Hypothesizing and Reasoning about Attacks Missed by Intrusion Detection Systems

PENG NING and DINGBANG XU
North Carolina State University

Several alert correlation methods have been proposed over the past several years to construct high-level attack scenarios from low-level intrusion alerts reported by intrusion detection systems (IDSs). However, all of these methods depend heavily on the underlying IDSs, and cannot deal with attacks missed by IDSs. 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 hypothesize and reason about attacks possibly missed by the IDSs. In addition, this paper also discusses techniques to infer attribute values for hypothesized attacks, to validate hypothesized attacks through raw audit data, and to consolidate 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.

Categories and Subject Descriptors: C.2.0 [**Computer and Communication Networks**]: General—*Security and Protection*; C.2.3 [**Computer and Communication Networks**]: Network Operations—*Network Monitoring*; D.4.6 [**Operating Systems**]: Security and Protection—*Invasive Software (e.g., viruses, worms, Trojan horses)*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: intrusion detection, intrusion alert correlation, missed attacks

1. INTRODUCTION

With the development of the Internet, more and more organizations manage their data in networked information systems. Due to the open nature of the Internet, network intrusions have become an increasingly serious problem in recent years. Intrusion detection, which is aimed at detecting activities violating the security policies of the networked information systems, has been considered a necessary component to protect these systems along with other prevention-based security mechanisms such as access control.

Intrusion detection techniques are generally classified into two categories: *anomaly de-*

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.
A preliminary version of this paper appeared as “Building Attack Scenarios through Integration of Complementary Alert Correlation Methods” in *Proceedings of 11th Annual Network and Distributed System Security Symposium (NDSS’04)*, pages 97–111, February 2004.
Authors’ address: Peng Ning and Dingbang Xu, Cyber Defense Laboratory, Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8207; emails: {pning, dxu}@ncsu.edu.

tection and *misuse detection*. Anomaly detection builds profiles (e.g., statistical models) for normal activities, and raises alerts when the monitored behaviors significantly deviate from the normal operations. Misuse detection constructs signatures (patterns) based on known attacks or vulnerabilities, and reports alerts if the monitored activities match the signatures.

Despite over 20 years' efforts on intrusion detection, current intrusion detection systems (IDSs) still have several well-known problems. First, existing IDSs cannot detect all intrusions. While a misuse detection system cannot detect an unknown attack (or an unknown variation of a known attack), an anomaly detection system may fail to recognize stealthy malicious activities, too. Second, current IDSs cannot ensure that all alerts reflect actual attacks; *true positives* (attacks detected as intrusive) are usually mixed with *false positives* (benign activities detected as intrusive). Third, an IDS usually produces a large number of alerts [Axelsson 2000; Julisch 2000; 2001; 2003]. As indicated in [Julisch 2000], five IDS sensors reported 40MB of alert data within ten days, and a large fraction of these alerts are false positives. The high volumes and low quality (i.e., missed attacks and false positives) of the intrusion alerts make it very challenging for human users or intrusion response systems to understand the alerts and take appropriate actions. Thus, it is necessary to develop techniques to deal with the large volumes and low quality of intrusion alerts.

Besides the aforementioned problems, current IDSs are not sufficiently prepared for several trends in attacks. According to a 2002 CERT report [CERT Coordinate Center 2002], there are increasingly more automated attack tools, which typically consist of several (evolving) phases such as scanning for potential victims, compromising vulnerable systems, propagating the attacks, and coordinated management of attack tools. Moreover, attack tools are increasingly more sophisticated. In particular, "today's automated attack tools can vary their patterns and behaviors based on random selection, predefined decision paths, or through direct intruder management" [CERT Coordinate Center 2002]. These attack trends require more capable systems than the current IDSs to handle large volumes of alerts that potentially belong to different complex attack scenarios.

Several alert correlation techniques have been proposed in recent years to facilitate the analysis of intrusion alerts. These methods attempt to correlate IDS alerts based on the similarity between alert attributes [Staniford et al. 2002; Valdes and Skinner 2001; Dain and Cunningham 2001a; Cuppens 2001], previously known (or partially known) attack scenarios [Debar and Wespi 2001; Dain and Cunningham 2001b], or prerequisites and consequences of known attacks [Cuppens and Miege 2002; Ning et al. 2002b; 2002a]. A common requirement of these approaches is that they all heavily depend on the underlying IDSs for alerts. As a result, the performance of alert correlation is strictly limited by the performance of IDSs. In particular, if the IDSs miss critical attacks, the correlated alerts cannot reflect the actual attack scenarios due to the lack of the corresponding alerts, and thus may provide misleading information.

