Correlating Alerts Using Prerequisites of Intrusions

Peng Ning Douglas S. Reeves Yun Cui
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534
Emails: ning@csc.ncsu.edu, reeves@csc.ncsu.edu, ycui4@eos.ncsu.edu

Abstract

Intrusion detection has been studied for about twenty years since the Anderson's report. However, intrusion detection techniques are still far from perfect. Current intrusion detection systems (IDSs) usually generate a large amount of false alerts and cannot fully detect novel attacks or variations of known attacks. In addition, all the existing IDSs focus on low-level attacks or anomalies; none of them can capture the logical steps or strategies behind these attacks. Consequently, the IDSs usually generate a large amount of alerts. In situations where there are intensive intrusive actions, 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 intrusions behind the alerts and take appropriate actions. This paper presents a novel approach to address these issues. The proposed technique is based on the observation that most intrusions are not isolated but related as different stages of attack sequences, *with the early stages preparing for the later ones*. In other words, there are often logical steps or strategies behind series of attacks. The proposed approach correlates alerts using *prerequisites of intrusions*. Intuitively, the prerequisite of an intrusion is the necessary condition for the intrusion to be successful. For example, the existence of a vulnerable service is the prerequisite of a remote buffer overflow attack against the service. The proposed approach identifies the prerequisite (e.g., existence of vulnerable services) and the consequence of each type of attacks, and correlates the corresponding alerts by matching the consequence of some previous alerts and the prerequisite of some later ones. The proposed approach has several advantages. First, it provides a high-level representation of the correlated alerts, and thus reveals the structure of series of attacks. Second, it can reduce the impact of false alerts by only keeping correlated alerts. Third, it can potentially be applied to predict attacks in progress, and allows the intrusion response systems to take appropriate actions to stop the on-going attacks. Our preliminary experiments have demonstrated the potential of the proposed approach in reducing false alerts and uncovering high-level attack strategies.

1 Introduction

Computer and network systems have become integral parts of national infrastructure upon which many critical information services and applications (e.g., stock market, electronic transactions) are relying. As required by these services and applications, it is necessary to ensure that the computer and network systems can survive natural disasters as well as malicious attacks.

Many techniques have been developed to protect computer and network systems against malicious attacks. These techniques can be classified into two broad categories: *prevention-based techniques* and *detection and response-based techniques*. Prevention-based techniques (e.g., access control, authentication) form the first line of defense for the computer and network systems. However, our past experience showed that prevention-based techniques cannot fully protect our systems due to the flaws and weaknesses in the design and development of the systems. As a result, the detection and response based techniques (i.e., intrusion detection and response systems) are used as a second line of defense along with the prevention-based ones.

Intrusion detection techniques can be roughly classified as *anomaly detection* and *misuse detection*. Anomaly detection (e.g., NIDES/STAT [10]) 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 [21]) detects intrusions 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.

Intrusion detection techniques are still far from perfect; current intrusion detection systems (IDSs) cannot fully detect novel attacks or variations of known attacks without generating a large amount of false alerts (i.e., high false alert rate and low detection rate). In addition, all the current IDSs focus on low-level attacks or anomalies; none of them can capture the logical steps or strategies behind these attacks. Consequently, the IDSs usually generate a large amount of alerts, and the alerts are raised independently though there may be logical connections between them. In situations where there are intensive intrusive actions, 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 intrusion behind the alerts and take appropriate actions.

An important fact has been ignored by most of the current intrusion detection techniques. That is, most intrusions are not isolated, but related as different stages of series of attacks, *with the early stages preparing for the later ones*. For example, hackers need to understand what vulnerable services a host is running before they can take advantage of these services. Thus, they typically scan for vulnerable services before they break into the system. As another example, in the 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 against another system. Therefore, in a series of attacks, one or more previous attacks usually prepare for the following attacks, and the success of the previous steps affects the success of the following ones. In other words, there are often logical steps or strategies behind series of attacks.

Based on this observation, we propose to correlate alerts using *prerequisites of intrusions* to address the aforementioned problems. Intuitively, the prerequisite of an intrusion is the necessary condition for the intrusion to be successful. For example, the existence of a vulnerable service is the prerequisite of a remote buffer overflow attack against the service. We assume that the attackers need to launch some attacks to prepare for the prerequisites of some later attacks. For example, they may perform a UDP port scan to discover the vulnerable services to be attacked. Our basic idea is to identify the prerequisites (e.g., existence of vulnerable services) and the consequences (e.g., discovery of vulnerable services) of each type of attacks and correlate the attacks (alerts) by matching the consequences of some previous attacks and the prerequisites of some later ones. (Some types of attacks are variations of each other, and thus should be treated as the same type.) For example, if we find a UDP port scan followed by a buffer overflow attack against one of the scanned ports, we can correlate them to be parts of the same series of attacks.

Our proposed approach has several advantages. First, by providing a high-level representation of the correlated alerts (see section 3.4), our approach reveals the structure of a series of attacks as well as the strategy behind them. Second, our approach can potentially reduce the impact caused by false alerts. False alerts, which do not correspond to any real attacks, tend to be more random than the actual attacks, and are less likely to be correlated to other alerts. Therefore, we can focus on the potentially actual attacks by picking out the correlated alerts. Finally, our approach can be potentially extended to predict attacks in progress. By profiling the past correlated alerts, we may predict the next move of on-going attacks and take appropriate actions to prevent it from happening. As we will show in section 4, our initial experiments have demonstrated the great potential of the proposed approach in reducing false alerts as well as uncovering high-leel attack strategies.

The remainder of this paper is organized as follows. The next section discusses the related work on intrusion detection as well as alert correlation. Section 3 presents our formal framework for correlating alerts using prerequisites of intrusions. Section 4 describes our initial experiments using the proposed approach. Section 5 discusses several research issues related to the proposed approach. Section 6 concludes this paper and points out future research directions.

2 Related Work

Intrusion detection has been studied for about twenty years, since Anderson's report [1]. A survey of the early work on intrusion detection is given in [13], and an excellent overview of current intrusion detection techniques and related issues can be found in a recent book [2].

Intrusion detection techniques can be classified as *anomaly detection* or *misuse detection*. Anomaly detection 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 catches intrusions in terms of the characteristics of known attacks or system vulnerabilities; any action that conforms to the pattern of a known attack or vulnerability is considered intrusive.

