

Towards Automating Intrusion Alert Analysis*

Peng Ning, Yun Cui, Douglas S. Reeves, and Dingbang Xu
Cyber Defense Laboratory
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8207

Email: {pning, ycui4}@ncsu.edu, reeves@csc.ncsu.edu, dxu@ncsu.edu

Abstract

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 summarizes a series of research efforts aimed at addressing this problem. These efforts start with an approach to constructing 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, this 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. Finally, this paper describes techniques to learn attacks strategies from correlated intrusion alerts.

1 Introduction

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 summarize a series of research efforts aimed at addressing the above problem [25–28]. These efforts start with an approach to constructing attack scenarios based on 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.

*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 grants CCR-0207297 and ITR-0219315.

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 [21], “encountering 10–20,000 alarms per sensor per day is common.” To facilitate the analysis of large sets of correlated alerts, we develop three utilities (called *adjustable graph reduction*, *focused analysis*, and *graph decomposition*). These utilities are intended for human users to analyze and understand the correlated alerts as well as the strategies behind them.

It is often desirable, and sometimes necessary, to understand attack strategies in security applications such as computer forensics and intrusion responses. For example, it is easier to predict an attacker’s next move, and decrease the damage caused by intrusions, if the attack strategy is known during intrusion response. To facilitate the extraction of attack strategies from intrusion alerts and complement static vulnerability analysis techniques (e.g., [1, 16, 31, 33]), we develop techniques to automatically learn attack strategies from intrusion alerts reported by IDSs. By examining correlated intrusion alerts, our method extracts the constraints intrinsic to the attack strategy automatically. Specifically, an attack strategy is represented as a directed acyclic graph (DAG), which we call an *attack strategy graph*, with nodes representing attacks, edges representing the (partial) temporal order of attacks, and constraints on the nodes and edges. These constraints represent the conditions that any attack instance must satisfy in order to use the strategy. To cope with variations in attacks, we use generalization techniques to hide the differences not intrinsic to the attack strategy. By controlling the degree of generalization, users may inspect attack strategies at different levels of details.

The remainder of this paper is organized as follows. The next section discusses related work. Section 2 presents our formal framework for correlating alerts using prerequisites and consequences of attacks, as well as a DBMS based implementation of this approach. Section 3 describes three utilities for analyzing attack scenarios constructed from large collections of alerts. Section 4 presents techniques to generalize attack strategies from correlated intrusion alerts. Section 6 concludes this paper and points out some future research directions.

2 Correlating Intrusion Alerts Based on Prerequisites and Consequences of Attacks

Our method for alert correlation 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 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 *Sadmin Ping* followed by a buffer overflow attack against the corresponding *Sadmin* 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.

2.1 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.

2.2 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 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 1 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 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 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 2 Consider the hyper-alert type *SadminBufferOverflow* discussed in example 1. There may be a hyper-alert $h_{SadminBOF}$ 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: $ExistHost(152.1.19.5) \wedge VulnerableSadmin(152.1.19.5)$, $ExistHost(152.1.19.7) \wedge VulnerableSadmin(152.1.19.7)$. 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 *sadmin* 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 *sadmin* 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 Consider a hyper-alert type $T = (\text{fact}, \text{prerequisite}, \text{consequence})$. 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 $p.\text{begin_time}$ and $p.\text{end_time}$ to refer to the timestamp associated with p .)

Example 3 Consider the *Sadmin Ping* attack through which an attacker discovers possibly vulnerable *sadmin* services. The corresponding alerts can be represented by a hyper-alert type $SadminPing = (\{VictimIP, VictimPort\}, \{ExistHost (VictimIP)\}, \{VulnerableSadmin (VictimIP)\})$.

Suppose a hyper-alert instance $h_{SadminPing}$ of type $SadminPing$ 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 $P(h_{SadminPing}) = \{ExistHost (152.1.19.5), ExistHost (152.1.19.7), ExistHost (152.1.19.9)\}$, and the consequence set $C(h_{SadminPing}) = \{VulnerableSadmin (152.1.19.5), VulnerableSadmin (152.1.19.7), VulnerableSadmin (152.1.19.9)\}$.

Example 4 Consider the hyper-alert $h_{SadminBOF}$ in example 2. We have $P(h_{SadminBOF}) = \{ExistHost (152.1.19.5), ExistHost (152.1.19.7), VulnerableSadmin (152.1.19.5), VulnerableSadmin (152.1.19.7)\}$, and $C(h_{SadminBOF}) = \{GainRootAccess (152.1.19.5), GainRootAccess (152.1.19.7)\}$.

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 $c \in C$, $c.\text{end_time} < p.\text{begin_time}$ 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 5 Let us continue examples 3 and 4. Assume that all tuples in $h_{SadminPing}$ have timestamps earlier than every tuple in $h_{SadminBOF}$. By comparing the contents of $C(h_{SadminPing})$ and $P(h_{SadminBOF})$, it is clear the instantiated predicate $VulnerableSadmin (152.1.19.5)$ (among others) in $P(h_{SadminBOF})$ is also in $C(h_{SadminPing})$. Thus, $h_{SadminPing}$ prepares for, and should be correlated with $h_{SadminBOF}$.