In this paper, we develop a series of techniques to hypothesize and reason about attacks possibly missed by IDSs, aiming at constructing high-level attack scenarios even if the underlying IDSs miss critical attacks. Our approach is to integrate the potentially relevant attack scenarios generated by the alert correlation technique in [Ning et al. 2002b], and use the intrinsic relationships between related attacks to hypothesize and reason about attacks missed by the IDSs. We observe that if two attacks are causally related, they usually satisfy certain constraints (e.g., sharing the same destination IP address), even if they are not directly adjacent to each other in a sequence of attacks. If the IDSs miss some critical attacks, alerts from the same attack scenario could be split into multiple attack scenarios. Thus, combining different attack scenarios and verifying the above constraints over possibly related alerts can potentially overcome the problem introduced by missed attacks.

Our approach works as follows. We first obtain (multiple) attack scenarios through a correlation method based on prerequisites and consequences of attacks such as those in [Cuppens and Miege 2002; Ning et al. 2002b], and identify what attack scenarios (and possibly individual, uncorrelated alerts) may be combined by examining the attributes of the alerts in different attack scenarios. If those attribute values satisfy the aforementioned constraints, we consider integrating the corresponding attack scenarios. We assume the missed attacks are most likely unknown variations of known attacks, or attacks equivalent to some known attacks. We then hypothesize and reason about attacks missed by IDSs based on possible causal relationships between known attacks, aiming at constructing more complete attack scenarios. 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 but no RPC traffic related to the target host, we can conclude that the latter hypothesis is incorrect. Finally, to improve the usability of the constructed attack scenarios, we consolidate the hypothesized attacks and generate concise representations of the combined attack scenarios.

Our main contribution in this paper is a series of techniques to combine multiple attack scenarios and to hypothesize and reason about attacks possibly missed by the IDSs. These techniques are critical to constructing high-level attack scenarios from low-level intrusion alerts in situations where the IDSs cannot detect all attacks. These techniques complement the underlying IDSs by hypothesizing and reasoning about missed attacks, and thus provide valuable additional evidence to support further intrusion investigation and response.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 briefly describes an alert correlation method based on prerequisites and consequences of attacks [Ning et al. 2002b], which is the basis of the techniques developed in this paper. Section 4 presents our techniques to hypothesize and reason about attacks possibly missed by IDSs, to infer attribute values for hypothesized attacks, to validate hypothesized attacks using raw audit data, and to consolidate hypothesized attacks. Section 5 reports our experiments, and Section 6 concludes this paper and points out some future research directions.

2. RELATED WORK

Our work in this paper is closely related to intrusion alert correlation and vulnerability analysis. In the following, we discuss the related research in these two areas, respectively.

2.1 Alert Correlation

Alert correlation, informally, is the process of discovering the relationships between alerts. Recent intrusion alert correlation techniques can be roughly classified into four categories.

Methods in the first category group intrusion alerts into different clusters based on the similarity between the alerts [Valdes and Skinner 2001; Staniford et al. 2002; Julisch 2001; 2003; Dain and Cunningham 2001a; Qin and Lee 2003; Cuppens 2001]. Such methods may facilitate alert analysis in that future analysis can be based on clusters instead of individual alerts. One critical issue in these approaches is how to measure the similarity between alerts. Traditional similarity measures used in data mining [Han and Kamber 2001; Kaufman and Rousseeuw 1990] may not be appropriate for alert correlation, because many alert attributes are categorical (e.g., TCP/UDP Port numbers) rather than numerical. Several techniques have been proposed to solve this problem. In particular, Julisch et al. [Julisch and Dacier 2002; Julisch 2001] use conceptual clustering and generalization hierarchy to aggregate alerts into clusters. These similarity based alert correlation approaches are complementary to ours; they could provide a way to identify what sets of correlated alerts may be further integrated based on the similarity between alerts.