Numerous research as well as commercial IDSs have been developed using anomaly and/or misuse detection techniques, including host-based IDSs (e.g., USTAT [8]), network based IDSs (e.g., NetSTAT [21], NFR [16]), and distributed IDSs (e.g., AAFID [18], EMERALD [15]).

All current IDSs are aimed at detecting low-level attacks or anomalies; none can capture the logical steps or attack strategies behind these attacks. It is usually up to human users to discover the connections between alerts. However, in intrusion intensive situations, IDSs may generate large numbers of alerts, making manual correlation of alerts a very difficult task.

Several alert correlation techniques have been proposed recently to address the aforementioned problem. In [20], a probabilistic method was used to correlate alerts using similarity between their features. However, this method depends on parameters selected by human experts (*e.g.*, similarity between classes of alerts), and is not suitable for fully discovering causal relationships between alerts. In [19], a similar approach was applied to detect stealthy portscans along with several heuristics. Though some such heuristics (*e.g.*, feature separation heuristics [19]) may be extended to the general alert correlation problem, the approach cannot fully recover the causal relationships between alerts, either.

Techniques for aggregating and correlating alerts have been proposed by others [6]. In particular, the correlation method in [6] uses a *consequence mechanism* to specify what types of alerts may follow a given alert type. This is similar to the specification of misuse signatures. However, the consequence mechanism only uses alert types, the probes that generate the alerts, the severity level, and the time interval between the two alerts involved in a consequence definition, which do not provide sufficient information to correlate all possibly related alerts. Moreover, it is not easy to predict how an attacker may arrange a sequence of attacks. In other words, developing a sufficient set of consequence definitions for alert correlation is not a solved problem.

Another approach has been proposed to "learn" alert correlation models by applying machine learning techniques to training data sets embedded with known intrusion scenarios [5]. This approach can automatically build models for alert correlation; however, it requires training in every deployment, and the resulting models may overfit the training data, thereby missing attack scenarios not seen in the training data sets. The alert correlation techniques that we present in this paper address this same problem from a novel angle, overcoming the limitations of the above approaches.

3 Correlating Alerts Using Prerequisites Of Attacks

3.1 Overview

Our approach is based on the observation that in series of attacks, the component attacks are usually not isolated, but related as different stages of the attacks, with the early ones preparing for the later ones. For example, the attackers often use certain probing attacks (e.g., IP Sweep, UDP port scan) to identify their targets before hacking into these systems.

As discussed in the introduction, we propose to correlate the alerts generated by IDSs using prerequisites 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 the prerequisite of a remote buffer overflow attack against the service. Moreover, the attacker may make some progress in intruding 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 former attack should at least make a part of the prerequisite of the later attack true.

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 attacks and correlate alerts, which are attacks detected by IDSs, by matching the consequences of some previous alerts and the prerequisites of some later ones. (Some types of attacks are variations of each other, and thus should be treated as the same type.) 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 try to correlate alerts by discovering the causal relationships between them.

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

3.2 Prerequisite and Consequence of Attacks

In order to make our approach computable, 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 formula, i.e., logical combination of predicates, to describe the prerequisite of an attack. For example, a certain network born 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*) *AND UDPAccessibleViaFirewall* (*VictimIP, VictimPort*). To simplify the discussion, we restrict the logical operations to *AND, OR* and *NOT*, and require that *NOT* only appear directly before a predicate. For example, we allow logical formula in the form of (*NOT A* (*x, y*)) *OR (NOT B* (*z*)), but not in the form of *NOT* (*A* (*x, y*) *AND B* (*z*)), though they are logically equivalent.

We use a set of logical formulas 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 set of logical formulas to represent the corresponding consequence: {*GainRootAccess* (*VictimIP*), *rhostModified* (*VictimIP*)}. This example says that as a result of the attack, the attacker may gain root access to the system and the .rhost file may be modified. Note that the set of logical formulas used to represent the consequence is essentially the conjunction of the formulas and can be represented by a single logical formula. However, allowing a set of logical formulas gives us the convenience and flexibility to avoid long formulas in consequences.

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 an attacker launches a buffer overflow attack against a service, he may or may not gain the root access, depending on whether the service is vulnerable to the attack or not.

We propose to use *possible* consequence instead of *actual* consequence due to the following two reasons. First, an IDS may not have enough information to decide whether 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 some specific information from the related hosts. Thus, it may not be feasible to correlate alerts using the actual consequence of attacks. In contrast, the possible consequence of a type of attack can be analyzed and made available for 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 successfulness of the previous attack or the attacker uses a script to launch a series of attacks. Thus, using possible consequences of attacks will lead to better opportunity to correlate such attacks.

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

3.3 Hyper-alert Type and Hyper-alert

Having predicate as the basic construct, we introduce the notion of *hyper-alert type* to represent the prerequisite and the consequence of each type of alerts.

Definition 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 formula whose free variables are all in *fact*, and (3) *consequence* is a set of logical formulas such that all the free variables in *consequence* are in *fact*.

Each hyper-alert type encodes the knowledge about a type of attacks. The component *fact* of a hyper-alert type tells what kind of information is reported along with the detected attack (i.e., alert), *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 1 An attacker may perform an IPSweep attack to discover the existing hosts in a certain network. We may have a hyper-alert type *IPSweep* = (*fact, prerequisite, consequence*) for such attacks, where (1) *fact* = {*SweepedIP*}, (2) *prerequisite* = $\emptyset$, and (3) *consequence* = {*ExistHost*(*IP*)}. This hyper-alert type says that IPSweep is to discover whether there is a host running at IP address *SweepedIP*. (We expect an IDS reports the values of *SweepedIP* along with the corresponding IPSweep alerts.) As a result of this type of attacks, the attacker may learn the existing host(s) at the IP address(es) identified by the value(s) of *SweepedIP*. □

Example 2 Consider the buffer overflow attack against the *sadmind* remote administration tool. We may have a hyper-alert type *SadmindBufferOverflow* = (*fact, prerequisite, consequence*) for such attacks, where (1) *fact* = $\{$*VictimIP, VictimPort*$\}$, (2) *prerequisite* = *ExistHost* (*VictimIP*) *AND VulnerableSadmind* (*VictimIP*), and (3) *consequence* = $\{$*GainRootAccess*(*VictimIP*)$\}$. Intuitively, this hyper-alert type says that such an attack is against the host running at IP address *VictimIP*. (Similar to the previoius example, we expect the actual values of *VictimIP* are reported by an IDS.) As the prerequisite of a successful attack, there must exist a host at IP address *VictimIP* and the corresponding *sadmind* service should 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 an alert that describes an *Sadmind Buffer Overflow* attack. The notion of hyper-alert instance is formally defined as follows.