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 .

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 5 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 .

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.

Figure 1 shows a hyper-alert correlation graph discovered in our experiments [26]. Each node in Figure 1 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.)

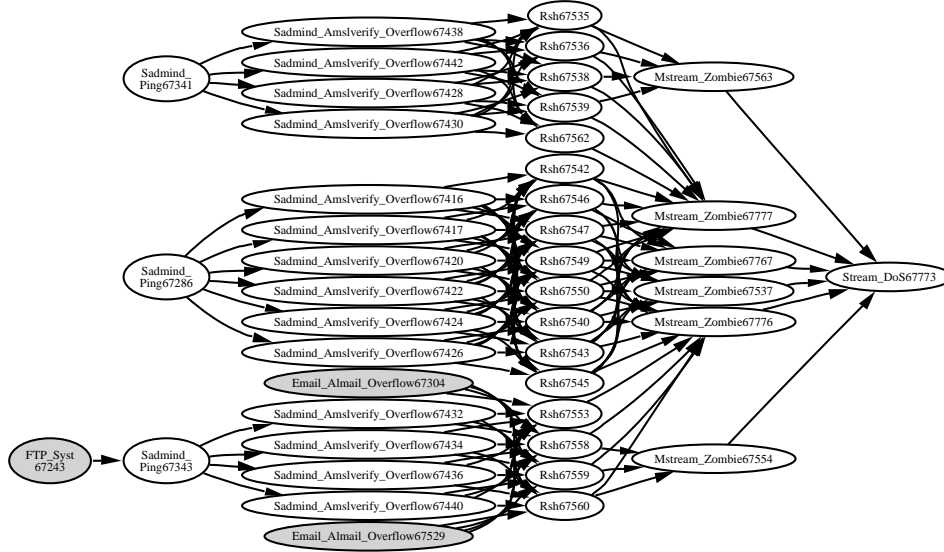


Figure 1: A hyper-alert correlation graph discovered in our experiments

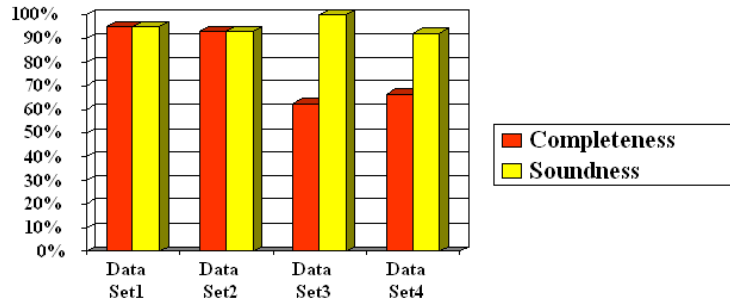
The hyper-alerts can be divided into five stages horizontally. The first stage consists of three *Sadmin_Ping* alerts, which the attacker used to find out the vulnerable *Sadmin* services. The three alerts are from source IP address 202.077.162.213, and to destination IP addresses 172.016.112.010, 172.016.115.020, and 172.016.112.050, respectively. The second stage consists of fourteen *Sadmin_Amslverify_Overflow* alerts. According to the description of the attack scenario, the attacker tried three different stack pointers and two commands in *Sadmin_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.

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. Figure 2(a) shows the completeness and the soundness measures computed in our experiments [26]. We also examine the effect of alert correlation in differentiating true and false alerts. Figures 2(b) and 2(c) show the false alert rate and the detection rate before and after alert correlation. We can see a significant reduction in false alert rate, but a slight reduction in detection rate after correlation.

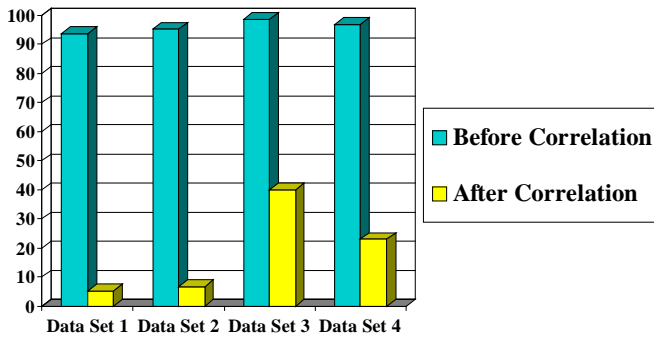
3 Analyzing Intensive Alerts

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 [9]) has revealed several problems.

