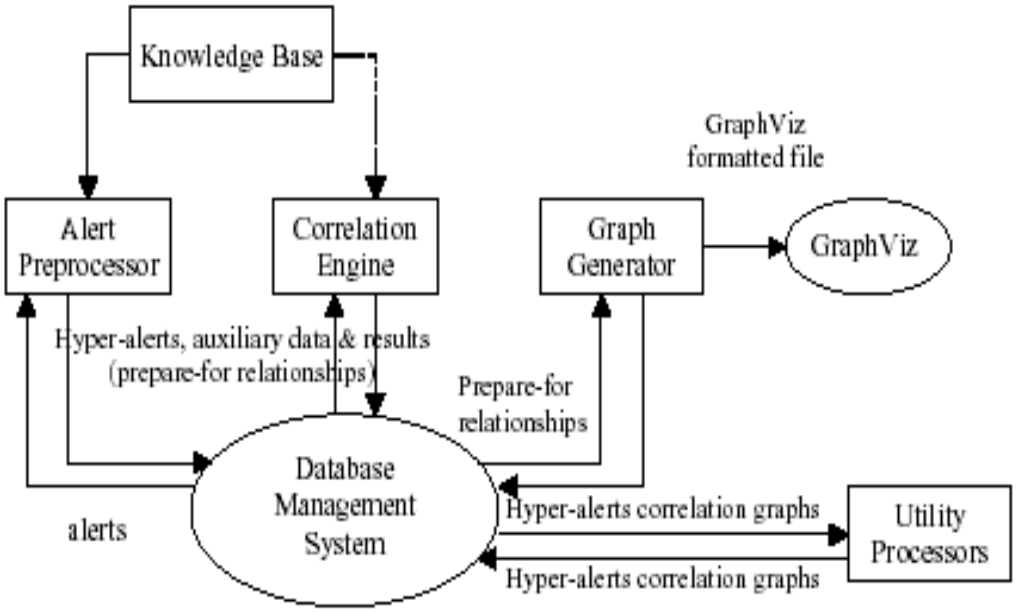

Figure 3-2: Alert Correlator Architecture

Figure 3-2 shows the architecture used to the implemented the offline correlator. The model includes:

> Database Management System: The DBMS system that maintains the alerts, and the necessary table to the correlation and the correlated results. The actual correlation occurs, through a series of SQL queries to the DBMS. Tables in the DBMS maintain all results for correlations.

Alert Preprocessor: The Alert Preprocessor generates hyper-alerts and instantiates the prerequisite and consequence sets of each hyper-alert from the IDS alerts.

Knowledge Base: Knowledge base contains the information about hyper-alert types and relationships between predicates. In the implementation, the knowledge base is defined in an XML file.

Correlation Engine: Actually does the correlation by finding all the prepare-for relationships between hyper-alerts. The correlation is done through interactions with the DBMS using JDBC and SQL quarries.

Graph Generator: Generates the hyper-alert graphs, outputs the graphs in a GraphViz formatted file.

Utility Processors: The implementation of graph utilities, adjustable graph reduction, focused analysis, and graph decomposition.

GraphViz: used to generate the actually graph files from GraphViz formatted text files.

3.7.2 Results

Early evaluation of the NCSU correlation method was done using the 2000 DARPA intrusion detection scenario specific data sets (LLDOS 1.0 and LLDOS 2.0.2)[1]. The experiments had two goals, first to show the effectiveness of the intrusion alert correlation method in constructing attack scenarios. Second, to see how well the alert correlation can be used to differentiate true and false alerts. For a more detailed description of the experiments and results read [40] or[14].

The experiments showed that the alert correlation method was effective at revealing attack scenarios from low-level alerts. Hyper-alerts correlation graphs showed the structure and the high level strategy of the sequence of attacks.

Completeness of alert correlation is how well related alerts are correlated, and soundness evaluates how correctly alerts are correlated. Table 3.1 shows the completeness and soundness values for each data set.

Table 3.1: Completeness and soundness of alert correlation. [14]

	LLDOS 1.0		LLDOS 2.0.2	
	DMZ	Inside	DMZ	Inside
# Correctly correlated alerts	54	41	5	12
#Related alerts	57	44	8	18
#Correlated alerts	57	44	5	13
Completeness measure R_c	94.74%	93.18%	62.5%	66.7%
Soundness measure R_s	94.74%	93.18%	100%	92.3%

For both data sets, most of the correlated alerts were correlated correctly. R_s value is above 92%. This implies that if two alerts are correlated, there is a significant probability that the correlation is done correctly. The completeness measure for alerts of LLDOS 1.0 was also significant, R_c being above 93%. However, there was a significant reduction in R_c for LLDOS2.0.2 when compared to LLDOS 1.0.

As mentioned earlier, false alert rates are high for most IDSs, and alert correlation is also an attempt to reduce the false alert rate. Early experiments also showed the ability to differentiate true and false alerts using alert correlation. For, the traffic of LLDOS 1.0 the detection rate was 56% for DMZ and 60% for inside. For LLDOS 1.0 the detection rate were 42% for DMZ and 67% for the inside. The false alert rates for LLDOS 1.0 were 5.36, 6.82 for DMZ and inside respectfully, and 40% and 23% for LLDOS 2.02. This compares to above 93% for RealSecure Alerts without correlation. Therefore, this method shows the potential for reducing false positive rates.

Chapter 4

Correlating Alerts from Homogeneous IDSs

This chapter provides an overview of the modified NCSU intrusion alert correlator framework, in which alerts from multiple homogeneous IDSs can be correlated together. In this chapter, we present an extension to the correlator framework, and then end with the presentation and discussion of the results from our experiments.

On a large or complicated network, intrusion detection normally requires multiple IDSs or sensors of single IDS to be distributed over the network. This is because a sensor (here we refer to a single IDS system deployed at one physical location as a sensor) has a limited view of the network traffic. An IDS only sees the traffic that is going or coming from the network segment it is on. In order to get a better view of what is occurring on the network, the placement of IDSs is important. High traffic locations like switches and routers; locations where the network is separated into segments give the best sites to detect traffic. Although the use of multiple IDSs can give a better view of the traffic on the network and thereby be able to detect more intrusions, it can also complicate matters for the user. The sheer number of alerts the user has to respond to can be overwhelming. As stated in chapter 3, the intrusion alert correlator shows the potential to reduce false alerts and help the user by uncovering high-level attack strategies behind the attacks. Yet, in experiments from the initial implementation of the intrusion alert correlator, the traffic from multiple IDSs was not correlated together. Processing traffic from different IDSs independently could possibly lead to a loss of information; where alerts from one IDS are related to alerts from another, but not correlated. It is also possible that in a large attack one IDS may miss a stage of that attack, if the relating traffic does not pass through its network segment or the IDS is overloaded. This is especially true, if the attack is carried out from multiple locations or against multiple hosts

on the network, but the hosts involved are not all on the same network segment. Therefore correlating alerts from multiple IDSs could provide a better high-level view of attacks, and a more complete view of attack strategies.

In addition to the need to determine if alerts from multiple IDSs are correlated, there is a need to determine if alerts refer to the same event. For example, take a network in which there is an IDS in the DMZ and an internal IDS. An attack from the outside to an internal machine can cause the DMZ IDS to produce alerts for the attack, and the same attack can cause the internal IDS to produce alerts. In this case, there is a need to identify that the alerts refer to the same attack. The benefit of this is obvious, the reduction in the number of alerts that a security administrator has to address, and a more complete picture of what is occurring on the network.

4.1 Homogenous Integration Framework

The integration framework is an extension of the basic intrusion alert correlator framework described in chapter two. Since, we are using homogenous IDSs, we assume that there is a one to one mapping of alert types from each IDS. That is that an event will generate the exact same alert(s) when any of the IDSs detects it. However, we do not assume clock synchronization between IDSs. The result is that timestamps generated for the same attack in different IDSs may not be equal.

The reason, we do not assume clock synchronization is that in the real world (and often in lab environments) it is very difficult to guarantee system synchronization in a network environment. This time difference between systems we consider a delay. There are many factors that can introduce delays:

1. Clock Discrepancy: There can exist a discrepancy between system clocks on the different IDS. Work has been done in this area, such as Secure NTP[13], and Secure Partial order Time[47] for solutions to clock synchronization.

2. Network Delay: An attack may be observed at different times by different IDSs. For example, the IDS deployed in the DMZ part of a network can observe an incoming attack earlier than the IDS deployed inside the network due to network transmission delay. Network delay in the context of this project is only the delay between an event being reported on one IDS and then on another IDS. The delay is calculated as the difference between the two timestamps. We do not consider the time it takes to store alert in the relational database, since we are performing offline correlation.

3. System Workload: Different IDSs have different workloads. Therefore, an IDS with a heavy workload may experience delay in generating a timestamp for a detected attack, while an IDS with a light workload would quickly generate the timestamp, resulting in a discrepancy between the timestamps for the same event.

To deal with the timestamp differences, we introduce the notion of *inter-IDS timestamp difference*. The inter-IDS timestamp difference for two alerts a_1 and a_2 (denoted as $\Delta(a_1, a_2)$), in which a_1 and a_2 represent alerts for the same attack detected by different IDSs, is the max ($|a_1.begin\_time - a_2.begin\_time|$, $|a_1.end\_time - a_2.end\_time|$). In addition, we assume that there exists a maximum inter-IDS timestamp difference $\Delta_{max,}$ such that for all alerts a_1, a_2, for the same attack, $\Delta(a_1, a_2) < \Delta_{max}$.

We also modify the definition of *prepares for* to take account of the fact that an alert a_1 that prepares for alert a_2 may have been generated by a different sensor; therefore, the inter-IDS timestamp difference should be taken into account when deciding if one alert prepares for the other, and should be correlated. We now define *prepares for* as follows:

Prepares for: Given the maximum inter-IDS timestamp difference Δ_{max}, hyper-alert h_1 prepares for hyper-alert h_2 if there exist $p \subseteq P(h_2)$ and $C \subseteq C(h_1)$ such that for all $c \subseteq C$,

1. The conjunction of all the logical formulas in C implies p, and

2. Either c.sid = p.sid $\wedge$ c.end_time < p.begin_time, or c.sid≠p.sid $\wedge$ c.end_time < p.begin_time + Δ_{max}. (sid is the IDS that produced the alert)

This simply says, that given two hyper-alerts h_1 and h_2, if the *consequence* of h1 imply the *precedent* of h_2 and either h_1 ends (the end time of h1 < begin time of h2) before h2, if the alerts are from the same sensor, or h1 ends before the beginning of h_2 and the end time of h_1 is more than the begin time of h_2 plus the maximum inter-IDS timestamp difference, then the hyper-alert h_1 prepares for h_2.

Given the updated definition of *prepares for*, all our previous definitions remain the same as in the original framework. Our new definition allows for the relaxation of the temporal constraints imposed on the correlation of hyper-alerts, when they are generated by different IDSs to account for any possible timestamp discrepancies.

Let's consider an example with the extended concepts.

Example: Given the following hyper-alerts

Hyper-Alert type(s): *SadmindPing* = {(*VictimIP*, *VictimPort*);

SadmindBOF = ({*VictimIP*, *VictimPort* }, *ExistHost*(*VictimIP*) □ *VulnerableSadmind*(*VictimIP*), { *GainRootAccess*(*VictimIP*)})

Hyper-Alerts: $h_{SadmindPing}$ = {(VictimIP = 152.141.129.5, VictimPort = 1235)}

$h_{SadmindBOF}$={(VictimIP = 152.142.129.5, VictimPort = 1235), (VictimIP = 152.141.129.37, VictimPort = 1235)}

The *prerequisite set* of $h_{SadmindPing}$ is $P(h_{SadmindPing})$ = (ExistHost(152.141.129.5)} and the *consequence set* is $C(h_{SadmindPing})$ = {VulnerableSadmind(152.141.129.5)}.

The *prerequisite set* of $h_{SadmindBOF}$ is $P(h_{SadmindBOF})$ = { ExistHost (152.141.129.5), ExistHost(152.141.129.37), VulnerableSadmind (152,141.129.5), VulnerableSadmind (152.141.129.37)}, and the consequence set is $C(h_{SadmindBOF})$ = {GainRootAccess (152.141.129.5), GainRootAccess (152.141.129.37)}.

Also assume that the two events actually happened 20 milliseconds apart, from the end of $h_{SadmindPing}$ to start of $h_{SadmindBOF}$. It is easy to see that $h_{SadmindPing}$ prepares for $h_{SadmindBOF}$, by comparing $C(h_{SadmindPing})$ and $P(h_{SadmindBOF})$.

Now, assume the two hyper-alerts are received from different IDSs, and there exists an inter-IDS timestamp difference of 30 milliseconds. Therefore, the timestamp of $h_{SadmindPing}$ is 30 milliseconds off from the actual time.

The two hyper-alerts should be correlated. If we try to correlate using the original definition of *prepares for,* the temporal constraints are too strict to allow the two hyper-alerts to be correlated. Using the extended definition of *prepares for,* the two alerts will be correlated, when the max inter-IDS timestamp difference is equal or greater then 30.