Methods in the second category perform correlation according to pre-defined attack scenarios, which are patterns of known sequences of attacks consisting of individual attack steps. Such methods then match IDS alerts to attack steps in the attack scenarios (in a similar way to misuse detection). Examples in this category include [Debar and Wespi 2001; Dain and Cunningham 2001b; Morin and Debar 2003]. Some approaches in this category specify attack scenarios through attack languages such as STATL [Eckmann et al. 2002] and Chronicles [Morin and Debar 2003]. A limitation of such techniques is that they cannot discover unknown attack scenarios. Nevertheless, the pre-defined attack scenarios can potentially help us hypothesize what attacks are possibly missed by the IDSs when only partial scenarios are observed.

Methods in the third category [Templeton and Levitt 2000; Cuppens and Miege 2002; Ning et al. 2002b] model attacks by specifying their pre-conditions (prerequisites) and post-conditions (consequences), and then correlates alerts (i.e., detected attacks) together when an earlier alert's post-condition (partially) matches a later one's pre-condition. These techniques have the potential to discover novel attack scenarios. However, specifying pre-conditions and post-conditions of attacks requires knowledge of individual attacks, and is time-consuming and error-prone. Our techniques in this paper further enhance this category of methods so that they can construct (partial) attack scenarios even if some critical attacks are missed by IDSs.

Methods in the fourth category [Porras et al. 2002; Morin et al. 2002] correlate alerts from multiple heterogeneous information sources, such as IDS sensors, firewalls, vulnerability scanners and anti-virus tools. The mission-impact-based approach [Porras et al. 2002] ranks the alerts (incidents) in terms of the overall impact to the mission of the networks; thus, it is feasible to differentiate the alerts based on their impacts. M2D2 [Morin et al. 2002] uses a formal model to describe the concepts and relations about the security of various information systems, so that security analysts can use this model to facilitate correlating (aggregating) the alerts from multiple sources. Since these heterogeneous information sources put different emphases on protecting the network components and applications, combining them can potentially obtain more comprehensive understanding about the security of the protected system. However, multiple information sources also bring challenges to the current correlation research. The information provided by these sources is often syntactically or semantically different, or even conflict with each other. We consider these approaches complementary to ours.

As we mentioned in our earlier discussion, all the above correlation techniques depend on the underlying IDSs to provide alerts. Thus, the correlation results are limited to the alerts triggered by the IDSs. The techniques proposed in this paper allow us to hypothesize and reason about missed attacks, and thus can potentially exceed the limitation of the underlying IDSs.

The work closest to ours is the abductive correlation discussed in [Cuppens and Miege 2002], which identifies missed attacks by matching correlated alerts with predefined attack scenarios. Our method differs from abductive correlation in that our method does not require predefined attack scenarios, but automatically constructs a type graph, which consists of all possible ways to arrange known attacks, to guide the hypothesis of missed attacks. Moreover, our method also includes a series of techniques to further reason about the hypothesized attacks. In particular, our method allows pruning unreasonable hypotheses by using the relationships between attacks as well as raw audit data.

2.2 Vulnerability Analysis

Our approach is also related to techniques for vulnerability analysis [Sheyner et al. 2002; Jha et al. 2002; Ammann et al. 2002; Ramakrishnan and Sekar 1998; Dacier et al. 1996; Ramakrishnan and Sekar 2002; Phillips and Swiler 1998; Swiler et al. 2001; Gruschke 1998], which focus on how adversaries can exploit a sequence of security flaws to achieve their intrusion goals.

Attack graphs have been proposed to discover possible sequences of attacks from individual exploits [Sheyner et al. 2002; Jha et al. 2002; Ammann et al. 2002; Ramakrishnan and Sekar 1998; Dacier et al. 1996; Ramakrishnan and Sekar 2002; Phillips and Swiler 1998]. In an attack graph, each node is a state representing the status of the system of concern, and each edge is associated with an exploit (possible attacks) or an action which can lead to the corresponding state transition. A path in an attack graph describes a possible sequence of exploits which may be applied by an adversary to achieve the intrusion goal.

