Analyzing Intensive Intrusion Alerts Via Correlation

Peng Ning, Yun Cui, and Douglas S. Reeves

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534
ning@csc.ncsu.edu, ycui4@eos.ncsu.edu, reeves@csc.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 intrusions, 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. Several complementary alert correlation methods have been proposed to address this problem. As one of these methods, we have developed a framework to correlate intrusion alerts using *prerequisites of intrusions*. In this paper, we continue this work to study the feasibility of this method in analyzing real-world, intensive intrusions. In particular, we develop three utilities (called *adjustable graph reduction, focused analysis*, and *graph decomposition*) to facilitate the analysis of large sets of correlated alerts. We study the effectiveness of the alert correlation method and these utilities through a case study with the network traffic captured at the DEF CON 8 Capture the Flag (CTF) event. Our results show that these utilities can simplify the analysis of large amounts of alerts, and also reveals several attack strategies that were repeatedly used in the DEF CON 8 CTF event.

Key Words: Intrusion Detection, Alert Correlation, Attack Scenario Analysis

1 Introduction

Intrusion detection has been considered the second line of defense for computer and network systems along with the prevention-based techniques such as authentication and access control. Intrusion detection techniques can be roughly classified as *anomaly detection* (e.g., NIDES/STAT [1]) and *misuse detection* (e.g., NetSTAT [2]).

Traditional intrusion detection systems (IDSs) focus on low-level attacks or anomalies; they cannot capture the logical steps or attacking 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 intrusions, 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.

To assist the analysis of intrusion alerts, several alert correlation methods (e.g., [3–6]) have been proposed recently to process the alerts reported by IDSs. (Please see

Section 2 for details.) As one of these proposals, we have developed a framework to correlate intrusion alerts using *prerequisites of intrusions* [6].

Our approach is based on the observation that most intrusions are not isolated, but related as different stages of series of attacks, with *the early stages preparing for the later ones*. 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. Our method identifies 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 corresponding alerts by matching the consequences of some previous alerts and the prerequisites of some later ones. For example, if we find a port scan followed by a buffer overflow attack against one of the scanned ports, we can correlate them into the same series of attacks. We have developed an intrusion alert correlator based on this framework [7]. Our initial experiments with the 2000 DARPA intrusion evaluation datasets [8] have shown that our method can successfully correlate related alerts together [6, 7].

This paper continues the aforementioned work to study the effectiveness of this method in analyzing 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 [9], "encountering 10-20,000 alarms per sensor per day is common."

In this paper, we present three utilities (called *adjustable graph reduction, focused analysis*, and *graph decomposition*) that we have developed to facilitate the analysis of large sets of correlated alerts. These utilities are intended for human users to analyze and understand the correlated alerts as well as the strategies behind them. We study the effectiveness of these utilities through a case study with the network traffic captured at the DEF CON 8 Capture the Flag (CTF) event [10]. Our results show that they can effectively simplify the analysis of large amounts of alerts. Our analysis also reveals several attack strategies that appeared in the DEF CON 8 CTF event.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 briefly describes our model for alert correlation. Section 4 introduces three utilities for analyzing hyper-alert correlation graphs. Section 5 describes our case study with the DEF CON 8 CTF dataset. Section 6 concludes this paper and points out some future work.

2 Related Work

Intrusion detection has been studied for about twenty years. An excellent overview of intrusion detection techniques and related issues can be found in a recent book [11].

Several alert correlation techniques have been proposed to facilitate analysis of intrusions. In [3], a probabilistic method was used to correlate alerts using similarity between their features. However, this method is not suitable for fully discovering causal relationships between alerts. In [12], a similar approach was applied to detect stealthy portscans along with several heuristics. Though some such heuristics (e.g., feature sep-

aration heuristics [12]) may be extended to 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 [4]. In particular, the correlation method in [4] 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.

Our method can be considered as a variation of JIGSAW [13]. Both methods try to uncover attack scenarios based on specifications of individual attacks. However, our method also differs from JIGSAW in the following two ways. First, our method allows partial satisfaction of prerequisites (i.e., required capabilities in JIGSAW [13]), 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.

Several languages have been proposed to represent attacks, including STAT [2, 14], Colored-Petri Automata (CPA), LAMBDA [15], and MuSig [16] and its successor [17]. In particular, LAMBDA uses a logic-based method to specify the precondition and postcondition of attack scenarios, which is similar to our method. (See Section 3.) However, all these languages specify entire attack scenarios, which are limited to known scenarios. In contrast, our method (as well as JIGSAW) describes prerequisites and consequences of individual attacks, and correlate detected attacks (i.e., alerts) based on the relationship between these prerequisites and consequences. Thus, our method can potentially correlate alerts from unknown attack scenarios.

3 Preliminary: Alert Correlation Using Prerequisites of Intrusions

In this section, we briefly describe our model for correlating alerts using prerequisites of intrusions. Please read [6] for further details.

The alert correlation model 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, an attacker needs

to install the Distributed Denial of Service (DDOS) daemon programs before he can launch a DDOS attack.