The recognition of duplicate alerts is not the focus of this thesis, but we are aware that duplicate alerts can occur. Merging Alerts from multiple IDSs can result in one event being represented by multiple alerts from different IDSs. In the case of duplicate alerts from different IDSs, recognizing that the alerts refer to the same event, could possibly simplify the work for the security administrator. In this thesis, we assume a simplified definition of a duplicate alert, so that we do not spend a large amount of time trying to find out whether two alerts are actually caused by the same event. We just note that the two alerts are similar enough that they may refer to the same event. Therefore recognizing that alerts possibly refer to the same event is straightforward. We take into account the alert features and the fact that there may be some inter-IDS timestamp difference. Formally, we define the simplified duplicate alerts as:

Duplicate: Given two alerts generated by different IDSs, we say that they are about the same attack if they have the same values on for all *fact* attributes and the inter-IDS timestamp difference between them is more than $\Delta_{max.}$

4.1.1 Effects of Δ_{max} value on Correlation

In a real network the inter-IDS timestamp difference will vary depending on a number of factors (e.g., network traffic load, IDS system load, system misconfiguration, etc.). The actual value of Δ_{max} is unknown because we do not know the exact time when the event occurs. The only thing we can go by is the timestamp of the alerts. Therefore, choosing the correct value of Δ_{max} can be difficult. There are a number of approaches that can be used to determine what value to use for Δ_{max}. You could choose a random value, or base the value of Δ_{max} on a previous session, or do some type of statistical calculation. However, even this would not likely be sufficient since the actual inter-IDS timestamp difference will likely vary during the day or month. Therefore, it is important to understand the effects of Δ_{max} on the effectiveness of correlation.

When the value of Δ_{max} is less than the actual delay, the *prepares for relationship,* for alerts from different IDSs, cannot be satisfied. For example, for two alerts a_1 and a_2, where a_1 prepares for a_2; if the delay for a_1 is 3 seconds and there is no delay in the recording of a_2, then using a value less 3 will not fulfill the requirement that a_1.end_time < a_2.begin_time.

The introduction of Δ_{max} to relax the timing constraint in the *prepare for* relationship, may allow the false correlation of alerts. When the value of Δ_{max} exceeds the delay, the false correlation of alerts may occur unnecessarily. The reason for this is when Δ_{max} is slightly more than the delay, the maximum number of true correlations can occur. Increasing the value of Δ_{max} beyond this point will further relax the timing constraint, possibly allowing the false correlation of alerts. At the same time, increasing Δ_{max} beyond the actual delay will not improve the true correlation rate since the maximum true correlation rate occurs when Δ_{max} is set to the actual inter-IDS timestamp difference.

4.1.2 Limitation of Correlation Method

A number of factors limit the performance of our method. First, our method is limited by the actual deployment of IDSs used. Since, an IDS is only able to monitor traffic on its network

segment, to monitor traffic correctly requires that no network segment goes unmonitored. Intrusive traffic in an unmonitored segment will not have corresponding alerts in the database that can be correlated. This can also lead to alerts in the database that are not correlated because they appear to be independent from the rest of the alerts although the alerts may be related to intrusions in the unmonitored segment of the network.

Second, the correlation performance is limited by the underlying IDS used. As mentioned earlier each IDS has certain strengths and weaknesses; these same strengths and weaknesses influence how well the correlation can be done. Since, the correlator works above the IDSs and processes the low-level alerts, the correlator will also miss any intrusions missed by the IDSs. In addition, if the IDS generates some alerts but fails to generate alerts for an entire series of related attacks, the correlator will be limited in its ability to correlate the resulting alerts. The resulting alerts may appear to be independent from any series of attacks because of the lack of related alerts that can link them to an attack scenario.

Third, anytime alerts reported by an IDS do not correctly represent the underlying network or the traffic on the network, correlation is reduced. One factor that can miss-represent traffic is the use of NAT (Network Address Translation). In NAT, a network address is mapped to different addresses. For example: if you have a system in a protected network then you may want use an address for communications within the protected network and another one for communications with the external network. Instead of using multiple network interfaces, you translate the internal address to an external address. Therefore, the system may have address 138.201.12.16 in the internal network and appear to have address 94.64.15.27 for external communications. There are a number of reasons why NAT may be used: save address space, virtual servers (load balancing), address masquerading. Yet, the use of NAT will affect our ability to correlate alerts. If the underlying IDSs do not take into account the use of NAT, it can be difficult to correlate alerts that are about the same host(s). Therefore, it can be difficult to relate alerts when they appear to be for different network addresses, but actually refer to the same host.

4.2 Implementation

To evaluate the performance of our method, we used Alert Correlator version 2.0[1] with a few modifications. The Alert correlator version 2.0 is implemented based on the architecture and concepts in chapter 2. To allow the alert correlator to handle alerts from multiple IDSs, changes were needed to the correlation engine so that sensor ID and inter-IDS timestamp differences were considered. No, changes were required in the database structure. We assume that all alerts are in the DBMS, and that each alert has a feature (sensorID) that indicates which IDS generated the alert.

4.2.1 Correlation Engine

The correlation engine finds all the prepare-for relationships between hyper-alerts. The found relationships are stored in the *CorrelatedAlerts* table of the DBMS. The *CorrelatedAlerts* table maintains the correlated hyper-alert information; each entry in the *CorrelatedAlerts* table represents a relationship between two alerts and consists of two attributes, *PreparingHyperAlertID* and *PreparedHyperAlertID*. The entries indicate that the hyper-alert with the ID *PreparingHyperAlertID* prepares for the hyper-alert with ID *PreparedHyperAlertID*.

Using two auxiliary tables *PrereqSet* and *ConseqSet* and SQL queries are used to select the prepares for relations. The resulting relationships are inserted into the *CorrelatedAlerts* table. In the original implementation of the correlator, the following query was used to generate all the *prepares for* relationships:

```
SELECT DISTINCT c.HyperAlertID, p.HyperAlertID
FROM PrereqSet p, ConseqSet c
WHERE p.EndodedPredicate = c. EncodedPredicate
AND c.end_time < p.begin_time;
```

[1] NCSU Intrusion Alert Correlator 0.2, available at discovery.csc.ncsu.edu/correlator/ver0.2/iac.html

Since the data sets used in earlier experiments did not process alerts from multiple IDSs together, this worked fine. However, with the inclusion of alerts from multiple IDSs this query fails to take into account the existence of a possible inter-IDS timestamp. We modify the query to account for alerts from multiple IDS and to take the inter-IDS timestamp into account. The updated query used to generate all the prepare-for relationships is:

```
SELECT DISTINCT c.HyperAlertID, p.HyperAlertID
FROM PrereqSet p, ConseqSet c
WHERE p.EndodedPredicate = c. EncodedPredicate
AND (c.end_time < p.begin_time AND c.sid=c.sid)
OR ((p.sid<>c.sid) AND (C.end_time <(p.begin_time  +" + delta_max)))
```

Notice that one condition is used for alerts that are generated by the same IDS, and another for alerts from different IDSs. The hyper-alert feature sid identifies the IDS which initiated the alert. The reason is that we still want to maintain a tighter temporal constraint on correlation between alert from the same IDS then for correlations between alerts from different IDSs. Here delta_max is the max inter-IDS timestamp difference, the value being predefined.

Our earlier implementation of the alert correlator provides a number of utilities to simplify hyper-alert correlation data for the user. Among these is graph decomposition. As mentioned in chapter two, graph decomposition allows for the simplification of graphs according to the common features shared by hyper-alerts. The graph is simplified by clustering alerts based on a clustering constraint (the specified common features). Graph decomposition, provides an easy method to aggregate duplicate alerts, through the clustering constraints.

The clustering constraint is specified in an SQL statement. The constraint must be a valid relationship between two alerts in the query, h1 and h2. For example, to cluster alerts based

on an alerts' source IP address, then h1.SrcIPAddress=h2.SrcIPAddress would be used as the clustering constraint. The features used must be valid attributes in the knowledge base.

A link list keeps all the alerts that belong to the same cluster and a dynamic array is used to keep the entire linked list, with each element in the array containing the head of a linked list. Entries in the link list consist of the hyper-alert IDs. The size of the dynamic array is the total number of sub-graphs after the graph decomposition, and the size of each linked list is the number of alerts in each sub-graph.

Decomposition is performed as follows:

Generate all the distinct hyper-alerts in the graph and save the results in auxiliary table *GraphAlerts* .

```
SELECT DISTINT PreparingHyperAlertID AS HyperAlertID
FROM GraphEdges WHERE graph ID=user-input-graphID
UNION
SELECT DISTINCT PreparedHyperAlertID AS HyperAlertID
From GraphEdges WHERE graphID=user-input-graphID;
```

Then we generate all alert pairs. An alert pair is two alerts that match the graph clustering constraint. The following query is used to find all alert pairs that exist in the graph:

```
SELET h1.HyperAlertID, h2.HyperAlertID
From GraphAlerts a, HyperAlertFact h1, HyperAlertFact h2
WHERE h1.HyperAlertID IN (SELECT * FROM a)
And H2.HyperAlertID IN (SELECT * FROM a)
And h1.HyperAlertID NOT = h2.HyperAlertID
And user-input-constraint;
```

The clustering constraint used to aggregate duplicates is: h1.fact=h2.fact. Fact is an SQL statement of the *fact* attributes of the hyper-alerts. The alert pairs are then assigned to a cluster. An aggregated hyper-alert is used to represent the cluster in the simplified graphs.

4.3 Experiment results

A number of experiments were done to evaluate the effectiveness of the extended method. Evaluation was done using two data sets from DARPA 2000 Intrusion Detection Scenario Specific Data sets [1]. The objectives in our experiments were:

1. To see the effects of merged data on the ability to discover and construct attack scenarios

2. To see the effects on the ability to differentiate true and false alerts

3. To see effects of the value of the max inter-IDS timestamp difference on correlation and detection rates.

4. To compare the performance of the extended method of correlation to the original method.

4.3.1 Experiment Setup

The data sets were replayed using NetPoke[2] in an isolated network monitored by a RealSecure Network Sensor 6.0[5]. The Network Sensor was configured to use the *Maximum_coverage* policy, and was set to save all the reported alerts.

Each part of the data set was run separately, that is the Inside and DMZ files for each data set were not run concurrently. This led to a large difference in timestamps between alerts from the inside network alerts and those from the DMZ. The difference was larger than would occur under normal conditions if you were running a network with two IDSs. To adjust for

[2] Netpoke is a tool to replay packets to a live network from tcpdump files.. Available from MIT Lincoln Labs at http://www.ll.mit.edu/IST/ideval/tools/tools/_index.html.

this large difference, we then created a small program to adjust the timestamps by updating the "EventDate" feature for alerts in the database. We calculated the time difference between the data sets and adjusted the timestamps of the sets to reduce the inter-IDS time difference.

For the experiments, each alert type reported by RealSecure Network Sensor was mapped to a hyper-alert type. The names of the hyper-alert types were those generated by RealSecure Network Sensor. The *prerequisite* and *consequence* for each hyper-alert type was specified according to the descriptions of the attack signatures provided with the RealSecure Network Sensor 6.0. The hyper-alert types and implication relationships between predicates are defined in the knowledge base used in the experiments. The knowledge base is the sample knowledge base available with the Intrusion Alert Correlator version 0.2.

4.3.2 Data set

The data sets used for the experiments were the 2000 DARPA Intrusion Detection Scenarios Specific Data Sets. There are two sets consisting of an attack scenario each.

The LLDOS 1.0 includes a distributed denial of service attack run by a novice attacker. The attack scenario is carried out over multiple networks and audit sessions. The sessions can be grouped into five phases,:

1. IPsweep
2. Probe of live IP's to look for the Sadmind daemon running.
3. Breaks in via the Sadmind vulnerability,
4. Installation of the Trojan mstream DDoS (Distributed Denial of Service) software.
5. Launching the DDoS

LLDOS 2.0.2 data set includes a distributed denial of service attack run by an attacker that is more sophisticated than the attacker in LLDOS 1.0. The scenario followed in the LLDOS 2.0.2 is similar to the LLDOS 1.0 scenario. The main difference being that IPSweep and Sadmind_Ping are used to find vulnerable hosts in LLDOS 1.0, and LLDOS 2.0.2 uses DNS_HINFO; and while in the first data set the attacker attacks each host individually, in LLDOS 2.0.2 the attack starts at one host then fans out. The five phases of the attack scenario in LLDOS 2.20 are:

1. Probe of public DNS server, via the HINFO query
2. Break-in via the Sadmind exploit
3. FTP upload of mstream DDoS software and attack script, to break-into more Eyrie hosts.
4. Initiate attack on other hosts
5. Launch the DDoS:

4.3.3 Effectiveness of Alert Correlation

The first goal was to see the effects of correlating data from multiple IDSs on constructing attack scenarios from alerts. Figures 4-1,4-2, and 4-3 are the hyper-alert correlation graphs generated during the experiment. Each node in the figures represents a hyper-alert. The label of the alert is the hyper-alert type followed by the hyper-alert ID. Figure 4-1, shows the largest of the hyper-alert correlation graphs discovered in data set LLDOS 1.0. This graph shows there exists a causal relationship between some of the alerts from the DMZ (shaded nodes) and alerts from the inside network. Figure 4-2 shows the other hyper-alert graphs from LLDOS 1.0. All hyper-alerts in these graphs come from the DMZ. There is a reduction in the number of hyper-alert graphs generated from 7 to 4, compared to processing the DMZ and inside traffic separately. As correlations between hyper-alerts from different IDSs are discovered, their respective hyper-alert correlation graphs are merged.

The hyper-alert correlation graph reveals the structure and high-level strategy of the sequence of attacks. A five-stage scenario, dividing the hyper-alerts horizontally, is visible in figure 4-1. This scenario corresponds closely with the description of the attack scenario given with the data set. Stage one consists of six *Sadmind_Ping* alerts, which indicate that the attacker used ping to find the vulnerable *Sadmind* services. Three of the alerts are from source IP address 202.077.162.213, and all six are to the destination IP addresses 172.016.122.010, 172,016.155.020, or 172,016,122,050. Stage two consists of 28 hyper-alerts of type Sadmind_Amslverify_Overflow, which the attacker used to break into the victim host. Stage three consists of 33 Rsh alerts, which were used to install *mstream* daemon and master programs. Stage four consists of the 6 Mstream_Zombie alerts, which corresponds to the communications between the DDOS master and Daemon programs. The fifth stage is the actual DDOS attack, represented by the Stream_DoS hyper-alert.