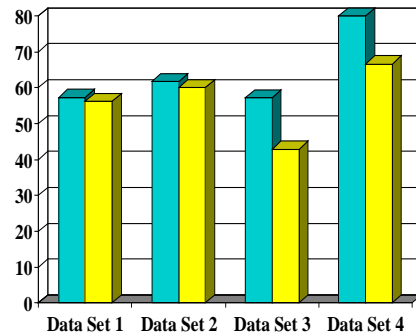
First, let us consider the following scenario. Suppose an IDS detected an *SadminPing* attack, which discovered the vulnerable *Sadmin* service on host V , and later an *SadminBufferOverflow* attack against the *Sadmin* service. Assuming that they were launched from different hosts, should we correlate 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.



(a) Completeness and soundness



(b) False alert rate



(c) Detection rate

Figure 2: Performance data

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 [9] 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 [21].” 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.

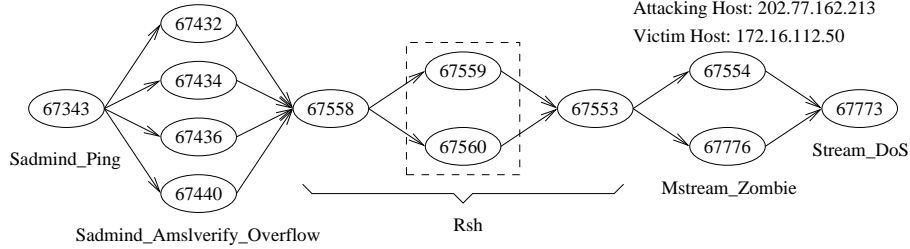


Figure 3: A hyper-alert correlation graph discovered in the 2000 DARPA intrusion detection evaluation datasets

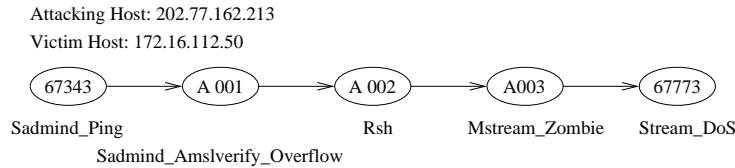


Figure 4: A hyper-alert correlation graph reduced from Fig. 3

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 [36] with our method.

3.1 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 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.

3.2 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 $C_f = \text{True}$ 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.

3.3 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 $C_c(h_3, h_1)$.

The three utilities presented 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.

We have performed a series of experiments with the DEFCON 8 CTF dataset to study the usefulness of these utilities [25]. Our experience indicates that the utilities have greatly simplified the analysis process and 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.

4 Learning Attack Strategies from Correlated Alerts

The correlation model can be used to construct attack scenarios (represented as hyper-alert correlation graphs) from intrusion alerts. Although such attack scenarios *reflect* attack strategies, they do not capture the essence of the strategies. Indeed, even with the same attack strategy, if an attacker changes certain details during attacks, the correlation model will generate different hyper-alert correlation graphs. For example, an attacker may repeat (unnecessarily) one step in a sequence of attacks many times, and the correlation model will generate a much more complex attack scenario. As another example, if an attacker uses equivalent, but different attacks, the correlation model will generate different hyper-alert correlation graphs as well. It is then up to the user to figure out manually the common strategy used in two sequences of attacks. This certainly increases the overhead in intrusion alert analysis.

In the following, we present a model to represent and automatically extract attack strategies from correlated alerts. The goal of this model is to capture the invariants in attack strategies that do not change across multiple instances of attacks.

4.1 Attack Strategy Graph

The strategy behind a sequence of attacks is indeed about how to arrange earlier attacks to prepare for the later ones so that the attacker can reach his/her final goal. Thus, the prepare for relations between the intrusion alerts (*i.e.*, detected attacks) is intrinsic to attack strategies. However, in the correlation model, the prepare for relations are between specific intrusion alerts; they do not directly capture the conditions that have to be met by related attacks. To facilitate the representation of the invariant attack strategy, we transform the prepare for relation into some common conditions that have to be satisfied by *all* possible instances of the same strategy. In the following, we formally represent such a condition as an *equality constraint*.

Definition 6 Given a pair of hyper-alert types (T_1, T_2) , an *equality constraint for (T_1, T_2)* is a conjunction of equalities in the form $u_1 = v_1 \wedge \dots \wedge u_n = v_n$, where u_1, \dots, u_n are attribute names in T_1 and v_1, \dots, v_n are attribute names in T_2 , such that there exist $p(u_1, \dots, u_n)$ and $p(v_1, \dots, v_n)$, which are the same predicate with possibly different arguments, in $ExpConseq(T_1)$ and $Prereq(T_2)$, respectively. Given a type T_1 hyper-alert h_1 and a type T_2 hyper-alert h_2 , h_1 and h_2 *satisfy the equality constraint* if there exist $t_1 \in h_1$ and $t_2 \in h_2$ such that $t_1.u_1 = t_2.v_1 \wedge \dots \wedge t_1.u_n = t_2.v_n$ evaluates to True.

