Building Attack Scenarios through Integration of Complementary Alert Correlation Methods *

Peng Ning, Dingbang Xu, Christopher G. Healey, and Robert St. Amant
Cyber Defense Laboratory
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8207

Abstract

Several alert correlation methods were proposed in the past several years to construct high-level attack scenarios from low-level intrusion alerts reported by intrusion detection systems (IDSs). These correlation methods have different strengths and limitations; none of them clearly dominate the others. However, all of these methods depend heavily on the underlying IDSs, and perform poorly when the IDSs miss critical attacks. In order to improve the performance of intrusion alert correlation and reduce the impact of missed attacks, this paper presents a series of techniques to integrate two complementary *types of alert correlation methods: (1) those based on the similarity between alert attributes, and (2) those based on prerequisites and consequences of attacks. In particular, this paper presents techniques to hypothesize and reason about attacks possibly missed by IDSs based on the indirect causal relationship between intrusion alerts and the constraints they must satisfy. This paper also discusses additional techniques to validate the hypothesized attacks through raw audit data and to consolidate the hypothesized attacks to generate concise attack scenarios. The experimental results in this paper demonstrate the potential of these techniques in building high-level attack scenarios and reasoning about possibly missed attacks.*

1. Introduction

It is well-known that current intrusion detection systems (IDSs) produce large numbers of alerts, including both actual and false alerts. The high volume and the low quality of intrusion alerts (i.e., missed attacks and false alerts) make it a very challenging task for human users or intrusion response systems to understand the alerts and take appropriate actions. Thus, it is usually necessary to construct high-level attack scenarios from a large collection of low-level intrusion alerts.

Several alert correlation techniques have been proposed to facilitate the analysis of intrusion alerts, including approaches based on the similarity between alert attributes [4, 7, 25, 27], previously known (or partially known) attack scenarios [8, 9], and prerequisites and consequences of known attacks [5, 19]. A common requirement of these approaches is that they all depend on underlying IDSs for alerts. This means the results of alert correlation are limited to the abilities of the IDSs. In particular, if the IDSs miss critical attacks, the results of alert correlation will not reflect the true attack scenario, and thus provide misleading information.

In this paper, we develop a series of techniques to *construct high-level attack scenarios even if the IDSs miss critical attacks.* Our approach is to integrate complementary intrusion alert correlation methods and use the intrinsic relationships between possibly related attacks to hypothesize and reason about attacks missed by the IDSs. We are particularly interested in two types of alert correlation methods: correlation based on prerequisites and consequences of attacks [5, 19] (which we call *causal correlation methods*, since they are intended to discover the causal relationships between alerts), and correlation based on similarity between alert attribute values [4, 7, 25, 27] (which we call *clustering correlation method*). Because these two methods correlate alerts using different mechanisms, combining them can potentially lead to better correlation results. Our main contribution in this paper is a series of techniques to integrate the causal and the clustering correlation methods, and to use the results to hypothesize and reason about attacks possibly missed by the IDSs. These techniques are critical to constructing high-

*The authors would like to thank the anonymous reviewers for their valuable comments. This work is partially supported by the National Science Foundation (NSF) under grants ITR-0219315 and CCR-0207297, and by the U.S. Army Research Office (ARO) under grant DAAD19-02-1-0219.

level attack scenarios from low-level intrusion alerts in situations where the IDSs cannot guarantee to detect all attacks. These techniques complement the underlying IDSs by hypothesizing and reasoning about missed attacks, and thus can provide valuable additional evidences to support further intrusion investigation and response.

Our approach starts with a straightforward combination of the causal and the clustering correlation methods. Specifically, we first correlate the same set of intrusion alerts with both methods independently, then combine results from the causal correlation method (represented as correlation graphs) using results from the clustering correlation method. For example, if the clustering correlation method decides that two alerts in two separate correlation graphs are very similar (e.g., same target, close timestamps, etc.), we combine these two correlation graphs into one. We then develop techniques to hypothesize about attacks possibly missed by the IDSs, especially unknown variations of known attacks.

We observe that if two attacks are causally related through some intermediate attacks, they usually satisfy certain conditions (e.g., sharing the same destination IP address), even if they are not directly adjacent in a sequence of attacks. The attribute values of the corresponding alerts should also satisfy such conditions. This observation provides another opportunity to reason about the hypothesized attacks by inferring their attribute values. Moreover, the hypothesized attacks can be further validated through raw audit data. For example, we might hypothesize that variations of `IMAP_Authen_Overflow` and/or `RPC_Cmsd_Overflow` were missed by the IDSs. However, if during the target time frame, there is only IMAP traffic and no RPC traffic to the target host, we can easily conclude that the latter hypothesis is impossible. Finally, to improve the usability of the constructed attack scenarios, we present a technique to consolidate the hypothesized attacks and generate concise representations of the high-level attack scenarios.

The remainder of this paper is organized as follows. The next section briefly describes a specific causal correlation method, on the basis of which our techniques are developed. Section 3 presents our techniques to integrate clustering and causal correlation methods, including approaches to hypothesizing and reasoning about attacks possibly missed by IDSs, methods to validate hypothesized attacks using raw audit data, and ways to consolidate hypothesized attacks. Section 4 reports our experimental results to test the effectiveness of our techniques. Section 5 discusses related work, and Section 6 concludes this paper and points out some future research directions. The Appendix gives more information about the alert types used in our experiments.

2. Previous Work: Alert Correlation Using Prerequisites of Attacks

The new techniques in this paper are built on the basis of the causal correlation method proposed in [19], which we briefly describe below.

The approach in [19] correlates intrusion alerts using the prerequisites and consequences of 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. The *consequence* of an attack is the possible outcome of the attack. For example, gaining local access as root from a remote machine may be the consequence of a ftp buffer overflow attack. In a series of attacks where earlier ones are launched to prepare for later ones, there are usually 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 attacks, and correlate detected attacks (i.e., alerts) by matching the consequences of previous alerts to the prerequisites of later ones.

The correlation method uses logical formulas, which are logical combinations of predicates, to represent the prerequisites and consequences of attacks. For example, a scanning attack may discover UDP services vulnerable to certain buffer overflow attacks. Then the predicate *UDPVulnerableToBOF* (*VictimIP, VictimPort*) may be used to represent this discovery.

The correlation model formally represents the prerequisites and consequences of known attacks as hyper-alert types. A *hyper-alert type* is a triple (*fact, prerequisite, consequence*), where *fact* is a set of alert attribute names, *prerequisite* is a logical formula whose free variables are all in *fact*, and *consequence* is a set of logical formulas such that all the free variables in *consequence* are in *fact*. Intuitively, a hyper-alert type encodes the knowledge about the corresponding type of attacks. Given a hyper-alert type *T* = (*fact, prerequisite, consequence*), a *type T hyper-alert h* 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 correlation process is used to identify *prepare-for* relations between hyper-alerts. Intuitively, this relation exists if an earlier hyper-alert *contributes* to the prerequisite of a later one. In the formal model, correlations are identified via prerequisite and consequence sets. Given a hyper-alert type *T* = (*fact, prerequisite, consequence*), the *prerequisite set* (or *consequence set*) of T, denoted $Prereq(T)$ (or $Conseq(T)$),

(a) CG_1 (b) CG_2

Figure 1. Two correlation graphs

is the set of all predicates that appear in *prerequisite* (or *consequence*). The *expanded consequence set* of T, denoted $ExpConseq(T)$, is the set of all predicates that are implied by $Conseq(T)$. (This is computed using the implication relationships between predicates [19].) Thus, $Conseq(T) \subseteq ExpConseq(T)$. Given a type T hyper-alert h, the *prerequisite set*, *consequence set*, and *expanded consequence set* of h, denoted $Prereq(h)$, $Conseq(h)$, and $ExpConseq(h)$), respectively, are the predicates in $Prereq(T)$, $Conseq(T)$, and $ExpConseq(T)$ with arguments replaced by the corresponding attribute values of each tuple in h. Each element in $Prereq(h)$, $Conseq(h)$, or $ExpConseq(h)$ is associated with the timestamp of the corresponding tuple in h. Hyper-alert h_1 *prepares for* hyper-alert h_2 if there exist $p \in Prereq(h_2)$ and $c \in ExpConseq(h_1)$ such that $p = c$ and $c.end\_time < p.begin\_time$.