One method to construct attack graphs (e.g., [Phillips and Swiler 1998]) uses a backward approach based on pre-defined attack templates (attack scenarios). Starting from a goal state, such approaches search all attack templates, looking for the desirable exploits (edges) and the states (nodes). This process continues until the initial states are reached. This method can build all possible sequences of exploits based on given attack templates. However, defining attack templates is time-consuming and error-prone. In addition, this approach also has scalability problems. To more efficiently construct attack graphs, recent approaches [Sheyner et al. 2002; Jha et al. 2002] model each exploit based on its pre-conditions and post-conditions, and apply model checking tools such as SMV [SMV] and NuSMV [NuSMV] to automatically generate attack graphs. One limitation of such approaches is that they suffer from the same scalability problem that the model checking tools have [Burch et al. 1992]. To analyze exploits in large networks, Ammann et al. [2002] proposed a scalable network vulnerability analysis method based on the assumption of monotonicity, which dramatically decreases the analysis complexity from exponential to polynomial.

Our approach proposed in this paper uses a *type graph* to help us hypothesize and reason about attacks possibly missed by the IDSs. *Type graph* is a concept similar to *attack graph* in that it provides us with possible combinations of attacks. However, a type graph captures the constraints that any two consecutive attacks should satisfy, which are not provided by an attack graph. In addition, constructing a type graph does not have the scalability problem, since attack types (rather than attack instances) are used as nodes in a type graph. Finally, our method employs intrusion alerts and raw audit data to hypothesize and reason about missed attacks, and thus presents additional opportunities that cannot be provided by vulnerability analysis.

3. PREVIOUS WORK: ALERT CORRELATION USING PREREQUISITES AND CONSEQUENCES OF ATTACKS

The new techniques in this paper are based on the alert correlation method proposed in [Ning et al. 2002b]. In the following, we briefly describe this method with a slight modification, which is required by the newly proposed techniques.

The approach in [Ning et al. 2002b] 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 (partially) matching the consequences of earlier alerts to the prerequisites of later ones.

The correlation method uses logical formulas, i.e., 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. For simplicity, we only consider logical AND ("$\wedge$") and OR ("$\vee$") in logical formulas.

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, and *consequence* is a set of logical formulas, and all the free variables in *prerequisite* and *consequence* are in *fact*. Intuitively, *prerequisite* and *consequence* encode the pre-condition and the post-condition of a known type of attacks, and *fact* consists of the alert attributes that are relevant to the pre-condition and the post-condition.

To illustrate the notion of hyper-alert types, we use the following example from [Ning et al. 2002b]. A buffer overflow attack against the *sadmind* remote administration tool can be represented as *SadmindBufferOverflow* = ($\{$*VictimIP, VictimPort*$\}$, *ExistHost* (*VictimIP*) $\wedge$ *VulnerableSadmind* (*VictimIP*), $\{$*GainRootAccess*(*VictimIP*)$\}$). Intuitively, this hyper-alert type says that such attacks are against the host at IP address *VictimIP*. For the attack to be successful, there must exist a host at IP address *VictimIP*, and the corresponding *sadmind* service must be vulnerable to buffer overflow attacks. The attacker may gain root privilege as a result of the attack.

Specifying hyper-alert types from known attacks is essentially a knowledge engineering process, which is error-prone and requires substantial efforts from human experts. To improve the quality of hyper-alert types, in practice, we use a pre-defined set of predicates, which may be extended when necessary, classify known attacks into different classes based on their pre-conditions and post-conditions, and specify a hyper-alert type for each class.

Given a hyper-alert type *T* = (*fact, prerequisite, consequence*), a *type T alert t* is a tuple on *fact*, where this tuple is associated with an interval-based timestamp [*begin_time, end_time*]. The existence of an alert t implies that *prerequisite* must evaluate to True and all the logical formulas in *consequence* might evaluate to True for this alert. A *type T* *hyper-alert h* is a finite set of type *T* alerts. The notion of hyper-alert allows multiple alerts of the same type to be treated collectively.

The original model in [Ning et al. 2002b] is aimed at identifying the *prepare-for* relations between hyper-alerts, aimed at allowing flexible manipulation of collections of alerts. However, the techniques proposed in this paper require such *prepare-for* relations between *alerts* explicitly. (Informally, an alert can be considered a special case of a hyper-alert, i.e., a hyper-alert with a single alert.) Thus, we present a slightly modified model below to help the presentation of the new techniques.