There may be several equality constraints for a pair of hyper-alert types. However, if a type T_1 hyper-alert h_1 prepares for a type T_2 hyper-alert h_2 , then h_1 and h_2 must satisfy at least one of the equality constraints. Indeed, h_1 preparing for h_2 is equivalent to the conjunction of h_1 and h_2 satisfying at least one equivalent constraint and h_1 occurring before h_2 . Assume that h_1 occurs before h_2 . If h_1 and h_2 satisfy an equality constraint for (T_1, T_2) , then by Definition 6, there must be a predicate $p(u_1, \dots, u_n)$ in $ExpConseq(T_1)$ such that the same predicate with possibly different arguments, $p(v_1, \dots, v_n)$, is in $Prereq(T_2)$. Since h_1 and h_2 satisfy the equality constraint, $p(u_1, \dots, u_n)$ and $p(v_1, \dots, v_n)$ will be instantiated to the same predicate in $ExpConseq(h_1)$ and $Prereq(h_2)$. This implies that h_1 prepares for h_2 . Similarly, if h_1 prepares for h_2 , there must be an instantiated predicate that appears in $ExpConseq(h_1)$ and $Prereq(h_2)$. This implies that there must be a predicate with possibly different arguments in $ExpConseq(T_1)$ and $Prereq(T_2)$ and that this predicate leads to an equality constraint for (T_1, T_2) satisfied by h_1 and h_2 .

Example 6 The notion of equality constraint can be illustrated with an example. Consider the following hyper-alert types: $SadminPing = (\{VictimIP, VictimPort\}, ExistsHost(VictimIP), \{VulnerableSadmin(VictimIP)\})$, and $SadminBufferOverflow = (\{VictimIP, VictimPort\}, ExistHost(VictimIP) \wedge VulnerableSadmin(VictimIP), \{GainRootAccess(VictimIP)\})$. The first hyper-alert type indicates that $SadminPing$ is a type of attacks that requires the existence of a host at the $VictimIP$, and as a result, the attacker may find out that this host has a vulnerable $Sadmin$ service. The second hyper-alert type indicates that this type of attacks requires a vulnerable $Sadmin$ service at the $VictimIP$, and as a result, the attack may gain root access. Obviously, the predicate $VulnerableSadmin$ is in both $Prereq(SadminBufferOverflow)$ and $ExpConseq(SadminPing)$. Thus, we have an equality constraint $VictimIP = VictimIP$ for $(SadminPing, SadminBufferOverflow)$, where the first $VictimIP$ comes from $SadminPing$, and the second $VictimIP$ comes from $SadminBufferOverflow$.

We observe many times that one step in a sequence of attacks may trigger multiple intrusion alerts, and the number of alerts may vary in different situations. This is partially due to the existing vulnerabilities and the hacking tools. For example, `unicode_shell` [29], which is a hacking tool against Microsoft IIS web server, checks about 20 vulnerabilities at the scanning stage and usually triggers the same number of alerts. As another example, in the attack scenario reported in [26], the attacker tried 3 different stack pointers and 2 commands in $Sadmin_Amslverify_Overflow$ attacks for each victim host until one attempt succeeded. Even if not necessary, an attacker may still deliberately repeat the same step multiple times to confuse IDSs and/or system administrators. However, such variations do not change the corresponding attack strategy. Indeed, these variations make the attack scenarios unnecessarily complex, and may hinder manual or automatic analysis of the attack strategy. Thus, we decide to disallow such situations in our representation of attack strategies.

In the following, an attack strategy is formally represented as an attack strategy graph.

Definition 7 Given a set \mathcal{S} of hyper-alert types, an *attack strategy graph* over \mathcal{S} is a quadruple (N, E, T, C) , where (1) (N, E) is a connected DAG (directed acyclic graph); (2) T is a mapping that maps each $n \in N$ to a hyper-alert type in \mathcal{S} ; (3) C is a mapping that maps each edge $(n_1, n_2) \in E$ to a set of equality constraints for $(T(n_1), T(n_2))$; (4) For any $n_1, n_2 \in N$, $T(n_1) = T(n_2)$ implies that there exists $n_3 \in N$ such that $T(n_3) \neq T(n_1)$ and n_3 is in a path between n_1 and n_2 .

In an attack strategy graph, each node represents a step in a sequence of related attacks. Each edge (n_1, n_2) represents that a type $T(n_1)$ attack is needed to prepare for a successful type $T(n_2)$ attack. Each edge may also be associated with a set of equality constraints satisfied by the intrusion alerts. These equality constraints indicate how one attack prepares for another. Finally, as represented by condition 4 in Definition 7, same type of attacks should be considered as one step, unless they depend on each other through other types of attacks.

Now let us see an example of an attack strategy graph.

Example 7 Figure 5 is the attack strategy graph extracted from the hyper-alert correlation graph in Figure 3. The hyper-alert types are marked above the corresponding nodes, and the equality constraints are labeled near the corresponding edges. This attack strategy graph clearly shows the component attacks and the constraints that the component attacks must satisfy.

