Constructing Attack Scenarios through Correlation of Intrusion Alerts

PENG NING
North Carolina State University
YUN CUI
MCNC Research & Development Institute
and
DOUGLAS S. REEVES
North Carolina State University

Traditional intrusion detection systems (IDSs) focus on low-level attacks or anomalies, and raise alerts independently, though there may be logical connections between them. In situations where there are intensive attacks, not only will actual alerts be mixed with false alerts, but the amount of alerts will also become unmanageable. As a result, it is difficult for human users or intrusion response systems to understand the alerts and take appropriate actions. This paper presents a practical technique to address this issue. The proposed approach constructs attack scenarios by correlating alerts on the basis of *prerequisites* and *consequences* of attacks. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful, while the consequence of an attack is the possible outcome of the attack. Based on the prerequisites and consequences of different types of attacks, our method correlates alerts by (partially) matching the consequences of some prior alerts with the prerequisites of some later ones. Moreover, to handle large collections of alerts, this paper presents three interactive analysis utilities aimed at reducing the complexity of the constructed attack scenarios without losing the structure of the attacks. This paper also reports the experiments conducted to validate the proposed techniques with the 2000 DARPA intrusion detection scenario-specific datasets, and the data collected at the DEFCON 8 Capture The Flag (CTF) event.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses)*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: intrusion detection, security management, alert correlation

Parts of this paper appeared in preliminary form in *Proceedings of the 9th ACM Conference on Computer and Communications Security* [Ning et al. 2002b] and *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection* [Ning et al. 2002a].
This work is partially supported by the U.S. Army Research Office (ARO) under grant DAAD19-02-1-0219, and by the National Science Foundation (NSF) under grant CCR-0207297. Ning's work is also supported by the NSF under grant ITR-0219315.
Authors' addresses: Peng Ning and Douglas S. Reeves, Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8207, emails: {ning, reeves}@csc.ncsu.edu; Yun Cui, Advanced Networking Research, MCNC Research & Development Institute, Research Triangle Park, NC 27709; email: ycui@anr.mcnc.org.

1. INTRODUCTION

Intrusion detection has been studied for more than twenty years since Anderson's report [Anderson 1980]. Intrusion detection techniques can be roughly classified as *anomaly detection* and *misuse detection*. Anomaly detection (e.g., NIDES/STAT [Javits and Valdes 1993]) is based on the normal behavior of a subject (e.g., a user or a system); any action that significantly deviates from the normal behavior is considered intrusive. Misuse detection (e.g., NetSTAT [Vigna and Kemmerer 1999]) detects attacks based on the characteristics of known attacks or system vulnerabilities; any action that conforms to the pattern of a known attack or vulnerability is considered intrusive.

Traditional intrusion detection systems (IDSs) focus on low-level attacks or anomalies, and raise alerts independently, though there may be logical connections between them. In situations where there are intensive attacks, not only will actual alerts be mixed with false alerts, but the amount of alerts will also become unmanageable. As a result, it is difficult for human users or intrusion response systems to understand the alerts and take appropriate actions. Therefore, it is necessary to develop techniques to construct *attack scenarios* (i.e., steps that attackers use in their attacks) from alerts to facilitate intrusion analysis.

In this paper, we develop a practical alert correlation technique that can be used to construct attack scenarios for real-life intrusion analysis. Our method can be explained through the following observation: most attacks are not isolated, but related as different stages of attack sequences, *with the early stages preparing for the later ones*. For example, in Distributed Denial of Service (DDOS) attacks, the attacker has to install the DDOS daemon programs in vulnerable hosts before he can instruct the daemons to launch an attack. In other words, an attacker has to (or usually does) reach a certain state before he can carry out certain attacks, and usually reaches the state by launching some other attacks.

Based on this observation, we correlate alerts using *prerequisites and consequences of attacks*. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful, while the consequence of an attack is the possible outcome of the attack. For example, the existence of a vulnerable service is the prerequisite of a remote buffer overflow attack against the service, and as the consequence of the attack, the attacker may gain access to the host. Accordingly, we correlate the alerts together when the attackers launch some early attacks to prepare for the prerequisites of some later ones. For example, if they use a UDP port scan to discover the vulnerable services, followed by an attack against one of the services, we can correlate the corresponding alerts together. It is well-known that current IDSs often miss unknown attacks, or variations of known attacks. To tolerate missing detections, our method allows partial satisfaction of prerequisites of an attack. In addition, our method allows flexible alert aggregation, and provides intuitive representations of correlated alerts.

We apply this alert correlation method to analyze real-world, intrusion intensive data sets. In particular, we would like to see how well the alert correlation method can help human users organize and understand intrusion alerts, especially when IDSs report a large amount of alerts. We argue that this is a practical problem that the intrusion detection community is facing. As indicated in [Manganaris et al. 2000], "encountering 10–20,000 alarms per sensor per day is common." In this paper, we also present three utilities (called *adjustable graph reduction, focused analysis*, and *graph decomposition*) to facilitate the analysis of large sets of correlated alerts. These utilities are intended for human users to analyze and understand the correlated alerts as well as the strategies behind them.

The contribution of this paper is as follows. First, we develop a formal framework for alert correlation. Our method can deal with attack attempts and correlate alerts as long as there are signs of connections between them, even if some related attacks fail or bypass the IDSs. In addition, our method provides an intuitive mechanism (called hyper-alert correlation graph) to represent the attack scenarios constructed through alert correlation. Second, we develop an off-line tool that implements our alert correlation method. Based on the information about different types of attacks, our tool processes the alerts reported by IDSs and generates hyper-alert correlation graphs as the output. As we will see in Section 5, these hyper-alert correlation graphs reveal the structure of series of attacks, and thus the strategy behind them. Third, we develop three interactive utilities to facilitate the analysis of very large attack scenarios. These utilities reduce the size of large attack scenarios while keeping the structure of the attacks. Finally, we perform a series of experiments to validate our methods using 2000 DARPA intrusion detection scenario specific datasets [MIT Lincoln Lab 2000] and the network traffic captured at the DEFCON 8 Capture the Flag (CTF) event [DEFCON 2000]. Our results show that our correlation method not only correlates related alerts and uncovers the attack strategies, but also provides a way to differentiate between alerts, and that the interactive analysis utilities can effectively simplify the analysis of large amounts of alerts. Our analysis also reveals several attack strategies that appeared in the DEFCON 8 CTF event.

The remainder of this paper is organized as follows. The next section discusses related work. Section 3 presents our formal framework for correlating alerts using prerequisites and consequences of attacks, as well as a DBMS based implementation of this approach. Section 4 describes three utilities for analyzing attack scenarios constructed from large collections of alerts. Section 5 reports our experiments with the 2000 DARPA intrusion detection scenario specific datasets and the DEFCON 8 CTF dataset. Section 6 discusses the advantages and limitations of our approach, and Section 7 concludes this paper and points out some future research directions.

2. RELATED WORK

Intrusion detection has been studied for more than twenty years, since Anderson's report [Anderson 1980]. A survey of the early work on intrusion detection is given in [Mukherjee et al. 1994], and an excellent overview of current intrusion detection techniques and related issues can be found in a recent book [Bace 2000].

Research on intrusion alert correlation has been rather active recently. The first class of approaches (e.g., Spice [Staniford et al. 2002], probabilistic alert correlation [Valdes and Skinner 2001], and the alert clustering methods in [Cuppens 2001] and

[Julisch 2001]) correlates alerts based on the similarities between alert attributes. Though they are effective for clustering similar alerts (e.g., alerts with the same source and destination IP addresses), they cannot fully discover the causal relationships between related alerts.

Another class of methods (e.g., correlation based on STATL [Eckmann et al. 2002] or LAMBDA [Cuppens and Ortalo 2000], and the data mining approach [Dain and Cunningham 2001]) performs alert correlation based on attack scenarios specified by human users, or learned from training datasets. A limitation of these methods is that they are restricted to *known* attack scenarios, or those that can be generalized from known scenarios. A variation in this class uses a consequence mechanism to specify what types of attacks may follow a given attack, partially addressing this problem [Debar and Wespi 2001].

A third class of methods, including JIGSAW [Templeton and Levitt 2000], the MIRADOR correlation method [Cuppens and Miege 2002], and our approach, targets recognition of multi-stage attacks; it correlates alerts if the prerequisites of some later alerts are satisfied by the consequences of some earlier alerts. Such methods can potentially uncover the causal relationship between alerts, and are not restricted to known attack scenarios.

Our method can be considered as a variation of JIGSAW [Templeton and Levitt 2000]. Both methods try to uncover attack scenarios based on specifications of individual attacks. However, our method also differs from JIGSAW. First, our method allows partial satisfaction of prerequisites (i.e., required capabilities in JIGSAW [Templeton and Levitt 2000]), recognizing the possibility of undetected attacks and that of attackers gaining information through non-intrusive ways (e.g., talking to a friend working in the victim organization), while JIGSAW requires all required capabilities be satisfied. Second, our method allows aggregation of alerts, and thus can reduce the complexity involved in alert analysis, while JIGSAW currently does not have any similar mechanisms. Third, we develop a set of utilities for alert correlation and interactive analysis of correlated alerts, which is not provided by JIGSAW.

The work closest to ours is the MIRADOR correlation method proposed in [Cuppens and Miege 2002], which was developed independently and in parallel to ours. These two methods share substantial similarity. The MIRADOR approach also correlates alerts using partial match of prerequisites (pre-conditions) and consequences (post-conditions) of attacks. However, the MIRADOR approach uses a different formalism than ours. In particular, the MIRADOR approach treats alert aggregation as an individual stage before alert correlation, while our method allows alert aggregation during and after correlation. As we will see in Section 4, our treatment of alert aggregation leads to the three utilities for interactive alert analysis.

A formal model named M2D2 was proposed in [Morin et al. 2002] to correlate alerts by using multiple information sources, including the characteristics of the monitored systems, the vulnerability information, the information about the monitoring tools, and information of the observed events. Due to the multiple information sources used in alert correlation, this method can potentially lead to better results than those simply looking at intrusion alerts. A mission-impact-based approach was proposed in [Porras et al. 2002] to correlate alerts raised by INFOSEC devices such as IDSs and firewalls. A distinguishing feature of this approach is that it correlates the alerts with the importance of system assets so that attention can be focused on critical resources. These methods are complementary to ours.

Several techniques have used the prerequisites and consequences of attacks for vulnerability analysis purposes. In [Ritchey and Ammann 2000], a model checking technique was applied to analyze network vulnerabilities on the basis of prerequisites and results (i.e., consequences) of exploits (i.e., attacks), along with hosts and network connectivity information. In [Sheyner et al. 2002] and [Jha et al. 2002], the technique in [Ritchey and Ammann 2000] was further extended to generate and analyze all possible attacks against a vulnerable networked system. These techniques are focused on analyzing what attacks *may happen* to violate a *given* security property. In contrast, our purpose is to reconstruct what *has happened* according to the alerts reported by IDSs, and our technique has to deal with the inaccuracy of IDSs (i.e., false alerts and undetected attacks). We consider our method as complementary to these vulnerability analysis techniques.

Several languages have been proposed to represent attacks, including STAT [Ilgun et al. 1995; Vigna and Kemmerer 1999; Eckmann et al. 2002], Colored-Petri Automata (CPA) [Kumar and Spafford 1994; Kumar 1995], LAMBDA [Cuppens and Ortalo 2000], and MuSig [Lin et al. 1998] and its successor [Ning et al. 2001]. In particular, LAMBDA uses a logic-based method to specify the precondition and postcondition of attack scenarios, which is similar to our method. However, all these languages specify entire attack scenarios, which are limited to known scenarios. In contrast, our method (as well as JIGSAW and the MIRADOR correlation method) describes prerequisites and consequences of individual attacks, and correlates detected attacks (i.e., alerts) based on the relationship between these prerequisites and consequences. Thus, our method can potentially correlate alerts in unknown attack scenarios.

GrIDS uses activity graphs to represent the causal structure of network activities and detect propagation of large-scale attacks [Staniford-Chen et al. 1996]. Our method also uses graphs to represent correlated alerts. However, unlike GrIDS, in which nodes represent hosts or departments and edges represent network traffic between them, our method uses nodes to represent alerts, and edges the relationships between the alerts.

Alert correlation has been studied in the context of network management (e.g., [Gruschke 1998], [Ricciulli and Shacham 1997], and [Gardner and Harle 1998]). In theory, alert correlation methods for network management are applicable to intrusion alert correlation. However, intrusion alert correlation faces more challenges than its counter part in network management: While alert correlation for network management deals with alerts about natural faults, which usually exhibits regular patterns, intrusion alert correlation has to cope with less predictable, malicious intruders.