Definition 2 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 logical formulas in *consequence* might evaluate to True for each of the tuples.

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 sweeped IP addresses, which cannot be represented as a single record. Thus, we believe the current notion of a hyper-alert is a more appropriate choice.

A hyper-alert also 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 hyper-alert type *IPSweep* (see example 1) 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 Consider the hyper-alert type *IPSweep* defined in example 1. We may have a hyper-alert $h_{IPSweep}$ that includes the following tuples: $\{$(*SweepedIP* = 152.141.129.1), (*SweepedIP* = 152.141.129.2), ..., (*SweepedIP* = 152.141.129.254)$\}$. As possible consequences of the attack, the following predicates might be True: *ExistHost* (152.141.129.1), *ExistHost* (152.141.129.2), ..., *ExistHost* (152.141.129.254). This hyper-alert says that there is an IP Sweep attack, and as a result, the attacker may learn the existence of the hosts at these IP addresses. □

Example 4 Consider the hyper-alert type *SadmindBufferOverflow* defined in example 2. We may have a hyper-alert $h_{SadmindBOF}$ that includes the following tuples: $\{$(*VictimIP* = 152.141.129.5, *VictimPort* = 1235), (*VictimIP* = 152.141.129.37, *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* (152.141.129.5) *AND VulnerableSadmind* (152.141.129.5), *ExistHost* (152.141.129.37) *AND VulnerableSadmind* (152.141.129.37), and the following two predicates might be True as possible consequences of the attack: *GainRootAccess* (152.141.129.5), *GainRootAccess* (152.141.129.37). This hyper-alert says that there are buffer overflow attacks against *sadmind* at IP addresses 152.141.129.5 and 152.141.129.37, 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 reports 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 of the 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 together if the consequences of the former ones imply the prerequisite of the latter one, as illustrated by the following example.

Example 5 First 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* = (*fact, prerequisite, consequence*), where (1) *fact* = $\{$*VictimIP, VictimPort*$\}$, (2) *prerequisite* = *ExistHost* (*VictimIP*), and (3) *consequence* = $\{$*VulnerableSadmind* (*VictimIP*)$\}$.

Suppose a hyper-alert $h_{SadmindPing}$, which is an instance of type *SadmindPing*, has the following tuples: $\{$(*VictimIP* = 152.141.129.5, *VictimPort* = 1235), (*VictimIP* = 152.141.129.37, *VictimPort* = 1235), (*VictimIP* = 152.141.129.39, *VictimPort* = 1235)$\}$. Thus, the following predicates might be True as the consequences of the attack: *VulnerableSadmind* (152.141.129.5), *VulnerableSadmind* (152.141.129.37), *VulnerableSadmind* (152.141.129.39). Further consider the hyper-alerts $h_{IPSweep}$ and $h_{SadmindBOF}$, which were discussed in examples 3 and 4, respectively. It is easy to see that the consequences of $h_{IPSweep}$ and $h_{SadmindPing}$ imply the prerequisite of $h_{SadmindBOF}$. Thus, we can correlate these three hyper-alerts together. $\square$

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 satisfy (or more precisely, imply) the prerequisite of an alert.

Having considered the above issues, we adopt an alternative approach. Instead of examining whether a set of hyper-alerts satisfies the prerequisite of another hyper-alert, 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 pieces of predicates and test whether the consequence of an earlier hyper-alert makes some pieces of the prerequisite True (i.e., make the prerequisite easier to satisfy). If the result is positive, then we correlate the hyper-alerts together.

Definition 3 Given a hyper-alert h, which is an instance of type *T* = (*fact, prerequisite, consequence*), the *prerequisite set of* h, denoted as $P(h)$, is the set of all such predicates (with the preceding negation if there is one) that appear in *prerequisite* and whose arguments are replaced with the corresponding attribute values of a tuple in h. Each element in $P(h)$ is associated with the timestamp of the corresponding tuple in h. The *consequence set of* h, denoted as $C(h)$, is the set of all such logical formulas in *consequence* whose arguments are replaced with the corresponding attribute values of a tuple in h. Each element in $C(h)$ is associated with the timestamp of the corresponding tuple in h.

Example 6 Consider the hyper-alert $h_{SadmindBOF}$ discussed in example 4. The corresponding prerequisite set is $P(h_{SadmindBOF})$ = $\{$*ExistHost* (152.141.129.5), *ExistHost* (152.141.129.37), *VulnerableSadmind* (152.141.129.5), *VulnerableSadmind* (152.141.129.37)$\}$, and the consequence set is $C(h_{SadmindBOF})$ =

$\{$*GainRootAccess* (152.141.129.5), *GainRootAccess* (152.141.129.37)$\}$.

Further consider the hyper-alert $h_{SadmindPing}$ in example 5. The prerequisite set is $P(h_{SadmindPing})$ = $\{$*ExistHost* (152.141.129.5), *ExistHost* (152.141.129.37), *ExistHost* (152.141.129.39)$\}$, and the consequence set is $C(h_{SadmindPing})$ = $\{$*VulnerableSadmind* (152.141.129.5), *VulnerableSadmind* (152.141.129.37), *VulnerableSadmind* (152.141.129.39)$\}$. $\square$

Definition 4 Hyper-alert h_1 *prepares for* hyper-alert h_2, if there exist $p \in P(h_2)$ and $C \subseteq C(h_1)$ such that for all $c \in C$, $c.end\_time < p.begin\_time$ and the conjunction of all the logical formulas in C implies p.

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 7 Let us continue example 6. 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 that the element *VulnerableSadmind* (152.141.129.5) (among others) in $P(h_{SadmindBOF})$ is also in $C(h_{SadmindPing})$. Thus, $h_{SadmindPing}$ prepares for, and should be correlated with $h_{SadmindBOF}$. $\square$

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' such that either h prepares for h' or h' prepares for h. Otherwise, h is called an *isolated hyper-alert*. Our goal is to discover all pairs of hyper-alerts h_1 and h_2 in S such that h_1 prepares for h_2.