A hyper-alert correlation graph is used to represent a set of correlated hyper-alerts. A *hyper-alert correlation graph* $CG = (N, E)$ is a connected directed acyclic graph, where N 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. For brevity, we refer to a hyper-alert correlation graph as a *correlation graph* in this paper.

3. Integrating Complementary Alert Correlation Methods

To integrate the causal and the clustering correlation methods, we combine correlation graphs generated by the causal method using the results from the clustering methods. Intuitively, if the IDSs miss certain attacks, alerts from the same attack scenario may be split across several correlation graphs. The clustering methods have the potential to identify the common features shared by these alerts, and therefore to help re-integrate the relevant correlation graphs together.

The integration process can be conceptually divided into two steps: (1) identify the correlation graphs to be integrated, and (2) determine possible causal relationships between alerts in different correlation graphs. In this paper, we choose a simple technique for the first step: we integrate two correlation graphs when they both contain alerts from a common cluster. For example, given the two correlation graphs shown in Figures 1(a) and 1(b) (The string inside each node is a hyper-alert type name followed by an alert ID), if the clustering method groups `SCAN_NMAP_TCP2` and `Rsh3` in the same cluster based on their common source and destination IP addresses, we consider integrating these two graphs together.

Because this simple technique works well in practice, our main focus in this paper is on the second step: how to identify causal relationships between alerts in different correlation graphs. This is challenging, since we must deal with missed attacks that cause an attack scenario to split into multiple correlation graphs.

To meet our goal, we developed a series of techniques to integrate multiple correlation graphs. We start with a straightforward method, and gradually add more sophisticated techniques into our approach. Without loss of generality, we assume we integrate two correlation graphs in the following discussion.

3.1. Combining Related Correlation Graphs

A straightforward approach to combining two correlation graphs is to use the prior knowledge of attacks and the alert timestamp information to hypothesize about the possible causal relationships between alerts in different correlation graphs. For example, suppose an attacker uses *nmap* to find out a vulnerable service, then uses a buffer overflow attack to compromise that service, and finally installs and starts a DDOS daemon program. When we observe an earlier `SCAN_NMAP_TCP` and a later `Mstream_Zombie` alert in two correlation graphs that are identified for integration, we may hypothesize that the `SCAN_NMAP_TCP` alert indirectly prepares for the `Mstream_Zombie` alert through an unknown attack (or an unknown variation of a known attack, e.g., a variation of the above buffer overflow attack). As a result, we would add a hypothesized indirect causal relationship between these two alerts.

To further characterize this intuition and facilitate later discussion, we introduce two definitions. (Note that Definition 1 is based on the model in [19], which has been described briefly in Section 2.) For convenience, we denote the type of a hyper-alert h as $Type(h)$.

Definition 1 Given two hyper-alert types T and T', we say T *may prepare for* T' if $ExpConseq(T)$ and $Prereq(T')$ share at least one predicate (with possibly different arguments). Given a set $\mathcal{T}$ of hyper-alert types, we say T *may indirectly prepare for* T' *w.r.t.* $\mathcal{T}$ if there exists a sequence of hyper-alert types $T, T_1, ..., T_k, T'$ such that (1) all these hyper-alert types are in $\mathcal{T}$, and (2) T *may prepare for* T_1, T_i *may prepare for* T_{i+1}, where $i = 1, 2, ..., k-1$, and T_k *may prepare for* T'.

Intuitively, given two hyper-alert types T and T', T *may*

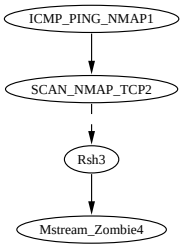


Figure 2. A straightforward combination of CG_1 and CG_2

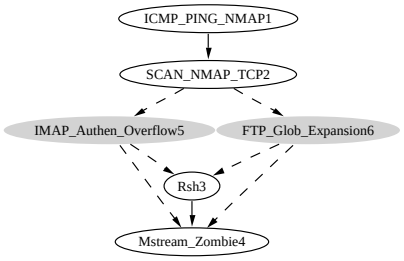


Figure 3. Integration of CG_1 and CG_2 with hypotheses of missed attacks

prepare for T' if there exist a type T hyper-alert h and a type T' hyper-alert h' such that h *prepares for* h'.

Definition 2 Given a set $\mathcal{T}$ of hyper-alert types and two hyper-alerts h and h', where $Type(h)$ and $Type(h') \in \mathcal{T}$ and $h'.begin\_time > h.end\_time$, h *may indirectly prepare for* h' if $Type(h)$ *may indirectly prepare for* $Type(h')$ w.r.t. $\mathcal{T}$. Given a sequence of hyper-alerts h, h_1, ..., h_k, h' where $k > 0$, h *indirectly prepares for* h' if h *prepares for* h_1, h_i *prepares for* h_{i+1} for $i = 1, ..., k-1$, and h_k *prepares for* h'.

Intuitively, h *may indirectly prepare for* h' if there *could* exist a path from h to h' in a correlation graph (with additional hyper-alerts), while h *indirectly prepares for* h' if such hyper-alerts do exist. We are particularly interested in the case where h *may indirectly prepare for* h' but there do not exist additional hyper-alerts showing that h *indirectly prepares for* h'. Indeed, a possible reason for such a situation is that the IDSs miss some critical attacks, which, if detected, would lead to additional hyper-alerts showing that h *indirectly prepares for* h'.

A simple way to take advantage of the above observation is to assume a possible causal relationship between hyper-alerts h and h' if they belong to different correlation graphs and h *may indirectly prepare for* h'. Let us continue the example in Figure 1. If `SCAN_NMAP_TCP` *may prepare for* `FTP_Glob_Expansion`, which *may prepare for* `Rsh`, then we have `SCAN_NMAP_TCP` *may indirectly prepare for* `Rsh`. Thus, we may hypothesize that `SCAN_NMAP_TCP2` *indirectly prepares for* `Rsh3`. We add a *virtual edge*, displayed in a dashed line, from `SCAN_NMAP_TCP2` to `Rsh3` in Figure 2, indicating that there may be some attacks between them that are missed by the IDSs.

Though this simple approach can integrate related hyper-alert correlation graphs and hypothesize about possible causal relationships between alerts, it is limited in several ways. First, the virtual edges generated with this approach provide no information about attacks possibly missed by the IDSs. Second, the virtual edges are determined solely on the basis of prior knowledge about attacks. There is no "reality check." Though a clustering correlation method combines several hyper-alert correlation graphs together, the hypothesized virtual edges are not necessarily true due to the limitations of the clustering correlation method and the lack of information about the missed attacks.

3.2. Hypothesizing Missed Attacks

The *may-prepare-for* and *may-indirectly-prepare-for* relations identified in Definitions 1 and 2 provide additional opportunities to hypothesize and reason about missed attacks, especially unknown variations of known attacks.

Consider two hyper-alerts h and h' that belong to different correlation graphs prior to integration. If h *may indirectly prepare for* h', we can then identify possible sequences of hyper-alert types in the form of $T_1, T_2, ..., T_k$ such that $Type(h)$ *may prepare for* T_1, T_i *may prepare for* T_{i+1}, $i = 1, 2, ..., k-1$, and T_k *may prepare for* $Type(h')$. These sequences of hyper-alert types are candidates of attacks possibly missed by the IDSs. (More precisely, variations of these attacks, which could be used by an attacker and then missed by the IDSs, are the actual candidates of missed attacks.) We can then search in the alerts and/or the raw audit data between h and h' to check for signs of these attacks (or their variations). For example, to continue the example in Figure 2, we may hypothesize that variations of either `IMAP_Authen_Overflow`, or `FTP_Glob_Expansion`, or both may have been missed by the IDSs based on our prior knowledge about attacks. To better present these hypotheses, we may add the hypothesized attacks into the correlation graph as virtual nodes (displayed in gray). Figure 3 shows the resulting correlation graph.