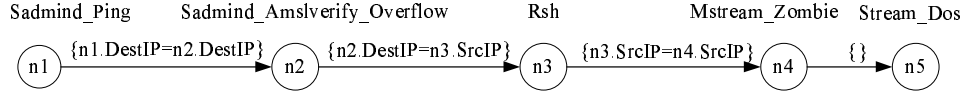


Figure 5: An example of attack strategy graph

4.2 Learning Attack Strategies

As discussed earlier, our goal is to learn attack strategies automatically from correlated intrusion alerts. This requires we extract the constraints intrinsic to attack strategy from alerts so that the constraints apply to all instances of the same strategy.

Our strategy to achieve this goal is to process the correlated intrusion alerts in two steps. First, we aggregate intrusion alerts that belong to the same step of a sequence of attacks into one hyper-alert. For example, in Figure 3, alerts 67432, 67434, 67436, and 67440 are indeed attempts of the same attack with different parameters, and thus they should be aggregated as one step in the attack sequence. Second, we extract the constraints between the attack steps and represent them as an attack strategy graph. For example, after we aggregate the hyper-alerts in the first step, we may extract the attack strategy graph shown in Figure 5.

The challenge lies in the first step. Because of the variations of attacks as well as the signatures that IDSs use to recognize attacks, there is no clear way to identify intrusion alerts that belong to the same step in a sequence of attacks. In the following, we first attempt to use the attack type information to do so. The notion of *aggregatable* hyper-alerts is introduced formally to clarify when the same type of hyper-alerts can be aggregated.

Definition 8 Given a hyper-alert correlation graph $CG = (N, E)$, a subset $N' \subseteq N$ is *aggregatable*, if (1) all nodes in N' are the same type of hyper-alerts, and (2) $\forall n_1, n_2 \in N'$, if there is a path from n_1 to n_2 , then all nodes in this path must be in N' .

Intuitively, in a hyper-alert correlation graph, where intrusion alerts have been correlated together, the same type of hyper-alerts can be aggregated as long as they are not used in different stages in the attack sequence. Condition 1 in Definition 8 is quite straightforward, but condition 2 deserves more explanation. Consider the same type of hyper-alerts h_1 and h_2 . If h_1 prepares for a different type of hyper-alert h' (directly or indirectly), and h' further prepares for h_2 (directly or indirectly), h_1 and h_2 obviously belong to different steps in the same sequence of attacks. Thus, we shouldn't allow them to be aggregated together. Although we have never observed such situations, we cannot rule out such possibilities.

Based on the notion of aggregatable hyper-alerts, the first step in learning attack strategy from a hyper-alert correlation graph is quite straightforward. We only need to identify and merge all aggregatable hyper-alerts. To proceed to the second step in strategy learning, we need a hyper-alert correlation graph in which each hyper-alert represents a separate step in the attack sequence. Formally, we call such a hyper-alert correlation graph an *irreducible hyper-alert correlation graph*.

Definition 9 A hyper-alert correlation graph $CG = (N, E)$ is *irreducible* if for all $N' \subseteq N$, where $|N'| > 1$, N' is not aggregatable.

Figure 6 shows the algorithm to extract attack strategy graphs from hyper-alert correlation graphs. The subroutine *GraphReduction* is used to generate an irreducible hyper-alert correlation graph, and the rest of the algorithm extracts the components of the output attack strategy graph. The steps in this algorithm are self-explanatory; we do not repeat them in the text.

4.3 Dealing with Variations of Attacks

Algorithm 1 in Figure 6 has ignored equivalent but different attacks in sequences of attacks. For example, an attacker may use either *pmap_dump* or *Sadmind_Ping* to find a vulnerable Sadmind service. As another