In LLDOS 1.0 the Email_Almail_Overflow hyper-alerts are false alerts, and are miss-correlated with the Rsh alerts. According to the data set these are not used in the attack, even though it is possible that an attacker could use them to gain access to the victim system. In addition, the FTP_Syst hyper-alerts are also false and are not part of the attack.

In data set LLDOS 2.0.2, a four-stage scenario is easily visible (Figure 4-3). Stage one consists of six Sadmind_Amslvery_Overflow alerts, which are used to access host mill of the eyrie network. Stage two consists of three ftp_put alerts, which corresponds to the upload of the *mstream* DDoS software and attack script. Stage three consists of three Mstream Zombie alerts, which correspond to the communications to DDoS(Distributed Denial of Service) master, and last stage consists of the StreamDoS hyper-alert, which represents the actual DDoS attack.

In LLDOS 2.0.2, we get an increase in the number of false alerts, compared to correlating the DMZ and inside traffic separately. Correlated separately we get an Email_Almail_Overflow, an Email_Ehlo and a Email_Turn alert that are false in the internal traffic, and an Email_Ehlo and a Email_Turn alert in the DMZ traffic. When correlated together we get an additional Email_Almail_Overflow alert from the DMZ that is correlated to an internal alert. This

represents a 20% increase in the number of false alerts. Since the number of alerts in the data set is small, we cannot definitely make any general assumptions. It does however suggest that large increases in false alerts are possible when the traffic is merged under certain conditions. We should note that we get the complete opposite indication from LLDOS 1.0. Where the false rate does not change compared to correlating the DMZ and inside alerts independent from each other.

In both cases, the hyper-alert correlation graph reveals the same high-level strategy of the sequence of attacks as correlating the DMZ and inside traffic separately. This is something that would require security administrator much work to obtain from the uncorrelated alerts. What we do not see by correlating the traffic separately is the relationships between the internal traffic and DMZ traffic. This would require the network administrator to investigate the hyper-alerts to find any relationships. Correlating the alert from the DMZ sensor and internal sensor together provides an automated method to show their relationship. It also shows that some of the DMZ traffic is involved in the overall attack. This can help clarify how the attacks proceeded on the network.

Since the IDSs used are homogenous, we expected that the IDSs generate alerts for the same events. This can be seen as a way to provide sensor reinforcement. Our experiment shows a high level of agreement between the IDSs on the attack scenarios. In LLDOS 1.0 figure 4-1, there is clear agreement on three stages of the attack scenario, considering only stages that are part of the actual attack scenario. As expected, many of the incidents generate alerts at the DMZ and internal IDS. The DMZ IDS does not generate alerts for the last two stages, but this is expected since the traffic at the last two stages does not go through the DMZ. Figure 4-2, shows that for these graphs there is no correlation with alerts from internal IDS, but on closer review shows that the reason for this is that the destination for the alerts in the graphs are address on the 172.016.114.0 sub-network, which corresponds to the DMZ. Therefore, the internal IDS would not scan traffic going to these hosts from an external host. Similarly, in LLDOS 2.0.2 we have agreement on three stages for traffic that goes through the DMZ and internal IDSs.

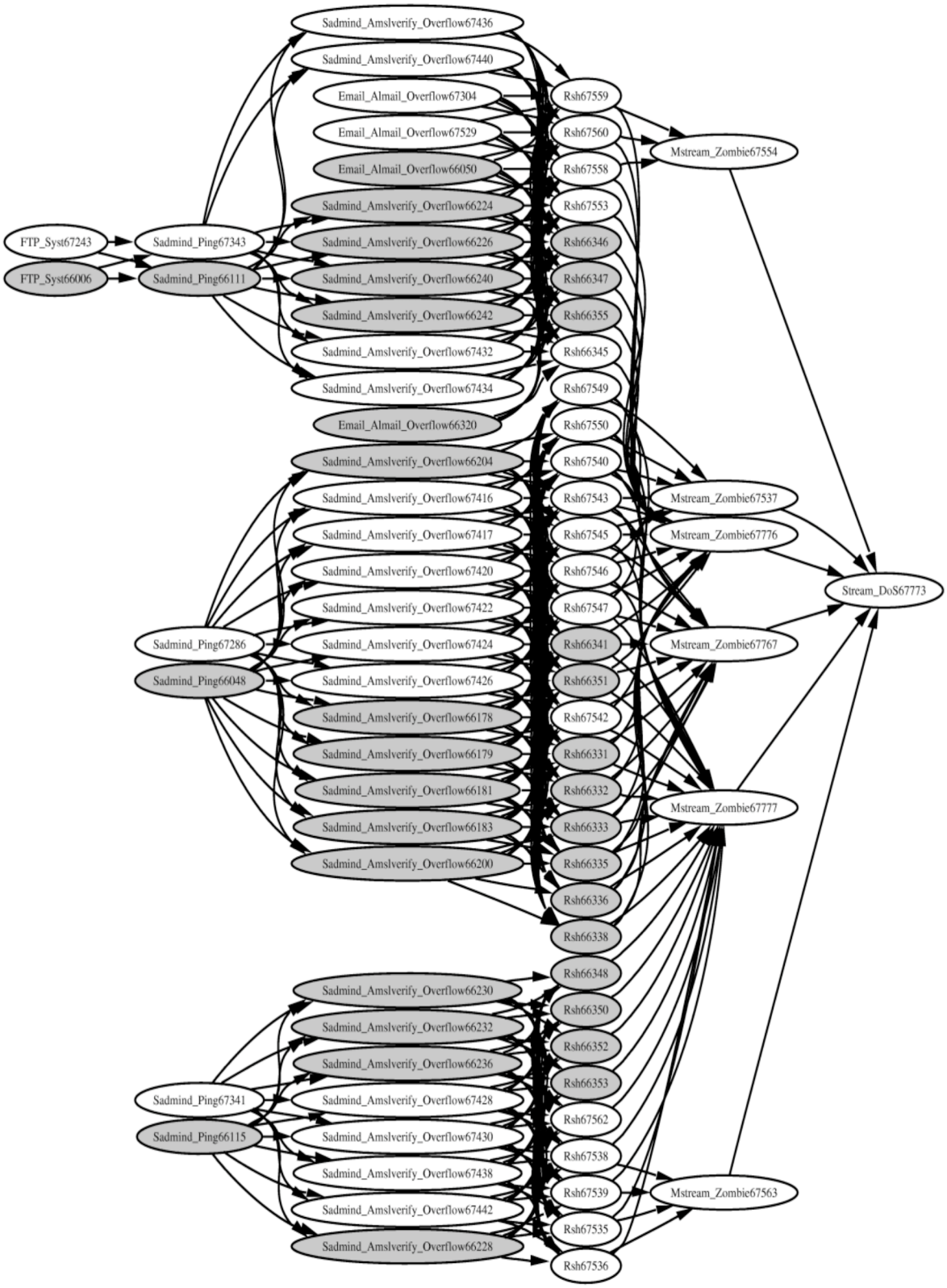

Figure 4-1: Largest hyper-alert correlation graph discovered from LLDOS 1.0 (shaded hyper-alerts are from DMZ traffic)

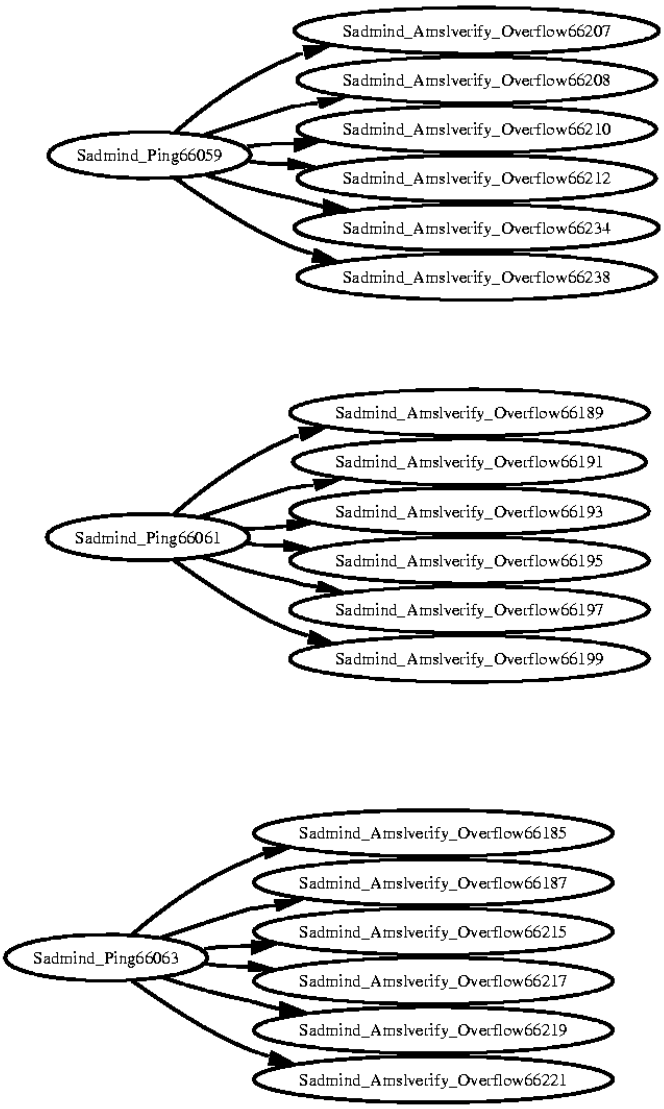

Figure 4-2: Additional Hyper-alert correlation graphs discovered in LLDOS 1.0 (all hyper-alerts are from DMZ traffic), The top graph: destination IP=172.016.114.010, middle graph: destination IP = 172.016.114.010, bottom graph: destination IP=172.016.114.010

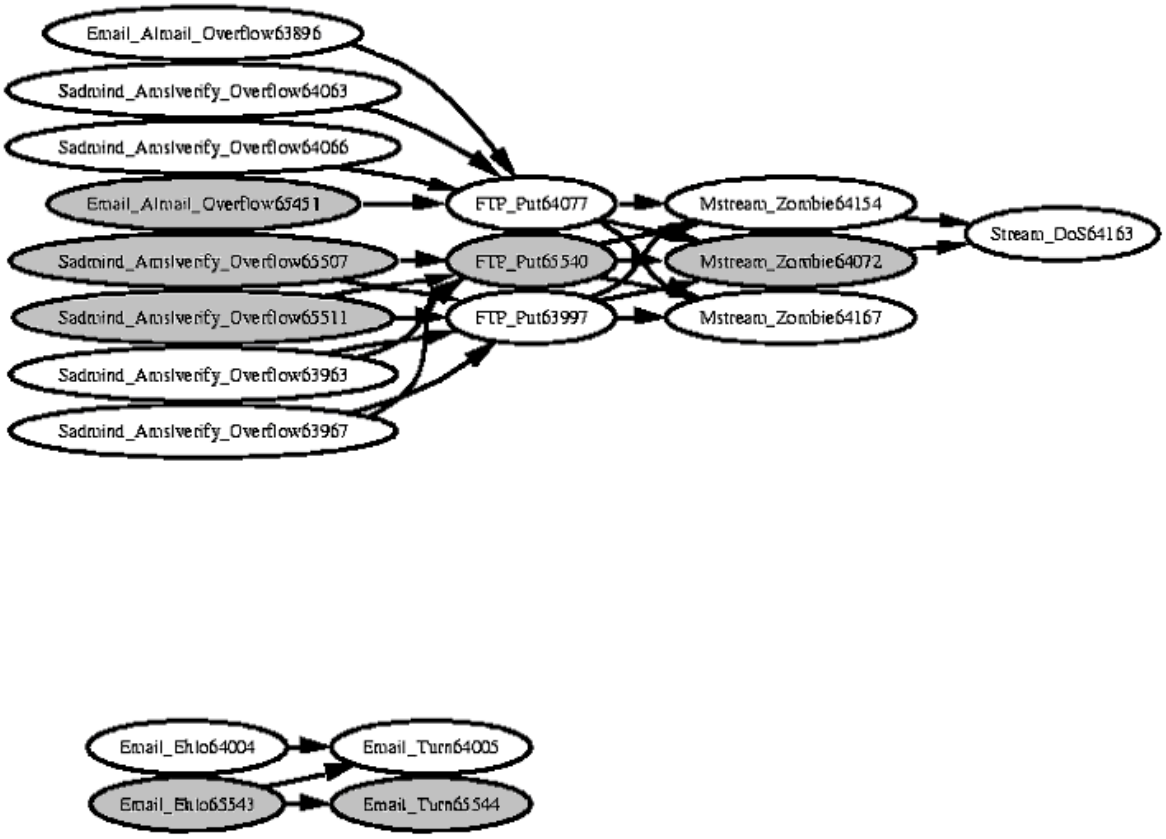

Figure 4-3: LLDOS 2.0.2 hyper-alert correlation graphs (shaded alerts are from the DMZ)

To evaluate the effectiveness of our correlation, the measures of completeness(R_c) and soundness (R_s) are used. They are calculated as:

$$R_c = \frac{\#\,\text{correctly correlated alert}}{\#\,\text{related alerts}}$$

$$R_s = \frac{\#\,\text{correctly correlated alerts}}{\#\,\text{correleated alerts}}$$

Completeness (R_c) of the alert correlation assesses how well related alerts are correlated. Soundness (R_s) evaluates how correctly the alerts are correlated. In our experiments, we

counted the duplicate alerts as different alerts. False alerts are counted as incorrectly correlated alerts, if they are correlated. Non-intrusive alerts (Email_Ehlo and Email_Turn), which are not attacks, are counted as correctly correlated if they related activities. For both data sets the completeness measure was equal to the average of the completeness of correlating each data set separately. This indicates that we did not observe a loss of information by integrating the alerts. The soundness of our correlation did not fair as well. For, LLDOS 1.0, the soundness was equal to average of doing the correlation separately; but the soundness of LLDOS 2.0.2 was more than expected. This is due to the increase in false correlated alerts. The hyper-alerts missed are those triggered by the telnets that the attacker used to access a victim host. Each telnet triggers three alerts, TelnetEnvAll, TelnetXDisplay and Telnet TerminalType. RealSecure's documentation describes theses alerts as attacks launched using environmental variables (TelnetEnvAll) in a telnet session, including Xdisplay and TerminalType. In the DARPA data sets the attacker does not launch these type of attacks. He does telnet to a victim host after gaining access to it. To stay consistent with earlier experiments, we consider them as related alerts in our evaluation.