To facilitate hypothesizing about missed attacks, we encode our knowledge of the relationships between hyper-alert types in a *hyper-alert type graph*, or simply a *type graph*. Let us first introduce the concept of *equality constraint*, which was originally defined in [20], to help for-

mally describe the notion of type graph.

Definition 3 (equality constraint [20]) 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 of $u_1 = v_1 \wedge \cdots \wedge u_n = v_n$, where $u_1, \cdots, u_n$ are attribute names in T_1 and $v_1, \cdots, v_n$ are attribute names in T_2, such that there exist $p(u_1, \cdots, u_n)$ and $p(v_1, \cdots, v_n)$, 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 $t_1.u_1 = t_2.v_1 \wedge \cdots \wedge t_1.u_n = t_2.v_n$ evaluates to True.

An equality constraint characterizes the relationships between the attribute values of two hyper-alerts when one of them *prepares for* the other. 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 equality constraint. 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 [20].

Given a set of hyper-alert types (representing the known attacks), we can derive all possible *may-prepare-for* relations between them together with the corresponding equality constraints. This information can help us understand how these known attacks may be combined to launch sequences of attacks, and thus hypothesize about which attacks (more precisely, their variations) may be missed when we observe alerts that *may indirectly prepare for* each other. The following definition formally captures this intuition.

Definition 4 Given a set $\mathcal{T}$ of hyper-alert types, a *(hyper-alert) type graph* TG over $\mathcal{T}$ is a quadruple (N, E, T, C), where

(1) (N, E) is a directed graph,

(2) T is a bijective mapping from N to $\mathcal{T}$, which maps each node in N to a hyper-alert type in $\mathcal{T}$,

(3) there is an edge (n_1, n_2) in E if and only if $T(n_1)$ *may prepare for* $T(n_2)$, and

(4) C is a mapping that maps each edge (n_1, n_2) in E to a set of equality constraints associated with $(T(n_1), T(n_2))$.

Example 1 Consider the following set of hyper-alert types: $\mathcal{T}$={`ICMP_PING_NMAP`, `SCAN_NMAP_TCP`, `IMAP_Authen_Overflow`, `FTP_Glob_Expansion`, `Rsh`, `Mstream_Zombie`}. (We include the specifications of these hyper-alert types in Appendix A.) We can compute the type graph over $\mathcal{T}$ as shown in Figure 4. The string inside each node is the node name followed by the hyper-alert type name. The label of each edge is the corresponding set of equality constraints.

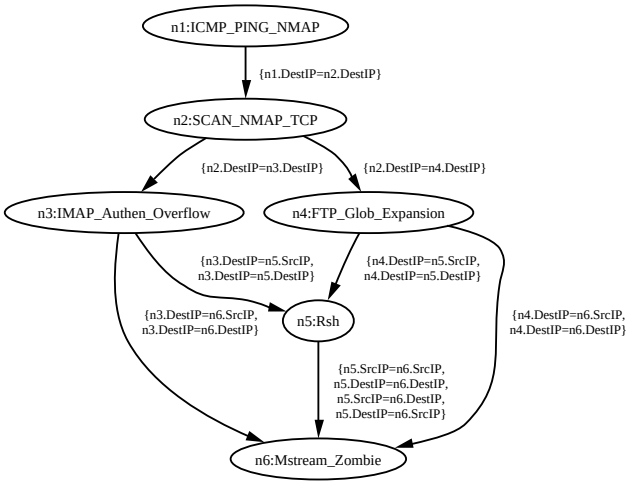


Figure 4. An example type graph

Obviously, given multiple correlation graphs that may be integrated together, we can hypothesize about possibly missed attacks that break the attack scenario according to the type graph. Let us revisit the example in Figure 1. Given the type graph in Figure 4, we can *systematically* hypothesize that the IDSs may have missed variations of `IMAP_Authen_Overflow` and/or `FTP_Glob_Expansion` attacks. As a result, we obtain the integrated correlation graph shown in Figure 3.

3.3. Reasoning about Missed Attacks

Consider nodes $n2$, $n3$, and $n5$ in Figure 4. There is an equality constraint $n2.DestIP = n3.DestIP$ associated with ($n2$, $n3$), and two equality constraints $n3.DestIP = n5.SrcIP$ and $n3.DestIP = n5.DestIP$ associated with ($n3$, $n5$). These imply that $n2.DestIP = n5.SrcIP$ or $n2.DestIP = n5.DestIP$. In other words, if a type `SCAN_NMAP_TCP` hyper-alert *indirectly prepares for* a type `Rsh` hyper-alert (through a type `IMAP_Authen_Overflow` hyper-alert), they must satisfy one of these two equations. We obtain the same equations if we consider nodes $n2$, $n4$, and $n5$ in Figure 4. In general, we can derive constraints for two hyper-alert types when one of them *may indirectly prepare for* the other. Informally, we call such a constraint an *indirect equality constraint*. These constraints can be used to study whether two hyper-alerts in two different correlation

graphs could be indirectly related. This in turn allows us to filter out incorrectly hypothesized attacks.

Indirect equality constraints can be considered a generalization of the equality constraints specified in Definition 3. In this paper, we combine the terminology and simply refer to an indirect equality constraint as an equality constraint when it is not necessary to distinguish between them.

To take advantage of the above observation, we must derive indirect equality constraints. Algorithm 1 (shown in Figure 5) outlines an approach for generating the set of indirect equality constraints between two hyper-alert types T and T' where T *may indirectly prepare for* T'. Intuitively, for each pair of hyper-alert types T and T', Algorithm 1 identifies all paths from T to T' in the type graph, and computes an indirect equality constraint for each combination of equality constraints between consecutive hyper-alert types along the path. The basic idea is to propagate the equality relationships between attributes of hyper-alert types. Each indirect equality constraint is labeled with the corresponding path that produces the constraint. This information provides further guideline in hypothesizing missed attacks. The usefulness of Algorithm 1 is guaranteed by Lemma 1.

Lemma 1 *Consider a type graph TG and two hyper-alerts h and h', where $Type(h)$ and $Type(h')$ are in TG. Assume Algorithm 1 outputs a set C of equality constraints for $Type(h)$ and $Type(h')$. If C is non-empty and h indirectly prepares for h', then h and h' must satisfy at least one equality constraint in C.*

PROOF. According to Definition 2, if h *indirectly prepares for* h', there must exists a sequence of hyper-alerts h_1, ..., h_k, where $k > 0$, such that h *prepares for* h_1, h_i *prepares for* h_{i+1} for $i = 1, ..., k-1$, and h_k *prepares for* h'. Thus, we have $Type(h)$ *may prepare for* $Type(h_1)$, $Type(h_i)$ *may prepare for* $Type(h_{i+1})$ for $i = 1, ..., k-1$, and $Type(h_k)$ *may prepare for* $Type(h')$. Following the convention of Algorithm 1, we denote $Type(h)$ as T_0, $Type(h_i)$ as T_i, where $i = 1, ..., k$, and $Type(h')$ as T_{k+1}. It is easy to see there must be a path $T_0, T_1, ..., T_{k+1}$ in the corresponding type graph TG. For convenience, we also denote h as h_0, and h' as h_{k+1}.

According to [20], if h_i *prepares for* h_{i+1}, then h_i and h_{i+1} must satisfy at least one equality constraint for (T_i, T_{i+1}). For $i = 0, 1, ..., k$, we denote the constraint h_i and h_{i+1} satisfy as C_i. According to Figure 5, Algorithm 1 will process the path $T_0, T_1, ..., T_{k+1}$ (in step 2) and the combination of equality constraints $C_0, C_1, ..., C_{k+1}$ that $h_0, h_1, ..., h_{k+1}$ satisfy (in step 4).