To take advantage of this observation, we correlate alerts 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, an attacker may make progress (e.g., discover a vulnerable service, 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 attackers launch earlier ones to prepare for later ones, there are usually strong connections between the consequences of the earlier attacks and the prerequisites of the later ones.

Accordingly, we identify the prerequisites (e.g., existence of vulnerable services) and the consequences (e.g., discovery of vulnerable services) of each type of attacks and correlate detected attacks (i.e., alerts) by matching the consequences of some previous alerts and the prerequisites of some later ones.

Note that an attacker does not have to perform early attacks to prepare for later ones. For example, an attacker may launch an individual buffer overflow attack against the service blindly. In this case, we cannot, and should not correlate it with others. However, if the attacker does launch attacks with earlier ones preparing for later ones, our method can correlate them, provided the attacks are detected by IDSs.

3.1 Prerequisite and Consequence of Attacks

We use predicates as basic constructs to represent prerequisites and (possible) consequences of attacks. For example, a scanning attack may discover UDP services vulnerable to certain buffer overflow attacks. We can use the predicate *UDPVulnerableToBOF* (*VictimIP, VictimPort*) to represent this discovery. In general, we use a logical formula, i.e., logical combination of predicates, to represent the prerequisite of an attack. Thus, we may have a prerequisite of the form *UDPVulnerableToBOF* (*VictimIP, VictimPort*) $\wedge$ *UDPAccessibleViaFirewall* (*VictimIP, VictimPort*). To simplify the discussion, we restrict the logical operators to $\wedge$ (conjunction) and $\vee$ (disjunction).

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 account as well as modification of the .rhost file. We may use the following set of logical formulas to represent the consequence of this attack: $\{$*GainRootAccess* (*IP*), *rhostModified* (*IP*)$\}$. 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 consequence of an attack is the *possible* result of the attack. In other words, the consequence of an attack is indeed the worst consequence. (Please read [6] for details.) For brevity, we refer to *possible consequence* simply as *consequence* throughout this paper.

3.2 Hyper-alerts and Hyper-alert Correlation Graphs

With predicates as basic constructs, we use a *hyper-alert type* to encode our knowledge about each type of attacks.

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*.

Intuitively, the *fact* component of a hyper-alert type gives the information associated with the alert, *prerequisite* specifies what must be true for the attack to be successful, and *consequence* describes what could be true if the attack indeed succeeds. For brevity, we omit the domains associated with attribute names when they are clear from context.

Example 1 Consider the buffer overflow attack against the *sadmind* remote administration tool. We may have the following hyper-alert type for such attacks: *SadmindBufferOverflow* = ($\{$*VictimIP, VictimPort*$\}$, *ExistHost* (*VictimIP*) $\wedge$ *VulnerableSadmind* (*VictimIP*), $\{$*GainRootAccess*(*VictimIP*)$\}$). Intuitively, this hyper-alert type says that such an attack is against the host running at IP address *VictimIP*. (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 the 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 such an attack.

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 swept 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 *instantiates* its *prerequisite* and *consequence* by replacing the free variables in *prerequisite* and *consequence* with its specific values. 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 *SadmindBufferOverflow* defined in example 1. 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 for-

mulas must be True as the prerequisites of the attack: *ExistHost* (152.141.129.5) $\wedge$ *VulnerableSadmind* (152.141.129.5), *ExistHost* (152.141.129.37) $\wedge$ *VulnerableSadmind* (152.141.129.37), and the following two predicates might be True as 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. □

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

Definition 3 Consider a hyper-alert type T = (*fact, prerequisite, consequence*). The *prerequisite set (or consequence set, resp.) of* T, denoted $P(T)$ (or $C(T)$, resp.), is the set of all such 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.

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.

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*.

Let us further explain the alert correlation method with the following example.

Example 3 Consider the *Sadmind Ping* attack with which an attacker discovers possibly vulnerable *sadmind* services. The corresponding hyper-alert type can be represented by *SadmindPing* = ({*VictimIP, VictimPort*}, *ExistsHost* (*VictimIP*), {*VulnerableSadmind* (*VictimIP*)}). It is easy to see that $P(SadmindPing) = \{ExistHost(VictimIP)\}$, and $C(SadmindPing) = \{VulnerableSadmind(VictimIP)\}$.

Suppose a hyper-alert $h_{SadmindPing}$ of type *SadmindPing* has the following tuples: {(*VictimIP* = 152.141.129.5, *VictimPort* = 1235)}. Then the prerequisite set of $h_{SadmindPing}$ is $P(h_{SadmindPing})$ = {*ExistsHost* (152.141.129.5)}, and the consequence set is $C(h_{SadmindPing})$ = {*VulnerableSadmind* (152.141.129.5)}.

Now consider the hyper-alert $h_{SadmindBOF}$ discussed in Example 2. Similar to $h_{SadmindPing}$, we can easily get $P(h_{SadmindBOF})$ = {*ExistsHost* (152.141.129.5), *ExistsHost* (152.141.129.37), *VulnerableSadmind* (152.141.129.5), *VulnerableSadmind* (152.141.129.37)}, and $C(h_{SadmindBOF})$ = {*GainRootAccess* (152.141.129.5), *GainRootAccess* (152.141.129.37)}.