Intuitively, a *prepare-for* relation exists if an earlier alert *contributes* to the prerequisite of a later one. In the formal model, alert correlations are performed 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)), 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 implied by *Conseq*(T). (This can be computed with the implication relationships between predicates [Ning et al. 2002b].) Thus, *Conseq*$(T) \subseteq$ *ExpConseq*(T). Given a type T alert t, the *prerequisite set*, *consequence set*, and *expanded consequence set* of t, denoted *Prereq*(t), *Conseq*(t), and *ExpConseq*(t), respectively, consist of the predicates in *Prereq*(T), *Conseq*(T), and *ExpConseq*(T) with arguments replaced by the corresponding attribute values of t. Alert t_1 *prepares for* alert t_2 if $t_1.end\_time < t_2.begin\_time$ and there exist $p \in$ *Prereq*(t_2) and $c \in$ *ExpConseq*(t_1) such that $p = c$. The *prepare-for* relation between hyper-alerts is defined similarly in [Ning et al. 2002b]; we do not repeat it here.

An alert (or hyper-alert) correlation graph is used to represent a set of correlated alerts (or hyper-alerts). An *alert (or hyper-alert) correlation graph* CG = (N, E) is a connected directed acyclic graph, where N is a set of alerts (or 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. Note that a hyper-alert correlation graph must be acyclic, since if one attack step prepares for the other, then the former must occur before the latter. For brevity, we refer to an alert correlation graph or a hyper-alert correlation graph as a *correlation graph* in this paper. For brevity, we also refer to this correlation method as the *causal correlation method*, since its goal is to discover the causal relationships between alerts.

4. HYPOTHESIZING AND REASONING ABOUT ATTACKS MISSED BY IDS

If IDSs miss some critical attacks, an attack scenario (represented as a correlation graph) may be split into multiple smaller ones, each of which only reflects a part of the original attack scenario. To better understand the whole attack scenario, it is desirable to integrate related attack scenarios, and hypothesize and reason about the attacks possibly missed by the IDSs. In this section, we develop a sequence of techniques for these purposes. We assume we have applied the causal correlation method to the alerts before using the newly proposed techniques.

In the following, we start with a straightforward approach to integrating possibly related attack scenarios, and gradually develop more sophisticated techniques to enhance this approach.

4.1 Integrating Possibly Related Correlation Graphs

We observe that the causal correlation method can be potentially enhanced by a similarity-based correlation method (e.g., [Valdes and Skinner 2001; Staniford et al. 2002; Julisch

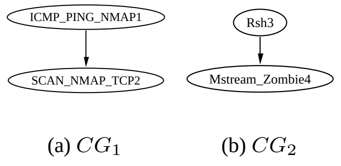


Fig. 1. Two correlation graphs

2001; 2003]), which clusters alerts based on the similarity between alert attribute values. Intuitively, when the IDSs miss certain attacks, though the causal correlation method may split the alerts from the same attack scenario into several correlation graphs, a similarity-based correlation method still has the potential to identify the common features shared by these alerts, and thus help re-integrate the related correlation graphs together. To take advantage of this observation, we integrate correlation graphs based on the alert clusters generated by a similarity-based correlation method.

The integration process may 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 generated by a similarity-based correlation method. For example, given the two correlation graphs shown in Figures 1(a) and 1(b)[1], 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.

The first step is pretty straightforward once we select a similarity-based correlation method. However, the second step remains challenging, since we must deal with missed attacks that cause an attack scenario to split into multiple correlation graphs. Thus, we focus on the second step in the following discussion. As we will see later, the first step becomes unnecessary as we develop our approach. Without loss of generality, we assume that we integrate two correlation graphs.

We propose to harness the prior knowledge of attacks and the alert timestamp information to hypothesize about possible causal relationships between alerts in different correlation graphs. For example, suppose an attacker uses *nmap* [Fyodor 2003] 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 the above buffer overflow attack). As a result, we would hypothesize an indirect causal relationship between these two alerts.

To further characterize this intuition and facilitate later discussion, we introduce two definitions. (Note that Definition 4.1 is based on the model in [Ning et al. 2002b], which has been briefly described in Section 3.) For convenience, we denote the type of an alert t (or a hyper-alert h) as $Type(t)$ (or $Type(h)$).

[1]The string inside each node is a hyper-alert type name followed by an alert ID.

Definition 4.1. 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).