Table 4.1: Evaluation Statistics for extended correlation method and uncorrelated alerts.

	LLDOS 1.0 (RealSecure)	LLDOS1.0 (Extended Correlator)	LLDOS 2.0.2 (RealSecure)	LLDOS2.02 (Extended Correlator)
#observable attacks	149	149	22	22
#alerts	1813	101	914	19
Detected attacks	88	86	16	13
Detection rate (%)	59	57	72.7	59
False alerts rate (%)	94.4	5.9	97.66	31.5
Correlation Completeness measure R_c	N/A	0.941	N/A	0.57.6
Correlation Soundness measure R_s	N/A	0.941	N/A	0.789

4.3.4 Ability to Differentiate Alerts

The second goal was to see how well integrating the alerts affects the ability to differentiate between false and true alerts. Correlation has shown the ability to reduce the false rate by not correlating false alerts, which tend to be more random than actual alerts, see results in chapter 2. The correlation graphs only show false alerts that are correlated with other alerts (some true). We computed the false rate of the correlated alerts.

Table 4.1 shows the false alert rate for the extended correlation method and for the uncorrelated alerts (RealSecure). False and true alerts were determined according to the description of the data sets. For example, the initial phase of the attack in LLDOS 1.0 is the IPSweep. From this we know that any alerts before the IPSweep are not valid parts of the attack and can count them as false alerts. We also know that the email related alerts were not part of the scenario and can count them as false. Similarly, we can look at the correlated alerts and can compare them to the description of the scenario to see if they are true alerts.

RealSecure Network Sensor 6.0 generated duplicate alerts for certain attacks. For example, the same rsh connection that the attacker used to access the compromised host triggered two alerts. As a result, the number of true alerts (i.e., the alerts corresponding to actual attacks) is greater than the number of detected attacks. The detection rates were calculated as $\frac{\#\text{detected attacks}}{\#\text{observable attacks}}$, and the false alert rates were calculated as (1- $\frac{\#\text{ true alerts}}{\#\text{alerts}}$). Table 4.1 summarizes the detection rates of our experiments. The experiment did not show any change in false rate (5.9%) for LLDOS 1.0, compared to correlating the DMZ and internal traffic independently. The false rate for LLDOS 2.0.2 was 31.5%. For LLDOS 2.0.2 correlation allowed for an increase in the number of false alerts. The increase was due to an additional Email_Almail_Overflow. Since the number of hyper-alerts was small 19, with 6 false positives, this one alert represents an increase of 16 percent of the false alerts. LLDOS 2.0.2 is a small data set; therefore small changes have a larger impact on the detection and false alert rates. This does however; indicate that under certain conditions merged correlation may cause the false detection rate to increase. The false rates are still much lower then for the uncorrelated data sets, which have false rates above 90 percent.

4.3.5 Effects of the value Δ_{max} on effectiveness

Our experiment showed for the given data sets that small changes in the value of Δ_{max} had little effect on correlation. Figures 4-4 and 4-5 show the impact of Δ_{max} on completeness and soundness of correlation on LLDOS 1.0 and LLDOS 2.0.2.

As stated in section 4.1, our method defines Δ_{max} as the maximum difference between any two alerts from different IDSs representing related events in the same attack. This would be the optimal value of Δ_{max} and would allow the maximum number of related alerts to be correlated. Yet, finding the exact value of Δ_{max} can be time consuming at minimum, requiring the calculation of all inter-IDS timestamp differences. It also requires that you know which alerts are correlated before hand. So, not only is the optimal value of Δ_{max} hard to obtain, but also if you could calculate it you would not need it, since you would already know which alerts are related. So, what occurs if the optimal value of Δ_{max} is not optimal?

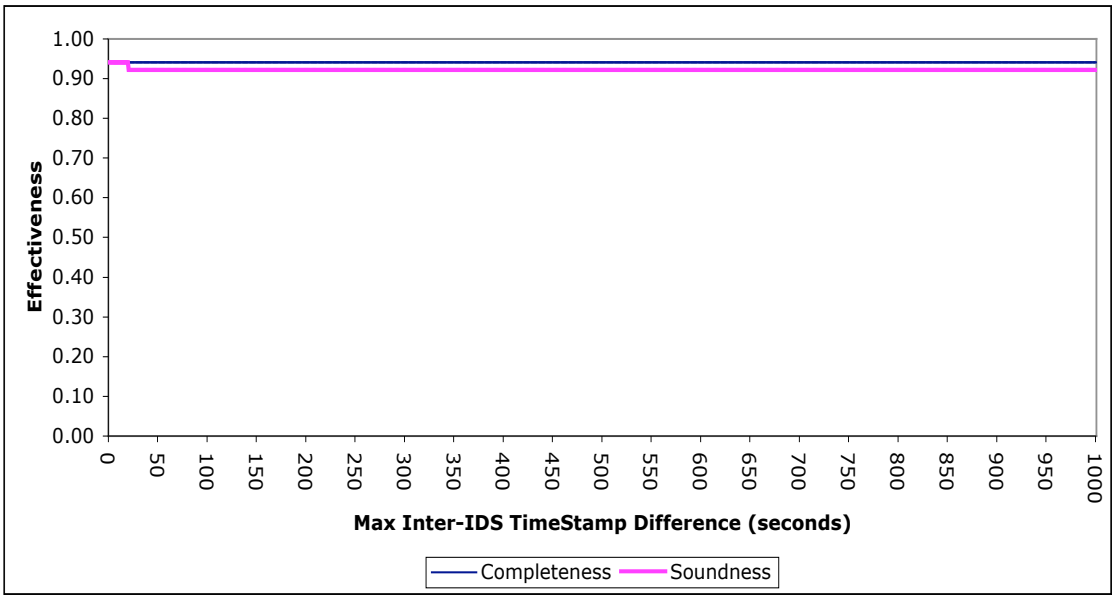

Figure 4-4: Effectiveness LLDOS 1.0

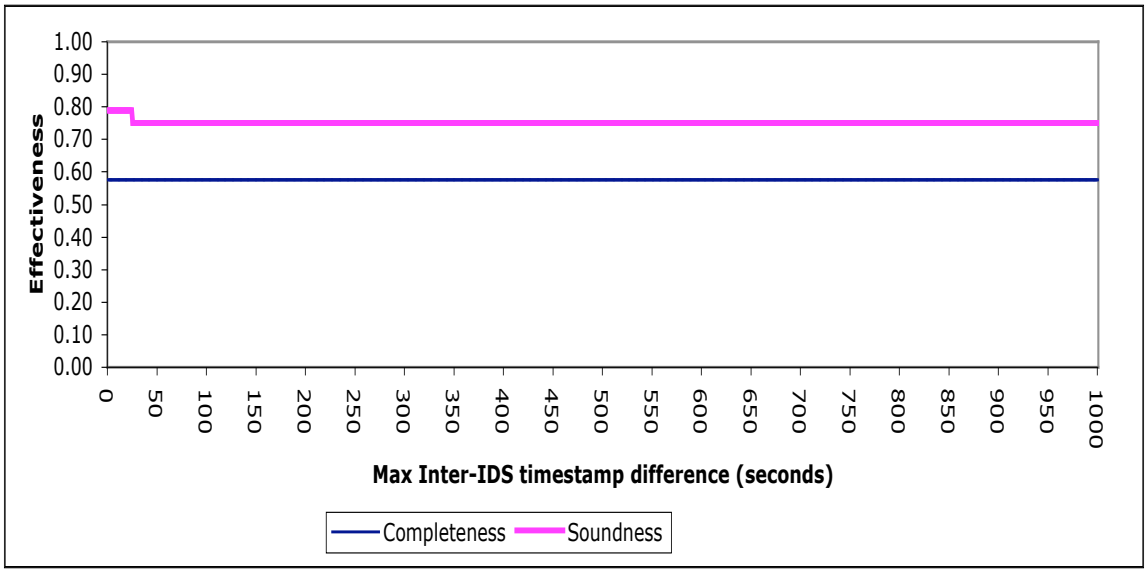

Figure 4-5: Effectiveness LLDOS 2.0.2

It is easy to see that the value of Δ_{max} should be equal or greater then the actual value of the maximum timestamp difference between any two alerts representing the same event. . If Δ_{max} is not greater then $\Delta(a_1, a_2)$ for any two related alerts, we expect that correlation cannot occur between the alerts. If Δ_{max} is more than the minimum of inter-timestamp differences between any two related alerts, then the results of doing the correlation should be equivalent to correlating the alerts for each IDS independently. Our experiments show that this is not always the case. For example, at Δ_{max} set to 0, correlation still occurs. On examining the data set more closely, we notice that the stages of the attack are separated by more time then the actual Δ_{max}. Therefore, the correlation of events between two different stages, still occurs, because the preparing events timestamp is more than the prepared for events timestamp, even if the inter-IDS timestamp difference is not taken into account.

The value of used Δ_{max} may be greater then Δ_{min} and more than the actual Δ_{max}. What we expected was a drop in the number of correlated alerts when the value was more than the actual Δ_{max}, but this did not occur. On closer look, our actual Δ_{max} was quite small. We judged this based on the timestamp differences between duplicated alerts. In most cases, the

timestamp difference was no more than 1-2 seconds. Although this is a large period of time in computer terms, for our data sets this was not the case. Doing the correlation of the DMZ and inside separately and correlation after merging the traffic, we did not see a significant drop in the number of hyper-alert correlations when Δ_{max} was below the actual Δ_{max}. Similarly, we did not see an increase in correlation when Δ_{max} was larger then Δ_{min}. Again this is likely due to the large time period between stages in the scenario.

The question remains what occurs when Δ_{max} is greater then the actual inter-timestamp difference. The actual value of Δ_{max} for the data sets is 3 seconds for LLDOS 1.0, and 2 seconds for LLDOS 2.0.2, yet, as the value of Δ_{max} used increases beyond the actual Δ_{max}, there is no change in the confidence or soundness. It is not until Δ_{max} reaches 20 seconds for data set LLDOS 1.0 and 50 seconds for data set LLDOS 2.0.2 that we see a fall in the soundness. For LLDOS 1.0 we see a fall from 94 to 92 percent. In LLDOS 2.0.2, the change is from 78 to75. The drop in soundness is due to the addition of falsely correlated alerts. As Δ_{max} continues to increase, for LLSOD 2.0.2, the soundness falls from 75% to 71% at 1500 seconds (not shown in graph). While, at first this indicates that the soundness is very resilient to increases in the value of Δ_{max}, on closer look it looks that this may only hold true for the given data sets. First, there seems to be a large time period between all stages of the attack, so instead of milliseconds or seconds between stages there are minutes. Therefore, the inter-IDS timestamp difference is small compared to the time between two related events. For example, stage one of LLDOS 1.0 scenario starts as 15.03, and phase two starts at 15:16. Thirteen minutes is larger than most inter-IDS time differences that might occur in the real world. Second, there seems to be very little similarity between the attack traffic and the background traffic. Therefore, the correlation of background traffic is minimal at all values of Δ_{max}.

There is no change in the completeness value for the two data sets. This is expected because we initially get high completeness values with low values of Δ_{max}. Increasing the value of Δ_{max} will not cause previous defined relationships to end. We had hoped that increasing Δ_{max} might show an increase in completeness. With these data sets, the fact that the phases are

farther apart in time then the value of Δ_{max} means that increasing Δ_{max} does not cause any increase in completeness.

Although not part of the actual attack scenario, the Email_Ehlo and Email_Turn hyper-alerts from LLDOS 2.0.2, lead us to believe that if the scenario stages were closer together (in time) the value of Δ_{max} would have a larger impact on correlation. The Email_Ehlo and Email_Turn alerts are false alerts in our scenario, not because they could not be used as part of the attack, but because we know according to the description of the scenario that they are not. Yet, they provide a two stage attack sequence with short time delay between stages (1 second), and a one second inter-IDS timestamp difference. Figure 4-6 shows the resulting graphs when Δ_{max} is set to 0. Hyper-alert 65543 and 65544 are from the DMZ and 64004 and 64005 are from the internal network sensor. According to our simplified definition of duplicate, 65543 and 64004 may refer to the same event, as do 65544 and 64005. Although 64004 and 65544 should be correlated, the inter-IDS timestamp difference keeps them from being correlated. The same graph (figure 4-6) results when Δ_{max} is set to 1. Figure 4-7 shows the graph that results when Δ_{max} is set to two. At this point, Email_Ehlo64004 and Email_Turn65544 are correlated.