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) in $P(h_{SadmindBOF})$

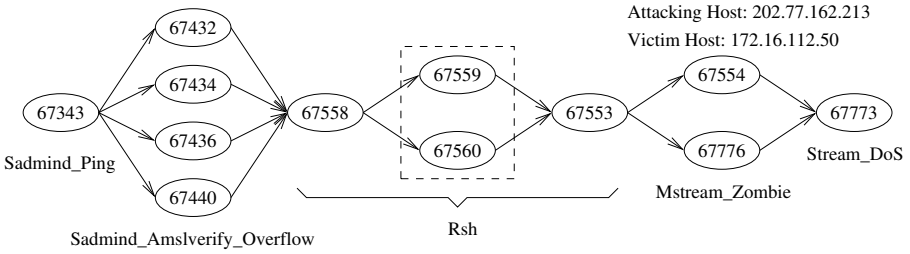

Fig. 1. A hyper-alert correlation graph discovered in the 2000 DARPA intrusion detection evaluation datasets

(among others) is also in $C(h_{SadmindPing})$. Thus, $h_{SadmindPing}$ prepares for, and should be correlated with $h_{SadmindBOF}$. □

The prepare-for relation between hyper-alerts provides a natural way to represent the causal relationship between correlated hyper-alerts. We also introduce the notion of a *hyper-alert correlation graph* to represent a set of correlated hyper-alerts.

Definition 5 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 $n_1, n_2 \in N$, there is a directed edge from n_1 to n_2 in E if and only if n_1 prepares for n_2.

A hyper-alert correlation graph is an intuitive representation of correlated alerts. It can potentially reveal intrusion strategies behind a series of attacks, and thus lead to better understanding of the attacker's intension. We have performed a series of experiments with the 2000 DARPA intrusion detection evaluation datasets [7]. Figure 1 shows one of the hyper-alert correlation graphs discovered from these datasets. Each node in Figure 1 represents a hyper-alert. The numbers inside the nodes are the alert IDs generated by the IDS. This hyper-alert correlation graph clearly shows the strategy behind the sequence of attacks. (For details please refer to [7].)

4 Utilities for Analyzing Intensive Alerts

As demonstrated in [7], the alert correlation method is effective in analyzing small amount of alerts. However, our experience with intrusion intensive datasets (e.g., the DEF CON 8 CTF dataset [10]) has revealed several problems.

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

One may suggest to use time to solve this problem. For example, we may correlate the aforementioned attacks if they happened within t seconds. However, knowing

this method, an attacker may introduce arbitrary delay between these attacks to bypass correlation.

The second problem is the overwhelming information encoded by hyper-alert correlation graphs when intensive intrusions trigger a large amount of alerts. Our initial attempt to correlate the alerts generated for the DEF CON 8 CTF dataset [10] 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 DEF CON 8 dataset involves intensive intrusions which are 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 [9]." 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, 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 user's specification. This may generate hyper-alert correlation graphs much smaller and more comprehensible than the original ones.
3. *Graph decomposition.* Cluster the hyper-alerts in a hyper-alert correlation graph based on the common features shared by the hyper-alerts, and decompose the graph into smaller graphs according to the clusters. This can be considered to combine a variation of the method proposed in [3] with our method.

4.1 Adjustable Reduction of Hyper-alert Correlation Graphs

A natural way to reduce the complexity of a hyper-alert correlation graph is to reduce the number of nodes and edges. However, to make the reduced graph useful, any reasonable reduction should maintain the structure of the corresponding 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 1, 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 1, the edges between the *Rsh* hyper-alerts are subsumed into the aggregated hyper-alert, while the

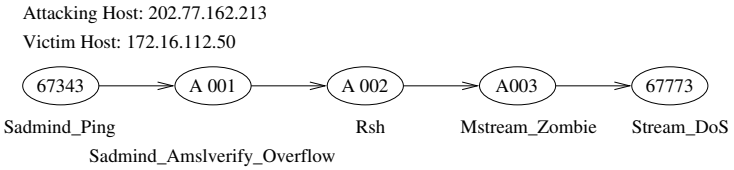

Fig. 2. A hyper-alert correlation graph reduced from Fig. 1

.

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 2.

Reduction of a hyper-alert correlation graph may loose certain 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 misleading results. Nevertheless, our goal is to loose 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, or over simplified. We would like to give a human user more control over the graph reduction process. In the following, we use a simple mechanism to control this process, based on the notion of an interval constraint [6].

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 $|T - T'| < I$.

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

4.2 Focused Analysis

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

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

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

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

4.3 Graph Decomposition Based on Hyper-alert Clusters

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

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

A clustering constraint $C_c(A_1, A_2)$ is *enforceable w.r.t. hyper-alert types* T_1 *and* T_2 if when we represent $C_c(A_1, A_2)$ in a disjunctive normal form, at least for one disjunct C_{ci}, all the attribute names in A_1 appear in T_1 and all the attribute names in A_2 appear in T_2. For example, the above clustering constraint is enforceable w.r.t. T_1 and T_2 if both of them have $SrcIP$ and $DestIP$ in the $fact$ component. Intuitively, a focusing constraint is enforceable w.r.t. T 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 $C_c(A_1, A_2)$ = True for h_1 and h_2. For brevity, we write $C_c(h_1, h_2)$ = True if $C_c(A_1, A_2)$ = 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 h_1 and h_2 in the same cluster do not always imply $C_c(h_1, h_2)$ = True or $C_c(h_2, h_1)$ = True. This is because $C_c(h_1, h_2) \wedge C_c(h_2, h_3)$ does not imply $C_c(h_1, h_3)$, nor $C_c(h_3, h_1)$.