3. A FRAMEWORK FOR ALERT CORRELATION

As discussed in the introduction, our method is based on the observation that in a series of attacks, the attacks are usually not isolated, but related as different stages of the attack sequence, with the early ones preparing for the later ones. To take advantage of this observation, we propose to correlate the alerts generated by IDSs using prerequisites and consequences of the corresponding attacks. Intuitively, the *prerequisite* of an attack is the necessary condition for the attack to be successful. For example, the existence of a vulnerable service is a prerequisite for a remote buffer overflow attack against the service. Moreover, the attacker may make progress in gaining access to the victim system (e.g., discover the vulnerable services, install a Trojan horse program) as a result of an attack. Informally, we call the possible outcome of an attack the (possible) *consequence* of the attack. In a series of attacks where the attackers launch earlier attacks to prepare for later ones, there are usually strong connections between the consequences of the earlier attacks and the prerequisites of the later ones. Indeed, if an earlier attack is to prepare for a later attack, the consequence of the earlier attack should at least partly satisfy the prerequisite of the later attack.

Accordingly, we propose to identify the prerequisites (e.g., existence of vulnerable services) and the consequences (e.g., discovery of vulnerable services) of each type of attack. These are then used to correlate alerts, which are attacks detected by IDSs, by matching the consequences of (the attacks corresponding to) some previous alerts and the prerequisites of (the attacks corresponding to) some later ones. For example, if we find a *Sadmind Ping* followed by a buffer overflow attack against the corresponding *Sadmind* service, we can correlate them to be parts of the same series of attacks. In other words, we model the knowledge (or state) of attackers in terms of individual attacks, and correlate alerts if they indicate the progress of attacks.

Note that an attacker does not *have to* perform early attacks to prepare for a later attack, even though the later attack has certain prerequisites. For example, an attacker may launch an individual buffer overflow attack against a service blindly, without knowing if the service exists. In other words, the prerequisite of an attack should not be mistaken for the necessary existence of an earlier attack. However, if the attacker does launch attacks with earlier ones preparing for later ones, our method can correlate them, provided that the attacks are detected by IDSs.

In the following subsections, we adopt a formal approach to develop our alert correlation method.

3.1 Prerequisite and Consequence of Attacks

We propose to use predicates as basic constructs to represent the prerequisites and (possible) consequences of attacks. For example, a scanning attack may discover UDP services vulnerable to a certain buffer overflow attack. We can use the predicate *UDPVulnerableToBOF* (*VictimIP, VictimPort*) to represent the attacker's discovery (i.e., the consequence of the attack) that the host having the IP address *VictimIP* runs a service (e.g., *sadmind*) at UDP port *VictimPort* and that the service is vulnerable to the buffer overflow attack. Similarly, if an attack requires a UDP service vulnerable to the buffer overflow attack, we can use the same predicate to represent the prerequisite.

Some attacks may require several conditions be satisfied at the same time in order to be successful. To represent such complex conditions, we use a logical combination of predicates to describe the prerequisite of an attack. For example, a certain network launched buffer overflow attack may require that the target host have a vulnerable UDP service accessible to the attacker through the firewall. This prerequisite can be represented by *UDPVulnerableToBOF* (*VictimIP, VictimPort*) $\wedge$ *UDPAccessibleViaFirewall* (*VictimIP, VictimPort*). To simplify the following discussion, we restrict the logical operators in predicates to $\wedge$ (conjunction) and $\vee$ (disjunction).

We also use a set of predicates to represent the (possible) consequence of an attack. For example, an attack may result in compromise of the root privilege as well as modification of the *.rhost* file. Thus, we may use the following to represent the corresponding consequence: {*GainRootAccess* (*VictimIP*), *rhostModified* (*VictimIP*)}. Note that the set of predicates used to represent the consequence is essentially the logical combination of these predicates and can be represented by a single logical formula. However, representing the consequence as a set of predicates rather than a long formula is more convenient and will be used here.

The consequence of an attack is indeed the *possible* result of the attack. In other words, the attack may or may not generate the stated consequence. For example, after a buffer overflow attack against a service, an attacker may or may not gain the root access, depending on if the service is vulnerable to the attack.

We use *possible* consequences instead of *actual* consequences due to the following two reasons. First, an IDS may not have enough information to decide if an attack is effective or not. For example, a network based IDS can detect certain buffer overflow attacks by matching the patterns of the attacks; however, it cannot decide whether the attempts succeed or not without more information from the related hosts. Thus, it may not be practical to correlate alerts using the actual consequences of attacks. In contrast, the possible consequence of a type of attack can be analyzed and made available for the IDS. Second, even if an attack fails to prepare for the follow-up attacks, the follow-up attacks may still occur simply because, for example, the attacker wants to test the success of the previous attack, or the attacker uses a script to launch a series of attacks. Using possible consequences of attacks will lead to better opportunity to correlate such attacks.

For the sake of brevity, we refer to a *possible consequence* simply as a *consequence* throughout this paper.

3.2 Hyper-alert Type and Hyper-alert

Using predicates as the basic construct, we introduce the notion of a *hyper-alert type* to represent the prerequisite and the consequence of each type of alert.

Definition 3.1. 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*, and (3) *consequence* is a set of predicates such that all the free variables in *consequence* are in *fact*.

Each hyper-alert type encodes the knowledge about a type of attack. The component *fact* of a hyper-alert type tells what kind of information is reported along with the alert (i.e., detected attack), *prerequisite* specifies what must be true in order for the attack to be successful, and *consequence* describes what could be true if the attack indeed succeeds. For the sake of brevity, we omit the domains associated with the attribute names when they are clear from the context.

Example 3.2. Consider the buffer overflow attack against the *sadmind* remote administration tool. We may have a hyper-alert type *SadmindBufferOverflow* = ({*VictimIP, VictimPort*}, *ExistHost* (*VictimIP*) $\wedge$ *VulnerableSadmind* (*VictimIP*), {*GainRootAccess*(*VictimIP*)}) for such attacks. Intuitively, this hyper-alert type says that such an attack is against the host at IP address *VictimIP*. (We expect the actual values of *VictimIP* are reported by an IDS.) For the attack to be successful, there must exist a host at IP address *VictimIP*, and the corresponding *sadmind* service must be vulnerable to buffer overflow attacks. The attacker may gain root privilege as a result of the attack.

Given a hyper-alert type, a *hyper-alert instance* can be generated if the corresponding attack is detected and reported by an IDS. For example, we can generate a hyper-alert instance of type *SadmindBufferOverflow* from a corresponding alert. The notion of hyper-alert instance is formally defined as follows:

Definition 3.3. Given a hyper-alert type T = (*fact, prerequisite, consequence*), a *hyper-alert (instance)* h *of type* T 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 predicates in *consequence* might evaluate to True for each of the tuples. (Notation-wise, for each tuple t in h, we use *t.begin_time* and *t.end_time* to refer to the timestamp associated with t.)

The *fact* component of a hyper-alert type is essentially a relation schema (as in relational databases), and a hyper-alert is a relation instance of this schema. One may point out that an alternative way is to represent a hyper-alert as a record, which is equivalent to a single tuple on *fact*. However, such an alternative cannot accommodate certain alerts possibly reported by an IDS. For example, an IDS may report an IPSweep attack along with multiple swept IP addresses, which cannot be represented as a single record. In addition, our current formalism allows aggregation of alerts of the same type, and is flexible in reasoning about alerts. Therefore, we believe the current notion of a hyper-alert is an appropriate choice.

A hyper-alert instantiates its *prerequisite* and *consequence* by replacing the free variables in *prerequisite* and *consequence* with its specific values. Since all free variables in *prerequisite* and *consequence* must appear in *fact* in a hyper-alert type, the instantiated prerequisite and consequence will have no free variables. Note that *prerequisite* and *consequence* can be instantiated multiple times if *fact* consists of multiple tuples. For example, if an IPSweep attack involves several IP addresses, the *prerequisite* and *consequence* of the corresponding hyper-alert type will be instantiated for each of these addresses.

In the following, we treat timestamps implicitly and omit them if they are not necessary for our discussion.

Example 3.4. Consider the hyper-alert type *SadmindBufferOverflow* in example 3.2. There may be a hyper-alert $h_{SadmindBOF}$ as follows: {(*VictimIP* = 152.1.19.5, *VictimPort* = 1235), (*VictimIP* = 152.1.19.7, *VictimPort* = 1235)}. This implies that if the attack is successful, the following two logical formulas must be True as the prerequisites of the attack: *ExistHost* (152.1.19.5) ∧ *VulnerableSadmind* (152.1.19.5), *ExistHost* (152.1.19.7) ∧ *VulnerableSadmind* (152.1.19.7). Moreover, as possible consequences of the attack, the following might be True: *GainRootAccess* (152.1.19.5), *GainRootAccess* (152.1.19.7). This hyper-alert says that there are buffer overflow attacks against *sadmind* at IP addresses 152.1.19.5 and 152.1.19.7, and the attacker may gain root access as a result of the attacks.

A hyper-alert may correspond to one or several related alerts. If an IDS reports one alert for a certain attack and the alert has all the information needed to instantiate a hyper-alert, a hyper-alert can be generated from the alert. However, some IDSs may report a series of alerts for a single attack. For example, EMERALD may report several alerts (within the same thread) related to an attack that spreads over a period of time. In this case, a hyper-alert may correspond to the aggregation of all the related alerts. Moreover, several alerts may be reported for the same type of attack in a short period of time. Our definition of hyper-alert allows them to be treated as one hyper-alert, and thus provides flexibility in the reasoning about alerts. Certain constraints are necessary to make sure the hyper-alerts are reasonable. However, since our hyper-alert correlation method does not depend on them directly, we will discuss them after introducing our method.

Ideally, we may correlate a set of hyper-alerts with a later hyper-alert if the consequences of the former ones imply the prerequisite of the latter one. However, such an approach may not work in reality due to several reasons. First, the attacker may not always prepare for certain attacks by launching some other attacks. For example, the attacker may learn a vulnerable *sadmind* service by talking to people who work in the organization where the system is running. Second, the current IDSs may miss some attacks, and thus affect the alert correlation if the above approach is used. Third, due to the combinatorial nature of the aforementioned approach, it is computationally expensive to examine sets of alerts to find out whether their consequences imply the prerequisite of an alert.

Having considered these issues, we adopt an alternative approach. Instead of examining if several hyper-alerts imply the prerequisite of a later one, 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 individual predicates and test whether the consequence of an earlier hyper-alert makes some of the prerequisites True (i.e., make the prerequisite easier to satisfy). If the result is yes, then we correlate the hyper-alerts together. This idea is specified formally through the following Definitions.

Definition 3.5. Consider a hyper-alert type $T = (\textit{fact, prerequisite, consequence})$. The *prerequisite set (or consequence set, resp.) of* T, denoted $P(T)$ (or $C(T)$, resp.), is the set of all predicates that appear in *prerequisite* (or *consequence*, resp.). Given a hyper-alert instance h of type T, the *prerequisite set (or consequence set, resp.) of* h, denoted $P(h)$ (or $C(h)$, resp.), is the set of predicates in $P(T)$ (or $C(T)$, resp.) whose arguments are replaced with the corresponding attribute values of each tuple in h. Each element in $P(h)$ (or $C(h)$, resp.) is associated with the timestamp of the corresponding tuple in h. (Notation-wise, for each $p \in P(h)$ (or $C(h)$, resp.), we use $p.begin\_time$ and $p.end\_time$ to refer to the timestamp associated with p.)

Example 3.6. Consider the *Sadmind Ping* attack through which an attacker discovers possibly vulnerable *sadmind* services. The corresponding alerts can be represented by a hyper-alert type *SadmindPing* = ({*VictimIP*, *VictimPort*}, {*ExistHost* (*VictimIP*)}, {*VulnerableSadmind* (*VictimIP*)}).

Suppose a hyper-alert instance $h_{SadmindPing}$ of type *SadmindPing* has the following tuples: {(*VictimIP* = 152.1.19.5, *VictimPort* = 1235), (*VictimIP* = 152.1.19.7, *VictimPort* = 1235), (*VictimIP* = 152.1.19.9, *VictimPort* = 1235)}. Then we have the prerequisite set $P(h_{SadmindPing})$ = {*ExistHost* (152.1.19.5), *ExistHost* (152.1.19.7), *ExistHost* (152.1.19.9)}, and the consequence set $C(h_{SadmindPing})$ = {*VulnerableSadmind* (152.1.19.5), *VulnerableSadmind* (152.1.19.7), *VulnerableSadmind* (152.1.19.9)}.

Example 3.7. Consider the hyper-alert $h_{SadmindBOF}$ discussed in example 3.4. We have $P(h_{SadmindBOF})$ = {*ExistHost* (152.1.19.5), *ExistHost* (152.1.19.7), *VulnerableSadmind* (152.1.19.5), *VulnerableSadmind* (152.1.19.7)}, and $C(h_{SadmindBOF})$ = {*GainRootAccess* (152.1.19.5), *GainRootAccess* (152.1.19.7)}.