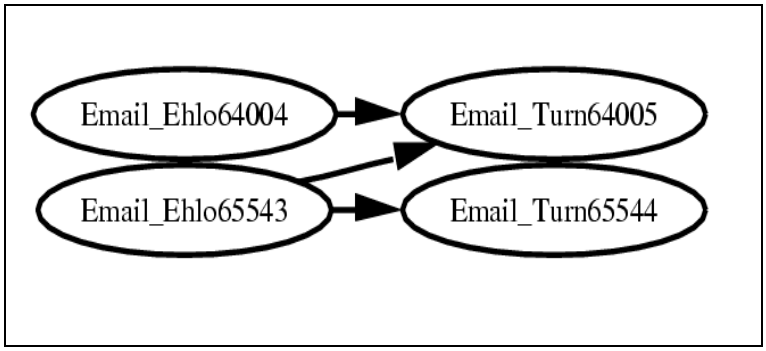

Figure 4-6: Hyper-alert correlation graph when Δ_{max} is set to less than 2.

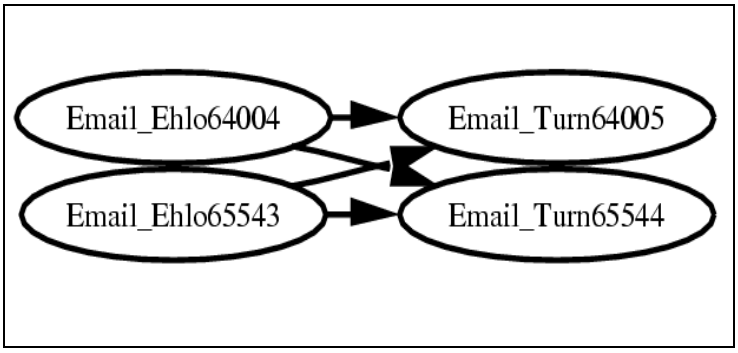

Figure 4-7: Hyper-alert correlation graph when δ_{max} is set to 2 or greater.

4.3.6 Effects of δ_{max} on ability to differentiate alerts

Figures 4-8 and 4-9 show the effect of the value of δ_{max} used on the ability to differentiate alerts. As can be seen for both, setting the value of δ_{max} has little effect on the detection rate. Again, this is most likely because the time between stages is larger then the actual value of δ_{max} Therefore, even when δ_{max} is more than optimal, there is no drop in the detection rate.

The false alert rate does increase for both data sets. In LLDOS 1.0, when δ_{max} equals 20 seconds, the false detection rate increases from about 5 percent to 8.7 percent. Then as the value of δ_{max} continues to increase, the false rate remains the same. LLDOS 2.0.2 has a higher false rate about 30 percent when δ_{max} equals 0, and increases to almost 36 percent at 45 seconds. The value that δ_{max} must reach before it affects false rate is much higher then the actual time difference between IDSs.

The two data sets appear to be resilient to the value of δ_{max}, for the most part because of the large delay between stages in the attack scenarios and because of the lack of major interference by the background traffic.

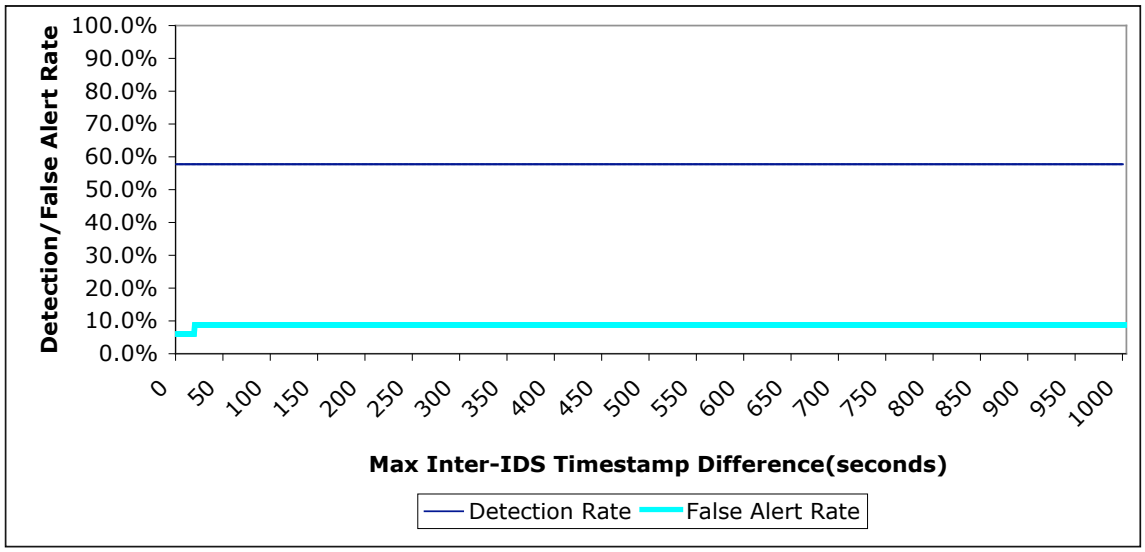

Figure 4-8: Data set LLDOS 1.0: ability to differentiate alerts

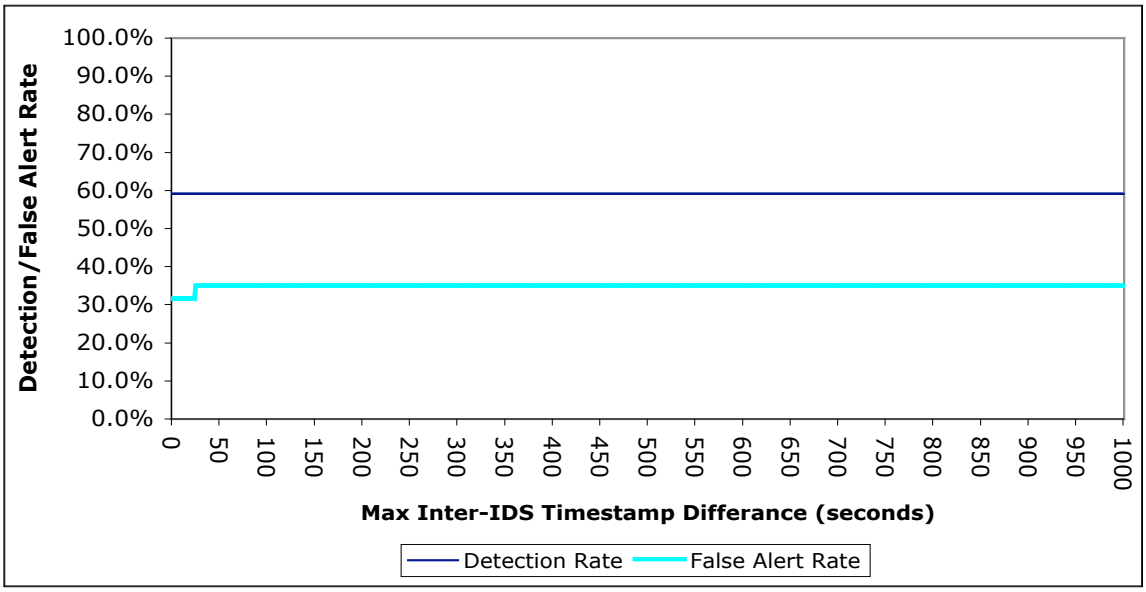

Figure 4-9: LLDOS 2.0.2 ability to differentiate alerts

4.3.7 Extended Method versus original Correlation Method

As mentioned earlier, the unmodified correlation method maintains stricter temporal constraints on the *prepares for* relationship then the extended method. Therefore, we expect that the original correlation method would miss more correlations as the delay between IDSs became larger. The result being that more of the relationships between alert from different IDSs would go undetected. The relaxing of the temporal constraints could also have negative consequences, as the constraints are relaxed there is a possibility that the extended correlation method would allow more false correlations (correlating of unrelated or false alerts). Yet, we believe that the benefits of relaxing the temporal constraints out weigh the cost. To show this we varied the actual delay between IDSs. In our experiments we increase the delay of the internal IDS, then correlated the alerts using the original correlation method and the extended method. We carried out two set of experiments, one set using the DARPA data sets and the second set with a modified version of the LLDOS 1.0 data set, in which the time between stages is reduced to one second.

4.3.7.1 Comparison using DARPA data sets.

In the original DARPA data sets, the delay between the end of one stage and the beginning of the next is large, for example, the delay between the first and the second stage of LLDOS 1.0 is 10 minutes. Since the delay between stages was large, there needs to be a large delay between IDSs in order to see where the original correlation method and extended method differ in performance. In our experiments, the delays between IDSs were 0 to 20 minutes. The tables 4.2 and 4.3 show points of interest in our results. The tables show correlation at delays equal 0, 10, and 20 minutes. We should note that a delay this large would not likely occur in a real world setting. The delay between stages is so large that at small delays between IDSs, both correlation methods yield the same results. This is because the timestamp difference is not significant compared to the delay between attack stages. For the experiments, the value of Δ_{max} used for our method was 0 when there was no delay, and the delay plus 1 second when delay was present.

As mentioned earlier, when there is no inter-IDS timestamp difference, both methods yield the same results. We expect this to be true regardless of the data set since setting Δ_{max} to 0 in the modified method is equivalent to the original correlation method. For our data sets, we obtained the same results for both methods as long as the inter-IDS timestamp difference was less than that of the minimum delay between stages of the attack. Therefore, as long as the inter-IDS timestamp difference was less than 10 minutes, which is the minimum delay between stages for LLDOS 1.0 and LLDOS 2.0.2, both methods yielded the same results as if no delay existed.

Table 4.2: Comparison of original correlation method and extended method with increasing delay for data set LLDOS 1.0

	LLDOS1.0 (Original)	LLDOS1.0 (Extended)	LLDOS1.0 (Original)	LLDOS1.0 (Extended)	LLDOS1.0 (Original)	LLDOS1.0 (Extended)
Delay (min)	0	0	10	10	20	20
Δ_{max}	N/A	0	N/A	10	N/A	20
#Alerts	101	101	101	102	101	103
True Alerts	95	95	95	95	95	95
False Alerts	6	6	6	8	6	8
#Correlations	538	538	535	540	450	541
False Correlations	36	36	37	38	49	39
False Correlation Rate (%)	6.7	6.7	6.9	7	10.9	7.2
Missed Correlations	31	31	35	31	119	31
Correct correlations	502	502	498	502	401	502
Missed Correlation Rate (%)	5.8	5.8	6.566	5.8	22.3	5.8

Table 4.3: Comparison of orignal correlation method and extended method with increasing delay for data set LLDOS 2.0.2

	LLDOS 2.0.2 (Original)	LLDOS 2.0.2 (Extended)	LLDOS 2.0.2 (Original)	LLDOS 2.0.2 (Extended)	LLDOS 2.0.2 (Original)	LLDOS 2.0.2 (Extended)
Delay (min)	0	0	10	10	20	20
Δ_{max}	N/A	0	N/A	10	N/A	20
#Alerts	19	19	19	21	19	21
True Alerts	13	13	13	13	13	13
False Alerts	6	6	6	8	6	8
#Correlations	26	26	26	31	24	31
False Correlations	5	5	8	10	8	10
False Correlations Rate (%)	19.2	19.2	30.7	32.2	33.3	32.2
Missed Correlation	11	11	14	11	16	11
Correct Correlations	21	21	18	21	16	21
Missed Correlations Rate (%)	34.3	34.3	43.7	34.3	50	34.3

As shown in tables 4.2 and 4.3, the number and percentage of missed correlations increase as the inter-IDS timestamp difference increases for the original correlation method, once there was a delay large enough to cause stages to appear to overlap. Missed correlations are true correlations that the correlator failed to make. In LLDOS 1.0 the missed correlations increase from 5.8 percent at inter-IDS timestamp difference equal 0, to 6.5 percent at 10 minutes, and to 22.3 percent at inter-IDS timestamp difference equal 20 minutes. For LLDOS 2.0.2 the missed correlations increase from 34.3 percent to 50 percent of total possible correlations, although the actual number of missed correlations increases only from 11 to 16. In LLDOS 1.0 missed correlations increase from 31 to 119. Our method performed much better; there was no increase in the number of missed correlations. For data set LLDOS 1.0, missed correlations were 31 (5.8 %) and 11 (34.2%) for data set LLDOS 2.0.2. Figures 4-10 and 4-11 show the progression of the missed correlation rate, calculated as $\frac{\#\text{correlations missed}}{\#\text{correlations observable}}$, for

LLDOS 1.0 and LLDOS 2.0.2 as the inter-IDS timestamp difference increases. The missed correlation rate is equivalent to the percentage of correlations missed.