Now consider steps 5 to 9. For each $S(T_0.a_i)$, we can prove by induction that all attributes $T_j.b$ added into $S(T_0.a_i)$ are equal to $T_0.a_i$, since each addition is based on a conjunct $T_{j-1}.a = T_j.b$, where $T_{j-1}.a$ is already in $S(T_0.a)$. Further because step 9 removes the attributes of T_{j-1}, only attributes of T_{k+1} remain in $S(T_0.a_i)$, $i = 1, 2, ..., l$. Thus, after step 9, each $S(T_0.a_i)$ includes all the attributes of T_{k+1} that are equal to $T_0.a_i$, where $i = 1, 2, ..., l$. Steps 10 to 13 then transform these equality relations into a conjunctive formula ec. Since the sequence of constraints C_i, $i = 0, 1, ..., k$, where each C_i is satisfied by h_i and h_{i+1}, is used in the above process, we can easily conclude that h_0 (h) and h_{k+1} (h') satisfy ec. Thus, if h *indirectly prepares for* h', they must satisfy at least one equality constraint in C. □

Note that Algorithm 1 is meant to illustrate the basic idea behind computing indirect equality constraints. In practice, we compute the indirect equality constraints for all pairs of hyper-alert types simultaneously as we traverse through the paths between a pair of hyper-alert types, and thus significantly reduce the required computation time. Moreover, we need to perform this computation only once after we determine the set of known attacks. Thus, the performance of this algorithm will not impact the performance of intrusion analysis greatly.

Example 2 Consider the type graph in Figure 4 and two hyper-alert types `SCAN_NMAP_TCP` (node $n2$) and `Rsh` (node $n5$). Using Algorithm 1, we can easily compute the indirect equality constraints for them: $\{n2.DestIP = n5.DestIP, n2.DestIP = n5.SrcIP\}$. Both indirect equality constraints are labeled with the following two paths: ⟨`SCAN_NMAP_TCP`, `IMAP_Authen_Overflow`, `Rsh`⟩ and ⟨`SCAN_NMAP_TCP`, `FTP_Glob_Expansion`, `Rsh`⟩. Moreover, we can derive the sets of equality constraints for all pairs of hyper-alert types in Figure 4 where one of the pair *may (indirectly) prepare for* the other. Table 1 shows the results. (To save space, we use node names to represent the corresponding hyper-alert types.) Each cell in the table contains the equality constraints for the hyper-alert types in the given row and the column, where the row *may (indirectly) prepare for* the column.

The equality constraints derived for indirectly related hyper-alert types can be used to improve the hypotheses of missed attacks. Given two correlation graphs G and G' that may be integrated together, we can check each pair of hyper-alerts h and h', where h *may indirectly prepare for* h', and h and h' belong to G and G', respectively. If h and h' satisfy at least one equality constraint for $(Type(h), Type(h'))$, then we have consistent evidence that supports the hypothesis that h *indirectly prepares for* h' (through some missed attacks). Moreover, for each equality constraint ec that h and h' satisfy, we

Algorithm 1. Computation of Indirect Equality Constraints
Input: A type graph TG, and two hyper-alert types T and T' in TG, where T *may indirectly prepare for* T'.
Output: A set of equality constraints for T and T'.
Method:

1. Let $Result = \emptyset$.
2. For each path $T, T_1, ..., T_k, T'$ from T to T' in TG
3. Denote T as T_0, and T' as T_{k+1}.
4. For each combination of constraints $C_1, C_2, ..., C_{k+1}$, where C_i is an equality constraint for (T_{i-1}, T_i)
5. Let $S(T_0.a_i) = \{T_0.a_i\}$, where $T_0.a_i$, $i = 1, 2, ..., l$, are all the attributes of T_0 that appear in C_1.
6. For $j = 1$ to $k + 1$
7. For each conjunct $T_{j-1}.a = T_j.b$ in C_j
8. For each $S(T_0.a_i)$ that contains $T_{j-1}.a$, let $S(T_0.a_i) = S(T_0.a_i) \cup \{T_j.b\}$.
9. Remove variables of T_{j-1} from each $S(T_0.a_i)$, $i = 1, 2, ..., l$.
10. Let $temp = \emptyset$.
11. For each non-empty $S(T_0.a_i)$ and each $T_{k+1}.b$ in $S(T_0.a_i)$
12. Let $temp = temp \cup \{T_0.a_i = T_{k+1}.b\}$.
13. Let ec be the conjunction of all elements in $temp$.
14. If ec is in $Result$ then
15. Let $Label(ec) = Label(ec) \cup \{\langle T, T_1, ..., T_k, T' \rangle\}$
16. else Let $Label(ec) = \{\langle T, T_1, ..., T_k, T' \rangle\}$, and $Result = Result \cup \{ec\}$.
17. Return $Result$.

Figure 5. Algorithm to compute indirect equality constraints for two hyper-alert types

Table 1. Equality constraints for hyper-alert types in Figure 4 where one *may (indirectly) prepare for* the other.

	n1	n2	n3	n4	n5	n6
n1	/	{n1.DestIP=n2.DestIP}	{n1.DestIP=n3.DestIP}	{n1.DestIP=n4.DestIP}	{n1.DestIP=n5.DestIP, n1.DestIP=n5.SrcIP}	{n1.DestIP=n6.DestIP, n1.DestIP=n6.SrcIP}
n2	/	/	{n2.DestIP=n3.DestIP}	{n2.DestIP=n4.DestIP}	{n2.DestIP=n5.DestIP, n2.DestIP=n5.SrcIP}	{n2.DestIP=n6.DestIP, n2.DestIP=n6.SrcIP}
n3	/	/	/	/	{n3.DestIP=n5.DestIP, n3.DestIP=n5.SrcIP}	{n3.DestIP=n6.DestIP, n3.DestIP=n6.SrcIP}
n4	/	/	/	/	{n4.DestIP=n5.DestIP, n4.DestIP=n5.SrcIP}	{n4.DestIP=n6.DestIP, n4.DestIP=n6.SrcIP}
n5	/	/	/	/	/	{n5.SrcIP=n6.SrcIP, n5.DestIP=n6.DestIP, n5.SrcIP=n6.DestIP, n5.DestIP=n6.SrcIP}
n6	/	/	/	/	/	/

can add the paths in $Label(ec)$ into the integrated correlation graph. Note that each path in $Label(ec)$ is in the form of $Type(h), T_1, ..., T_k, Type(h')$. $Type(h)$ and $Type(h')$ are then merged with h and h', respectively, and the rest of the path is added as a virtual path (i.e., virtual nodes and edges) from h to h'.

3.4. Validating through Raw Audit Data

The hypothesized attacks can be further validated using raw audit data. For example, we may hypothesize there is a variation of `FTP_Glob_Expansion` attack between a `SCAN_NMAP_TCP` alert and a `Rsh` alert. However, if there is no ftp activity related to the victim host between these two alerts, we can easily conclude that our hypothesis is incorrect. By doing so we can further narrow the hypothesized attacks down to meaningful ones.

To take advantage of this observation, we extend our model to associate a "filtering condition" with each hyper-alert type. Assume that the raw audit data set consists of a sequence of audit records, and we can extract a set of attributes from each audit record directly, or through inference. For example, we may extract the source IP address from a tcpdump record directly, or infer the type of service using the port and payload information. For the sake of presentation, we call such attributes obtained from the raw audit data *audit attributes*. Given a set A of audit attributes and a hyper-alert type T, a *filtering condition for T w.r.t. A* is a logical formula involving audit attribute names in A, which evaluates to True or False if the audit attribute names are replaced with specific values. Intuitively, a filtering condition for T w.r.t. A must be True if an attack corresponding to T or its variations indeed happen.