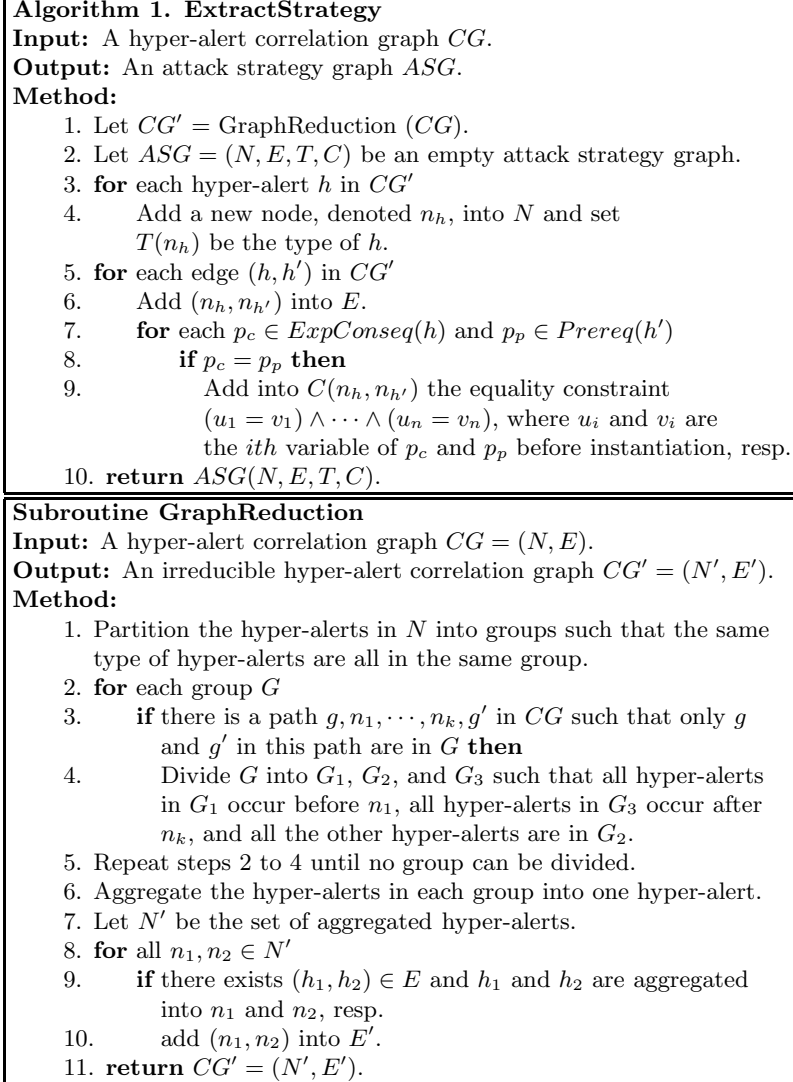


Figure 6: Extract attack strategy graph from a hyper-alert correlation graph

example, an attacker may use either *SadminBufferOverflow* or *TooltalkBufferOverflow* attack to gain remote access to a host. Obviously, at the same stage of two sequences of attacks, if an attacker uses equivalent but different attacks, Algorithm 1 will return two different attack strategy graphs, though the strategies behind them are the same.

We propose to generalize hyper-alert types so that the syntactic difference between equivalent hyper-alert types is hidden. For example, we may generalize both *SadminBufferOverflow* and *TooltalkBufferOverflow* attacks into *RPCBufferOverflow*.

A generalized hyper-alert type is created to hide the unnecessary difference between specific hyper-alert types. Thus, an occurrence of any of the specific hyper-alerts should imply an occurrence of the generalized one. This is to say that satisfaction of the prerequisite of a specific hyper-alert implies the satisfaction of the prerequisite of the generalized hyper-alert. Moreover, to cover all possible impact of all the specific hyper-alerts, the consequences of all the specific hyper-alert types should be included in the consequence of the generalized hyper-alert type. It is easy to see that this generalization may cause loss of information. Thus, generalization of hyper-alert types must be carefully handled so that information essential to attack strategy is not lost.

In the following, we formally clarify the relationship between specific and generalized hyper-alert types.

Definition 10 Given two hyper-alert types T_g and T_s , where $T_g = (fact_g, prereq_g, conseq_g)$ and $T_s = (fact_s, prereq_s, conseq_s)$, we say T_g is *more general than* T_s (or, equivalently, T_s is *more specific than* T_g) if there exists an injective mapping f from $fact_g$ to $fact_s$ such that the following conditions are satisfied:

- If we replace all variables x in $prereq_g$ with $f(x)$, $prereq_s$ implies $prereq_g$, and
- If we replace all variables x in $conseq_g$ with $f(x)$, then all formulas in $conseq_s$ are implied by $conseq_g$.

The mapping f is called the *generalization mapping* from T_s to T_g .

Example 8 Suppose hyper-alert types *SadmindBufferOverflow* and *TooltalkBufferOverflow* are specified as follows: *SadmindBufferOverflow* = ($\{VictimIP, VictimPort\}$, $ExistHost(VictimIP) \wedge VulnerableSadmind(VictimIP)$, $\{GainRootAccess(VictimIP)\}$), and *TooltalkBufferOverflow* = ($\{VictimIP, VictimPort\}$, $ExistHost(VictimIP) \wedge VulnerableTooltalk(VictimIP)$, $\{GainRootAccess(VictimIP)\}$). Assume that *VulnerableSadmind(VictimIP)* imply *VulnerableRPC(VictimIP)*. Intuitively, this represents that if there is a vulnerable Sadmin service at *VictimIP*, then there must be a vulnerable RPC service (i.e., the Sadmin service) at *VictimIP*. Similarly, we assume *VulnerableTooltalk(VictimIP)* also implies *VulnerableRPC(VictimIP)*. Then we can generalize both *SadmindBufferOverflow* and *TooltalkBufferOverflow* into *RPCBufferOverflow* = ($\{VictimIP\}$, $ExistHost(VictimIP) \wedge VulnerableRPC(VictimIP)$, $\{GainRootAccess(VictimIP)\}$), where the generalization mapping is $f(VictimIP) = VictimIP$.