Example 8 First consider the detection of a certain DDOS daemon which tries to contact its master program about its existence. This can be captured by the hyper-alert type *DDOSDaemon* = (*fact, prerequisite, consequence*), where *fact* = $\{$*IP*$\}$, *prerequisite* = *GainRootAccess* (*IP*), and *consequence* = $\emptyset$. Assume that there is a hyper-alert $h_{DDOSDaemon}$ which consists of the following tuple: (*IP* = 152.141.129.5).

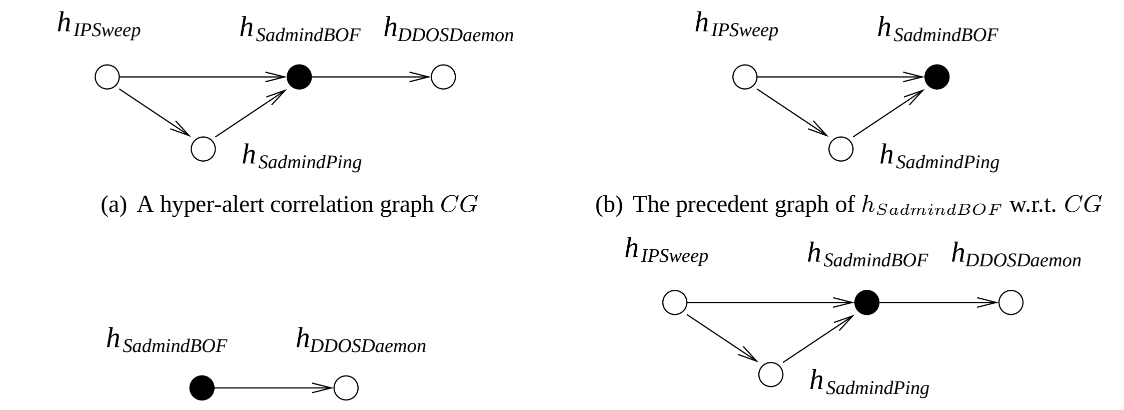
Suppose in a sequence of hyper-alerts we have the following ones: $h_{IPSweep}$, $h_{SadmindPing}$, $h_{SadmindBOF}$, and $h_{DDOSDaemon}$. (The first three hyper-alerts have been explained in examples 3, 4, and 5, respectively.) Assume that all tuples in a later hyper-alert have timestamps greater than every tuple in a previous hyper-alert. By comparing the prerequisite sets and the consequence sets of these hyper-alerts, we can easily get the following relationships: $h_{IPSweep}$ prepares for $h_{SadmindPing}$, $h_{IPSweep}$ prepares for $h_{SadmindBOF}$, $h_{SadmindPing}$ prepares for $h_{SadmindBOF}$, and finally $h_{SadmindBOF}$ prepares for $h_{DDOSDaemon}$. Thus, all of the four hyper-alerts should be correlated together. $\square$

3.3.1 Temporal Constraints for Hyper-alerts

One may point out that the definition of hyper-alert seems to be over flexible. It allows alerts that are far from each other in time to be treated as a single hyper-alert. For example, the definition allows to treat two UDP port scanning activities that are not close to each other in time as a single hyper-alert. The activities that occur far from each other in time tend not to relate to each other. Therefore, we have little reason to treat them as a single hyper-alert.

The reason that we allow a hyper-alert to be aggregated from multiple alerts is to have flexibility in reasoning about alerts. As we discussed earlier, an IDS (e.g., EMERALD [15]) may generate multiple alerts for the same attack. Thus, having the current definition of hyper-alert gives more opportunity for alert correlation than not allowing alert aggregation. Nevertheless, it is desirable to exclude irreasonable situations such as the one pointed out earlier.

In the following, we introduce two temporal constraints for hyper-alerts. 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.



(a) A hyper-alert correlation graph CG

(b) The precedent graph of $h_{SadmindBOF}$ w.r.t. CG

(c) The subsequent graph of $h_{SadmindBOF}$ w.r.t. CG

(d) The correlated graph of $h_{SadmindBOF}$ w.r.t. CG

Figure 1: Hyper-alerts correlation graphs

Definition 5 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 6 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 t in h, there exist another t' in h such that there exist $t.begin\_time < T < t.end\_time$, $t'.begin\_time < T' < t'.end\_time$, and $|T - T'| < I$.

There could be other meaningful constraints for hyper-alerts, depending on the semantics of the relevant contexts. We will identify such constraints when we encounter them in our research; however, they are not the current focus and will not be discussed in this paper.

3.4 Hyper-alert Correlation Graph

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

Definition 7 A *hyper-alert correlation graph* CG = (N, E) is a connected 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.

Hyper-alert correlation graph provides an intuitive representation of correlated hyper-alerts. For instance, figure 1(a) shows the hyper-alert correlation graph for the hyper-alerts in example 8. With the notion of hyper-alert correlation graph, our goal of alert correlation can be rephrased as the discovery of maximum hyper-alert correlation graphs 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 be helpful to inform the administrator how this happened, that is, report all the alerts directly or indirectly prepare for the DDOS daemon. Therefore, we define the following operations on hyper-alert correlation graphs.

Definition 8 Given a hyper-alert correlation graph CG = (N, E) and a hyper-alert n in N, $precedent$ (n, CG) is an operation that returns the maximum sub-graph PG = (N', E') of CG 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.* CG.

Definition 9 Given a hyper-alert correlation graph CG = (N, E) and a hyper-alert n in N, $subsequent$ (n, CG) is an operation that returns the maximum sub-graph PG = (N', E') of CG 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 PG is called the *subsequent graph of* n *w.r.t.* CG.

Definition 10 Given a hyper-alert correlation graph CG = (N, E) and a hyper-alert n in N, $correlated$ (n, CG) is an operation that returns the maximum sub-graph PG = (N', E') of CG 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 PG is called the *correlated graph of* n *w.r.t.* CG.

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

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 the hyper-alert of concern.

Hyper-alert correlation graph is not only an intuitive way to represent correlated attacks, 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 have patterns in their strategies, we can use hyper-alert correlation graph to profile previous series of 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. In addition, we may use previous hyper-alert correlation graphs to predict the next move of on-going attacks, and then take appropriate actions to prevent it from happening. Nevertheless, additional research is necessary to make these approaches possible.

3.5 Advantages of Our Approach

As a post-detection technique, our approach has the potential to complement the existing intrusion detection techniques. This can be seen through the following advantages of our approach.