Definition 3.8. 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 predicates in C implies p.

The prepare-for relation is developed to capture the causal relationships between hyper-alerts. Intuitively, h_1 prepares for h_2 if some attacks represented by h_1 make the attacks represented by h_2 easier to succeed.

Example 3.9. Let us continue examples 3.6 and 3.7. Assume 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 the instantiated predicate *VulnerableSadmind* (152.1.19.5) (among others) in $P(h_{SadmindBOF})$ is also in $C(h_{SadmindPing})$. Thus, $h_{SadmindPing}$ prepares for, and should be correlated with $h_{SadmindBOF}$.

Given a sequence S of hyper-alerts, a hyper-alert h in S is a *correlated hyper-alert*, if there exists another hyper-alert h' in S such that either h prepares for h' or h' prepares for h. If no such h' exists, h is called an *isolated hyper-alert.* The goal of the correlation process is to discover all pairs of hyper-alerts h_1 and h_2 in S such that h_1 prepares for h_2.

3.2.1 *Temporal Constraints for Hyper-alerts.*

As discussed earlier, we allow multiple alerts to be aggregated into a hyper-alert, which gives some flexibility in reasoning about alerts. However, the definition of hyper-alert is overly flexible in some situations; it allows alerts that occur at arbitrary points in time to be treated as a single hyper-alert. Although some attackers usually spread their intrusive activities over time, aggregating alerts at arbitrary time points is still overly broad, and may affect the effectiveness of alert correlation.

In the following, we introduce two temporal constraints for hyper-alerts. The purpose of these temporal constraints is to restrict the alert aggregation to meaningful ones. We are particularly interested in hyper-alerts that satisfy at least one of the constraints. However, most of our discussion in this paper applies to general hyper-alerts. Thus, we will not specifically indicate the constraints if it is not necessary.

Definition 3.10. Given a time duration D (e.g., 100 seconds), a hyper-alert h satisfies *duration constraint of* D if $Max\{t.end\_time|\forall t \in h\} - Min\{t.begin\_time|\forall t \in h\} < D$.

Definition 3.11. Given a time interval I (e.g., 10 seconds), a hyper-alert h satisfies *interval constraint of* I if (1) h has only one tuple, or (2) for all r in h, there exist another r' in h such that there exist $r.begin\_time < T < r.end\_time$, $r'.begin\_time < T' < r'.end\_time$, and $|T - T'| < I$.

Intuitively, a hyper-alert satisfies a duration constraint of D if all contributing alerts occur during a period of duration D. Similarly, a hyper-alert satisfies an interval constraint of I if the gap (in time) between two consecutive contributing alerts never exceeds I.

The temporal constraints are introduced to prevent unreasonable aggregation of alerts. However, this does not imply that alerts have to be aggregated. Indeed, in our initial experiments, we treat each alert as an individual hyper-alert. In other words, aggregation of alerts is an option provided by our model, and temporal constraints are restrictions that make the aggregated hyper-alerts meaningful.

3.3 Hyper-alert Correlation Graph

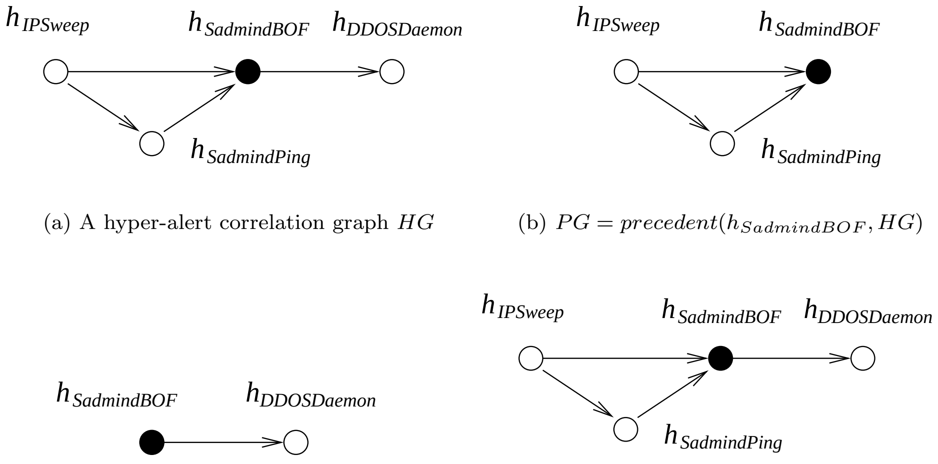
The prepare-for relation between hyper-alerts provides a natural way to represent the causal relationship between correlated hyper-alerts. In the following, we introduce the notion of a *hyper-alert correlation graph* to represent attack scenarios on the basis of the prepare-for relation. As we will see, the hyper-alert correlation graph reflects the high-level strategies or logical steps behind a sequence of attacks.

Definition 3.12. A *hyper-alert correlation graph* $HG = (N,\ E)$ is a connected DAG (directed acyclic graph), where the set N of nodes is a set of hyper-alerts, and for each pair of nodes $n_1, n_2 \in N$, there is an edge from n_1 to n_2 in E if and only if n_1 prepares for n_2.

Example 3.13. Suppose in a sequence of hyper-alerts we have the following ones: $h_{IPSweep}$, $h_{SadmindPing}$, $h_{SadmindBOF}$, and $h_{DDOSDaemon}$. The hyper-alerts $h_{SadmindBOF}$ and $h_{SadmindPing}$ have been explained in examples 3.4 and 3.6, respectively. Suppose $h_{IPSweep}$ represents an IP Sweep attack, and $h_{DDOSDaemon}$ represents the activity of a DDOS daemon program. Assume we have: $h_{IPSweep}$ prepares for $h_{SadmindPing}$ and $h_{SadmindBOF}$, respectively, $h_{SadmindPing}$ prepares for $h_{SadmindBOF}$, and $h_{SadmindBOF}$ prepares for $h_{DDOSDaemon}$. These are intuitively shown in a hyper-alert correlation graph in Figure 1(a).

The hyper-alert correlation graph provides an intuitive representation of correlated hyper-alerts. With this notion, the goal of the alert correlation process can be rephrased as the discovery of hyper-alert correlation graphs that have maximum number of nodes from a sequence of hyper-alerts.

In addition to getting all the correlated hyper-alerts, it is often desirable to discover those that are directly or indirectly correlated to one particular hyper-alert. For example, if an IDS detects a DDOS daemon running on a host, it would



(a) A hyper-alert correlation graph HG

(b) $PG = precedent(h_{SadmindBOF}, HG)$

(c) $SG = subsequent(h_{SadmindBOF}, HG)$

(d) $CG = correlated(h_{SadmindBOF}, HG)$

Fig. 1. Hyper-alert correlation graphs

be helpful to inform the administrator how this happened, that is, report all the alerts that directly or indirectly prepare for the DDOS daemon. Therefore, we define the following operations on hyper-alert correlation graphs.

Definition 3.14. 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 maximum sub-graph $PG = (N',\ E')$ of HG that satisfies the following conditions: (1) $n \in N'$, (2) for each $n' \in N'$ other than n, there is a directed path from n' to n, and (3) each edge $e \in E'$ 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.

Definition 3.15. 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) $n \in N'$, (2) for each $n' \in N'$ other than n, there is a directed path from n to n', and (3) each edge $e \in E'$ is in 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.

Definition 3.16. 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 maximum sub-graph $CG = (N',\ E')$ of HG that satisfies the following conditions: (1) $n \in N'$, (2) for each $n' \in N'$ other than n, there is either a path from n to n', or a path from n' to n, and (3) each edge $e \in E'$ is either in a path from a 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.

Intuitively, the precedent graph of n w.r.t. HG describes all the hyper-alerts in HG that prepare for n directly or indirectly, the subsequent graph of n w.r.t. HG

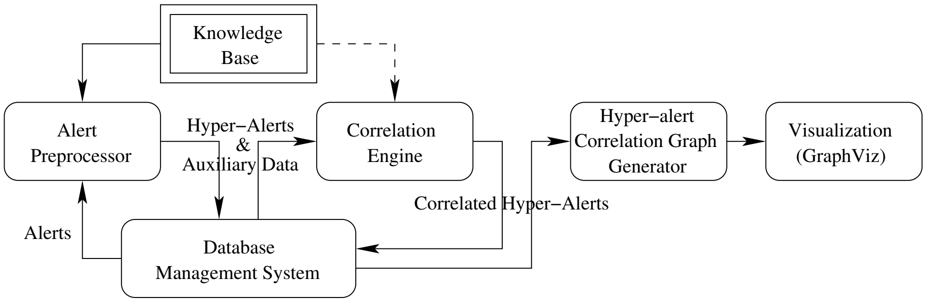


Fig. 2. The architecture of the intrusion alert correlator

describes all the hyper-alerts in HG for which n prepares directly or indirectly, and the correlated graph of n w.r.t. HG includes all the hyper-alerts in HG that are correlated to n directly or indirectly. It is easy to see that $correlated(n, HG) = precedent(n, HG) \cup subsequent(n, HG)$.

Assuming the black node $h_{SadmindBOF}$ in figure 1(a) is the hyper-alert of concern, figures 1(b) to 1(d) display the precedent graph, subsequent graph, and correlated graph of $h_{SadmindBOF}$ w.r.t. the hyper-alert correlation graph in figure 1(a), respectively. Note that figure 1(d) is the same as figure 1(a). This is because all the hyper-alerts in figure 1(a) are related to $h_{SadmindBOF}$ via the prepare-for relation. In reality, it is certainly possible that not all hyper-alerts are related to the hyper-alert of concern. In this case the correlated graph only reveals those directly or indirectly correlated to that hyper-alert.

The hyper-alert correlation graph is not only an intuitive way to represent attack scenarios constructed through alert correlation, but also reveals opportunities to improve intrusion detection. First, the hyper-alert correlation graph can potentially reveal the intrusion strategies behind the attacks, and thus lead to better understanding of the attacker's intention. Second, assuming some attackers exhibit patterns in their attack strategies, we can use the hyper-alert correlation graph to profile previous attacks and thus identify on-going attacks by matching to the profiles. A partial match to a profile may indicate attacks possibly missed by the IDSs, and thus lead to human investigation and improvement of the IDSs. Nevertheless, additional research is necessary to demonstrate the usefulness of hyper-alert correlation graphs for this purpose.

3.4 Implementation

We have implemented an off-line intrusion alert correlator using the method discussed earlier. Figure 2 shows the architecture. It consists of a knowledge base, an alert preprocessor, a correlation engine, a hyper-alert correlation graph generator, and a visualization component. All these components except for the visualization component interact with a DBMS, which provides persistent storage for the intermediate data as well as the correlated alerts. The program was written in Java, with JDBC to access the database. To save development effort, we use the GraphViz package [AT & T Research Labs] as the visualization component.

The knowledge base contains the necessary information about hyper-alert types as well as implication relationships between predicates. In our current implementation, the hyper-alert types and the relationship between predicates are specified in an XML file. When the alert correlator is initialized, it reads the XML file, and then converts and stores the information in the knowledge base.

Our current implementation assumes the alerts reported by IDSs are stored in the database. Using the information in the knowledge base, the alert preprocessor generates hyper-alerts as well as auxiliary data from the original alerts. The correlation engine then performs the actual correlation task using the hyper-alerts and the auxiliary data. After alert correlation, the hyper-alert correlation graph generator extracts the correlated alerts from the database, and generates the graph files in the format accepted by GraphViz. As the final step of alert correlation, GraphViz is used to visualize the hyper-alert correlation graphs.

3.4.1 *Preprocessing and Correlation of Hyper-alerts.* Preprocessing and correlation of hyper-alerts are the major tasks of the alert correlator. These tasks are performed using the hyper-alert type information stored in the knowledge base. As discussed earlier, the knowledge base stores two types of information: the implication relationships between predicates and the hyper-alert type information. When the alert correlator reads in the hyper-alert types, it generates the prerequisite and consequence sets of each hyper-alert type. In addition, it expands the consequence set of each hyper-alert type by including all the predicates in the knowledge base implied by the consequence set. We call the result the *expanded consequence set* of the corresponding hyper-alert type. (Similar to the consequence set of a hyper-alert type, we may instantiate the expanded consequence set with a hyper-alert instance and get the expanded consequence set of the hyper-alert.)

To simplify the preprocess and correlation of hyper-alerts, we make the following assumptions.

Assumption 3.17. Given a set $P = \{p_1(x_{11}, ..., x_{1k_1}), ..., p_m(x_{m1}, ..., x_{mk_m})\}$ of predicates, for any set of instantiations of the variables $x_{11}, ..., x_{1k_1}, ..., x_{m1}, ..., x_{mk_m}$, deriving all predicates implied by P followed by instantiating all the variables leads to the same result as instantiating all the variables and then deriving all the predicates implied by the instantiated predicates.