By identifying a generalization mapping, we can specify how a specific hyper-alert can be generalized into a more general hyper-alert. Following the generalization mapping, we can find out what attribute values of a specific hyper-alert should be assigned to the attributes of the generalized hyper-alert. The attack strategy learning algorithm can be easily modified: We first generalize the hyper-alerts in the input hyper-alert correlation graph into generalized hyper-alerts following the generalization mapping, and then apply Algorithm 1 to extract the attack strategy graph.

Although a hyper-alert can be generalized in different granularities, it is not an arbitrary process. In particular, if one hyper-alert prepares for another hyper-alert before generalization, the generalized hyper-alerts should maintain the same relationship. Otherwise, the dependency between different attack stages, which is intrinsic in an attack strategy, will be lost.

The remaining challenge is how to get the “right” generalized hyper-alert types and generalization mappings. The simplest way is to manually specify them. For example, *Apache2*, *Back*, and *Crashiis* are all Denial of Service attacks. We may simply generalize all of them into one *WebServiceDOS*. However, there are often different ways to generalize. To continue the above example, *Apache2* and *Back* attacks are against the apache web servers, while *Crashiis* is against the Microsoft IIS web server. To keep more information about the attacks, we may want to generalize *Apache* and *Back* into *ApacheDOS*, while generalize *Crashiis* and possibly other DOS attacks against the IIS web server into *IISDOS*. Nevertheless, this doesn’t affect the attack strategy graphs extracted from correlated intrusion alerts as long as the constraints on the related alerts are satisfied.

4.3.1 Automatic Generalization of Hyper-Alert Types

It is time-consuming and error-prone to manually generalize hyper-alert types. One way to partially automate this process is to use clustering techniques to identify the hyper-alert types that should be generalized into a common one. In our experiments, we use the bottom-up hierarchical clustering [15] to group hyper-alert types hierarchically on the basis of the similarity between them, which is derived from the similarity between the prerequisites and consequences of hyper-alert types. The method used to compute the similarity is described below.

To facilitate the computation of similarity between prerequisites of hyper-alert types, we convert each prerequisite into an *expanded prerequisite set*, which includes all the predicates that appear or are implied by the prerequisite. Similarly, we can get the expanded consequence set. Consider two sets of predicates, denoted S_1 and S_2 , respectively. We adopt the Jaccard similarity coefficient [14] to compute the similarity between S_1 and S_2 , denoted $Sim(S_1, S_2)$. That is, $Sim(S_1, S_2) = \frac{a}{a+b+c}$, where a is the number of predicates in both S_1 and S_2 , b is the number of predicates only in S_1 , and c is the number of predicates only in S_2 .

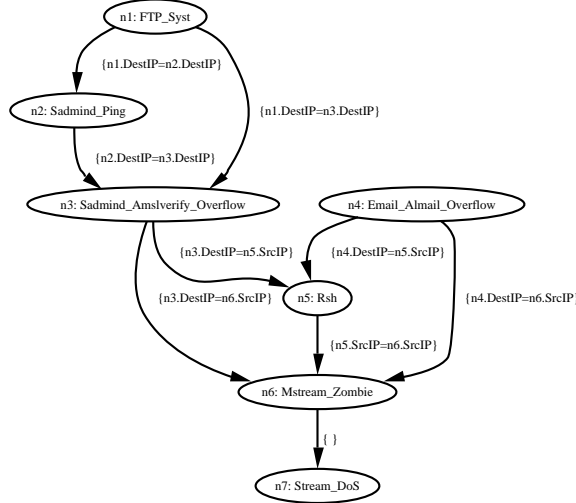


Figure 7: An attack strategy graph extracted in our experiments

Given two hyper-alert types T_1 and T_2 , the similarity between T_1 and T_2 , denoted $Sim(T_1, T_2)$, is then computed as $Sim(T_1, T_2) = Sim(XP_1, XP_2) \times w_p + Sim(XC_1, XC_2) \times w_c$, where XP_1 and XP_2 are the expanded prerequisite sets of T_1 and T_2 , XC_1 and XC_2 are the expanded consequence sets of T_1 and T_2 , and w_p and $w_c = 1 - w_p$ are the weights for prerequisite and consequence, respectively. (In our experiments, we use $w_p = w_c = 0.5$ to give equal weight to both prerequisite and consequence of hyper-alert types.) We may then set a threshold t so that two hyper-alert types are grouped into the same cluster only if their similarity measure is greater than or equal to t .

We have performed a series of experiments to study the proposed techniques [28]. Figure 7 shows one of the attack strategy graphs extracted from the 2000 DARPA intrusion detection scenario specific data set in our experiments. Based on the description of the data set [22], we know that Figure 7 has captured most of the attack strategy. The missing parts are due to the attacks missed by the IDSs. For more information, please refer to [28].

5 Related Work

Intrusion detection has been studied for more than twenty years, since Anderson’s report [2]. A survey of the early work on intrusion detection is given in [24], and an excellent overview of current intrusion detection techniques and related issues can be found in a recent book [3].