Example 4.2. Consider two hyper-alert types *SadmindPing* = (*fact, prereq, conseq*) and *SadmindBufferOverflow* = (*fact', prereq', conseq'*), where *fact* = {*VictimIP, VictimPort*}, *prereq* = *ExistHost(VictimIP)*, *conseq* = {*VulnerableSadmind (VictimIP)*}, *fact'* ={*VictimIP, VictimPort*}, *prereq'* = *ExistHost(VictimIP)* $\wedge$ *VulnerableSadmind(VictimIP)*, and *conseq'* = {*GainRootAccess(VictimIP)*}. We observe both *ExpConseq*(*SadmindPing*) and *Prereq*(*SadmindBufferOverflow*) include the predicate *VulnerableSadmind(VictimIP)*. Then we know that *SadmindPing may prepare for SadmindBufferOverflow*. □

Definition 4.3. 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'.

Example 4.4. Given a set $\mathcal{T}$ of hyper-alert types, where $\mathcal{T}$ ={*ICMP_PING_NMAP, SCAN_NMAP_TCP, FTP_Glob_Expansion, Rsh, Mstream_Zombie*}, assume the following *may-prepare-for* relations exist: *ICMP_PING_NMAP may prepare for SCAN_NMAP_TCP*, *SCAN_NMAP_TCP may prepare for FTP_Glob_Expansion*, *FTP_Glob_Expansion may prepare for Rsh*, and *Rsh may prepare for Mstream_Zombie*. Thus it is clear *ICMP_PING_NMAP may indirectly prepare for Mstream_Zombie* w.r.t. $\mathcal{T}$. □

Intuitively, given two hyper-alert types T and T', T *may prepare for* T' if there exist a type T alert t and a type T' alert t' such that t *prepares for* t'. *May-indirectly-prepare-for* relation, which is a natural extension of *may-prepare-for* relation, is defined through a sequence of *may-prepare-for* relations.

Definition 4.5. Given a set $\mathcal{T}$ of hyper-alert types and two alerts t and t', where $Type(t)$ and $Type(t') \in \mathcal{T}$ and $t.end\_time < t'.begin\_time$, t *may indirectly prepare for* t' if $Type(t)$ *may indirectly prepare for* $Type(t')$ w.r.t. $\mathcal{T}$. Given a sequence of alerts t, t_1, ..., t_k, t' where $k > 0$, t *indirectly prepares for* t' if t *prepares for* t_1, t_i *prepares for* t_{i+1} for $i = 1, ..., k-1$, and t_k *prepares for* t'.

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

A simple way to take advantage of the above observation is to assume a possible causal relationship between alerts t and t' if they belong to different correlation graphs and t *may indirectly prepare for* t'. Let us continue the example in Figure 1. If the hyper-alert type *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.

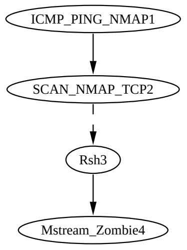


Fig. 2. A straightforward combination of CG_1 and CG_2

Though this simple approach can identify and integrate related 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." It is possible that the hypothesized virtual edges are not true due to the limitations of the expert knowledge and the lack of information about the missed attacks.

4.2 Hypothesizing about Missed Attacks

The *may-prepare-for* and *may-indirectly-prepare-for* relations identified in Definitions 4.1, 4.3 and 4.5 provide additional opportunities to hypothesize and reason about missed attacks, especially unknown variations of known attacks.

Consider two alerts t and t' that belong to different correlation graphs prior to integration. If t *may indirectly prepare for* t', we can then identify possible sequences of hyper-alert types in the form of $T_1, T_2, ..., T_k$ such that $Type(t)$ *may prepare for* T_1, T_i *may prepare for* T_{i+1}, $i = 1, 2, ..., k-1$, and T_k *may prepare for* $Type(t')$. 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 t and t' 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 adapted from [Ning and Xu 2003], to help formally describe the notion of type graph.

Definition 4.6. Given a pair of hyper-alert types T_1 and 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 alert t_1 and a type T_2 alert t_2, t_1 and t_2 *satisfy the equality constraint* if $t_1.u_1 = t_2.v_1 \wedge$

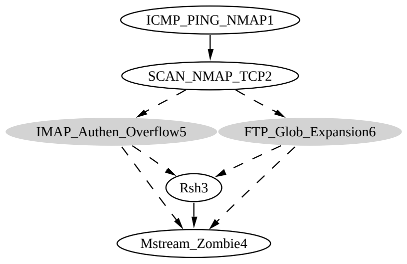


Fig. 3. Integration of CG_1 and CG_2 with hypotheses of missed attacks

$\cdots \wedge t_1.u_n = t_2.v_n$ evaluates to True.