Assumption 3.18. All predicates are uniquely identified by their names and the special characters "(", ")", and "," do not appear in predicate names and arguments.

The major preparation for alert correlation is performed at the preprocessing phase. The alert preprocessor generates hyper-alerts from the alerts reported by IDSs and instantiates the prerequisite and expanded consequence sets of each hyper-alert. The current implementation generates one hyper-alert from each alert, though our method allows aggregating multiple alerts into one hyper-alert. Note that having the expanded consequence set of a hyper-alert, we can test if some predicates in the consequence set imply a predicate by checking whether the latter predicate is included in the expanded consequence set.

We encode instantiated predicates as strings, and thus further transform the alert correlation problem to a string matching problem. Specifically, each instantiated predicate is encoded as the predicate name followed by the character "(", followed by the sequence of arguments separated with the character ",", and finally followed by the character ")". Thus, under assumption 3.18 and the fact that the order of arguments in a predicate is significant, comparing instantiated predicates is equivalent to comparing the encoded strings.

We store the encoded prerequisite and expanded consequence sets in two tables, *PrereqSet* and *ExpandedConseqSet*, along with the corresponding hyper-alert ID and timestamp, assuming that each hyper-alert is uniquely identified by its ID. (Note that one hyper-alert may have multiple tuples in these tables.) Both tables have attributes *HyperAlertID*, *EncodedPredicate*, *begin_time*, and *end_time*, with meanings as indicated by their names. As a result, alert correlation can be performed using the following SQL statement.

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

The correctness of our implementation method is guaranteed by the following theorem.

THEOREM 3.19. *Under assumptions 3.17 and 3.18, our implementation method discovers all and only the hyper-alert pairs such that the first one of the pair prepares for the second one.*

PROOF. Consider a pair of hyper-alerts h_1 and h_2 such that h_1 prepares for h_2. By Definition 3.8, there exists $p \in P(h_2)$ and $C \subseteq C(h_1)$ such that for all $c \in C$, $c.end\_time < p.begin\_time$ and the conjunction of all predicates in C implies p. By assumption 3.17, p should be in the expanded consequence set of h_1 (but associated with a different timestamp). Thus, both *PrereqSet* and *ExpandedConseqSet* have a tuple that has the same encoded predicate along with the appropriate hyper-alert ID and timestamp. As a result, the SQL statement will output that h_1 prepares for h_2.

Suppose the SQL statement outputs that h_1 prepares for h_2. Then there exist t_1 in *ExpandedConseqSet* and t_2 in *PrereqSet* such that $t_1.EncodedPredicate = t_2.EncodedPredicate$ and $t_1.end\_time < t_2.begin\_time$. According to assumption 3.18, $t_1.EncodedPredicate$ and $t_2.EncodedPredicate$ must be the same instantiated predicate. Let us refer to this predicate as p_1 when it is in the expanded consequence set of h_1, and as p_2 when it is in $P(h_2)$. Thus, $p_1 = p_2$ and $p_1.end\_time < p_2.begin\_time$. If p_1 is also in $C(h_1)$, let $C = \{p_1\}$. Otherwise, let C be the set of predicates in $C(h_1)$ that are instantiated by the same tuple in h_1 as p_1. By the way in which an expanded consequence set is constructed and assumption 3.17, p_1 is implied by the predicates in C. This is to say that p_2 is implied by $C \subseteq C(h_1)$ such that for all $c \in C$, $c.end\_time < p_2.begin\_time$. Thus, h_1 prepares for h_2 by Definition 3.8. □

4. UTILITIES FOR ANALYZING INTENSIVE ALERTS

Our experiments demonstrate that the alert correlation method is effective in analyzing small amount of alerts. However, our experience with intrusion intensive datasets (e.g., the DEFCON 8 CTF dataset [DEFCON 2000]) has revealed several problems.

First, let us consider the following scenario. Suppose an IDS detected an *SadmindPing* attack, which discovered the vulnerable *Sadmind* service on host V, and later an *SadmindBufferOverlfow* attack against the *Sadmind* service. Assuming that they were launched from different hosts, should we correlate them? On the one hand, it is possible that one or two attackers coordinated these two attacks from two different hosts, trying to avoid being correlated. On the other hand, it is also possible that these attacks belonged to two separate efforts. Such a scenario clearly introduces a dilemma, especially when there are a large amount of alerts.

One may suggest to use time to solve this problem. For example, we may correlate the aforementioned attacks if they happened within t seconds. However, knowing this method, an attacker may introduce delays between attacks to bypass correlation.

The second problem is the overwhelming information encoded by hyper-alert correlation graphs when intensive attacks trigger a large amount of alerts. Our initial attempt to correlate the alerts generated for the DEFCON 8 CTF dataset [DEFCON 2000] resulted in 450 hyper-alert correlation graphs, among which the largest hyper-alert correlation graph consists of 2,940 nodes and 25,321 edges. Such a graph is clearly too big for a human user to comprehend in a short period of time.

Although the DEFCON 8 dataset involves intensive attacks not usually seen in normal network traffic, the actual experience of intrusion detection practitioners indicates that "encountering 10-20,000 alarms per sensor per day is common [Manganaris et al. 2000]." Thus, it is necessary to develop techniques or tools to deal with the overwhelming information.

In this section, we propose three utilities, mainly to address the second problem. Regarding the first problem, we choose to correlate the alerts when it is possible, leaving the final decision to the user. We would like to clarify that these utilities are intended for human users to analyze alerts interactively, not for computer systems to draw any conclusion automatically, though some of the utilities may be adapted for automatic systems. These utilities are summarized as follows.

(1) *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. The graph reduction is adjustable in the sense that users are allowed to control the degree of reduction.

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

(3) *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. This can be considered to combine a variation of the method proposed in [Valdes and Skinner 2001] with our method.

4.1 Adjustable Reduction of Hyper-alert Correlation Graphs

A natural way to reduce the complexity of a hyper-alert correlation graph is to reduce the number of nodes and edges. However, to make the reduced graph useful, any reasonable reduction should maintain the structure of the corresponding

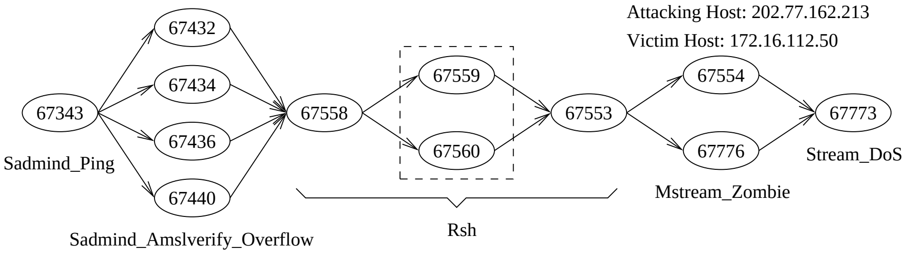


Fig. 3. A hyper-alert correlation graph discovered in the 2000 DARPA intrusion detection evaluation datasets

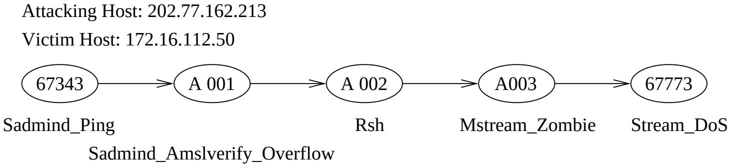

Attacking Host: 202.77.162.213
Victim Host: 172.16.112.50
67343
A 001
A 002
A003
67773
Sadmind_Ping
Sadmind_Amslverify_Overflow
Rsh
Mstream_Zombie
Stream_DoS

Fig. 4. A hyper-alert correlation graph reduced from Fig. 3

.

attacks.

We propose to aggregate hyper-alerts of the same type to reduce the number of nodes in a hyper-alert correlation graph. Due to the flexible definition of hyper-alerts, the result of hyper-alert aggregation will remain valid hyper-alerts. For example, in Figure 3, hyper-alerts 67432, 67434, 67436, and 67440 are all instances of hyper-alert type *Sadmind_Amslverify_Overflow*. Thus, we may aggregate them into one hyper-alert. As another example, hyper-alerts 67558, 67559, 67560, and 67553 are all instances of *Rsh*, and can be aggregated into a single hyper-alert.

Edges are reduced along with the aggregation of hyper-alerts. In Figure 3, the edges between the *Rsh* hyper-alerts are subsumed into the aggregated hyper-alert, while the edges between the *Sadmind_Ping* hyper-alert and the four *Sadmind_Amslverify_Overflow* hyper-alerts are merged into a single edge. As a result, we have a reduced hyper-alert correlation graph as shown in Figure 4.

Reduction of a hyper-alert correlation graph may lose information contained in the original graph. Indeed, hyper-alerts that are of the same type but belong to different sequences of attacks may be aggregated and thus provide overly simplified results. Nevertheless, our goal is to lose as little information of the structure of attacks as possible.

Depending on the actual alerts, the reduction of a hyper-alert correlation graph may be less simplified so that there is too much detail in the resulting graph, or overly simplified so that some structures are hidden. We would like to give a human user more control over the graph reduction process.

We allow hyper-alert aggregation only when the resulting hyper-alerts satisfy an interval constraint of a given threshold I. Intuitively, we allow hyper-alerts to be aggregated only when they are close to each other in time. The larger a threshold

I is, the more a hyper-alert correlation graph can be reduced. By adjusting the interval threshold, a user can control the degree to which a hyper-alert correlation graph is reduced.

4.2 Focused Analysis

Focused analysis is implemented on the basis of focusing constraints. A *focusing constraint* is a logical combination of comparisons between attribute names and constants. (In our work, we restrict logical operations to AND ($\wedge$), OR ($\vee$), and NOT ($\neg$).) For example, we may have a focusing constraint $SrcIP = 129.174.142.2 \vee DestIP = 129.174.142.2$. We say a focusing constraint C_f is *enforceable w.r.t. a hyper-alert type* T if when we represent C_f in a disjunctive normal form, at least for one disjunct C_{fi}, all the attribute names in C_{fi} appear in T. For example, the above focusing constraint is enforceable w.r.t. $T = (\{SrcIP, SrcPort\}, NULL, \emptyset)$, but not w.r.t. $T' = (\{VictimIP, VictimPort\}, NULL, \emptyset)$. Intuitively, a focusing constraint is enforceable w.r.t. T if it can be evaluated using a hyper-alert instance of type T.

We may *evaluate* a focusing constraint C_f with a hyper-alert h if C_f is enforceable w.r.t. the type of h. A focusing constraint C_f evaluates to True for h if there exists a tuple $t \in h$ such that C_f is True with the attribute names replaced with the values of the corresponding attributes of t; otherwise, C_f evaluates to False. For example, consider the aforementioned focusing constraint C_f, which is $SrcIP = 129.174.142.2 \vee DestIP = 129.174.142.2$, and a hyper-alert $h = \{(SrcIP = 129.174.142.2, SrcPort = 80)\}$, we can easily have that $C_f =$ True for h.

The idea of focused analysis is quite simple: we only analyze the hyper-alerts with which a focusing constraint evaluates to True. In other words, we would like to filter out irrelevant hyper-alerts, and concentrate on analyzing the remaining hyper-alerts. We are particularly interested in applying focusing constraints to *atomic hyper-alerts*, i.e., hyper-alerts with only one tuple. In our framework, atomic hyper-alerts correspond to the alerts reported by an IDS directly.

Focused analysis is particularly useful when we have certain knowledge of the alerts, the systems being protected, or the attacking computers. For example, if we are interested in the attacks against a critical server with IP address *Server_IP*, we may perform a focused analysis using *DestIPAddress* = *Server_IP*. However, focused analysis cannot take advantage of the intrinsic relationship among the hyper-alerts (e.g., hyper-alerts having the same IP address). In the following, we introduce the third utility, graph decomposition, to fill in this gap.

4.3 Graph Decomposition Based on Hyper-alert Clusters

The purpose of graph decomposition is to use the inherent relationship between (the attributes of) hyper-alerts to decompose a hyper-alert correlation graph. Conceptually, we cluster the hyper-alerts in a large correlation graph based on the "common features" shared by hyper-alerts, and then decompose the original correlation graphs into subgraphs on the basis of the clusters. In other words, hyper-alerts should remain in the same graph only when they share certain common features.

We use a *clustering constraint* to specify the "common features" for clustering hyper-alerts. Given two sets of attribute names A_1 and A_2, a *clustering constraint*

$C_c(A_1, A_2)$ is a logical combination of comparisons between constants and attribute names in A_1 and A_2. (In our work, we restrict logical operations to AND ($\wedge$), OR ($\vee$), and NOT ($\neg$).) A clustering constraint is a constraint for two hyper-alerts; the attribute sets A_1 and A_2 identify the attributes from the two hyper-alerts. For example, we may have two sets of attribute names $A_1 = \{SrcIP, DestIP\}$ and $A_2 = \{SrcIP, DestIP\}$, and $C_c(A_1, A_2) = (A_1.SrcIP = A_2.SrcIP) \wedge (A_1.DestIP = A_2.DestIP)$. Intuitively, this is to say two hyper-alerts should remain in the same cluster if they have the same source and destination IP addresses.