First, our approach provides a high-level representation of detected attacks that reveals the causal relationships between the alerts. Most of the current IDSs either report the detection of low-level attacks (e.g., IP sweep, UDP port scan) as independent alerts without identifying the connection between them, or correlate the alerts using the attribute values describing the alerts (e.g., [20]). In the later case, the result of alert correlation could be misleading and unable to fully discover the causal relationships between alerts. As a result, it is difficult for the human users to figure out the logical connection between the low-level alerts and understand the logical steps or attacking strategies behind the alerts. In an extreme situation where the information systems are under intensive intrusions, the IDSs could report tons of alerts, and it is very difficult for human users to understand them and take appropriate actions.

Our approach complements this deficiency by providing a high-level representation of the correlated alerts. Presenting correlated alerts in a hyper-alert correlation graph, our approach can reveal the causal structure of a series of attacks.

Second, our approach can potentially reduce the impact caused by false alerts. As discussed earlier, there are usually certain strategies or logical steps behind series of attacks. Indeed, in order to launch some destructive attacks, an attacker may have to perform some other attacks to either gather the necessary information or prepare in some other ways. If the alerts correspond to steps of actual attacks, our approach can correlate the corresponding alerts together by taking advantage of the prepare-for relationships between them. In contrast, the false alerts, which do not correspond to any real attacks, tend to be more random than the actual attacks, and are less likely to be correlated to other alerts. Therefore, we can easily identify the actual attacks by picking out the correlated alerts.

The effectiveness of our approach certainly relies on the performance of the underlying IDSs. If the IDS misses the intermediate steps of a series of attacks, it might be difficult to find the connection between alerts using the prerequisites of intrusion. Additional research is necessary to understand the impact of undetected attacks.

Third, our approach can potentially be extended to predict attacks in progress. Though attackers may have many choices at different stages of a series of attacks, they may still have certain patterns in their strategies, especially when the attacks are launched by scripts. Thus, the hyper-alert correlation graph can be extended to represent the patterns of intrusion strategies.

A potential way to predict attacks in progress is to profile the common attacking strategies using hyper-alert correlation graphs. By comparing the on-going alerts with the profiles of known attacking strategies, we may be able to predict the next step in a series of attacks and prevent it by taking appropriate actions. For example, suppose in our profile, more than 80% of *SadmindPing* alerts prepare for a certain type of *SadmindBOF* attacks. If the IDS generates an *SadmindPing* alert and we believe that our *sadmind* service is vulnerable to certain buffer overflow attacks, we can either shut down the service or inform the firewall not to forward any packets to the *sadmind* service. Nevertheless, predicting future attacks is a difficult problem, and one has to be careful with the predicted attack to avoid, for example, being misled by a cunning attacker.

4 Experimental Results

We have developed an off-line intrusion alert correlator based on the proposed approach and performed several experiments [14]. We report the experimental results in this section; please read [14] for further details.

We have performed a series of experiments using the two DARPA 2000 intrusion detection evaluation datasets [12]. The first dataset contains a series of attacks with which a novel attacker probes, breaks-in, installs DDOS daemon and master programs, and launches a DDOS attack against an off-site server. The second dataset includes a similar sequence of attacks run by an attacker who is a bit more sophisticated than the first one.

We used the network traffic data in the two datasets in our experiments. Each dataset includes the network traffic data 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. Due to the constraints of our testing network, we have not performed experiments with both the DMZ and the inside

Dataset		# observable attacks	Tool	# alerts	# detected attacks	Detection rate	# real alerts	False alert rate
1	DMZ	89	RealSecure	891	51	57.30%	57	93.60%
			Our method	57	50	56.18%	54	5.26%
	Inside	60	RealSecure	922	37	61.67%	44	95.23%
			Our method	44	36	60%	41	6.82%
2	DMZ	7	RealSecure	425	4	57.14%	6	98.59%
			Our method	5	3	42.86%	3	40%
	Inside	15	RealSecure	489	12	80.00%	16	96.73%
			Our method	13	10	66.67%	10	23.08%

Figure 2: The experimental results

network traffic at the same time. Nevertheless, such experiments are interesting and may lead to further insight into the alert correlation problem. We consider it as one of our future works.

In each experiment, we replayed the selected network traffic in an isolated network monitored by a RealSecure Network Sensor 6.0 [9]. We chose RealSecure 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. For simplicity, our alert correlator maps each alert generated by the RealSecure Network Sensor to a hyper-alert. (For details of the hyper-alert types, please see [14].) In all the experiments, our alert correlator only kept the correlated alerts, and discarded all the isolated ones.

Figure 2 shows the experimental results, including the detection and false alert rates for both RealSecure Network Sensor 6.0 and our method. We counted the number of actual attacks and false alerts according to the description included in the datasets. False alerts can be identified easily. However, counting the number of attacks is a subjective process, 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 Figure 2. Note that some activities such as *telnet* are not usually considered as attacks; however, we counted them here if the attacker used them as parts of the attacks.

RealSecure Network Sensor 6.0 generated duplicated alerts for certain attacks. For example, one *rsh* connection in the inside part of dataset one, which the attacker used to access the compromised host, triggered two alerts. As a result, the number of real alerts (i.e., the alerts corresponding to actual attacks) is greater than the number of detected attacks. Figure 2 shows both kinds of numbers. The detection rates were calculated using the number of detected attacks, while the false alert rates were computed with the number of real alerts.

For the DMZ network traffic in dataset one, the RealSecure Network Sensor generated 891 alerts. According to the description of the data set, 57 out of the 891 alerts were real alerts, 51 attacks were detected, and 38 attacks were missed. Thus, the detection rate of RealSecure Network Sensor was 57.30%, and the false alert rate was 93.60%.[1] Our intrusion alert correlator processed the alerts generated by the RealSecure Network Sensor. As shown in Figure 2, 57 alerts remained after correlation, 54 of them were real alerts, and

[1]Please note that choosing less aggressive policies than Maximum_Coverage may result in less false alert rates.