Using filtering conditions is indeed straightforward. When we hypothesize a sequence of missed attacks based on two hyper-alerts h and h', we know that all the times-

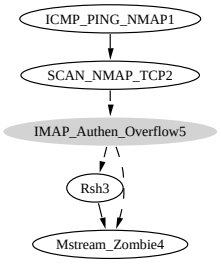


Figure 6. Integration of CG_1 and CG_2 after refinement with raw audit data

tamps of the hypothesized attacks are between those of h and h'. Thus, to validate a hypothesized attack, we can search through the raw audit records during the aforementioned time period, and evaluate the filtering condition for the hypothesized attack using the values of the audit attributes extracted from each raw audit record. To continue the earlier example, we may associate a filtering condition *protocol=ftp* with the hyper-alert type `FTP_Glob_Expansion`. If there is no ftp traffic between the hyper-alerts `SCAN_NMAP_TCP2` and `Rsh3`, this condition will evaluate to False for all audit records, and we can conclude that no ftp based attack is missed by the IDSs. As a result, the integrated correlation graph in Figure 3 can be refined to the one in Figure 6.

A limitation of using filtering conditions is that human users must specify the conditions associated with each hyper-alert type. It has at least two implications. First, it could be time consuming to specify such conditions for every known attack. Second, human users may make mistakes during the specification of filtering conditions. In particular, a filtering condition could be too specific to capture the invariant among the variations of a known attack, or too general to filter out enough incorrect hypotheses. Nevertheless, we observe that any filtering condition may help reduce incorrectly hypothesized attacks, even if it is very general. In our experiments, we simply use the protocols over which the attacks are carried out as filtering conditions. It is interesting to study how to get the "right" way to specify filtering conditions. We consider this problem outside of the scope of this paper; we will investigate it in our future work.

Another issue is the execution cost. To filter out a hypothesized attack with a filtering condition, we have to examine every audit record during the period of time when the attack could have happened. Though there are many ways to optimize the filtering process (e.g., indexing, concurrent examination), the cost is not negligible, especially when the time period is large. Thus, filtering conditions are more suitable for off-line analysis such as forensic analysis.

3.5. Consolidating Hypothesized Attacks

In the earlier discussion, we focused on hypothesizing and reasoning about missed attacks. However, this method does not consider the possibility that the same hypothesized attack could be related to multiple hyper-alerts in different correlation graphs. As a result, the integrated correlation graph with hypothesized attacks could be overly complex. In particular, there could be multiple hypothesized attacks for one missed attack. Though it is possible that the same hypothesized attacks are repeated multiple times, having too many uncertain details reduces the usability of the integrated correlation graph.

To deal with the above problem, we consolidate the hypothesized attacks based on their types and their inferred attribute values. The general idea is to merge the hypothesized attacks (and thus reduce the complexity of the integrated correlation graph) as long as it is possible that the hypothesized attacks to be merged are the same attack. In other words, we aggregate the same type of hypothesized attacks if their inferable attribute values are consistent.

The attribute values of a hypothesized attack can be inferred from the hyper-alerts that lead to this hypothesis. For example, if we hypothesize that an `IMAP_Authen_Overflow` attack after a `SCAN_NMAP_TCP` alert and before a `Rsh` alert such that `SCAN_NMAP_TCP` prepares for `IMAP_Authen_Overflow`, which then prepares for `Rsh`, then from Table 1 we know that `SCAN_NMAP_TCP` and `IMAP_Authen_Overflow` have the same destination IP address, and the destination IP address involved in `IMAP_Authen_Overflow` is the same as either the source or the destination IP address in `Rsh`. In general, we can use the equality constraints between the actual alerts and the hypothesized attacks to infer the possible attribute values of these attacks. As a special attribute, we estimate the timestamp of a hypothesized attack with a possible range. If an attack T is hypothesized with two actual alerts h and h', where h occurs before h', then the possible range of T's timestamp is $(h.end\_time, h'.begin\_time)$.

To reduce the number of hypothesized attacks and make the integrated correlation graph easy to understand, we take an aggressive approach. Specifically, we aggregate two hypothesized attacks together if they (1) belong to the same type, (2) share the same values on their common inferable attributes, and (3) the ranges of their timestamps overlap. An example of consolidation is what we performed in our experiments, which consolidated about 300 hypothesized attacks into 16. In the following, we outline the approach to consolidating hypothesized attacks.

The consolidation process is performed in three steps. First, we get the inferable attribute values for all hypothesized attacks. Second, we partition the hypothesized at-

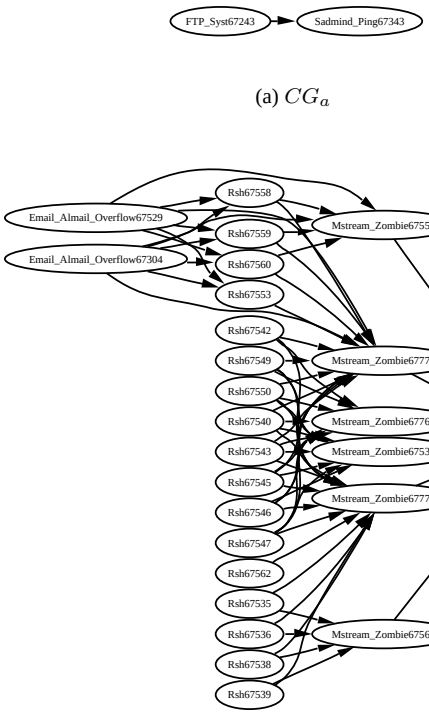
tacks into groups such that all hypothesized attacks in the same group have the same hyper-alert type, their attribute values do not have conflict values (i.e., if two hypothesized attacks have inferable values on the same attribute, these two values should be the same), and possible ranges of their timestamps overlap. We then aggregate each group into one hypothesized attack. Third, we merge virtual edges between the same pair of (aggregated) hypothesized attacks. The resulting correlation graph is the final integrated correlation graph.

Among the three steps, the only non-trivial step is the inference of attribute values for hypothesized attacks. We achieve this by revising Algorithm 1. In Steps 5 o 9, in addition to maintaining the sets $S(T_0.a_i)$ only for attributes of T_0, we construct such sets for all attributes of each hyper-alert type T_{j-1} that appear in the equality constraint C_j. As a result, for each combination of equality constraints mentioned in Step 4 and each T_j, $j = 1, 2, ..., k$, we can get an equality constraint for T and each T_j, denoted ec_j, and another equality constraint for T_j and T', denoted ec'_j. Given a type T hyper-alert h and a type T' hyper-alert h', if T, T_1, ..., T_k, T' are the hypothesized path that leads to h *indirectly preparing for* h' and C_1, C_2, ..., C_{k+1} are the corresponding sequence of equality constraints, then for each hypothesized attack T_j, T_j must satisfy both ec_j and ec'_j. Thus, we can infer the attributes involved in ec_j and ec'_j using h and h'. To keep this information, each T_j should maintain ec_j and ec'_j w.r.t. T and T'.

We shall point out that each hypothesized attack may correspond to multiple actual attacks in the consolidated correlation graph. In other words, each hypothesized attack in an integrated correlation graph is indeed a place holder for one or several possible attacks.

4. Experimental Results

To examine the effectiveness of the proposed techniques, we performed a series of experiments using one of the 2000 DARPA intrusion detection scenario specific data sets, LLDOS 1.0 [15]. LLDOS 1.0 contains a series of attacks in which an attacker probed, broke-in, installed the components necessary to launch a Distributed Denial of Service (DDOS) attack, and actually launched a DDOS attack against an off-site server. The network audit data were collected in both the DMZ and the inside parts of the evaluation network. We used RealSecure Network Sensor 6.0 [11] as the IDS sensor to generate alerts, and the NCSU Intrusion Alert Correlator [18] to correlate these alerts into correlation graphs. To validate the hypothesized attacks using raw audit data, we used Ethereal [3], a network protocol analyzer, to extract information from the raw tcpdump file (i.e., the network audit data). We used Graphviz [2] to visualize correlation graphs.