A clustering constraint $C_c(A_1, A_2)$ is *enforceable w.r.t. hyper-alert types* T_1 *and* T_2 if when we represent $C_c(A_1, A_2)$ in a disjunctive normal form, at least for one disjunct C_{ci}, all the attribute names in A_1 appear in T_1 and all the attribute names in A_2 appear in T_2. For example, the above clustering constraint is enforceable w.r.t. T_1 and T_2 if both of them have $SrcIP$ and $DestIP$ in the $fact$ component. Intuitively, a clustering constraint is enforceable w.r.t. T_1 and T_2 if it can be evaluated using two hyper-alerts of types T_1 and T_2, respectively.

If a clustering constraint $C_c(A_1, A_2)$ is enforceable w.r.t. T_1 and T_2, we can *evaluate* it with two hyper-alerts h_1 and h_2 that are of type T_1 and T_2, respectively. A clustering constraint $C_c(A_1, A_2)$ evaluates to True for h_1 and h_2 if there exists a tuple $t_1 \in h_1$ and $t_2 \in h_2$ such that $C_c(A_1, A_2)$ is True with the attribute names in A_1 and A_2 replaced with the values of the corresponding attributes of t_1 and t_2, respectively; otherwise, $C_c(A_1, A_2)$ evaluates to False. For example, consider the clustering constraint $C_c(A_1, A_2) : (A_1.SrcIP = A_2.SrcIP) \wedge (A_1.DestIP = A_2.DestIP)$, and hyper-alerts $h_1 = \{(SrcIP = 129.174.142.2, SrcPort = 1234, DestIP = 152.1.14.5, DestPort = 80)\}$, $h_2 = \{(SrcIP = 129.174.142.2, SrcPort = 65333, DestIP = 152.1.14.5, DestPort = 23)\}$, we can easily have that $C_c(A_1, A_2) = \text{True}$ for h_1 and h_2. For brevity, we write $C_c(h_1, h_2) = \text{True}$ if $C_c(A_1, A_2) = \text{True}$ for h_1 and h_2.

Our clustering method is very simple, with a user-specified clustering constraint $C_c(A_1, A_2)$. Two hyper-alerts h_1 and h_2 are in the same cluster if $C_c(A_1, A_2)$ evaluates to True for h_1 and h_2 (or h_2 and h_1). Note that $C_c(h_1, h_2)$ implies that h_1 and h_2 are in the same cluster, but the reverse is not true. This is because $C_c(h_1, h_2) \wedge C_c(h_2, h_3)$ implies neither $C_c(h_1, h_3)$ nor $C_c(h_3, h_1)$.

4.4 Discussion

The alert correlation method is developed to uncover the high-level strategies behind a sequence of attacks, not to replace the original alerts reported by an IDS. However, as indicated by our initial experiments (See Section 5), alert correlation does provide evidence to differentiate between alerts. If an alert is correlated with some others, it is more possible that the alert corresponds to an actual attack.

It is desirable to develop a technique which can comprehend a hyper-alert correlation graph and generate feedback to direct intrusion detection and response processes. We consider such a technique a part of our future research plan. However, given the current status of intrusion detection and response techniques, it is also necessary to allow human users to understand the attacks and take appropriate actions.

The three utilities developed in this section are intended to help human users analyze attacks behind large amounts of alerts. They can make attack strategies behind intensive alerts easier to understand, but cannot improve the performance of alert correlation.

5. EXPERIMENTAL RESULTS

In this section, we report the experiments used to validate our techniques. Due to the space limit, we can only report selected results in this paper. For complete and detailed experimental results, please refer to [Cui 2002].

We have performed two sets of experiments. The first set of experiments was aimed at evaluating the effectiveness of our method in constructing attack scenarios and its ability to differentiate true and false alerts[1]. These experiments were done with the 2000 DARPA intrusion detection scenario specific datasets [MIT Lincoln Lab 2000]. The second set of experiments was intended to evaluate the usefulness of the three analysis utilities in dealing with large collections of intrusion alerts. For the second set of experiments, we chose the dataset collected at the DEFCON 8 CTF event [DEFCON 2000], which contains intensive attacks launched by competing hackers during the contest.

In each experiment, we replayed selected network traffic with NetPoke[2] in an isolated network monitored by a RealSecure Network Sensor 6.0 [Internet Security Systems]. RealSecure was chosen because it has an extensive set of well documented attack signatures. In all the experiments, the Network Sensor was configured to use the *Maximum_Coverage* policy with a slight change, which forced the Network Sensor to save all the reported alerts. Our alert correlator was then used to process the alerts generated by RealSecure. The hyper-alert correlation graphs were visualized using the GraphViz package [AT & T Research Labs]. For the sake of readability, transitive edges are removed from the graphs.

We mapped each alert type reported by RealSecure to a hyper-alert type with the same name. The prerequisite and consequence of each hyper-alert type were specified according to the descriptions of the attack signatures provided with the RealSecure Network Sensor 6.0. The collection of predicates, the implication relationships between these predicates, and the hyper-alert types are given in [Cui 2002].

5.1 Experiments with the DARPA 2000 Datasets

The 2000 DARPA intrusion detection scenario specific datasets include LLDOS 1.0 and LLDOS 2.0.2 [MIT Lincoln Lab 2000]. LLDOS 1.0 contains a series of attacks in which an attacker probes, breaks in, installs the components necessary to launch a Distributed Denial of Service (DDOS) attack, and actually launches a DDOS attack against an off-site server. LLDOS 2.0.2 includes a similar sequence of attacks run by an attacker who is a bit more sophisticated than the first one. Each dataset includes the network traffic collected from both the DMZ and the inside part of the evaluation network. We have performed four sets of experiments, each with either the DMZ or the inside network traffic of one dataset.

[1] An alert is true if it is raised because the IDS detects an actual attack; otherwise, it is false.

[2] NetPoke is a utility to replay packets to a live network that were previously captured with the tcpdump program. `http://www.ll.mit.edu/IST/ideval/tools/tools_index.html`

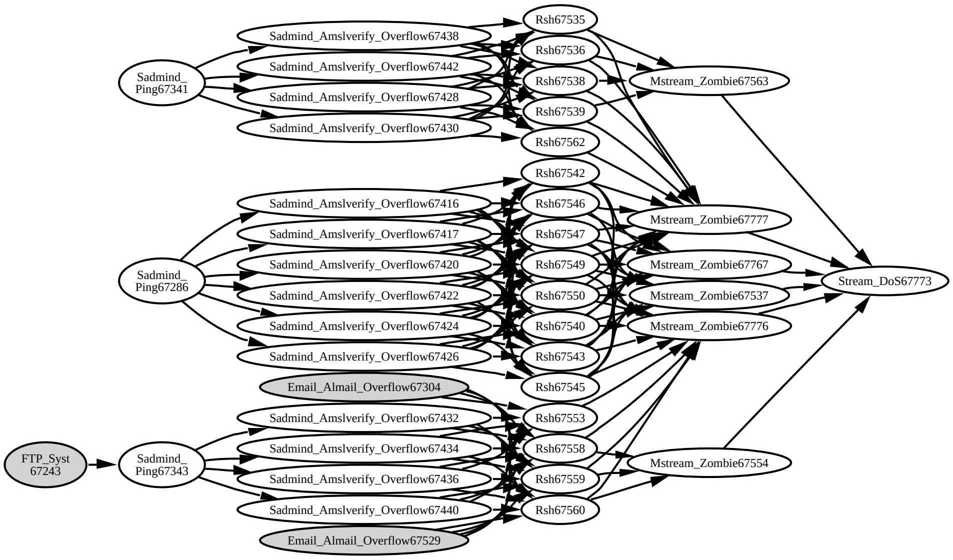

Sadmind_
Ping67341
Sadmind_Amslverify_Overflow67438
Sadmind_Amslverify_Overflow67442
Sadmind_Amslverify_Overflow67428
Sadmind_Amslverify_Overflow67430
Rsh67535
Rsh67536
Rsh67538
Rsh67539
Rsh67562
Mstream_Zombie67563
Sadmind_
Ping67286
Sadmind_Amslverify_Overflow67416
Sadmind_Amslverify_Overflow67417
Sadmind_Amslverify_Overflow67420
Sadmind_Amslverify_Overflow67422
Sadmind_Amslverify_Overflow67424
Sadmind_Amslverify_Overflow67426
Rsh67542
Rsh67546
Rsh67547
Rsh67549
Rsh67550
Rsh67540
Rsh67543
Rsh67545
Mstream_Zombie67777
Mstream_Zombie67767
Mstream_Zombie67537
Mstream_Zombie67776
Stream_DoS67773
Email_Almail_Overflow67304
FTP_Syst
67243
Sadmind_
Ping67343
Sadmind_Amslverify_Overflow67432
Sadmind_Amslverify_Overflow67434
Sadmind_Amslverify_Overflow67436
Sadmind_Amslverify_Overflow67440
Email_Almail_Overflow67529
Rsh67553
Rsh67558
Rsh67559
Rsh67560
Mstream_Zombie67554

Fig. 5. The (only) hyper-alert correlation graph discovered in the inside network traffic of LLDOS 1.0.

5.1.1 *Effectiveness of Alert Correlation.* Our first goal of these experiments is to evaluate the effectiveness of our method in constructing attack scenarios from alerts. Before discussing the quantitative measures, let us first look at one of the hyper-alert correlation graphs generated by the alert correlator.

Figure 5 shows the (only) hyper-alert correlation graph discovered from the inside network traffic in LLDOS 1.0. Each node in Figure 5 represents a hyper-alert. The text inside the node is the name of the hyper-alert type followed by the hyper-alert ID. (We will follow this convention for all the hyper-alert correlation graphs.)

There are 44 hyper-alerts in this graph, including 3 false alerts, which are shown in gray. We will discuss the false alerts later. The true hyper-alerts can be divided into five stages horizontally. The first stage consists of three *Sadmind_Ping* alerts, which the attacker used to find out the vulnerable *Sadmind* services. The three alerts are from source IP address 202.077.162.213, and to destination IP addresses 172.016.112.010, 172.016.115.020, and 172.016.112.050, respectively. The second stage consists of fourteen *Sadmind_Amslverify_Overflow* alerts. According to the description of the attack scenario, the attacker tried three different stack pointers and two commands in *Sadmind_Amslverify_Overflow* attacks for each victim host until one attempt succeeded. All the above three hosts were successfully broken into. The third stage consists of some *Rsh* alerts, with which the attacker installed and started the *mstream* daemon and master programs. The fourth stage consists of alerts corresponding to the communications between the DDOS master and daemon programs. Finally, the last stage consists of the DDOS attack.

We can see clearly that the hyper-alert correlation graph reveals the structure as well as the high-level strategy of the sequence of attacks. The other hyper-alert

Table I. Completeness and soundness of alert correlation.

	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%

correlation graphs and the corresponding analysis are included in [Cui 2002].

This hyper-alert correlation graph is still not perfect. Two *Email_Almail_Overflow* hyper-alerts (shown in gray in Figure 5) are false alerts, and are mis-correlated with the *Rsh* alerts, though it is possible that an attacker uses these attacks to gain access to the victim system and then copy the DDOS program with *Rsh*. The *FTP_Syst* hyper-alert is also a false one; it is correlated with one of the *Sadmind_Ping*s, because an attacker may use *FTP_Syst* to gain the OS information and then launch an *Sadmind_Ping* attack. Moreover, the attacker used a *telnet* as a part of the sequence of attacks, but this graph does not include the corresponding hyper-alert.

Another interesting issue is that we correlated alerts that are not attacks. In both DMZ and inside traffic of LLDOS 2.0.2, we correlated an *Email_Ehlo* with an *Email_Turn* from 135.013.216.191 to 172.016.113.105. Our further analysis indicated that these were normal and related activities between email servers.

To better understand the effectiveness of our method, we examine the *completeness* and the *soundness* of alert correlation. The completeness of alert correlation assesses how well we can correlate related alerts together, while the soundness evaluates how correctly the alerts are correlated. We introduce two simple measures, R_c and R_s, to quantitatively evaluate completeness and soundness, respectively:

$$R_c = \frac{\#correctly\ correlated\ alerts}{\#related\ alerts},\quad R_s = \frac{\#correctly\ correlated\ alerts}{\#correlated\ alerts}.$$

Counting the numbers in R_c and R_s is easy, given the description of the attacks in the DARPA datasets. However, RealSecure generated duplicate alerts for several attacks. In our experiments, we counted the duplicate alerts as different ones. False alerts are counted (as incorrectly correlated alerts) so long as they are correlated. Though non-intrusive alerts (e.g., the above *Email_Ehlo* and *Email_Turn*) are not attacks, if they are related activities, we counted them as correctly correlated ones.