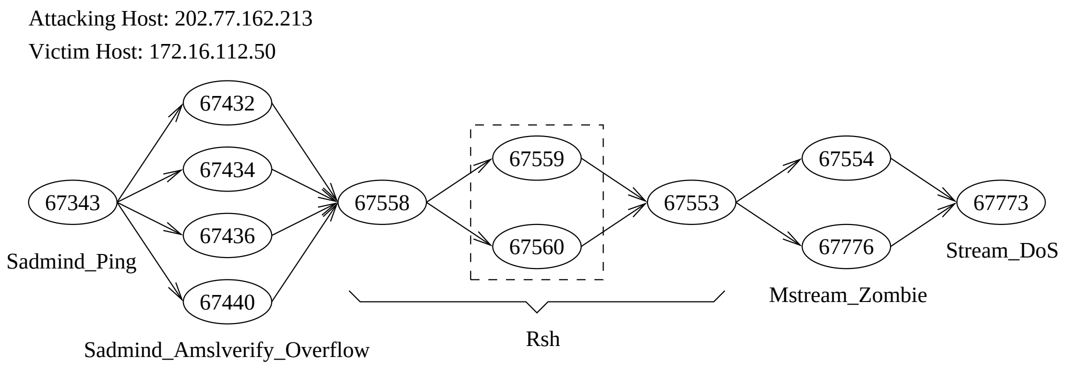


Figure 3: A hyper-alert correlation graph discovered in the inside network traffic of dataset one. (Transitive edges are removed for the sake of readability.)

50 attacks were covered by these alerts. Thus, the detection rate and the false alert rate after alert correlation were 56.18% and 5.26%, respectively. The results for the other datasets are also shown in Figure 2.

The experimental results in Figure 2 show that our approach greatly reduced the false alert rates. However, the detection rate was not affected very much. In the worst cases (the DMZ network traffic in datasets one and two), only one attack was missed after the correlation process.

Our experiments also revealed the high-level attack strategies via hyper-alert correlation graphs. Figure 3 shows one of the hyper-alert correlation graphs discovered from the inside network traffic in dataset one. Each node in Figure 3 represents a hyper-alert. The numbers inside the nodes are the alert IDs generated by the RealSecure Network Sensor. There are totally 12 hyper-alerts in figure 3, which can be divided into five stages. (Note that these five stages are not exactly the five phases stated in the description of the datasets [12].) The first stage has one *Sadmind_Ping* hyper-alert, with which the attacker discovered the possibly vulnerable *Sadmind* service running at 172.16.112.50. The second stage consists of four *Sadmind_Amslverify_Overflow* hyper-alerts, which correspond to four different variations of buffer overflow attacks against the *Sadmind* service at 172.16.112.50. The third stage consists of four *Rsh* hyper-alerts, with which the attacker copied a program and a .rhosts file and started the program on the victim system. (Hyper-alerts 67559 and 67560 correspond to the same *Rsh* event. They were here because RealSecure generated duplicated alerts for this *Rsh* event.) The fourth stage consists of two *Mstream_Zombie* hyper-alerts, which correspond to the communication between the DDOS daemon and master. Finally, the fifth stage consists of one *Stream_DoS* hyper-alert. This is the actual DDOS attack launched by the DDOS daemon programs. Figure 3 only shows the direct edges between these hyper-alerts for the sake of readability. For example, the hyper-alert 67432 also prepares for 67554, but the correspond edge is not included in Figure 3.

Our method depends on the underlying IDS for alert information. As shown in Figure 2, it reduced the false alert rate, but did not improve the detection rate. Indeed, the detection rate of our method cannot be better than the underlying IDS. This implies that if the underlying IDS is very poor in detecting attacks, our method cannot perform well, either. (This is why we decided to use the Maximum_Coverage policy in the RealSecure Network Sensor.) However, if the underlying IDS can detect the attacks but with a large amount of false alerts, our method can help identify the real alerts from the false ones.

The data sets in the above experiments include attacks with strong connections. In reality, our method may have worse false alert rate and detection rate. Nevertheless, the significant improvement over the RealSecure Network Sensor shown in our experiments has demonstrated the potential of the proposed approach.

5 Discussion

5.1 Systematic Development of Predicates and Hyper-alert Types

Since predicates are used to represent the prerequisites and consequences of hyper-alerts, they form the foundation of our approach. In order to make the proposed approach feasible, we need to make sure that we can develop the predicates required for hyper-alert correlation in a cost-effective way. In addition, we should avoid possible inconsistency among the predicates, and have certain ways to ensure the developed predicates can support the proposed approach.

It is worth noticing that the predicates can be represented in different granularities. For example, the situation that a host runs an *sadmind* service vulnerable to buffer overflow attack can be represented either by *UDPVulnerableToBOF* (*VictimIP, VictimPort*), which simply says that a service at IP address *VictimIP* and port number *VictimPort* is vulnerable to buffer overflow attacks, or by *SadmindVulnerableToBOF* (*VictimIP, VictimPort*), which specifically says that the *sadmind* service at IP address *VictimIP* and port number *VictimPort* is vulnerable to buffer overflow attacks, or even more specifically by *SadmindVulnerableToBOFType123* (*VictimIP, VictimPort*), which in addition specifies which type of buffer overflow attacks the *sadmind* service is vulnerable to.

The granularities in which the predicates are developed have impact on alert correlation. On the one hand, the finer grained the predicates are, the more precise we can expect the result to be. On the other hand, the cost for developing finer-grained predicates would be higher than coarser-grained ones. Moreover, the finer-grained the predicates are, the less they can accommodate variations of attack strategies.

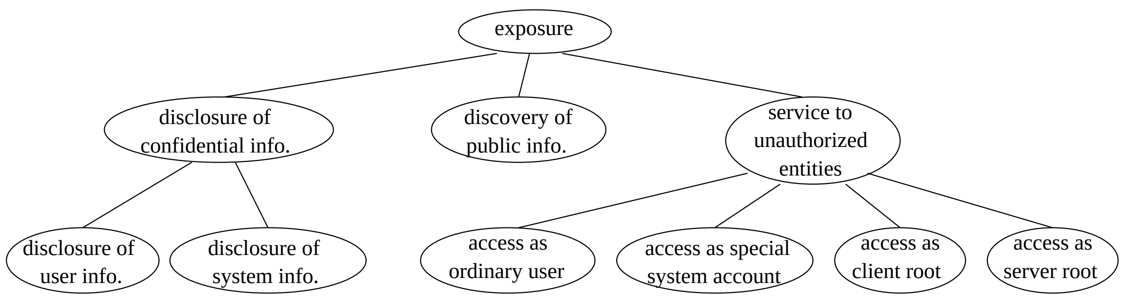
Each hyper-alert type corresponds to a class of hyper-alerts that share the same prerequisites and consequences. The development of hyper-alert types depends on what kinds of predicates are used. Although we developed the hyper-alert types and the involved predicates in an ad-hoc way, it is possible to develop them systematically.