4.4 Discussion

The alert correlation method is developed to uncover the high-level strategies behind a sequence of attacks, not to replace the original alerts reported by an IDS. However, as indicated by our initial experiments [7], alert correlation does provide evidence to differentiate between alerts. If an alert is correlated with some others, it is more possible that the alert corresponds to an actual attack.

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

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

5 Analyzing DEF CON 8 CTF Dataset: A Case Study

To study the effectiveness of the alert correlation method and the utilities proposed in Section 4, we performed a series of experiments on the network traffic collected at the DEF CON 8 CTF event [10]. In our experiments, we used NetPoke[1] to replay the network traffic in an isolated network monitored by a RealSecure Network Sensor 6.0 [18]. 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

[1] NetPoke is a utility used to replay packets to a live network that were previously captured with the tcpdump program. It can be downloaded at http://www.ll.mit.edu/IST/ideval/tools/tools_index.html

Table 1. General statistics of the initial analysis

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

Table 2. Statistics of top 10 uncorrelated hyper-alert types.

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

all the reported alerts. Our alert correlator [7] was then used to process the alerts to discover the hyper-alert correlation graphs. The hyper-alert correlation graphs were visualized using the GraphViz package [19]. For the sake of readability, transitive edges are removed from the graphs.

In these experiments, we mapped each alert type reported by the RealSecure Network Sensor to a hyper-alert type (with the same name). The prerequisite and consequence of each hyper-alert type were specified according to the descriptions of the attack signatures provided with the RealSecure Network Sensor 6.0.

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

5.1 Initial Attempt

In our initial analysis of the DEF CON 8 CTF dataset, we tried to correlate the hyper-alerts without reducing the complexity of any hyper-alert correlation graphs. The statistics of the initial analysis are shown in Table 1.

Table 1 shows that only 15% alerts generated by RealSecure are correlated. In addition, 20 out of 115 hyper-alert types that appear in this data set do not have any instances correlated. Among the remaining 95 hyper-alert types, 51 types have both correlated and uncorrelated instances.

Table 2 shows the statistics of the top 10 uncorrelated hyper-alert types (in terms of the number of uncorrelated hyper-alerts). Among these hyper-alert types, uncorrelated *IPHalfScan* counted 61% of all uncorrelated hyper-alerts. *Windows_Access_Error* counted 21% of all uncorrelated alerts. According to the description provided by RealSecure, a *Windows_Access_Error* represents an unsuccessful file sharing connection to a Windows or Samba server, which usually results from an attempt to brute-force a

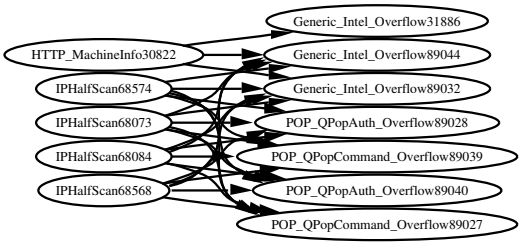

Fig. 3. A small hyper-alert correlation discovered in initial analysis

login under another account's privileges. It is easy to see that the corresponding attacks could hardly prepare for any other attacks (since they failed). The third largest hyper-alert type *HTTP_Cookie* counted for 3.3% of the total alerts. Though such alerts have certain privacy implications, we do not treat them as attacks, considering the nature of the DEF CON CTF events. These three hyper-alert types counted for 74.5% of all the alerts. We omit the discussion of the other uncorrelated hyper-alerts.

Figure 3 shows one of the small hyper-alert correlation graphs. The text in each node is the type followed by the ID of the hyper-alert. All the hyper-alerts in this figure were destined to the host at 010.020.001.024. All the *IPHalfScan* attacks were from source IP 010.020.011.240 at source port 55533 or 55534, and destined to port 110 at the victim host. After these attacks, all the attacks in the second stage except for 31886 were from 010.020.012.093 and targeted at port 110 of the victim host. The only two hyper-alerts that were not targeted at port 110 are hyper-alert 30882, which was destined to port 80 of the victim host, and hyper-alert 31886, which was destined to port 53. Thus, it is very possible that all the hyper-alerts except for 30882 and 31886 were related.

Not all of the hyper-alert correlation graphs are as small and comprehensible as Figure 3. In particular, the largest graph (in terms of the number of nodes) has 2,940 nodes and 25,321 edges, and on average, each graph has 21.75 nodes and 310.56 edges. Obviously, most of the hyper-alert correlation graphs are too big to understand for a human user.

5.2 Graph Reduction

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

We first applied graph reduction utility to the hyper-alert correlation graphs. Figure 4 shows the fully reduced graph. Compared with the original graph, which has 2,940 nodes and 25,321 edges, the fully reduced graph has 77 nodes and 347 edges (including transitive edges).