Table I shows the results about completeness and soundness of the alert correlation for the two datasets. As shown by the values of R_s, most of the hyper-alerts are correlated correctly. The completeness measures (R_c) are satisfactory for LLDOS 1.0. However, they are only 62.5% and 66.7% for the DMZ and inside traffic in LLDOS 2.0.2. (Though the results for LLDOS 2.0.2 are much more informative than not having the correlation capability, they are not as desirable as the results for LLDOS 1.0.) Our further analysis reveals that all the hyper-alerts missed are those triggered by the *telnet*s that the attacker used to access a victim host. Each *telnet* triggered three alerts, *TelnetEnvAll, TelnetXDisplay* and *TelnetTerminalType*. According to RealSecure's description, these alerts are about attacks that are launched

Table II. Ability to differentiate true and false alerts. RS corresponds to the results directly obtained with RealSecure Network Sensor 6.5; Cor corresponds to results obtained after correlation.

dataset		attacks	tool	alerts	detected attacks	detection rate	true alerts	false alert rate
LLDOS 1.0	DMZ	89	RS	891	51	57.30%	57	93.60%
			Cor	57	50	56.18%	54	5.26%
	Inside	60	RS	922	37	61.67%	44	95.23%
			Cor	44	36	60%	41	6.82%
LLDOS 2.0.2	DMZ	7	RS	425	4	57.14%	6	98.59%
			Cor	5	3	42.86%	3	40%
	Inside	15	RS	489	12	80.00%	16	96.73%
			Cor	13	10	66.67%	10	23.08%

using environmental variables (*TelnetEnvAll*) in a telnet session, including XDisplay (*TelnetXDisplay*) and TerminalType (*TelnetTerminalType*). However, according to the description of the datasets, the attacker did not launch these attacks, though he did telnet to one victim host after gaining access to it. Nevertheless, to be conservative, we consider them as related alerts in our evaluation. Considering these facts, we can conclude that our method is effective for these datasets.

5.1.2 *Ability to Differentiate Alerts.* Our second goal of these experiments is to see how well alert correlation can be used to differentiate false alerts and true alerts. As we conjectured in Section 3, false alerts, which do not correspond to any real attacks, tend to be more random than the actual alerts, and are less likely to be correlated to others. If this conjecture is true, we can divert more resources to deal with correlated alerts, and thus improve the effectiveness of intrusion response.

To understand this issue, we deliberately drop the uncorrelated alerts and then compare the resulting detection rate and false alert rate with the original ones of RealSecure.

We counted the number of actual attacks and false alerts according to the description included in the datasets. False alerts can be identified easily by comparing the alerts with the attack description provided with the datasets; however, counting the number of attacks is subjective, since the number of attacks depends on how one views the attacks. Having different views of the attacks may result in different numbers.

We adopted the following way to count the number of attacks in our experiments. The initial phase of the attacks involved an IP Sweep attack. Though many packets were involved, we counted them as a single attack. Similarly, the final phase had a DDOS attack, which generated many packets but was also counted as one attack. For the rest of the attacks, we counted each action (e.g., *telnet, Sadmind_Ping*) initiated by the attacker as one attack. The numbers of attacks observable in these datasets are shown in Table II. Note that some activities such as *telnet* are not usually considered as attacks; however, we counted them here if the attacker used them as part of the attacks.

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

Table III. General statistics of the initial analysis

# total hyper-alert types	115	# total hyper-alerts	65054
# correlated hyper-alert types	95	# correlated	9744
# uncorrelated hyper-alert types	20	# uncorrelated	55310
# partially correlated hyper-alert types	51	% correlated	15%

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 $\frac{\#detected\ attacks}{\#observable\ attacks}$, while the false alert rates were computed as $(1 - \frac{\#true\ alerts}{\#alerts})$.

Table II summarizes the results of these experiments. For the DMZ network traffic in LLDOS 1.0, RealSecure generated 891 alerts. According to the description of the data set, 57 out of the 891 alerts are true alerts, 51 attacks are detected, and 38 attacks are missed. Thus, the detection rate of RealSecure Network Sensor is 57.30%, and the false alert rate is 93.60%.[3] Our intrusion alert correlator processed the alerts generated by the RealSecure Network Sensor. As shown in Table II, 57 alerts remain after correlation, 54 of them are true alerts, and 50 attacks are covered by these alerts. Thus, the detection rate and the false alert rate after alert correlation are 56.18% and 5.26%, respectively. The results for the other datasets are also shown in Table II.

The experimental results in Table II show that discarding uncorrelated alerts reduces the false alert rates greatly without sacrificing the detection rate too much. Thus, it is reasonable to treat correlated alerts more seriously than uncorrelated ones. However, simply discarding uncorrelated alerts is dangerous, since some of them may be true alerts, which correspond to individual attacks, or attacks our method fails to correlate.

5.2 Experiments with the DEFCON 8 CTF Dataset

It would be helpful for the evaluation of our method if we could identify false alerts, alerts for sequences of attacks, and alerts for isolated attacks. Unfortunately, due to the nature of the dataset, we are unable to obtain any of them. Thus, in this study, we focus on the analysis of the attack strategies reflected by hyper-alert correlation graphs, but only discuss the uncorrelated alerts briefly.

5.2.1 *Initial Attempt.* In our initial analysis of the DEFCON 8 CTF dataset, we tried to correlate the hyper-alerts without reducing the complexity of any hyper-alert correlation graphs. The statistics of the initial analysis are shown in Table III.

Table III shows that only 15% alerts generated by RealSecure are correlated. In addition, 20 out of 115 hyper-alert types that appear in this data set do not have any instances correlated. Among the remaining 95 hyper-alert types, 51 types have both correlated and uncorrelated instances.

Table IV shows the statistics of the top 10 uncorrelated hyper-alert types (in terms of the number of uncorrelated hyper-alerts). Among these hyper-alert types,

[3] Choosing less aggressive policies than Maximum_Coverage can reduce the false alert rate; however, we may also lose the opportunity to detect some attacks.

Table IV. Statistics of top 10 uncorrelated hyper-alert types.

Hyper-alert type	# uncorrelated alerts	# correlated alerts
IPHalfScan	33745	958
Windows_Access_Error	11657	0
HTTP_Cookie	2119	0
SYNFlood	1306	406
IPDuplicate	1063	0
PingFlood	1009	495
SSH_Detected	731	0
Port_Scan	698	725
ServiceScan	667	2156
Satan	593	280

uncorrelated *IPHalfScan* accounted for 61% of all uncorrelated hyper-alerts. After analyzing the dataset, we believed that most, if not all, of the uncorrelated *IPHalfScans* were triggered by *SYNFlood* attacks. *Windows_Access_Error* accounted for 21% of all uncorrelated alerts. According to the description provided by RealSecure, a *Windows_Access_Error* represents an unsuccessful file sharing connection to a Windows or Samba server, which usually results from an attempt to brute-force a login under another account's privileges. It is easy to see that the corresponding attacks could hardly prepare for any other attacks (since they failed). The third largest hyper-alert type *HTTP_Cookie* counted for 3.8% of all uncorrelated hyper-alerts. Though such alerts have certain privacy implications, we do not treat them as attacks, considering the nature of the DEFCON CTF events. These three hyper-alert types counted for 74.5% of all the alerts. We omit the discussion of the other uncorrelated hyper-alerts.

Figure 6 shows one of the small hyper-alert correlation graphs. All the hyper-alerts in this figure were destined to the host at 010.020.001.024. All the *IPHalfScan* attacks were from source IP 010.020.011.240 at source port 55533 or 55534, and destined to port 110 at the victim host. After these attacks, all the attacks in the second stage except for 31886 were from 010.020.012.093 and targeted at port 110 of the victim host. The only two hyper-alerts that were not targeted at port 110 are hyper-alert 30882, which was destined to port 80 of the victim host, and hyper-alert 31886, which was destined to port 53. Thus, it is very possible that all the hyper-alerts except for 30882 and 31886 were related.

Not all of the hyper-alert correlation graphs are as small and comprehensible as Figure 6. In particular, the largest graph (in terms of the number of nodes) has 2,940 nodes and 25,321 edges, and on average, each graph has 21.75 nodes and 310.56 edges. Obviously, most of the hyper-alert correlation graphs are too big to understand for a human user.

5.2.2 *Adjustable Graph Reduction.* We further analyzed the hyper-alert correlation graphs with the three utilities proposed in Section 4. Due to space reasons, we only report our analysis results about the largest hyper-alert correlation graph in this section.

We first applied the graph reduction utility to the largest hyper-alert correlation

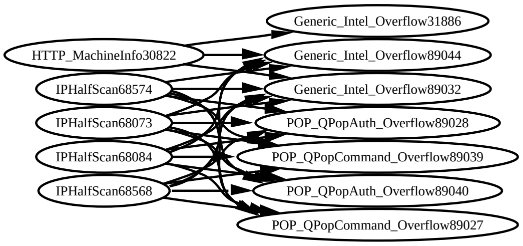

Generic_Intel_Overflow31886
HTTP_MachineInfo30822
Generic_Intel_Overflow89044
IPHalfScan68574
Generic_Intel_Overflow89032
IPHalfScan68073
POP_QPopAuth_Overflow89028
IPHalfScan68084
POP_QPopCommand_Overflow89039
IPHalfScan68568
POP_QPopAuth_Overflow89040
POP_QPopCommand_Overflow89027

Fig. 6. A small hyper-alert correlation discovered in initial analysis

graph. Figure 7 shows the fully reduced graph. Compared with the original graph, which has 2,940 nodes and 25,321 edges, the fully reduced graph has 77 nodes and 347 edges (including transitive edges).

The fully reduced graph in Figure 7 shows 7 stages of attacks. The layout of this graph was generated by GraphViz [AT & T Research Labs], which tries to reduce the number of cross edges and make the graph more balanced. As a result, the graph does not reflect the actual stages of attacks. Nevertheless, Figure 7 provides a much clearer outline of the attacks.

The hyper-alerts in stage 1 and about half of those in stage 2 correspond to scanning attacks or attacks to gain information of the target systems (e.g., *ISS, Port_Scan*). The upper part of stage 2 include attacks that may lead to execution of arbitrary code on a target system (e.g., *HTTP_WebSite_Sample*). Indeed, these hyper-alerts directly prepare for some hyper-alerts in stage 5, but GraphViz arranged them in stage 2, possibly to balance the graph. Stages 3 consists of a mix of scanning attacks (e.g., *Nmap_Scan*), attacks that reveal system information (e.g,, *HTTP_PHP_Read*), and attacks that may lead to execution of arbitrary code (e.g., *HTTP_Campas*). Stage 4 mainly consists of buffer overflow attacks (e.g., *POP_QPopCommand_Overflow*), detection of backdoor programs (e.g., *BackOrifice*), and attacks that may lead to execution of arbitrary code. The next 3 stages are much cleaner. Stage 5 consists of attacks that may be used to copy programs to target hosts, stage 6 consists of detection of two types of DDOS (Distributed Denial of Service) daemon programs, and finally, stage 7 consists of the detection of an actual DDOS attack.

Note that the fully reduced graph in Figure 7 is an approximation to the strategies used by the attackers. Hyper-alerts for different, independent sequences of attacks may be aggregated together in such a graph. For example, if two individual attackers use the sequence of attacks (e.g., using the same script downloaded from a website) to attack the same target, the corresponding hyper-alerts may be correlated and aggregated in the same fully reduced graph. Nevertheless, a fully reduced graph can clearly outline the attack strategies, and help a user understand the overall situation of attacks.