Lindqvist and Jonsson presented a classification of intrusions in [11]. In particular, they provided a hierarchical classification of intrusion results. The top-level classes of intrusion results include three classes: *exposure, denial of service (DOS)*, and *erroneous output*. We plan to use this classification (with a little modification) as the framework to systematically develop predicates as well as hyper-alert types. Specifically, we classify all possible consequences into groups and divide each group into more specific groups using their classification scheme. We can start to specify predicates when the arguments of a group can be fixed.

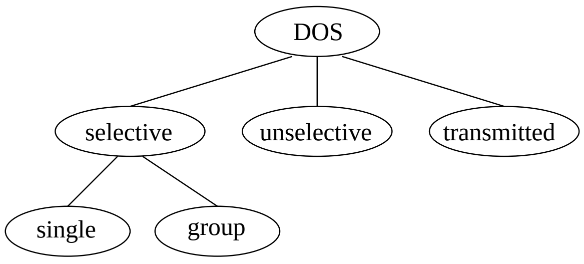
Figures 4(a) to 4(c) show our classification scheme. The leaf-level groups may be further partitioned when necessary. In the original classification, the top-level class *exposure* is classified into two classes: *disclosure of confidential information* and *service to unauthorized entities*. However, we observe that some attacks may reveal information that is not confidential but useful for latter attacks. For example, an IPSweep may discover hosts accessible from the network, though the existence of the hosts are not exactly confidential but available to the public. Thus, we added a third class named *discovery of public information*.

In addition, Common Vulnerabilities & Exposures (CVE) [3] may be used to assist predicates and hyper-alert types development. The current version (version 20010918) of CVE database has 1,604 entries. By applying the above classification scheme, we can group these entries into different classes and then develop the predicates and hyper-alert types within each final group. For example, we may have the predicates *ExistHost* (*IP*) and *ExistService* (*IP, port*) in the subgroups of *exposure.public_info*.

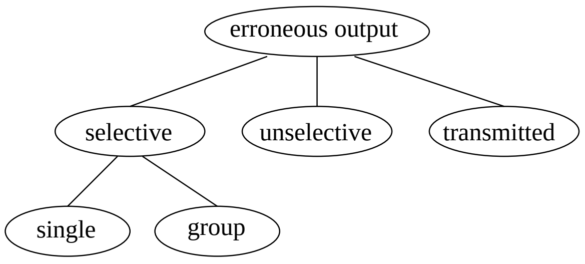
It is necessary to keep the implication relationships between predicates in order to reason about the prepare-for relations between hyper-alerts. In our implementation [14], human users are required to input


exposure
disclosure of confidential info.
discovery of public info.
service to unauthorized entities
disclosure of user info.
disclosure of system info.
access as ordinary user
access as special system account
access as client root
access as server root

(a) Classification of the top-level consequence *exposure*



(b) Classification of the top-level consequence *DOS*



(c) Classification of the top-level consequence *erroneous output*

Figure 4: Classification of the consequences of intrusions

the direct implications between predicates, and the intrusion alert correlator figures out the indirect ones automatically.

The development of predicates and hyper-alert types is a knowledge engineering process. Though the above proposed solutions and CVE provide general framework and guidelines for this process, substantial work is still necessary. To better understand the nature of predicate development, we plan to first identify the predicates that are necessary for our empirical study. The complete set of known predicates, if there is one, can be developed after a rigorous study of the proposed approach. Note that the set of predicates may have to be updated frequently like the vocabulary of natural languages, if more previously unknown attacks as well as their consequences are being discovered.

5.2 Generation of Hyper-alerts

In order to correlate hyper-alerts using prerequisites of attacks, our approach has to generate hyper-alerts from the alerts reported by the underlying IDSs. The hyper-alert generation process faces two challenges. The first challenge comes from the variety of the existing IDSs. Different IDSs usually generate alerts in different granularities and formats. To make our alert correlation approach applicable to as many IDSs as possible, we need certain ways to deal with alerts generated by heterogeneous IDSs and avoid being IDS specific.

Second, high-level hyper-alerts are preferable to capture the attacking strategies. However, alerts reported by IDSs usually correspond to low-level attacks. In particular, some IDSs (e.g., EMERALD [15, 20]) may generate multiple alerts for a single attack if the attack spreads over a period of time. (Our initial implementation maps each alert to a hyper-alert. However, the purpose is to gain better understanding of our approach. It is certainly not the best way to generate hyper-alerts.) To better support the proposed approach, we need to fill in the gap between the preference of high-level alerts and the low-level alerts provided by IDSs. In particular, we need to understand how to discover and recover the alerts that correspond to a single attack.

The problem in hyper-alert generation can be partially addressed by standards such as Intrusion Detection Message Exchange Format (IDMEF) [4]. We expect that the hyper-alert generation program can correctly understand the alerts generated by the IDSs that support IDMEF. However, not all the problems can be addressed by IDMEF. Alerts in different granularities are still allowed, and multiple alerts may be generated for a single attack. To address this problem, we propose to allow hyper-alert aggregation during the generation of hyper-alerts. For example, hyper-alerts of the same type can be aggregated into a single hyper-alert provided that the resulting hyper-alert satisfies certain duration constraint or interval constraint. Hyper-alerts of different types may also be aggregated into a compound hyper-alert if certain given conditions are satisfied. Existing alert correlation techniques [5, 20] may be adopted for this purpose. Additional research is necessary to decide when to use hyper-alert aggregation and what kind of aggregations are necessary.

5.3 Processing of Hyper-alerts

To correlate hyper-alerts using prerequisites of intrusions requires to discover all pairs of hyper-alerts such that one hyper-alert prepares for the other. A naive approach is to check each hyper-alert with all the others to find out all the hyper-alerts that it prepares for. However, this approach could be very expensive if a large amount of hyper-alerts are being discovered. Indeed, given n hyper-alerts, we will have n^2 pairs of hyper-alerts, each of which further requires reasoning about the predicates in the prerequisite and consequence sets. This is certainly not a scalable solution.

An engineering method to improve the naive approach is to only consider the hyper-alerts that are close to each other in time. In practice, a threshold T of time duration may be used; we check a hyper-alert with a previous one only if the time difference between them is less than T. This method is based on the assumption that attacks far from each other tend not to correlated to each other. Although the assumption could be true most of time, exceptions still exist. For example, intrusive activities involved in a series of stealthy attacks may be far from each other, simply because the attacker does not want to draw any attention. Relaxing the threshold T is one way to better correlate the attacks; however, the larger T is, the more expensive it is to process the hyper-alerts, and there will be a limit of T due to the constraint of system resources. Therefore, additional techniques are desirable to reduce the cost of hyper-alert processing.