The fully reduced graph in Figure 4 shows 7 stages of attacks. The layout of this graph was generated by GraphViz [19], which tries to reduce the number of cross edges and make the graph more balanced. As a result, the graph does not reflect the actual stages of attacks. Nevertheless, Figure 4 provides a much clearer outline of the attacks.

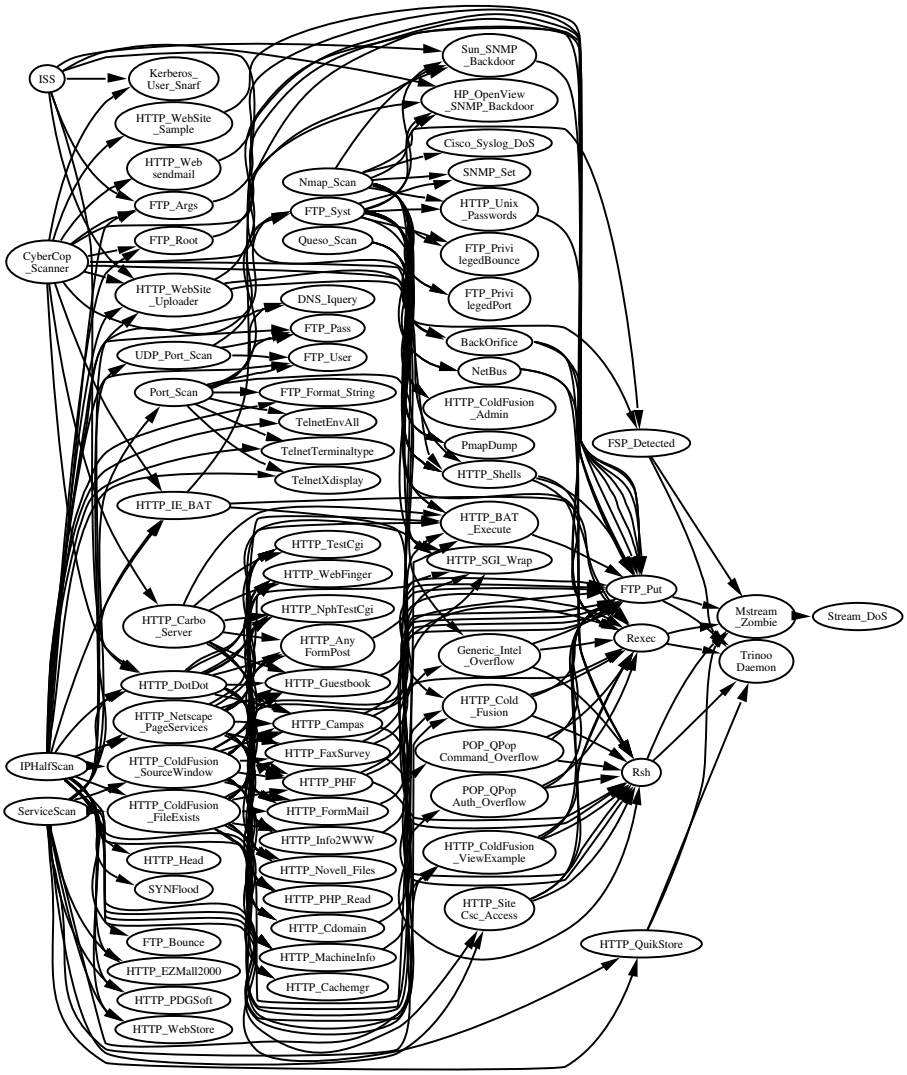

Fig. 4. The fully reduced graph for the largest aggregated hyper-alert correlation graph.

The hyper-alerts in stage 1 and about half of those in stage 2 correspond to scanning attacks or attacks to gain information of the target systems (e.g., *ISS, Port_Scan*). The upper part of stage 2 include attacks that may lead to execution of arbitrary code on a target system (e.g., *HTTP_WebSite_Sample*). Indeed, these hyper-alerts directly prepare for some hyper-alerts in stage 5, but GraphViz arranged them in stage 2, possibly to balance the graph. Stages 3 consists of a mix of scanning attacks (e.g., *Nmap_Scan*), attacks that reveal system information (e.g,, *HTTP_PHP_Read*), and attacks that may

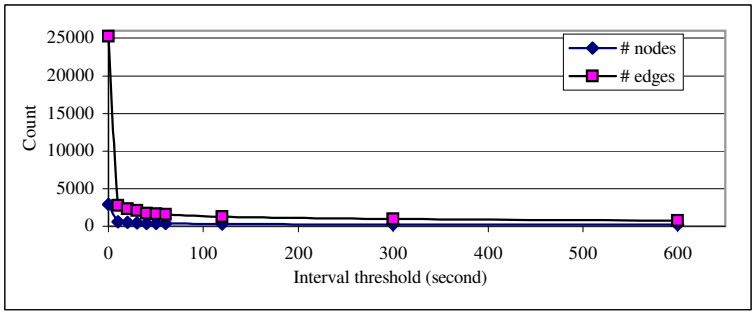

Fig. 5. Sizes of the reduced graphs w.r.t. the interval threshold for the largest hyper-alert correlation graph

lead to execution of arbitrary code (e.g., *HTTP_Campas*). Stage 4 mainly consists of buffer overflow attacks (e.g., *POP_QPopCommand_Overflow*), detection of backdoor programs (e.g., *BackOrifice*), and attacks that may lead to execution of arbitrary code. The next 3 stages are much cleaner. Stage 5 consists of attacks that may be used to copy programs to target hosts, stage 6 consists of detection of two types of DDOS (Distributed Denial of Service) daemon programs, and finally, stage 7 consists of the detection of an actual DDOS attack.