(a) CG_a

(b) CG_b

Figure 7. Two correlation graphs constructed from LLDOS 1.0 inside traffic

Our current implementation is still semi-automatic. Our prototype can automatically decide what hyper-alert correlation graphs to integrate, hypothesize possibly missed attacks, and consolidate the hypothesized attacks. However, the validation of hypothesized attacks with raw audit data still has to be handled semi-manually. Specifically, we use Ethereal [3] to extract the protocol information for each packet, and then validate the hypothesized attacks accordingly. We are currently refining our implementation so that all steps can be automated.

In our experiments, we use a simple clustering correlation method. That is, we cluster alerts together as long as they involve the same destination IP address. Since our goal is to examine the effectiveness of the integration techniques, we believe this is acceptable for our experiments. In practice, more sophisticated clustering correlation methods (e.g., those proposed in [27]) are certainly needed for better performance.

To test the ability of our techniques to hypothesize and reason about missed attacks, we dropped all `Sadmind_Amslverify_Overflow` alerts that RealSecure Network Sensor detected in LLDOS1.0. As a

result, the attack scenarios that the Intrusion Alert Correlator output before dropping these alerts are all split into multiple parts, some of which become individual, uncorrelated alerts. Figure 7 shows two of these correlation graphs constructed from the inside traffic in LLDOS 1.0.

Now let us focus on the correlation graphs in Figure 7. As mentioned earlier, we cluster alerts together if they share the same destination IP address. Since the destination IP addresses of both Sadmind_Ping67343 (in Figure 7(a)) and Rsh67553 (in Figure 7(b)) are 172.16.112.50, they belong to the same cluster. We integrate two correlation graphs as long as they both have alerts in the same cluster. Thus, Figures 7(a) and 7(b) should be integrated together.

We consider all types of alerts generated by the RealSecure Network Sensor 6.0 in our type graph. The specification of the corresponding hyper-alert types are given in Appendix A, and the type graph is given in Figure 8. For space reasons, we did not put the isolated nodes (the nodes which do not have edges connecting to them) into the type graph. Based on the type graph, we can easily hypothesize that variations of FTP_Syst, HTTP_Shells, and Sadmind_Amslverify_Overflow could have been missed by the IDS. For example, there could be variations of Sadmind_Amslverify_Overflow between Sadmind_Ping and any later Rsh alert. By reasoning about the hypothesized attacks using equality constraints, we rule out some of these hypothesized attacks. For example, the destination IP address of Sadmind_Ping67343 is 172.16.112.50, which is different from either the source or the destination IP address of Rsh67543. Thus, Sadmind_Ping67343 cannot *indirectly prepare for* Rsh67543 through a variation of Sadmind_Amslverify_Overflow.

The hypothesized attacks are further validated using the raw audit data. In our experiments, we use the inferred attribute values and the protocol that carries the corresponding attack as the filtering condition for each hyper-alert type. For example, the filtering condition for (variations of) FTP_Put is $protocol = ftp$ plus all the inferable attributes. All the hypothesized attacks are then checked using the audit records between the hyper-alerts that result in the hypothesized attacks. For example, we search all the IP packets between Sadmind_Ping67343 and Rsh67560 for RPC packets in order to validate a hypothesized (variation of) Sadmind_Amslverify_Overflow attack (which is a RPC based attack).

We continue the above process to integrate the resulting correlation graph with additional ones generated by the Intrusion Alert Correlator. We are able to further integrate hyper-alerts Sadmind_Ping67286 and Sadmind_Ping67341, which are both uncorrelated alerts. As a slight difference, FTP_Put are hypothesized during both integration processes, but all hypothesized FTP_Put attacks are invalidated later using raw audit data. In other words, we find no ftp activities involving the corresponding host during the time frame when the hypothesized attacks might happen. Figure 9 shows the integrated correlation graph after the hypothesized attacks are consolidated. The hypothesized attacks are shown in gray, and are labeled by the corresponding hyper-alert type followed by an ID to distinguish between different instances of the same type of attacks.

Now let us examine the integrated correlation graph in Figure 9. According to the description of the data sets [15], the three Sadmind_Amslverify_Overflow attacks and the *prepare-for* relations between these attacks and the other alerts are all hypothesized correctly. However, the FTP_Put and HTTP_Shells attacks are hypothesized incorrectly. In addition, the gray Rsh nodes in Figure 9 result from the different combinations of equality constraints between hyper-alert types (Their attribute values were inferred through existing alerts, and they cannot be further consolidated with other alerts or hypothesized attacks. In our future work, we may look for a way to further reduce them).

We also performed the experiments using the DMZ data set in LLDOS 1.0. Similar to the inside data set, we deliberately dropped all Sadmind_Amslverify_Overflow alerts from those generated by RealSecure Network Sensor 6.0. Using the type graph in Figure 8, we generated three integrated correlation graphs in Figure 10, in which hypothesized attacks are shown in gray. Based on the reasoning of the hypothesized attacks, we know the destination IP addresses of Sadmind_Amslverify_Overflow4, Sadmind_Amslverify_Overflow5 and Sadmind_Amslverify_Overflow6 are 172.16.115.20, 172.16.112.10 and 172.16.112.50, respectively. Similarly, the destination IP address of HTTP_Shells2 is 172.16.112.50. According to the description of the data sets [15], the Sadmind_Amslverify_Overflow attacks are all hypothesized correctly, while the HTTP_Shells attack is hypothesized incorrectly. These experiment results (including LLDOS 1.0 inside and DMZ data sets) indicate that though the proposed techniques can identify missed attacks, they are still not perfect. Nevertheless, the proposed techniques have already exceeded the limitation of IDSs.

The experimental evaluation reported in this paper is still preliminary, though they have demonstrated the potential of the proposed techniques. To further understand the capability of these techniques, a more detailed, quantitative evaluation is required. We will perform such eval-