As we discussed earlier, the reduction of hyper-alert correlation graphs can be controlled with interval constraints. Figure 8 shows the numbers of nodes and edges of the reduced graphs for different interval sizes. The shapes of the two curves in Figure 8 indicate that most of the hyper-alerts that are of the same type occurred

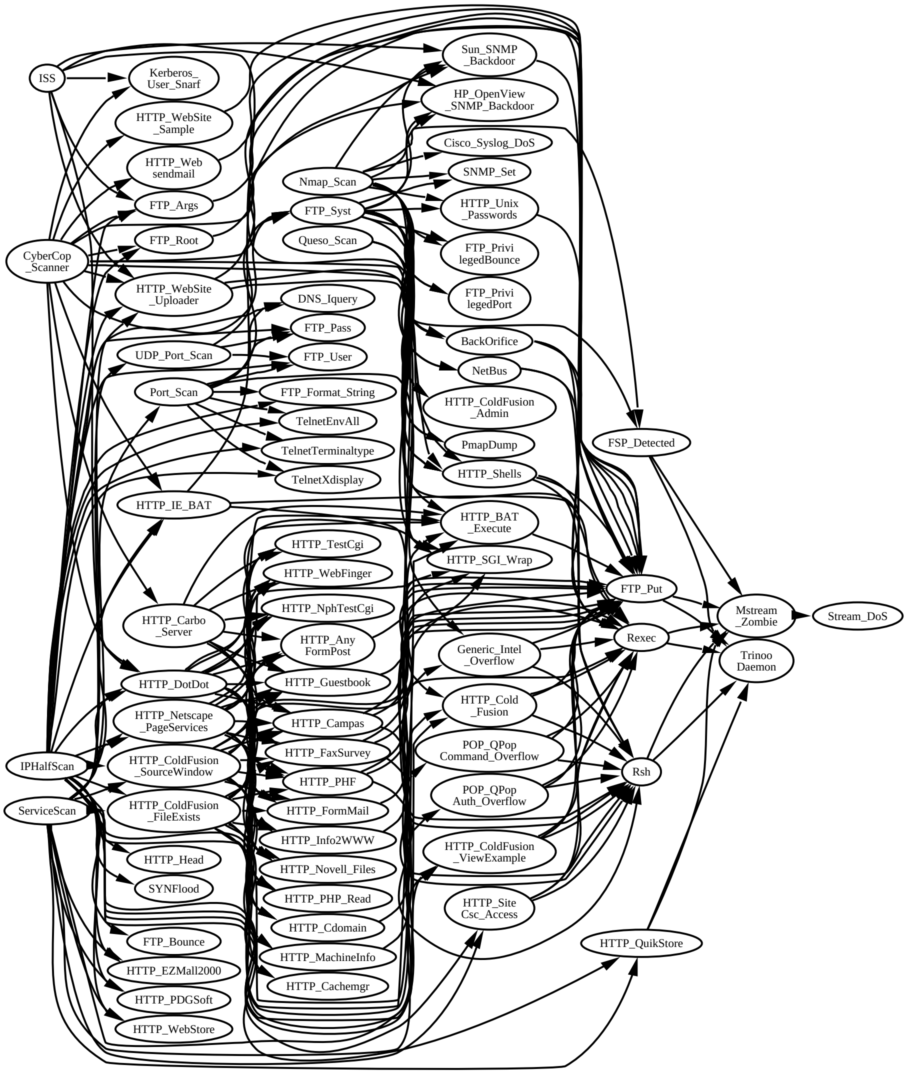


Fig. 7. The fully reduced graph for the largest aggregated hyper-alert correlation graph.

close to each other in time. Thus, the numbers of nodes and edges have a deep drop for small interval thresholds and a flat tail for large ones. A reasonable guess is that some attackers tried the same type of attacks several times before they succeeded or gave up. Due to space reasons, we do not show these reduced graphs.

5.2.3 *Focused Analysis.* Focused analysis can help filter out the interesting parts of a large hyper-alert correlation graph. It is particularly useful when a user knows the systems being protected or the potential on-going attacks. For example, a user may perform a focused analysis with focusing constraint $DestIP = ServerIP$,

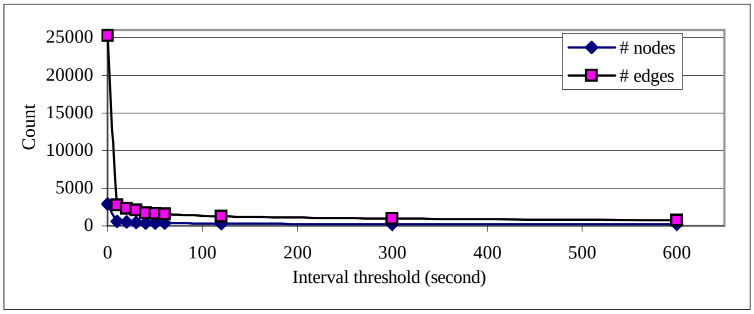


Fig. 8. Sizes of the reduced graphs w.r.t. the interval threshold for the largest hyper-alert correlation graph

where $ServerIP$ is the IP address of a critical server, to find out attacks targeted at the server. As another example, he/she may use $SrcIP = ServerIP \vee DestIP = ServerIP$ to find out attacks targeted at or originated from the server, suspecting that the server may have been compromised.

In our experiments, we tried a number of focusing constraints after we learned some information about the systems involved in the CTF event. Among these focusing constraints are (1) $C_{f1} : (DestIP = 010.020.001.010)$ and (2) $C_{f2} : (SrcIP = 010.020.011.251 \wedge DestIP = 010.020.001.010)$. We applied both focusing constraints to the largest hyper-alert correlation graph. The results consist of 2154 nodes and 19423 edges for C_{f1}, and 51 nodes and 28 edges for C_{f2}. The corresponding fully reduced graphs are shown in Figures 9 and 10, respectively. (Isolated nodes are shown in gray.) These two graphs can also be generated by graph decomposition (Section 5.2.4). To save space, we will reuse these two graphs to illustrate graph decomposition in the next subsection.

Focused analysis is an attempt to approximate a sequence of attacks that satisfy the focusing constraint. Its success depends on the closeness of focusing constraints to the invariants of the sequences of attacks. A cunning attacker would try to avoid being correlated by launching attacks from different sources (or stepping stones) and introducing delays between attacks. Thus, this utility should be used with caution.

5.2.4 *Graph Decomposition.* We applied three clustering constraints to decompose the largest hyper-alert correlation graph discussed in Section 5.2.1. In all these clustering constraints, we let $A_1 = A_2 = \{SrcIP, DestIP\}$.

(1) $C_{c1}(A_1, A_2)$: $A_1.DestIP = A_2.DestIP$. This is to cluster all hyper-alerts that share the same destination IP addresses. Since most of attacks are targeted at the hosts at the destination IP addresses, this is to cluster hyper-alerts in terms of the victim systems.

(2) $C_{c2}(A_1, A_2)$: $A_1.SrcIP = A_2.SrcIP \wedge A_1.DestIP = A_2.DestIP$. This is to cluster all the hyper-alerts that share the same source and destination IP addresses.

(3) $C_{c3}(A_1, A_2)$: $A_1.SrcIP = A_2.SrcIP \vee A_1.DestIP = A_2.DestIP \vee A_1.SrcIP$

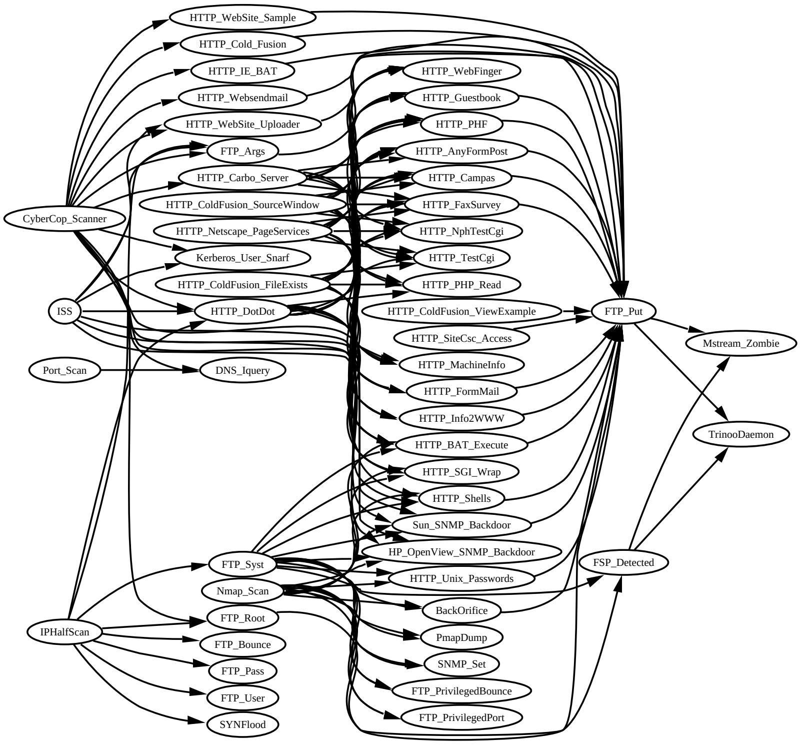


Fig. 9. A fully reduced hyper-alert correlation graph resulting from focused analysis with C_{f1} : $(DestIP = 010.020.001.010)$. This graph also appears in the result of graph decomposition with C_{c1}. (Cluster ID = 1; DestIP = 010.020.001.010.)

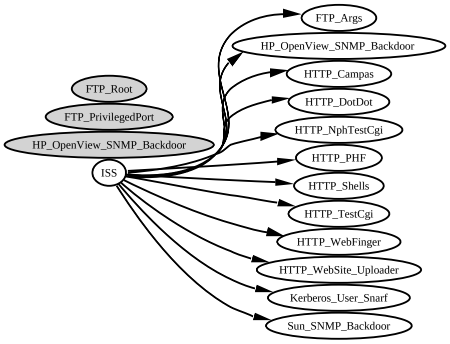


Fig. 10. A fully reduced hyper-alert correlation graph resulting from focused analysis with C_{f2} : $(SrcIP = 010.020.011.251 \wedge DestIP = 010.020.001.010)$. This graph also appears in the result of graph decomposition with C_{c2}. (Cluster ID = 10; SrcIP = 010.020.011.251; DestIP = 010.020.001.010.)

Table V. Partial statistics of decomposing the largest hyper-alert correlation graph.

	cluster ID	1	2	3	4	5	6	7	8	9
C_{c1}	# connected nodes	2154	244	105	227	83	11	54	28	0
	# edges	19423	1966	388	2741	412	30	251	51	0
	# isolated nodes	0	0	0	0	0	0	0	0	1
C_{c2}	# correlated nodes	1970	17	0	12	0	0	0	3	0
	# edges	2240	66	0	10	0	0	0	2	0
	# isolated nodes	3	0	21	17	35	26	15	12	4
C_{c3}	# connected nodes	2935	0	–	–	–	–	–	–	–
	# edges	25293	0	–	–	–	–	–	–	–
	# isolated nodes	4	1	–	–	–	–	–	–	–

$= A_2.DestIP \vee A_1.DestIP = A_2.SrcIP$. This is to cluster all the hyper-alerts that are connected via common IP addresses. Note that with this constraint, hyper-alerts in the same cluster do not necessarily share the same IP address directly, but they may connect to each other via other hyper-alerts.

Table V shows the statistics of the decomposed graphs. Constraint C_{c1} resulted in 12 clusters, among which 10 clusters contain edges. Constraint C_{c2} resulted in 185 clusters, among which 37 contain edges. Due to space reasons, we only show the first 9 clusters for C_{c1} and C_{c2} in Table V. Constraint C_{c3} in effect removes one hyper-alert from the original graph. This hyper-alert is *Stream_DoS*, which does not share the same IP address with any other hyper-alerts. This is because the source IP addresses of *Stream_DoS* are spoofed and the destination IP is different from any of the IP addresses involved in the earlier alerts. This result shows that all the hyper-alerts except for *Stream_DoS* share a common IP address with some others.

The isolated nodes in the resulting graphs are the hyper-alerts that prepare for or are prepared for by those that do not satisfy the same clustering constraints. Note that having isolated hyper-alerts in a decomposed graph does not imply that the isolated hyper-alerts are correlated incorrectly. For example, an attacker may hack into a host with a buffer overflow attack, install a DDOS daemon, and start the daemon program, which then tries to contact its master program. The corresponding alerts (i.e., the detection of the buffer overflow attack and the daemon's message) will certainly not have the same destination IP address, though they are related.

Figures 9 and 10 show a decomposed graph for C_{c1} and C_{c2}, respectively. Both graphs are fully reduced to save space. All the hyper-alerts in Figure 9 are destined to 010.020.001.010. Figure 9 shows several possible attack strategies. The most obvious ones are those that lead to the *Mstream_Zoombie* and *TrinooDaemon*. However, there are multiple paths that lead to these two hyper-alerts. Considering the fact that multiple attackers participated in the DEFCON 8 CTF event, we cannot conclude which path caused the installation of these daemon programs. Indeed, it is possible that none of them is the actual way, since the IDS may have missed some attacks.

Figure 9 involves 75 source IP addresses, including IP address 216.136.173.152, which does not belong to the CTF subnet. We believe that these attacks belong to different sequences of attacks, since there were intensive attacks from multiple attackers who participated in the CTF event.

Figure 10 is related to Figure 9, since they both are about destination IP address 010.020.001.010. Indeed, Figure 10 is a part of Figure 9, though in Figure 9, *ISS* prepares for *HTTP_Campas* through *HTTP_DotDot*. Since all the hyper-alerts in Figure 10 have the same source and destination IP addresses, it is very possible that the correlated ones belong to the same sequence of attacks. Note that *HP_OpenView_SNMP_Backdoor* appears as both connected and isolated nodes. This is because some instances are correlated, while the others are isolated.