Note that the fully reduce graph in Figure 4 is an approximation to the strategies used by the attackers. Hyper-alerts for different, independent sequences of attacks may be aggregated together in such a graph. For example, if two individual attackers use the sequence of attacks (e.g., using the same script downloaded from a website) to attack the same target, the corresponding hyper-alerts may be correlated and aggregated in the same fully reduced graph. Nevertheless, a fully reduced graph can clearly outline the attack strategies, and help a user understand the overall situation of attacks.

As we discussed earlier, the reduction of hyper-alert correlation graphs can be controlled with interval constraints. Figure 5 shows the numbers of nodes and edges of the reduced graphs for different interval sizes. The shapes of the two curves in Figure 5 indicate that most of the hyper-alerts that are of the same type occurred close to each other in time. Thus, the numbers of nodes and edges have a deep drop for small interval thresholds and a flat tail for large ones. A reasonable guess is that some attackers tried the same type of attacks several times before they succeeded or gave up. Due to space reasons, we do not show these reduced graphs.

5.3 Focused Analysis

Focused analysis can help filter out the interesting parts of a large hyper-alert correlation graph. It is particularly useful when a user knows the systems being protected or the potential on-going attacks. For example, a user may perform a focused analysis with focusing constraint $DestIP = ServerIP$, where $ServerIP$ is the IP address of a critical server, to find out attacks targeted at the server. As another example, he/she may

use $SrcIP = ServerIP \vee DestIP = ServerIP$ to find out attacks targeted at or originated from the server, suspecting that the server may have been compromised.

In our experiments, we tried a number of focusing constraints after we learned some information about the systems involved in the CTF event. Among these focusing constraints are (1) $C_{f1} : (DestIP = 010.020.001.010)$ and (2) $C_{f2} : (SrcIP = 010.020.011.251 \wedge DestIP = 010.020.001.010)$. We applied both focusing constraints to the largest hyper-alert correlation graph. The results consist of 2154 nodes and 19423 edges for C_{f1}, and 51 nodes and 28 edges for C_{f2}. The corresponding fully reduced graphs are shown in Fig. 6 and Fig. 7, respectively. (Isolated nodes are shown in gray.) These two graphs also appear in the results of graph decomposition (Section 5.4). We defer the discussion of these two graphs to the next subsection.

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

5.4 Graph Decomposition

We applied three clustering constraints to decompose the largest hyper-alert correlation graph discovered in Section 5.1. In all these clustering constraints, we let $A_1 = A_2 = \{SrcIP, DestIP\}$.

1. $C_{c1}(A_1, A_2)$: $A_1.DestIP = A_2.DestIP$. This is to cluster all hyper-alerts that share the same destination IP addresses. Since most of attacks are targeted at the hosts at the destination IP addresses, this is to cluster hyper-alerts in terms of the victim systems.
2. $C_{c2}(A_1, A_2)$: $A_1.SrcIP = A_2.SrcIP \wedge A_1.DestIP = A_2.DestIP$. This is to cluster all the hyper-alerts that share the same source and destination IP addresses.
3. $C_{c3}(A_1, A_2)$: $A_1.SrcIP = A_2.SrcIP \vee A_1.DestIP = A_2.DestIP \vee A_1.SrcIP = A_2.DestIP \vee A_1.DestIP = A_2.SrcIP$. This is to cluster all the hyper-alerts that are connected via common IP addresses. Note that with this constraint, hyper-alerts in the same cluster may not share the same IP address directly, but they may connect to each other via other hyper-alerts.

Table 3 shows the statistics of the decomposed graphs. C_{c1} resulted in 12 clusters, among which 10 clusters contain edges. C_{c2} resulted in 185 clusters, among which 37 contain edges. Due to space reasons, we only show the first 12 clusters for C_{c2}. C_{c3} in effect removes one hyper-alert from the original graph. This hyper-alert is *Stream_DoS*, which does not share the same IP address with any other hyper-alerts. This is because the source IPs of *Stream_DoS* are spoofed and the destination IP is the target of this attack. This result shows that all the hyper-alerts except for *Stream_DoS* share a common IP address with some others.

The isolated nodes in the resulting graphs are the hyper-alerts that prepare for or are prepared for by those that do not satisfy the same clustering constraints. Note that having isolated hyper-alerts in a decomposed graph does not imply that the isolated

Table 3. Statistics of decomposing the largest hyper-alert correlation graph.