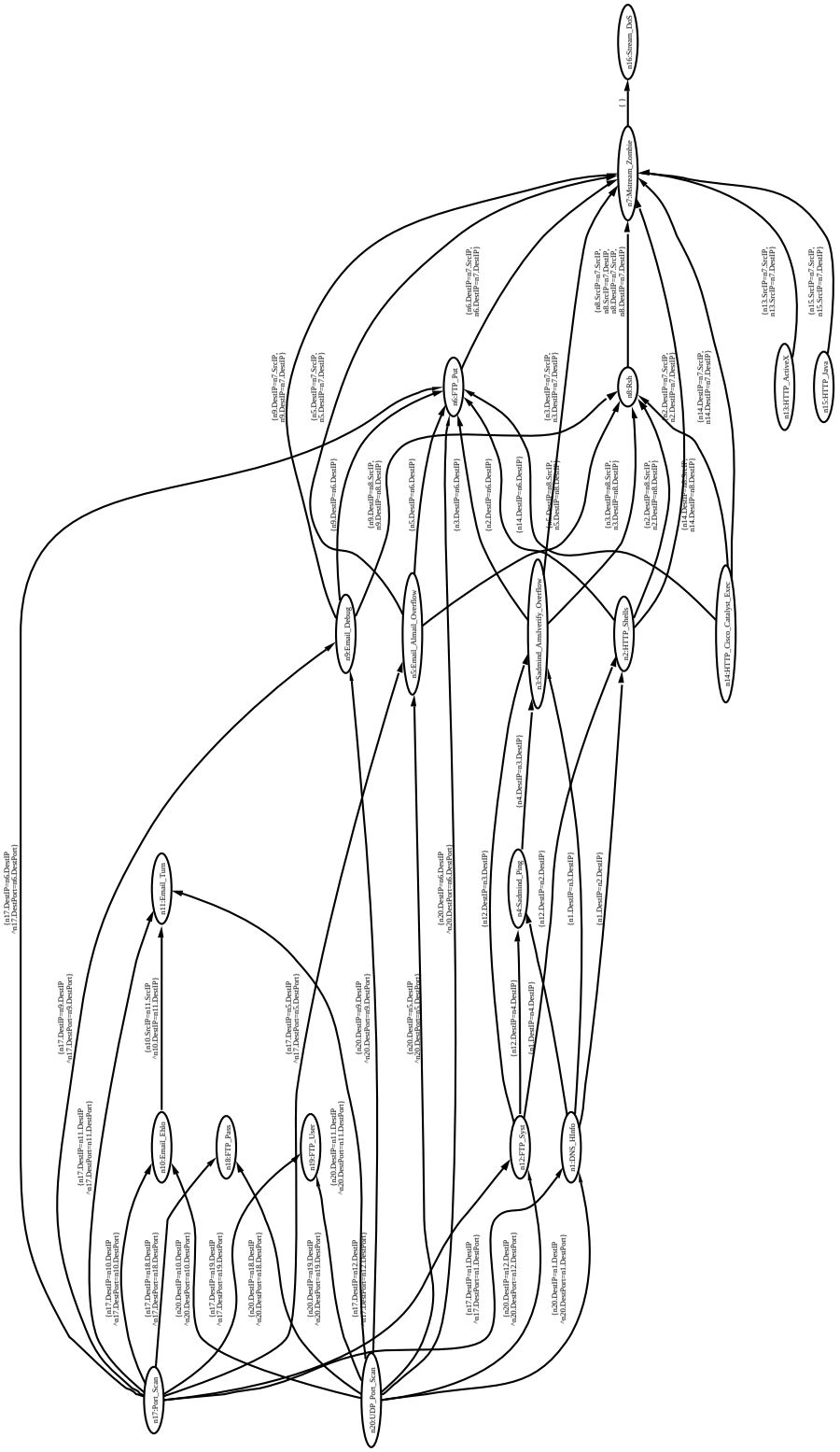


Figure 8. The type graph used in our experiments

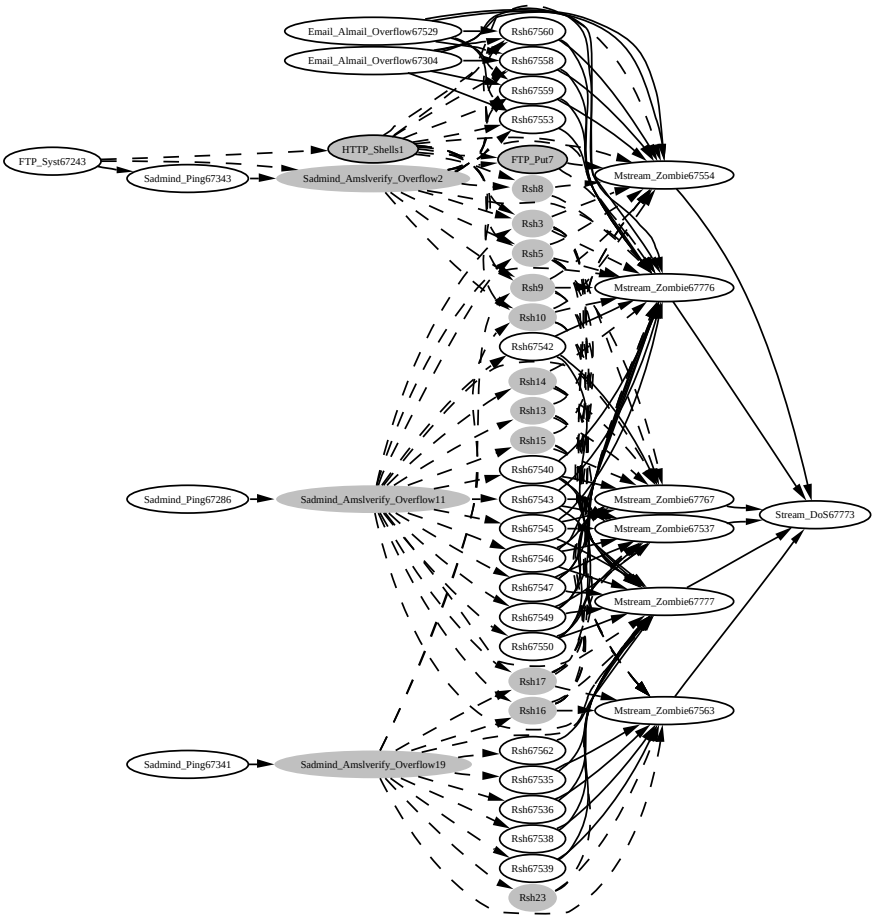


Figure 9. The integrated correlation graph constructed from LLDOS 1.0 inside traffic

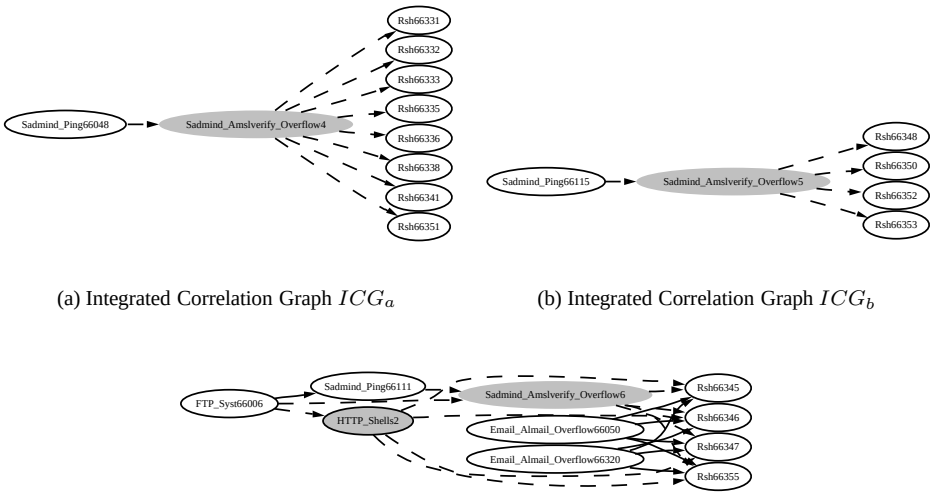
uations in our future research.

5. Related Work

Our work in this paper is closely related to the recent results in intrusion alert correlation. In particular, our techniques integrate the causal correlation methods in [5, 19] and the clustering correlation methods in [4, 7, 25, 27]. In addition to correlating intrusion alerts, our techniques allow to further hypothesize and reason about attacks possibly missed by the IDSs, and thus can potentially exceed the limitation of IDSs.

There are other alert correlation techniques. The Tivoli approach correlates alerts based on the observation that some alerts usually occur in sequence [9]. M2D2 correlates alerts by fusing information from multiple sources besides intrusion alerts, such as the characteristics of the monitored systems and the vulnerability information [17], thus having a potential to result in better results than those simply looking at intrusion alerts. The mission-impact-based approach correlates alerts raised by INFOSEC devices such as IDSs and firewalls with the importance of system assets [21]. The alert clustering techniques in [13, 14] use conceptual clustering and generalization hierarchy to aggregate alerts into clusters. An interesting approach was proposed recently to apply statistical tests to identify causal relationships among aggregated alerts [22]. Alert correlation may also be performed by matching attack scenarios specified by attack languages. Examples of such languages include STATL [10], LAMBDA [6], JIGSAW [26] and Chronicles [16]. We consider these techniques as complementary to ours.

Our approach is also related to techniques for static vulnerability analysis (e.g., [1, 12, 23, 24]). In particular, the methods in [1, 24] also use a model of exploits (possible attacks) in terms of their pre-conditions (prerequisites) and post-conditions (consequences) to construct possible sequences of attacks. However, our method aims to construct high-level attack scenarios from low-level intrusion alerts and reason about attacks possibly missed by the IDSs, while the vulnerability analysis techniques are in-



(a) Integrated Correlation Graph ICG_a

(b) Integrated Correlation Graph ICG_b

(c) Integrated Correlation Graph ICG_c

Figure 10. Experimental results using the DMZ data set in LLDOS 1.0

tended to understand possible ways of combining different attacks. In our method, the investigation of the actual alerts and raw audit data presents more opportunities that cannot be provided by static vulnerability analysis. Thus, we believe that our techniques are more suitable than static vulnerability analysis when real intrusion data is available.