In our initial implementation [14], we used the database management system (DBMS) to help process the alerts and hyper-alerts stored in the database. In other words, we represent the operations to be performed on hyper-alerts as SQL statements and let the DBMS handle all the operations. Such a method performs well for off-line correlation of alerts; however, it will suffer from severe performance penalty when it is used to process alerts in real-time.

Two approaches may be used to improve the hyper-alert processing. First, we may use the hyper-alert type information to help decide whether to check two hyper-alerts or not. Note that a hyper-alert instance of type T_1 may prepare for another hyper-alert instance of type T_2 only if the consequence set of T_1 may imply a predicate in the prerequisite set $P(T_2)$. Specifically, we say that hyper-alert type T_1 *may prepare for* hyper-alert type T_2 if there exist h_1, which is a hyper-alert of type T_1, and h_2, which is a hyper-alert of type T_2, such that h_1 prepares for h_2. Thus, we check hyper-alerts h_1 and h_2 only when the type of h_1 may prepare for the type of h_2 or vice versa.

The second possible enhancement is to adapt in-memory database query optimization techniques (e.g., in-memory hybrid hash join [7], T-Tree [17]) to assist hyper-alert processing. In the simplest case where we do not allow reasoning about the predicates, the discovery of prepare-for relation between hyper-alerts can be transformed to a search problem. Thus, database query optimization techniques can be applied directly to process hyper-alerts. Specifically, by building indexes on the elements of the consequence sets of hyper-alerts, we can search for the previous hyper-alerts whose consequence sets contain the elements in the prerequisite set of a new hyper-alert.

In general, when reasoning about predicates (e.g., *VulnerableSadmindService* (*VictimIP, VictimPort*) implies *VulnerableService* (*VictimIP, VictimPort*)) is allowed, the query optimization techniques have to be customized or improved in order to support hyper-alert correlation. One way to transform the general hyper-alert processing problem to a database search problem is to derive the implied predicates and build indexes on all the (instantiated) predicates (i.e., both original and implied predicates). (This is the method adopted by our initial implementation [14].) It may be an expensive method because of the computation involved in the derivation of implied predicates and the storage required for the indexes. Further research is needed to see if it works for real-time alert correlation.

5.4 Limitations of The Proposed Approach

Our approach has several limitations. First, as discussed earlier, the hyper-alert correlation method depends on the underlying IDSs to provide alerts. The attacks missed by the IDSs certainly has a negative effect on our approach. 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 separated 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 effective to alerts between which there is no prepare-for relation, even though they may be related. For example, an attacker may launch concurrent *Smurf* and *SYN flooding* attacks against the same target from different places; however, our approach will not correlate the corresponding alerts, though there is strong connections between them (i.e., same time and same target).

Given the above limitations, our approach should be used along with other intrusion detection techniques. In particular, it is still necessary to discover novel techniques that can detect unknown attacks or variations of known attacks. Nevertheless, our approach greatly reduces the impact of false alerts; we can have high confidence in the existence of attacks if some alerts are correlated by our approach. In addition, our approach can uncover the high-level attack strategies behind series of attacks. Therefore, we believe that the proposed approach advances the intrusion detection techniques despite of its limitations.

6 Conclusion and Future Work

This paper presented a novel approach to correlating alerts using prerequisites and consequences of intrusions. The approach was based on the observation that in series of attacks, component attacks were usually not isolated, but related as different stages, with the earlier stages preparing for the later ones. This paper adopted a formal framework to represent alerts along with their prerequisites and consequences and proposed a method to correlate related hyper-alerts together. Moreover, hyper-alert correlation graphs were developed to provide an intuitive representation of correlated alerts, which also reveal the high-level strategies behind the attacks. Our initial experiments have demonstrated the potential of the proposed technique in reducing false alerts and uncovering high-level attack strategies.

Several issues are worth future research. First, we will study and compare various ways to generate hyper-alerts from alerts reported by IDSs. Second, we will refine the off-line hyper-alert processing method and develop on-line algorithms so that alerts generatd by IDSs can be correlated in real-time. Third, we will further study the effectiveness of our approach. We expect to develop several quantative measures to assess the preformance of our approach.

Acknowledgement The first author would like to thank Robert Cunningham and David Fried for providing the NetPoke program.

References

[1] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PA, 1980.

[2] R.G. Bace. *Intrusion Detection*. Macmillan Technology Publishing, 2000.

[3] CVE Editorial Board. The common vulnerabilities & exposures. http://www.cve.mitre.org.

[4] D. Curry and H. Debar. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet Draft, draft-ietf-idwg-idmef-xml-03.txt, February 2001.

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

[6] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, number 2212 in Lecture Notes in Computer Science, pages 85 – 103, 2001.

[7] D.J. DeWitt, R.H. Katz, F. Olken, L.D. Shapiro, M.R. Stonebraker, and D. Wood. Implementation techniques for main memory database systems. *SIGMOD Record*, 14(2):1–8, 1984.

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

[9] Inc. ISS. RealSecure intrusion detection system. http://www.iss.net.

[10] H.S. Javits and A. Valdes. The NIDES statistical component: Description and justification. Technical report, SRI International, Computer Science Laboratory, 1993.

[11] U. Lindqvist and E. Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 154–163, Oakland, CA, May 1997.

[12] Lincoln Lab MIT. DARPA 2000 intrusion detection evaluation datasets. http://ideval.ll.mit.edu/2000_index.html, 2000.

[13] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May 1994.

[14] P. Ning and Y. Cui. An intrusion alert correlator based on prerequisites of intrusions. Submitted for publication. Available as Technical Report TR-2002-01, Department of Computer Science, North Carolina State University., January 2002.

[15] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, National Institute of Standards and Technology, 1997.

[16] M. J. Ranum, K. Landfield, M. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall. Implementing a generalized tool for network monitoring. In *Eleventh Systems Administration Conference (LISA '97)*, October 1997.

[17] TimesTen Performance Software. Architecture for real-time data management: Timesten's core in-memory database technology. White Paper, 2001.

[18] E.H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34:547–570, 2000.

[19] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. To appear in Journal of Computer Security.

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

[21] G. Vigna and R. A. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.