	# clusters	# graphs	cluster ID	1	2	3	4	5	6	7	8	9	10	11	12
C_{c1}	12	10	# connected nodes	2154	224	105	227	83	11	54	28	0	23	6	0
			# edges	19423	1966	388	2741	412	30	251	51	0	26	5	0
			# isolated nodes	0	0	0	0	0	0	0	0	1	0	0	4
C_{c2}	185	37	# correlated nodes	1970	17	0	12	0	0	0	3	0	29	0	0
			# edges	2240	66	0	10	0	0	0	2	0	28	0	0
			# isolated nodes	3	0	21	17	35	26	15	12	4	22	13	26
C_{c3}	2	1	# connected nodes	2935	0	–	–	–	–	–	–	–	–	–	–
			# edges	25293	0	–	–	–	–	–	–	–	–	–	–
			# isolated nodes	4	1	–	–	–	–	–	–	–	–	–	–

hyper-alerts are correlated incorrectly. For example, an attacker may hack into a host with a buffer overflow attack, install a DDOS daemon, and start the daemon program, which then tries to contact its master program. The corresponding alerts (i.e., the detection of the buffer overflow attack and the daemon's message) will certainly not have the same destination IP address, though they are related.

Figures 6 and 7 show a decomposed graph for C_{c1} and C_{c2}, respectively. Both graphs are fully reduced to save space. All the hyper-alerts in Figure 6 are destined to 010.020.001.010. Figure 6 shows several possible attack strategies. The most obvious ones are those that lead to the *Mstream_Zoombie* and *TrinooDaemon*. However, there are multiple paths that lead to these two hyper-alerts. Considering the fact that multiple attackers participated in the DEF CON 8 CTF event, we cannot conclude which path caused the installation of these daemon programs. Indeed, it is possible that none of them is the actual way, since the IDS may have missed some attacks.

Figure 6 involves 75 source IP addresses, including IP address 216.136.173.152, which does not belong to the CTF subnet. We believe that these attacks belong to different sequences of attacks, since there were intensive attacks from multiple attackers who participated in the CTF event.

Figure 7 is related to Figure 6, since they both are about destination IP address 010.020.001.010. Indeed, Figure 7 is a part of Figure 6, though in Figure 6, *ISS* prepares for *HTTP_Campas* through *HTTP_DotDot*. Since all the hyper-alerts in Figure 7 have the same source and destination IP addresses, it is very possible that the correlated ones belong to the same sequence of attacks. Note that *HP_OpenView_SNMP_Backdoor* appears as both connected and isolated node. This is because some instances are correlated, while the others are isolated.

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

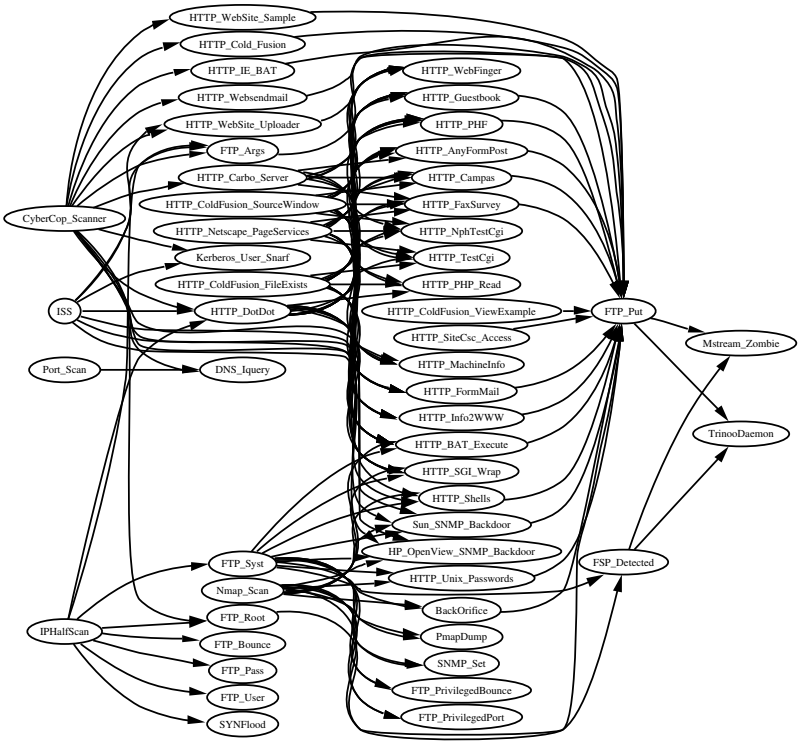

Fig. 6. A fully reduced hyper-alert correlation graph resulting from graph decomposition with C_{c1}. (Cluster ID = 1; DestIP = 010.020.001.010.)

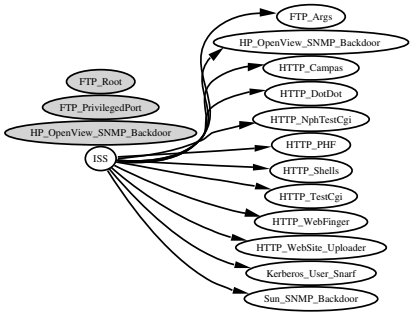

Fig. 7. A fully reduced hyper-alert correlation graph resulting from graph decomposition with C_{c2}. (Cluster ID = 10; SrcIP = 010.020.011.251; DestIP = 010.020.001.010.)