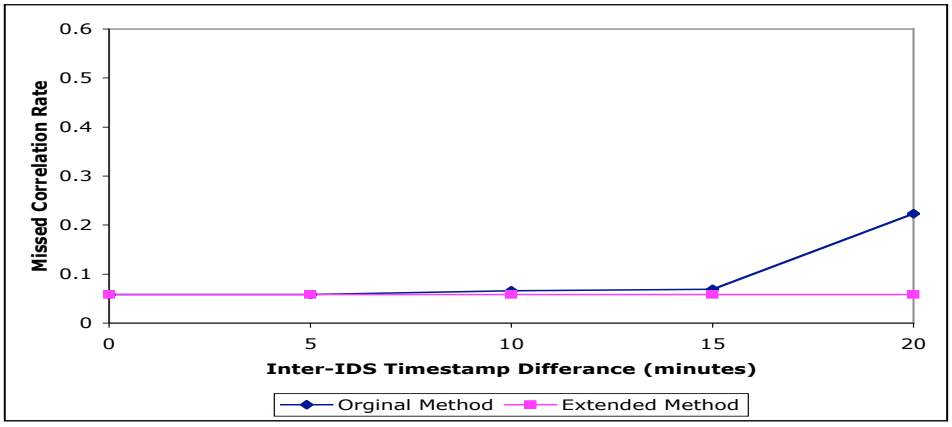

Figure 4-10: Comparison of Missed Correlation Rate for LLDOS 1.0 using extended and original correlation methods

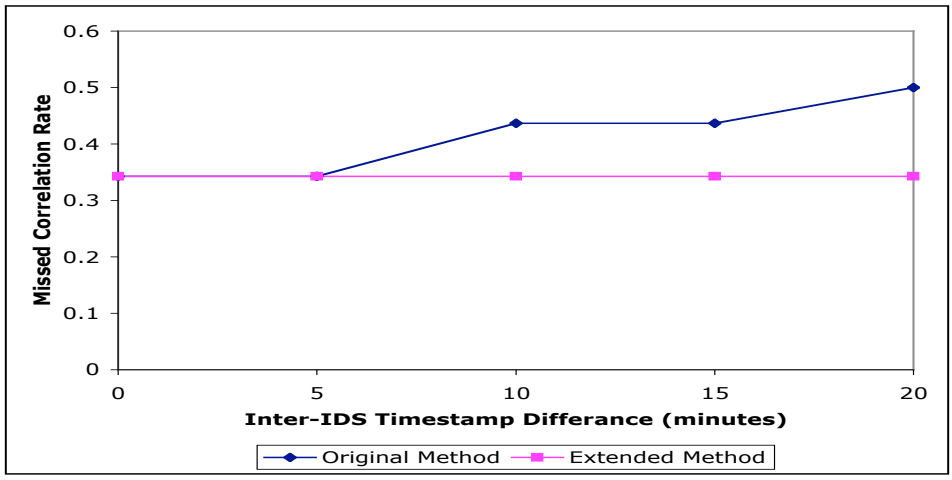

Figure 4-11: Comparison of Missed correlation Rate for LLDOS 2.0.2 using the extended and original correlation methods

Both methods showed an increase in the number of false correlations made. On close examination, the increase in false correlation occurred for different reasons. The original correlation method made an increasing number of false correlations between true alerts (alert involved in the attack) as the delay increased. Due to the timestamp difference, alerts that would not have satisfied the temporal constraints (when no delay exist) were correlated. The extended method's increase in false correlations is due to the relaxing of the temporal

constraints, more correlations with false alerts occurred. Alerts that were not part of the attack scenario were correlated. With a delay of 10 minutes, the original method was slightly better (6.9% versus 7% for LLDOS 1.0 and 30.7% versus 32.3% for data set LLDOS 2.0.2). Yet, at about 10 minutes the delay would start to force attack stages to overlap. At 20 minutes there is definite overlap between stages. At this point, the extended method performs better than using the original correlation method. At delay of 20 minutes, the extended method has a false correlation rate of 7.2% and the original method has a rate of 10.9% for data set LLDOS 1.0. For LLDOS 2.0.2, the extended method has false correlation rate of 32.2% versus 33.3% for the original method. Figures 4-12 and 4-13 graphs the false correlation rate, calculated as $\frac{\#\text{false correlations}}{\#\text{total correlations}}$, for LLDOS 1.0 and LLDOS 2.0.2 as the inter-IDS timestamp difference increases. The false correlation rate is equivalent to the percentage of false correlations made. Figure 4-12 shows that for data set LLDOS 1.0 the extended correlation method maintains a false correlation rate equal or lower than with the original correlation method. For data set LLDOS 2.0.2 that is a much smaller set with only 21 hyper-alerts, the extended method performs better only when sufficient overlap between stages of the attack occurs.

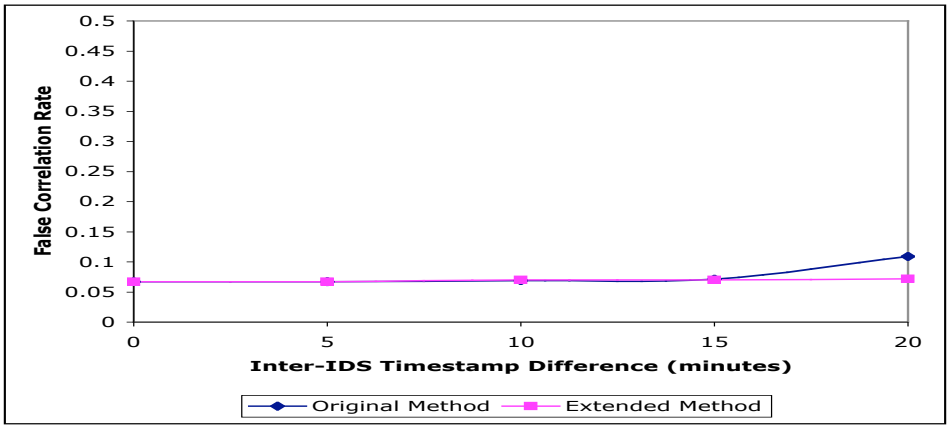

Figure 4-12: Comparison of False Correlation Rate for LLDOS 1.0 using extended and original correlation methods

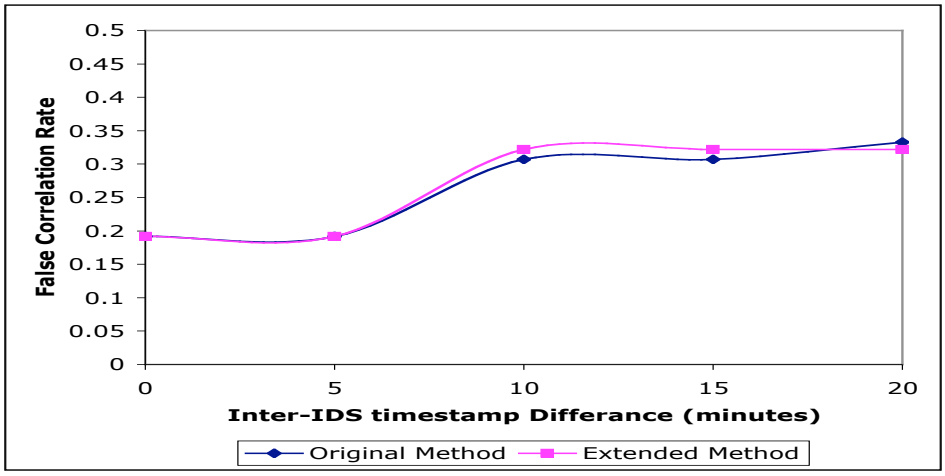

Figure 4-13: Comparison of False Correlation Rate for LLDOS 2.0.2 using extended and original correlation methods

The number of true alerts was equal for both methods. This is in large part because both data sets had well defined attack scenarios within the DMZ and internal traffic. Therefore, the correlation of alerts from the same IDS still occurred. Correlation of alerts from the same IDS is not affected by the inter-IDS timestamp difference. The original correlation method showed a significant drop in the number of correlations between alerts from different IDSs as the inter-IDS timestamp difference increased. The extended method shows no drop in correlations between alert from different IDSs. Therefore, the extended method more clearly shows the relationship between alerts from different IDSs.

With our data sets, there was a slight increase in the number of false alerts using our method as the delays increases, for each data set we had an increase of two false alerts. Since the data sets are small, this cannot be assumed to be a significant increase in false alerts compared to the original method. This does however indicate that false alerts could be a problem in the extended method under certain conditions.

4.3.7.2 Comparison using modified LLDOS 1.0

The conclusions we can make from our experiments with the DARPA data sets are limited by the large delay between attack stages in the scenarios. The delays between attack stages minimize the affects of any inter-IDS timestamp difference on correlation. Therefore, the delays used in the previous experiments are much larger than would normally occur in a real network. A delay of five or ten minutes could not be tolerated in a real networking environment.

The attacker carried out the LLDOS 1.0 and LLDOS 2.0.2 scenarios using limited help from automated scripts. If the attackers had automated all stages of the attack, the delay between stages could have been minimized. To show the effects of correlation on a more realistic scale we modified the LLDOS 1.0 data set to reduce the delays between stages. This was done by identifying the alerts in the attack and modifying the timestamps of the alerts, decreasing the time between stages. The resulting attack scenario had a one second delay between attack stages. We then performed our previous experiment to see the effects of the inter-IDS timestamp difference on correlation using the original and the extended correlation methods. As before, the max inter-IDS timestamp difference used was equal to the actual inter-IDS timestamp difference used. The inter-IDS timestamp differences used were values 0 to 20 seconds.

The results reinforced our earlier experiments. The detection rate and false alert rates were consistent with earlier results. The detection rate was 59% for both methods and did not change as the inter-IDS timestamp difference increased. This is because the scenarios are well defined in both the internal traffic and the DMZ traffic. Therefore, all hyper-alert that are correlated with at least one other hyper-alert from the same sensor, will not be dropped as the inter-IDS timestamp difference increases. The false alert rate (5.8%) for the original method is constant through out the experiment. Although, new false correlations are made as the inter-IDS timestamp difference increased, they do not result in new hyper-alerts being added. For the extended method the false rate is 5.8% when there is no delay and increases to 8.7% when the delay is two seconds. The relaxing of the temporal constraints results in

two additional false hyper-alerts being correlated. The false alert rate remains at 8.7% after two seconds.

Figure 4-14 shows the missed correlation rate as the inter-IDS timestamp difference increases. The missed correlation rate (6%) for the extended method is constant throughout the experiment. The same correlations are made as long as the value of Δ_{max} keeps up with the inter-IDS timestamp difference. The original method shows an increase in the number of correlations missed; the missed correlation rate goes from 6% at inter-IDS timestamp difference of zero, to 9.4% when the inter-IDS timestamp difference equals 20 seconds. This is expected because as the delay increases more and more correlations fail to satisfy the temporal constraints of the *prepares for* relationship.

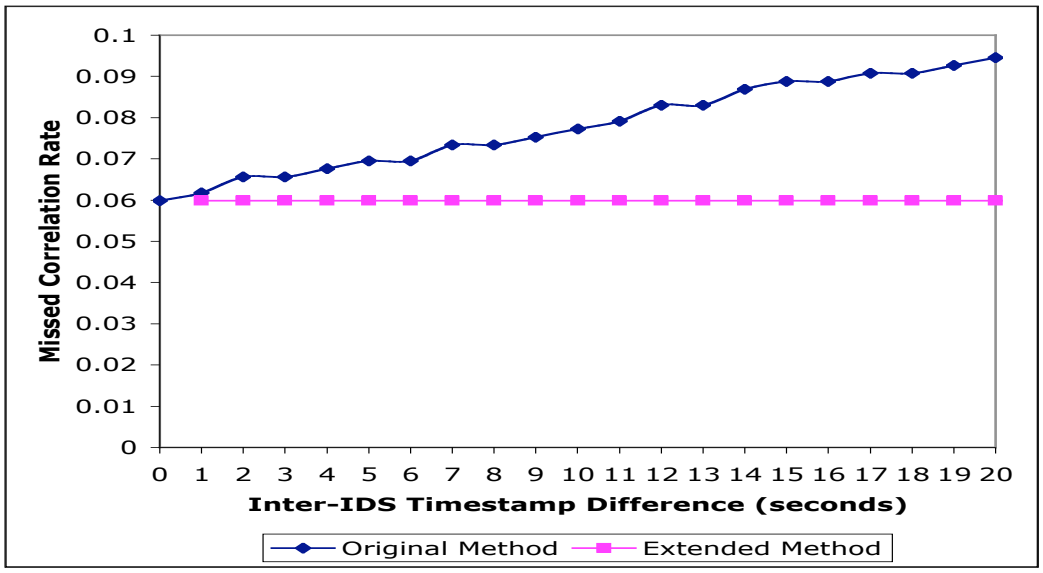

Figure 4-14: Comparison of missed correlation rate for original method and extended method with LLDOS 1.0 (modified)

The original method also shows a small increase in false correlations as the inter-IDS timestamp difference increases. Figure 4-15 shows the false correlation rate increased from 4.1% to 6.3%. The increase in false correlation rate is due to two factors; a small increase in

the number of false correlations and a reduction in the total number of correlations made. The number of correlations made decreases as the inter-IDS timestamp difference increases. Therefore, since the raw number of false correlations does not fall, they become a larger percentage of the total correlations made. For the extended method there is a small increase in the number of false correlations, the false correlation rate goes from 4.1% to 4.9%. This reinforces our earlier results, that as the delay increase, the relaxation of the temporal constraints may result a small increase in false correlations.