Research on intrusion alert correlation has been rather active recently. The first class of approaches (e.g., Spice [34], probabilistic alert correlation [36], and the alert clustering methods in [4] and [17]) 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 [10] or LAMBDA [6], and the data mining approach [7]) 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 [8].

A third class of methods, including JIGSAW [35], the MIRADOR correlation method [5], 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 [35]. 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 [35]), 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 [5], 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 3, our treatment of alert aggregation leads to the three utilities for interactive alert analysis.

A formal model named M2D2 was proposed in [23] 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 [30] 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 languages have been proposed to represent attacks, including STAT [10, 13, 37], Colored-Petri Automata (CPA) [18, 19], LAMBDA [6], and MuSig [20]. 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.

Alert correlation has been studied in the context of network management (e.g., [12], [32], and [11]). 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.

Our approach to learning attack strategies from correlated alerts is also closely related to techniques for static vulnerability analysis (e.g., [1, 16, 31, 33]). In particular, the methods in [1, 33] also use a model of exploits (possible attacks) in terms of their pre-conditions (prerequisites) and post-conditions (consequences). Our approach complements static vulnerability analysis methods by providing the capability of examining the actual execution of attack strategies in different details (e.g., an attacker tries different variations of the same attack), and thus gives human users more information to respond to attacks.

6 Conclusion and Future Work

This paper summarized a series of research efforts towards automating the analysis of intrusion alerts. These efforts start with a practical method for constructing attack scenarios through alert correlation, using prerequisites and consequences of attacks. We 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. 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*. Finally, to automate the analysis of intrusion alerts, we developed a method to extract attack strategies from correlated intrusion alerts.

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., [36]) 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

- [1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, November 2002.
- [2] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PA, 1980.
- [3] R.G. Bace. *Intrusion Detection*. Macmillan Technology Publishing, 2000.
- [4] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001.
- [5] F. Cuppens and A. Mieke. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.
- [6] F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Proc. of Recent Advances in Intrusion Detection (RAID 2000)*, pages 197–216, September 2000.
- [7] 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.
- [8] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, LNCS 2212, pages 85 – 103, 2001.
- [9] DEFCON. Def con capture the flag (CTF) contest. <http://www.defcon.org/html/defcon-8-post.html>, July 2000. Archive accessible at <http://wi2600.org/mediawhore/mirrors/shmoo/>.
- [10] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.
- [11] R. Gardner and D. Harle. Pattern discovery and specification translation for alarm correlation. In *Proceedings of Network Operations and Management Symposium (NOMS'98)*, pages 713–722, February 1998.
- [12] B. Gruschke. Integrated event management: Event correlation using dependency graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, October 1998.
- [13] 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.
- [14] D. A. Jackson, K. M. Somers, and H. H. Harvey. Similarity coefficients: Measures of co-occurrence and association or simply measures of occurrence? *The American Naturalist*, 133(3):436–453, March 1989.
- [15] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
- [16] S. Jha, O. Sheyner, and J.M. Wing. Two formal analyses of attack graphs. In *Proceedings of the 15th Computer Security Foundation Workshop*, June 2002.

- [17] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 12–21, December 2001.
- [18] S. Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, August 1995.
- [19] S. Kumar and E. H. Spafford. A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11–21, October 1994.
- [20] J. Lin, X. S. Wang, and S. Jajodia. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *Proceedings of the 11th Computer Security Foundations Workshop*, pages 190–201, Rockport, MA, June 1998.
- [21] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A data mining analysis of RTID alarms. *Computer Networks*, 34:571–577, 2000.
- [22] MIT Lincoln Lab. 2000 DARPA intrusion detection scenario specific datasets. http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html, 2000.
- [23] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.
- [24] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May 1994.
- [25] P. Ning, Y. Cui, and D. S. Reeves. Analyzing intensive intrusion alerts via correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 74–94, Zurich, Switzerland, October 2002.
- [26] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002.
- [27] P. Ning and D. Xu. Adapting query optimization techniques for efficient intrusion alert correlation. In *Proceedings of the 17th IFIP WG 11.3 Working Conference on Data and Application Security (DAS '03)*, August 2003.
- [28] P. Ning and D. Xu. Learning attack strategies from intrusion alerts. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, October 2003. To appear.
- [29] Packet storm. <http://packetstormsecurity.nl>. Accessed on April 30, 2003.
- [30] P.A. Porras, M.W. Fong, and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 95–114, 2002.
- [31] C.R. Ramakrishnan and R. Sekar. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security*, 10(1/2):189–209, 2002.
- [32] L. Ricciulli and N. Shacham. Modeling correlated alarms in network management systems. In *In Western Simulation Multiconference*, 1997.
- [33] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2002.
- [34] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.
- [35] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*, pages 31 – 38. ACM Press, September 2000.

- [36] 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.
- [37] G. Vigna and R. A. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.