As another strategy, the attacker(s) at 010.020.011.240 used a concise sequence of attacks against the host at 010.020.001.013: *Nmap_Scan* followed by *PmapDump* and

then *ToolTalk_Overflow*. Obviously, they used *Nmap_Scan* to find the *portmap* service, then used *PmapDump* to list the RPC services, and finally launched a *ToolTalk_Overflow* attack against the *ToolTalk* service. Indeed, the sequence *Nmap_Scan* followed by *PmapDump* with the same source and destination IP address appeared many times in this dataset.

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

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

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

When relax the restriction to allow hyper-alert correlation graphs involving different source IP but still with the same destination IP addresses (i.e., with clustering constraint C_{c1}), we have graphs with more stages. Figure 6 is one such fully reduced hyper-alert correlation graph. However, due to the amount of alerts and source IP addresses involved in this graph, it is difficult to conclude which hyper-alerts belong to the same sequences of attacks.

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

6 Conclusion and Future Work

In this paper, we presented three utilities, *adjustable graph reduction, focused analysis*, and *graph decomposition*), which were developed to facilitate the analysis of large sets of correlated alerts. We studied the effectiveness of these utilities through a case study with the DEF CON 8 CTF dataset [10]. Our results show that these utilities can simplify the analysis of large amounts of alerts. In addition, our analysis reveals several attack strategies that are repeatedly used in the DEF CON 8 CTF event.

Due to the nature of the DEF CON 8 CTF dataset, we were unable to evaluate the successful rate of the alert correlation method. Although we have found several attack strategies, we also encountered several situations where it is difficult to conclude about attack strategies. Indeed, a hyper-alert correlation graph is an approximation to a real attack strategy. Thus, the hyper-alert correlation method and the three utilities should be used with caution.

Our future work includes several problems, including seeking more techniques to improve alert correlation, automating the analysis process, further refinement of our toolkit, and systematic development of hyper-alert types.

References

1. Javits, H., Valdes, A.: The NIDES statistical component: Description and justification. Technical report, SRI International, Computer Science Laboratory (1993)
2. Vigna, G., Kemmerer, R.A.: NetSTAT: A network-based intrusion detection system. Journal of Computer Security **7** (1999) 37–71
3. Valdes, A., Skinner, K.: Probabilistic alert correlation. In: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001). (2001) 54–68
4. Debar, H., Wespi, A.: Aggregation and correlation of intrusion-detection alerts. In: Recent Advances in Intrusion Detection. LNCS 2212 (2001) 85 – 103
5. Dain, O., Cunningham, R.: Fusing a heterogeneous alert stream into scenarios. In: Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications. (2001) 1–13
6. Ning, P., Reeves, D., Cui, Y.: Correlating alerts using prerequisites of intrusions. Technical Report TR-2001-13, North Carolina State University, Dept. of Computer Science (2001)
7. Ning, P., Cui, Y.: An intrusion alert correlator based on prerequisites of intrusions. Technical Report TR-2002-01, North Carolina State University, Dept. of Computer Science (2002)
8. MIT Lincoln Lab: DARPA 2000 intrusion detection evaluation datasets. http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html (2000)
9. Manganaris, S., Christensen, M., Zerkle, D., Hermiz, K.: A data mining analysis of RTID alarms. Computer Networks **34** (2000) 571–577
10. DEFCON: Def con capture the flag (CTF) contest. http://www.defcon.org/html/defcon-8-post.html (2000) Archive accessible at http://wi2600.org/mediawhore/mirrors/shmoo/.
11. Bace, R.: Intrusion Detection. Macmillan Technology Publishing (2000)
12. Staniford, S., Hoagland, J., McAlerney, J.: Practical automated detection of stealthy portscans. To appear in Journal of Computer Security (2002)
13. Templeton, S., Levit, K.: A requires/provides model for computer attacks. In: Proceedings of New Security Paradigms Workshop, ACM Press (2000) 31 – 38
14. Ilgun, K., Kemmerer, R.A., Porras, P.A.: State transition analysis: A rule-based intrusion detection approach. IEEE Transaction on Software Engineering **21** (1995) 181–199
15. Cuppens, F., Ortalo, R.: LAMBDA: A language to model a database for detection of attacks. In: Proc. of Recent Advances in Intrusion Detection (RAID 2000). (2000) 197–216
16. Lin, J., Wang, X.S., Jajodia, S.: Abstraction-based misuse detection: High-level specifications and adaptable strategies. In: Proceedings of the 11th Computer Security Foundations Workshop, Rockport, MA (1998) 190–201
17. Ning, P., Jajodia, S., Wang, X.S.: Abstraction-based intrusion detection in distributed environments. ACM Transactions on Information and System Security **4** (2001) 407–452
18. ISS, I.: RealSecure intrusion detection system. (http://www.iss.net)
19. AT & T Research Labs: Graphviz - open source graph layout and drawing software. (http://www.research.att.com/sw/tools/graphviz/)