6. Conclusion and Future Work

In this paper, we presented a series of techniques to construct high-level attack scenarios even if the underlying IDSs miss critical attacks. Our approach integrates two complementary intrusion alert correlation methods: (1) correlation based on similarity between alert attributes, and (2) correlation based on prerequisites and consequences of attacks. Moreover, our approach uses the intrinsic relationships between possibly related attacks to hypothesize missed attacks. To reason about hypothesized attacks, we developed techniques to compute constraints that indirectly related attacks must satisfy and proposed to further validate hypothesized attacks through raw audit data. Finally, we presented a technique to consolidate hypothesized attacks to generate concise representations of constructed attack scenarios. Our experimental results demonstrated the potential of these techniques.

The proposed techniques can provide meaningful "guesses" of attacks possibly missed by the IDSs, and thus supply good starting points as well as supporting evidences to facilitate investigation of unknown intrusions. A limitation of these techniques is that they depend on known attacks used together with unknown attacks to identify those unknown ones. These techniques will fail if all attacks involved in a sequence attacks are unknown. Moreover, the effectiveness of these techniques for different mixtures of known and unknown attacks still requires further investigation.

This paper is a starting point for improving intrusion detection through alert correlation. In our future research, we plan to continue our investigation in this direction. In particular, we will develop additional techniques to validate and reason about hypothesized attacks and perform a more thorough, quantitative evaluation of the proposed techniques.

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

[3] G. Combs. The ethereal network analyzer. `http://www.ethereal.com`.

[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. Miege. 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. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, pages 231–235, June 2001.

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

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

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

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

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

[13] 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.

[14] K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *The 8th ACM International Conference on Knowledge Discovery and Data Mining*, July 2002.

[15] MIT Lincoln Lab. 2000 DARPA intrusion detection scenario specific datasets. `http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html`, 2000.

[16] B. Morin and H. Debar. Correlation of intrusion symptoms: an application of chronicles. In *Proceedings of the 6th International Conference on Recent Advances in Intrusion Detection (RAID'03)*, September 2003.

[17] 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.

[18] P. Ning and Y. Cui. Intrusion alert correlator (version 0.2). `http://discovery.csc.ncsu.edu/software/correlator/ver0.2/iac.html`, 2002.

[19] 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.

[20] P. Ning and D. Xu. Learning attack stratagies from intrusion alerts. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, October 2003. To appear.

[21] P. Porras, M. 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.

[22] X. Qin and W. Lee. Statistical causality analysis of infosec alert data. In *Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, Sept. 2003.

[23] C. Ramakrishnan and R. Sekar. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security*, 10(1/2):189–209, 2002.

[24] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2002.

[25] S. Staniford, J. Hoagland, and J. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[26] 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.

[27] 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.

A. The Hyper-alert Types

In this appendix, we give more information about the hyper-alert types used in our examples and experiments. We present the implication relationships between predicates in Table 2. Table 3 shows all hyper-alert types used in our examples, and Table 4 describes all hyper-alert types used in our experiments. In both Tables 3 and 4, the set of *fact* attributes for each hyper-alert type is {*SrcIP*, *SrcPort*, *DestIP*, *DestPort*}.

Table 2. Implication relationships between the predicates

Predicate	Implied Predicate
ExistService(IP,Port)	GainInformation(IP)
GainOSInfo(IP)	GainInformation(IP)
GainOSInfo(IP)	OSSolaris(IP)
OSSolaris(IP)	OSUNIX(IP)
GainSMTPInfo(SrcIP,DestIP)	SMTPSupportTurn(SrcIP,DestIP)
GainAccess(IP)	SystemCompromised(IP)
SystemCompromised(IP)	SystemAttack(IP)
ReadyForDDOSAttack(IP)	ReadyForDDOSAttack

Table 3. Hyper-alert types used in the examples

Hyper-alert Type	Prerequisite	Consequence
ICMP_PING_NMAP		ExistHost(DestIP)
SCAN_NMAP_TCP	ExistHost(DestIP)	{ExistService(DestIP,DestPort)}
IMAP_Authen_Overflow	ExistService(DestIP,DestPort) ∧VulnerableAuthenticate(DestIP)	{GainAccess(DestIP)}
FTP_Glob_Expansion	ExistService(DestIP,DestPort) ∧VulnerableFTPRequest(DestIP)	{GainAccess(DestIP)}
Rsh	GainAccess(DestIP) ∧GainAccess(SrcIP)	{SystemCompromised(DestIP), SystemCompromised(SrcIP)}
Mstream_Zombie	SystemCompromised(DestIP) ∧SystemCompromised(SrcIP)	{ReadyForDDOSAttack(DestIP), ReadyForDDOSAttack(SrcIP)}

Table 4. Hyper-alert types used in our experiments

Hyper-alert Type	Prerequisite	Consequence
Admind		
DNS_HInfo	ExistService(DestIP,DestPort)	{GainOSInfo(DestIP)}
Email_Almail_Overflow	ExistService(DestIP,DestPort) ∧VulnerableAlMailPOP3Server(DestIP)	{GainAccess(DestIP)}
Email_Debug	ExistService(DestIP,DestPort) ∧SendMailInDebugMode(DestIP)	{GainAccess(DestIP)}
Email_Ehlo	ExistService(DestIP,DestPort) ∧SMTPSupportEhlo(DestIP)	{GainSMTPInfo(SrcIP,DestIP)}
Email_Turn	ExistService(DestIP,DestPort) ∧SMTPSupportTurn(SrcIP,DestIP)	{MailLeakage(DestIP)}
FTP_Pass	ExistService(DestIP,DestPort)	
FTP_Put	ExistService(DestIP,DestPort) ∧GainAccess(DestIP)	{SystemCompromised(DestIP)}
FTP_Syst	ExistService(DestIP,DestPort)	{GainOSInfo(DestIP)}
FTP_User	ExistService(DestIP,DestPort)	
HTTP_ActiveX	ActiveXEnabledBrowser(SrcIP)	{SystemCompromised(SrcIP)}
HTTP_Cisco_Catalyst_Exec	CiscoCatalyst3500XL(DestIP)	{GainAccess(DestIP)}
HTTP_Java	JavaEnabledBrowser(SrcIP)	{SystemCompromised(SrcIP)}
HTTP_Shells	VulnerableCGIBin(DestIP) ∧OSUNIX(DestIP)	{GainAccess(DestIP)}
Mstream_Zombie	SystemCompromised(DestIP) ∧SystemCompromised(SrcIP)	{ReadyForDDOSAttack(SrcIP), ReadyForDDOSAttack(DestIP)}
Port_Scan		{ExistService(DestIP,DestPort)}
RIPAdd		
RIPExpire		
Rsh	GainAccess(DestIP) ∧GainAccess(SrcIP)	{SystemCompromised(DestIP), SystemCompromised(SrcIP)}
Sadmind_Amslverify_Overflow	VulnerableSadmind(DestIP) ∧OSSolaris(DestIP)	{GainAccess(DestIP)}
Sadmind_Ping	OSSolaris(DestIP)	{VulnerableSadmind(DestIP)}
SSH_Detected		
Stream_DoS	ReadyForDDOSAttack	{DDOSAgainst(DestIP)}
TCP_Urgent_Data		{SystemAttacked(DestIP)}
TelnetEnvAll		{SystemAttacked(DestIP)}
TelnetTerminaltype		{GainTerminalType(DestIP)}
TelnetXdisplay		{SystemAttacked(DestIP)}
UDP_Port_Scan		{ExistService(DestIP,DestPort)}