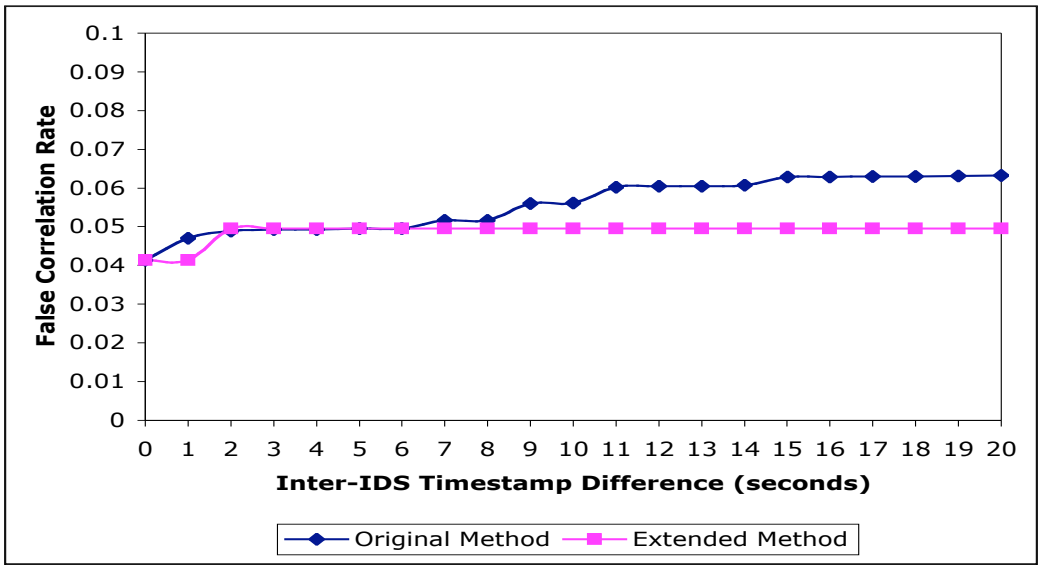

Figure 4-15: False correlation rates for LLDOS 1.0 (modifies) using extended and original correlation methods

4.3.8 Graph simplification:

In our experiments, the resulting graphs were not overly complicated. There was no need to simplify the graphs in order to understand the resulting attack scenarios. For both data sets, the multi-stage scenario was easily visible. This is not always true, for larger data set the number of hyper-alerts and correlations can make the attack scenario hard to understand. One way to simplify hyper-alert correlation graphs is to aggregate duplicate alerts. Figure 4-17,

shows the hyper-alert graph that results after clustering duplicate alerts together for the largest graph of LLDOS 1.0. Figure 4-16 shows the graph before aggregation. For LLDOS 1.0, the graph was reduced from 80 to 45 hyper-alerts. Similar aggregation of LLDOS 2.0.2 shows a reduction in the number of hyper-alerts from 15 to 12 hyper-alerts. The graphs look very similar to the correlation graph of the correlation of the inside traffic. For both sets, most of the hyper-alerts from the DMZ that appear in the largest graph are duplicates of alerts already present from the internal IDSs.

While the clustering of duplicate alerts can simplify the hyper-alert correlation graph, it can be difficult to find a one to one match for duplicate alerts. For example, in data set LLDOS 1.0 there are 9 Rsh alerts from the DMZ and 6 from the internal traffic, all with the same hyper–alert *fact* values and timestamps very close or equal to each other. Therefore, it is difficult to say that one alert corresponds to another. In fact, it can be difficult to tell one alert from another without the alert ID. For example in LLDOS 1.0 three of the DMZ rsh alerts have timestamps of 12:35:51, and all other features are the same (source, destination, etc). Finding the corresponding alerts from the internal traffic is difficult. Recognition of duplicate alerts is farther complicated when inter-IDS delays exist, since the timestamps are no longer a match for duplicate alerts.

In our experiments, we did not attempt to find a one to one match. Our definition of duplicate only states that alerts sufficiently similar to each other (based on fact values) may represent the same event. Our definition of duplicate provides a method for simplifying the aggregating of duplicate alerts. In addition, within stages of the attack scenarios for LLDOS 1.0 and LLDOS 2.0.2, the hyper-alerts occur close enough to each other that aggregating by hyper-alert time is as effective at simplifying the hyper-alert correlation graphs as aggregating by hyper-alert *fact* values. This is true only when there is no delay between IDSs, or the delay is sufficiently small. For large delays, aggregating by time may not produce the same results as aggregating by duplicates.

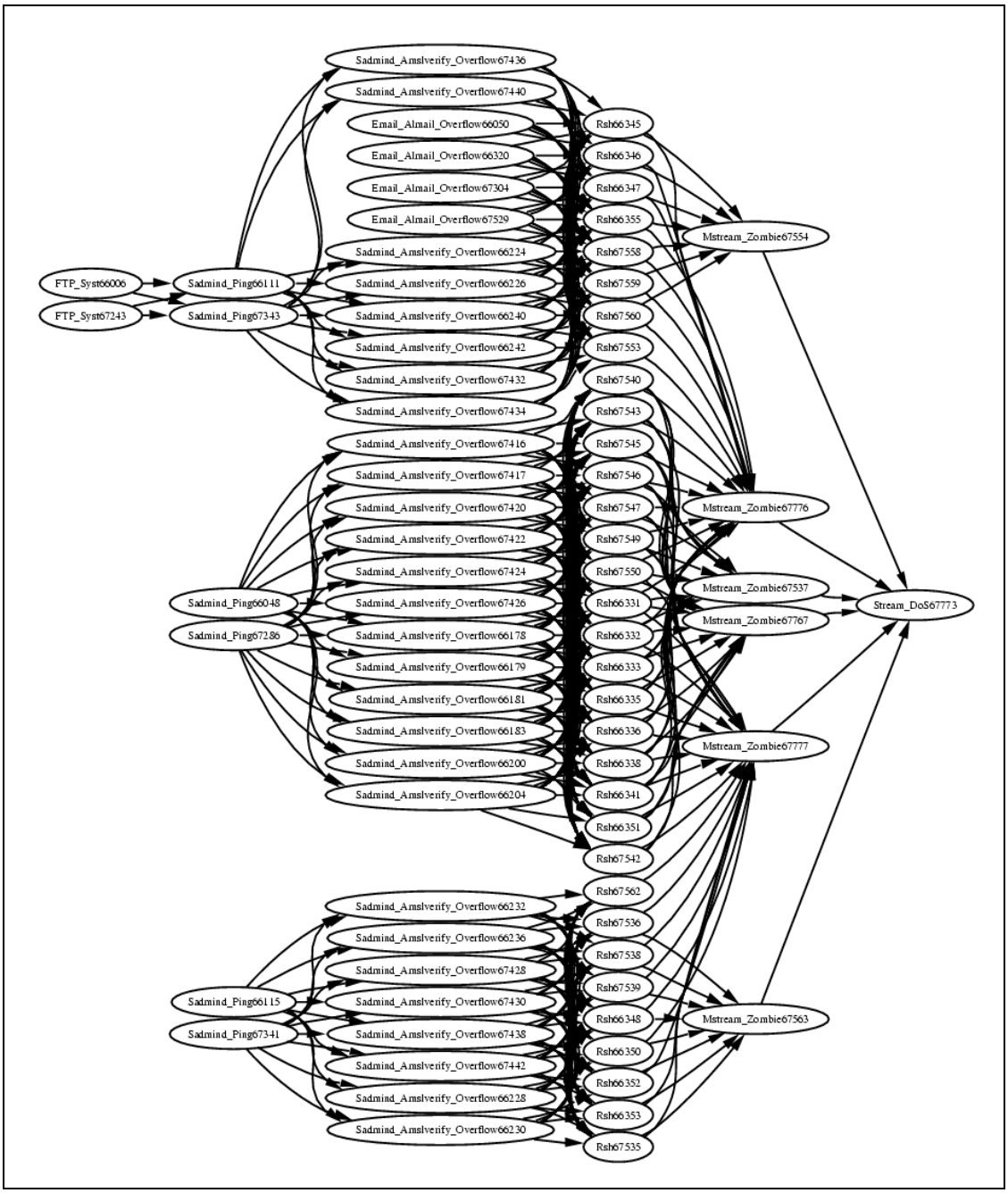

Figure 4-16: LLDOS 1.0 Hyper-alert correlation graph before duplicate aggregation

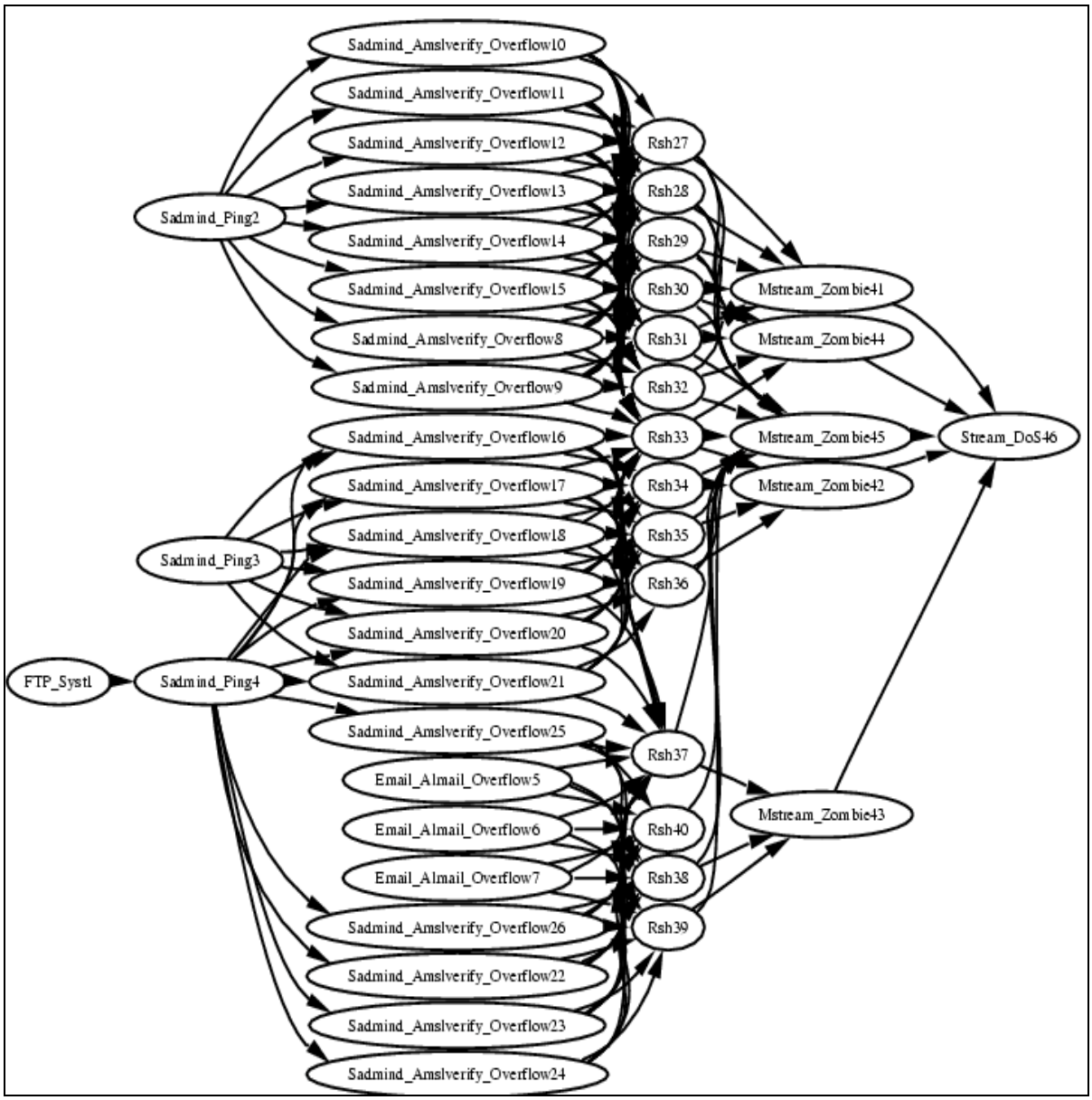

Figure 4-17: LLDOS 1.0 Hyper-alert correlation graph decomposed to cluster duplicates

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we presented an experimental study of intrusion alert correlation using multiple homogeneous IDSs. We extended the basic intrusion alert correlation framework developed at NCSU to account for possible timing issues (network delay, host overloading, etc.) that can cause timestamp differences between alerts from different IDSs. We propose relaxing the temporal constraints when considering whether two alerts from different IDSs should be correlated. We introduced the maximum inter-IDS timestamp difference (Δ_{max}) to relax the timing constraints on the *prepares for* relationship. The maximum inter-IDS timestamp difference is the maximum delay between alerts from different IDSs for the same event. Two alerts are correlated if the preparing alert's termination timestamp is more than the prepared for alert's starting timestamp plus the maximum inter-IDS timestamp.

Experiments performed using the DARPA 2000 intrusion detection scenario specific data sets showed that our method was an improvement over RealSecure's uncorrelated alerts. Our extended method was successful at identifying multi-staged attack scenarios. It showed that correlating alerts from multiple IDSs provides an automated method for showing how an attack progressed though the network and presenting sensor reinforcement (sensor agreement on the attacks). Correlation also showed a significant reduction in false detection rates (6% for LLDOS 1.0 and 32% for LLDOS 2.0.2) compared to uncorrelated alerts (above 90% for both data sets).

Our experiments also showed that compared to the original correlation method, the extended method was better at identifying correlations when inter-IDS timestamp differences occurred and stages of an attack overlapped. The original correlation method showed a significant increase in correlations missed, while the extended method showed no increase in the missed correlation rate.

Our experiments also showed that the time delay between stages could be as important as the value of Δ_{max} in determining correlation between alerts. In our experiments using the DARPA 2000 data sets, the value of Δ_{max} had minimal effect on correlation strength and false rates. The delays between stages in the DARPA data sets were sufficiently large to minimize the effects of Δ. When the delay between stages is larger than the inter-IDS timestamp difference, the extended and original correlation methods performed the equally well. The delay between IDSs had to increase to 10 minutes before the value of Δ_{max} had significant impact on correlation. Our experiments using a modified LLDOS 1.0 scenario showed that Δ_{max} plays a larger influence on scenarios in which there is little or no delay between stages. The extended method performs better then the original method under these conditions. The extended method showed no increase in missed correlations and only a small increase in false correlations. The original method showed an increase in false correlations and a larger increase in missed correlations.