5.2.5 *Attack Strategies.* We analyzed the correlated hyper-alerts using the three utilities and discovered several strategies used by the attackers. We first restricted our attention to the hyper-alert correlation graphs that satisfy the clustering constraint C_{c2}. One common strategy reflected by these graphs is to use scanning attacks followed by attacks that may lead to execution of arbitrary code. For example, the attacker(s) at 010.020.011.099 scanned host 010.020.001.010 with *CyberCop_Scanner, IPHalfScan, Nmap_Scan*, and *Port_Scan* and then launched a sequence of HTTP-based attacks (e.g., *HTTP_DotDot*) and FTP based attacks (e.g., *FTP_Root*). The attacker(s) at 010.020.011.093 and 010.020.011.227 also used a similar sequence of attacks against the host 010.020.001.008.

As another strategy, the attacker(s) at 010.020.011.240 used a concise sequence of attacks against the host at 010.020.001.013: *Nmap_Scan* followed by *PmapDump* and then *ToolTalk_Overflow*. Obviously, they used *Nmap_Scan* to find the *portmap* service, then used *PmapDump* to list the RPC services, and finally launched a *ToolTalk_Overflow* attack against the *ToolTalk* service. Indeed, the sequence of two alerts, *Nmap_Scan* followed by *PmapDump* with the same source and destination IP address appeared many times in this dataset.

The attacker(s) at 010.020.011.074 used the same sequence of HTTP-based attacks (e.g., *HTTP_DotDot* and *HTTP_TestCgi*) against multiple web servers (e.g., servers at 010.020.001.014, 010.020.001.015, 010.020.001.019, etc.). Our hyper-alert correlation graphs shows that *HTTP_DotDot* prepares for the HTTP-based attacks that follow. However, our further analysis of the dataset shows that this may be an incorrect correlation. Though it is possible that the attacker used *HTTP_DotDot* to collect necessary information for the later attacks, the timestamps of these alerts indicate that the attacker(s) used a script to launch all these attacks. Thus, it is possible that the attacker(s) simply launched all the attacks, hoping one of them would succeed. Though these alerts are indeed related, these prepare-for relations reveal that our method is aggressive in correlating alerts. Indeed, alert correlation is designed to recover the relationships between the attacks behind alerts; any alert correlation method may make mistakes when there is not enough information.

There are several other interesting strategies; however, due to space reasons, we do not list them here.

One interesting observation is that with clustering constraint C_{c2}, there are not many hyper-alert correlation graphs with more than 3 stages. Considering the fact that there are many alerts about *BackOrifice* and *NetBus* (which are tools to remotely manage hosts), we suspect that many attackers used multiple machines during their attacks. Thus, their strategies cannot be reflected by the restricted hyper-alert correlation graphs.

When we relax the restriction to allow hyper-alert correlation graphs for alerts with different source IP addresses, but still with the same destination IP addresses (i.e., with clustering constraint C_{c1}), we have graphs with more stages. Figure 9 is one such fully reduced hyper-alert correlation graph. However, due to the amount of alerts and source IP addresses involved in this graph, it is difficult to conclude which hyper-alerts belong to the same sequences of attacks.

In summary, during the analysis of the DEFCON 8 CTF dataset, the utilities have greatly simplified the analysis process. We have discovered several attack strategies that were possibly used during the attacks. However, there are a number of situations where we could not separate multiple sequences of attacks. This implies that additional work is necessary to address this problem.

6. DISCUSSION

Our method has several advantages. First, our approach provides a high-level representation of correlated alerts that reveals the causal relationships between them. As we have seen in Section 5, the hyper-alert correlation graphs generated by our implementation clearly show the strategies behind these attacks. Second, our approach can potentially reduce the impact caused by false alerts by providing a way to differentiate alerts. While true alerts are more likely to be correlated with other alerts, false alerts, which do not correspond to any actual attacks, tend to be more random than the true alerts, and are less likely to be correlated to others.

Our method does not depend on predefined attack scenarios to discover sequences of related attacks. This is a common feature that our method shares with JIGSAW [Templeton and Levitt 2000] and the MIRADOR approach [Cuppens and Miege 2002]. However, unlike JIGSAW, our method can correlate alerts even if some alerts correspond to failed attack attempts or the IDSs fail to detect some related attacks. Compared with the MIRADOR approach, our method allows flexible aggregation of the same type of alerts, while the MIRADOR approach treats alert aggregation as a pre-correlation process. This difference enables us to develop the three interactive analysis utilities presented in Section 4. In addition, our method provides an intuitive representation (i.e., hyper-alert correlation graph) of correlated alerts, which reveals the high-level strategy behind the attacks.

Our decision certainly has its pros and cons. On the positive side, our method is simple and yet able to correlate related alerts even when the IDSs miss certain attacks. However, on the negative side, our method may correlate alerts incorrectly when one attack seemingly prepares for another. In other words, our method has both a higher true correlation rate and a higher false correlation rate than JIGSAW. Nevertheless, Section 5 shows experimentally that our method has a low false correlation rate at least with the 2000 DARPA datasets.

Our approach has several limitations. First, our method depends on the underlying IDSs to provide alerts. Though our reasoning process can compensate for undetected attacks, the attacks missed by the IDSs certainly have a negative effect on the results. If the IDS misses a critical attack that links two stages of a series of attacks, the related hyper-alerts may be split into two separate hyper-alert correlation graphs. In the extreme case where the underlying IDSs missed all the attacks, our approach cannot do anything.

Second, our approach is not fully effective for alerts between which there is no prepare-for relationship, even if they may be related. For example, an attacker may launch concurrent *Smurf* and *SYN flooding* attacks against the same target; however, our approach will not correlate the corresponding alerts, though there are connections between them (i.e., same time and same target). Therefore, our method should be used along with other, complementary techniques (e.g., the probabilistic alert correlation [Valdes and Skinner 2001]).

Third, our method may falsely correlate alerts belonging to different attack sequences if they are launched close to each other in time. Take Figure 9 as an example, which shows a fully reduced hyper-alert correlation graph involving the destination IP address 010.020.001.010. Because of the nature of the DEFCON CTF contest, it is very likely that multiple hacker teams were attacking 010.020.001.010 during the same period of time. They were not necessarily cooperating with each other; however, because of the *possible* connections between these alerts, our method may correlate them into the same attack scenario. Nevertheless, it's not clear whether *any* method can correlate coordinated attacks without making the same mistake. We will investigate techniques that can cope with this problem in our future work.

Finally, it is worth mentioning that the results produced by our correlation techniques are only as good as the hyper-alert information provided by the user, and the interactive analysis utilities require expert users who have certain skills and insights in the attacks and the system configuration.

7. CONCLUSIONS AND FUTURE WORK

This paper presented a practical method for constructing attack scenarios through alert correlation, using prerequisites and consequences of attacks. The approach was based on the observation that in a series of attacks, the attacks were usually not isolated, but related as different stages, with the earlier stages preparing for the later ones. This paper proposed a formal framework to represent alerts along with their prerequisites and consequences, and developed a method to correlate related hyper-alerts together, including an intuitive representation of correlated alerts that reveals the attack scenario of the corresponding attacks. An off-line intrusion alert correlator was developed on the basis of the formal framework. To facilitate the analysis of large sets of correlated alerts, we also developed another three interactive utilities, *adjustable graph reduction, focused analysis*, and *graph decomposition.* We studied the effectiveness of our techniques through experiments with the 2000 DARPA intrusion detection scenario specific datasets [MIT Lincoln Lab 2000] and the DEFCON 8 CTF dataset [DEFCON 2000]. Our experimental results showed that our correlation method not only correlated related alerts and uncovered the attack strategies, but also provided a way to differentiate between alerts, and that the interactive analysis utilities could effectively simplify the analysis of large amounts of alerts. Our analysis also revealed several attack strategies that appeared in the DEFCON 8 CTF event.

Several issues are worth future research. First, we plan to develop better ways to specify hyper-alert types, especially how to represent predicates to be included in their prerequisite and consequence sets to get the best performance for alert correlation. Second, we will study possible way to integrate our method with complementary correlation methods (e.g., [Valdes and Skinner 2001]) for better performance. In particular, we are interested in methods that can better tolerate false alerts and missing detections typically seen in current IDSs. Third, we will extend our method to seek the possibility to identify attacks possibly missed by the IDSs and predict attacks in progress. In general, we would like to develop a suite of comprehensive techniques that facilitate the analysis and management of intensive intrusion alerts.

REFERENCES

Anderson, J. P. 1980. Computer security threat monitoring and surveillance. Tech. rep., James P. Anderson Co., Fort Washington, PA.

AT & T Research Labs. Graphviz - open source graph layout and drawing software. `http://www.research.att.com/sw/tools/graphviz/`.

Bace, R. 2000. *Intrusion Detection.* Macmillan Technology Publishing.

Cui, Y. 2002. A toolkit for intrusion alerts correlation based on prerequisites and consequences of attacks. M.S. thesis, North Carolina State University. Available at `http://www.lib.ncsu.edu/theses/available/etd-12052002-193803/`.

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

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

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

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

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

DEFCON. 2000. Def con capture the flag (CTF) contest. http://www.defcon.org/html/defcon-8-post.html. Archive accessible at http://wi2600.org/mediawhore/mirrors/shmoo/.

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

Gardner, R. and Harle, D. 1998. Pattern discovery and specification translation for alarm correlation. In *Proceedings of Network Operations and Management Symposium (NOMS'98).* 713–722.

Gruschke, B. 1998. Integrated event management: Event correlation using dependency graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management.*

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

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

Javits, H. and Valdes, A. 1993. The NIDES statistical component: Description and justification. Tech. rep., SRI International, Computer Science Laboratory.

Jha, S., Sheyner, O., and Wing, J. 2002. Two formal analyses of attack graphs. In *Proceedings of the 15th Computer Security Foundation Workshop.*

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

Kumar, S. 1995. Classification and detection of computer intrusions. Ph.D. thesis, Purdue University.

Kumar, S. and Spafford, E. H. 1994. A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer Security Conference.* 11–21.

LIN, J., WANG, X. S., AND JAJODIA, S. 1998. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *Proceedings of the 11th Computer Security Foundations Workshop*. Rockport, MA, 190–201.

MANGANARIS, S., CHRISTENSEN, M., ZERKLE, D., AND HERMIZ, K. 2000. A data mining analysis of RTID alarms. *Computer Networks 34*, 571–577.

MIT LINCOLN LAB. 2000. 2000 DARPA intrusion detection scenario specific datasets. `http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html`.

MORIN, B., MÉ, L., DEBAR, H., AND DUCASSÉ, M. 2002. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*. 115–137.

MUKHERJEE, B., HEBERLEIN, L. T., AND LEVITT, K. N. 1994. Network intrusion detection. *IEEE Network 8,* 3 (May), 26–41.

NING, P. AND CUI, Y. 2002. An intrusion alert correlator based on prerequisites of intrusions. Tech. Rep. TR-2002-01, North Carolina State University, Department of Computer Science. January.

NING, P., CUI, Y., AND REEVES, D. S. 2002a. Analyzing intensive intrusion alerts via correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*. Zurich, Switzerland, 74–94.

NING, P., CUI, Y., AND REEVES, D. S. 2002b. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. Washington, D.C., 245–254.

NING, P., JAJODIA, S., AND WANG, X. S. 2001. Abstraction-based intrusion detection in distributed environments. *ACM Transactions on Information and System Security 4,* 4 (November), 407–452.

PORRAS, P., FONG, M., AND VALDES, A. 2002. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*. 95–114.

RICCIULLI, L. AND SHACHAM, N. 1997. Modeling correlated alarms in network management systems. In *In Western Simulation Multiconference*.

RITCHEY, R. AND AMMANN, P. 2000. Using model checking to analyze network vulnerabilities. In *Proceedings of IEEE Symposium on Security and Privacy*. 156–165.

SHEYNER, O., HAINES, J., JHA, S., LIPPMANN, R., AND WING, J. 2002. Automated generation and analysis of attack graphs. In *Proceedings of IEEE Symposium on Security and Privacy*.

STANIFORD, S., HOAGLAND, J., AND MCALERNEY, J. 2002. Practical automated detection of stealthy portscans. *Journal of Computer Security 10,* 1/2, 105–136.

STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., DILGER, M., FRANK, J., HOAGLAND, J., LEVITT, K., WEE, C., YIP, R., AND ZERKLE, D. 1996. GrIDS - a graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*. Vol. 1. 361–370.

TEMPLETON, S. AND LEVITT, K. 2000. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*. ACM Press, 31 – 38.

VALDES, A. AND SKINNER, K. 2001. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*. 54–68.

VIGNA, G. AND KEMMERER, R. A. 1999. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security 7,* 1, 37–71.