However, the experiments showed that under certain conditions our methods performance might decrease in terms of false alerts. First, in data set LLDOS 2.0.2, just correlating the DMZ alerts and the internal alerts together resulted in an increase in false alerts, compared to correlating the alert independently. Although the increase cannot be considered significant because of the small size of the data set, it does suggest that false rates could be a problem when the background traffic is very similar to the attack traffic. In both data sets there is a small increase in false alert rate when we relax the temporal constraints (two additional alerts for LLDOS 1.0 and one for LLDOS 2.0.2).

5.2 Future Work

5.2.1 Correlation of Alerts From multiple Heterogeneous IDSs

In our experiments, we limited correlation to alerts generated by homogeneous IDSs. The effectiveness of detecting attacks could be farther improved with the correlation of alerts from heterogeneous IDSs. This is because heterogeneous IDSs have different strengths and weaknesses. Therefore, one IDS may be able to detect attacks that another IDS misses.

One method for correlation of alerts from heterogeneous IDSs is to handle heterogeneous alerts by transforming them into a case of homogenous alerts. This could be done if all alerts were converted to a single common representation. The main issues that remain are how to handle the conversion. First, we need a single representation for alerts, such as IDMEF. Then we need to be able to map the alerts from each IDS to its IDMEF representation. The structure should be sufficiently strict, that we do not end up in a situation in which each IDS developer is able to identify hundreds or even thousands of unique hyper-alert types. Therefore, in addition to a common representation of the hyper-alert types there is a need for a complete and consistent alert taxonomy. Finally, there is a need for tools to automate the lookup, classification, and conversion of alerts from vender specific representations to the common representation.

5.2.2 Recognizing Duplicate Alerts

In this thesis, we assumed a simplified definition of what constitutes a duplicate alert. We assumed that if two alerts from different IDSs have the same *fact* attributes and they occur at the same time (taking in to account any possible inter-IDS timestamp difference), that they refer to the same event. Yet, the issue of duplicate alerts is much more complicated. First, it can be much more difficult to recognized duplicate alerts. For example, in LLDOS 1.0 there are a number of Sadmind_Overflow alerts that occur with in a short period of time. Of theses alerts, 5 alerts from the internal traffic that have the same *fact* values and same timestamp, and there are 3 alerts from the DMZ that also have the same timestamp and *fact* values.

Finding the true one to one mapping in this case is not possible using our simplified definition of duplicate alerts.

In addition, the implication of duplicate alerts needs to be looked at. What do duplicate alerts mean? Can we for example use duplicate alerts to provide a confidence measure of an alert. If we are providing a confidence measure based on or influenced by whether the alerts are duplicates then we may want a one to one mapping of duplicate alerts. Are alerts with duplicates more likely to be true positives? With homogenous IDS, this would not mean much since the IDSs are likely to classify the same traffic as alerts, but with heterogeneous IDSs, this could be different.

5.2.3 Real-time Correlation

This thesis work was done, with minor modifications, using the offline toolkit to correlate intrusion alerts. A DBMS was used to process the preprocessed data. Yet, the main payoff of intrusion alert correlation will come when it is possible to do the correlations in real time. However, there is a performance penalty using the DBMS that limits real-time processing.

How do we improve performance enough to allow real time correlation to occur? One way to improve the efficiency of the underlying system, for example using in-memory queries could help. Our group has already started some work on this area. [39] looks at techniques for improving the efficiency of alert correlation by looking at main memory index structures (e.g., B Trees, T Trees. Linear Hashing) and database query optimization techniques. Yet, there is still much work to be done, before real-time correlation is a viable alternative.

5.2.4 Improving Detections Rate

In our experiments using the DARPA data set, there is a decrease in the detection rate compared to RealSecure. Some alerts that are part of the attack sequence are not correlated. What we want to accomplish through correlation is that we want reduce the false alert rate

without decreasing the detection rate of true positives. So, the issue remains, how can we continue to reduce the false alert rate with out decreasing the detection rate?

It would also be beneficial if we could use our correlator to improve the detection rate of the underlying IDS. Could the information from the correlator be used to predict where to look for other attacks that should be correlated but are not because the underlying IDS did not detect them? Can the attack scenarios be sued to predict the next action to be taken by an attacker?

LIST OF REFERENCES

1. 2000 DARPA Intrusion Detection Scenario Specific Data sets, MIT Lincoln Lab, 2000. http://www.ll.mit.edu/ISt/ideval/data/2000/2000_data_index.html
2. CERT/CC Statistics 1988-2002, CERT, 2003. www.cert.org/stats/cert_stats.html
3. Intrusion Detection Working Group. http://www.ietf.org/html.charters/idwg-charter.html
4. The open source intrusion detection system, 2002. http://www.snort.org.
5. RealSecure Intrusion Detection System, ISS, Inc., 2003. http://realsecure.com
6. Anderson, D., Frivold, T. and Valdes, A. Next-generation intrusion-detection expert system (NEDES), Computer Science Laboratory, 1995.
7. Anderson, J. Computer Security Threat Monitoring and Surveillance, 1980.
8. Axelsson, S. The Base-Rate Fallacy and the difficulty of Intrusion Detection. *ACM Transactions on Information and System Security*, *3* (3). 186-205.
9. Axelsson, S. Research in intrusion-detection systems: A survey, Department of Computer Engineering, Chalmers University of Technology, Gotenborg, Sweden, 1999.
10. Balasubramaniyan, J., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E.H. and Zamboni, D. An architecture for intrusion detection using autonomous agents, The Coast Project, Department of Computer Science, Purdue University, West Lafayette, 1998.
11. Balthrop, J., Esponda, F., Forrest, S. and Glickman, M., Coverage and Generalization in an Artificial Immune System. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, (New York, NY, 2002), 3-10.
12. Barbara, D., Couto, J., Jajodia, S. and Wu, N. ADAM: A test-bed for exploring the use of data mining in intrusion detection. *ACM SIGMOD Record*, *30* (4). 15-24.
13. Bishop, M., A security analysis of the NTP protocol version 2. In *6th Annual Computer Security Application Conference*, (1990), 20-29.
14. Cui, Y. A Toolkit For Intrusion Alert Correlation Based on Prerequisites and Consequences of Attacks *Computer Science*, North Carolina State University, Raleigh, 2002.
15. Cuppens, F. and Miege, A., Alert Correlation in a cooperative Intrusion Detection Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, (Oakland, 2002).
16. Cuppens, F., Miege, A. and Benferhat, S. Correlation in an intrusion detection process. *Atelier Secrite des Communications sur Internet (SECI' 02)*. 153-171.
17. Cuppens, F. and Ortalo, R. LAMDA: A language to model a database for detection of attacks. *Proc. of Recent Advances in Intrusion Detection (RAID 2000)*. 197-216.
18. Dain, O. and Cunningham, R., Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshops on Data Mining for Security Applications*, (2001), 1-13.

19. Feiertag, R., Rho, S.M., Redmond, T., Zhang, C., Levitt, K., Peticolas, D., Heckmann, M., Satniford, S. and McAlernery, J. Intrusion detection inter-component adaptive negotiation. *Computer Networks* (34). 605-621.
20. Fox, K.L., Henning, R.R., Reed, J.H. and Simonian, R.P., A neural network approach towards intrusion detection. In *Proceedings of the d13th National Computer Security Conference*, (Baltimore, MD, 1990), National Institute of Standards and Technology (NIST), 125-134.
21. Garvey, T.D. and Lunt, T.F., Model-based intrusion detection. in *Proceedings of the 14th national Computer Security Conference*, (Baltimore MD, USA, 1991), NIST, National Institute of Standards and Technology/Nation Computer Security Center, 372-385.
22. Ghosh, A.K., Wanken, J. and Charron, F., Detecting anomalous and unknown intrusion against programs. in *Proceedings of the 14th Annual Computer Security Applications Conference*, (Los Alamitos, CA, 1998), IEEE Computer Society, 259-267.
23. Halme, L.R. and Bauer, K.R., AINT misbehaving- A Taxonomy of anti-intrusion techniques. in *Proceedings of the 18th National Information Systems Security Conference*, (Baltimore, MD,USA, 1995), National Institute of Standards and Technology/National Computer Security Center, 163-172.
24. Hoagland, J., McAlemy, J. and S., S. Practical Automated Detection of Stealthy Portscans.
25. Hofmeyr, S., Forrest, S. and Somayaji, A. Intrusion detection using sequences of system calls. *Journal of Computer Security* (6). 151-180.
26. Julisch, K., Mining Alarm Clusters to Improve Alarm Handling Efficiency. in *Proceedings of the a5th Computer Security Foundation Workshop*, (2002).
27. Kathari, P. Intrusion Detection Interoperability and Standardization, 2002.
28. Kreugel, C., Valeur, F. and Kemmerer, R.A., Statefull intrusion detection for high-speed networks. in *Proceedings of 2002 IEEE Symposium on Security and Privacy*, (2002).
29. Lane, T. and Brodley, C.E., Temporal sequence learning and data reduction for anomaly detection. in *Proceedings of 5th conference on computer and communications security*, (New York, NY, 1998), ACM press, 150-158.
30. Lin, J., Wang, X.S. and Jajodia, S., Abstraction-based misuse detection: High-level specifications and adaptable strategies. in *Proceedings of the 5th Int'l symposium on Recent Advances in Intrusion Detection (RAID 2002)*, (2002).
31. Lindqvist, U. and Porras, P.A., Detecting computer and network misuse through the production-based expert system toolset(P-BEST). in *Proceedings of the 11th computer security foundations workshop*, (Los Alamitos, CA, 1998), IEEE Computer Society, 146-161.
32. Lippman, R., et al., Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. in *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'00)*, (Hilton Head, SC USA, 2000), 12-26.
33. Lunt, T., Jagannathan, R., Lee, R., Listgarten, S., Edwards, D., Neumann, P., Javitz, H. and Valdes, A. IDES: The Enhanced prototype, A Real-time Intrusion Detection

System, CSL SRI International, Computer Science Laboratory, Menlo Park, CA, 1988.
34. McHugh, J. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, *3* (4).
35. Mounji, A., Charlier, B.L., Zampunieris, D. and Habra, N., Distributed audit trail analysis. in *Proceedings of the ISOC'95 symposium on network and distributed system security*, (1995), IEEE Computer Society, 102-112.
36. Ning, P., Cui, Y. and Reeves, D.S. Constructing attack scenarios through correlation of intrusion alerts. *9th ACM conference on Computer & Communications Security*.
37. Ning, P. and Jajodia, S. *Intrusion Detection Techniques*, 2003.
38. Ning, P., Jajodia, S. and Wang, X.S. Abstraction-based intrusion detection in distributed environments. *ACM Transactions on Information and System Security*, *4* (4). 497-452.
39. Ning, P. and Xu, D. Adapting query optimization techniques for efficient intrusion alert correlation, North Carolina State University, Department of Computer Science, Raleigh, 2002.
40. Ning, P. and Cui, Y. An intrusion alert correlator based on prerequisites of intrusions, North Carolina State University, Raleigh, 2002.
41. Paxson, V. Bro: A system for detecting network intruders in real-time. *Computer Networks*. 23-24.
42. Porras, P., Schnackernberg, D., Staniford-Chen, S., Stillman, M. and Wu, F. Common Intrusion Detection Framework architecture, 1999.
43. Roesch, M., Snort -- Lightweight Intrusion Detection For Networks. in *Proceedings of the 13th Systems administration Conference*, (1999).
44. Sekar, R., Bendre, M., Dhurjati, D. and Bollineni, P., A fast automaton based method for detection anomalous program behaviors. in *Proceedings of the 2001 IEEE symposium on security and privacy*, (Los Almaitos, CA, 2001), IEEE Computer Society, 144-155.
45. Sekar, R., Guang, Y. and Shanbhag, T., A high-performance network intrusion detection system. in *Proceedings of the 6th ACM conference on computer and communications security*, (New York, NY, 1999), ACM Press, 8-17.
46. Smaha, S.E., Haystack: An Intrusion Detection System. in *Proceedings of the IEEE Fourth Aerospace Computer Security applications Conference*, (1988).
47. Smith, S.W. and Tygar, T.J., Security and privacy for partial order time. in *ISCA Seventh International Conference on Parallel and Distributed Computing Systems*, (2000).
48. Snapp, S.R., et al., DIDS (distributed intrusion detection system) - Motivation, architecture, and an early prototype. in *Proceedings of the 14th National Computer Security Conference*, (1991), 167-176.
49. Spafford, E.H. and Zamboni, D. Intrusion detection using autonomous agents. *Computer Networks*, *34*. 547-570.
50. Stallings, W. *Network Security Essential- Application and Standards*. Prentice-Hall, 1999.

51. Templeton, S. and Levit, K., A requires/provides model for computer attacks. in *Proc. of New Security Paradigms Workshops*, (2000), 31-38.
52. Teng, H.S., Chen, K. and Lu, S.C., Adaptive real-time anomaly detection using inductively generated sequential patterns. in *Proceedings of 1990 IEEE Symposium on Security and Privacy*, (1990), 278-284.
53. Valdes, A. and Skinner, K., Probabilistic Alert Correlation. in *Proceedings of the Fourth International Workshop on the Recent advances in intrusion detection (RAID 2001)*, (Davis, CA, USA, 2001), 54-68.
54. Vigna, G. and Kemmerer, R.A. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, *7* (1). 37-71.
55. Wagner, D. and Dean, D., Intrusion detection via static analysis. in *Proceedings of 2001 IEEE symposium on security and privacy*, (Los Alamitos, CA, 2001), IEEE Computer Society, 156-168.