Example 4.7. Consider the hyper-alert types *SadmindPing* and *SadmindBufferOverflow* in Example 4.2. *ExpConseq*(*SadmindPing*) and *Prereq*(*SadmindBufferOverflow*) both contain the predicate *VulnerableSadmind(VictimIP)*. Thus, it is easy to see that *SadmindPing.VictimIP* $=$ *SadmindBufferOverflow.VictimIP* is an equality constraint for (*SadmindPing*, *SadmindBufferOverflow*). Further consider a type *SadmindPing* alert t_1 and a type *SadmindBufferOverflow* alert t_2. If t_1 and t_2 both have *VictimIP* $= 152.1.19.5$, we can conclude that t_1 and t_2 satisfy the equality constraint. ☐

An equality constraint characterizes the equality relations between attribute values of two alerts when one *prepares for* the other. There may exist several equality constraints for a pair of hyper-alert types. However, if a type T_1 alert t_1 *prepares for* a type T_2 alert t_2, then t_1 and t_2 must satisfy at least one equality constraint. Indeed, t_1 *preparing for* t_2 is equivalent to the conjunction of t_1 and t_2 satisfying at least one equivalent constraint and t_1 occurring before t_2.

Given a set of hyper-alert types (representing the known attacks), by matching all possible predicates in the expanded consequence set and the prerequisite set, 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 4.8. 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 DAG (directed acyclic 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 4.9. 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*$\}$. (The specification of these hyper-alert types are given in Table I.) We can compute the type

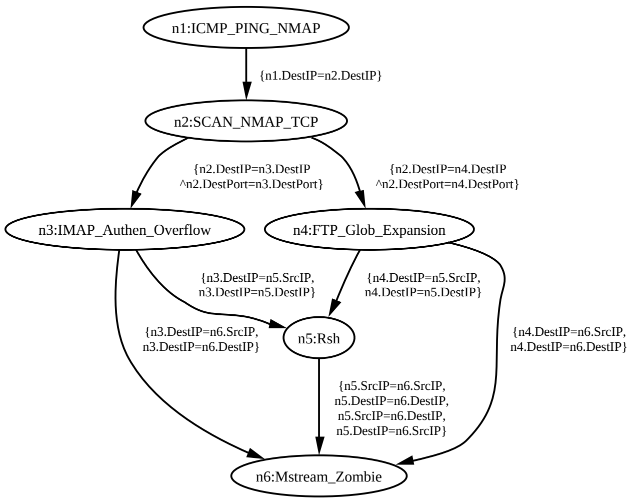

n1:ICMP_PING_NMAP
{n1.DestIP=n2.DestIP}
n2:SCAN_NMAP_TCP
{n2.DestIP=n3.DestIP ^n2.DestPort=n3.DestPort}
{n2.DestIP=n4.DestIP ^n2.DestPort=n4.DestPort}
n3:IMAP_Authen_Overflow
n4:FTP_Glob_Expansion
{n3.DestIP=n5.SrcIP, n3.DestIP=n5.DestIP}
{n4.DestIP=n5.SrcIP, n4.DestIP=n5.DestIP}
{n3.DestIP=n6.SrcIP, n3.DestIP=n6.DestIP}
n5:Rsh
{n4.DestIP=n6.SrcIP, n4.DestIP=n6.DestIP}
{n5.SrcIP=n6.SrcIP, n5.DestIP=n6.DestIP, n5.SrcIP=n6.DestIP, n5.DestIP=n6.SrcIP}
n6:Mstream_Zombie

Fig. 4. An example type graph

Table I. Hyper-alert types used in Example 4.9 (The set of *fact* attributes for each hyper-alert type is {*SrcIP, SrcPort, DestIP, DestPort*})

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

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. $\square$

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.

4.3 Reasoning about Missed Attacks

In a type graph, the label of an edge encodes all possible equality constraints for the corresponding pair of hyper-alert types. Moreover, even if two hyper-alert types are not adjacent to each other, they may still satisfy some constraints if they are connected through some intermediate nodes (hyper-alert types) in the type graph (due to the equality constraints those intermediate nodes must satisfy). For example, consider nodes $n2$, $n3$, and $n5$ in Figure 4. There is an equality constraint $n2.DestIP = n3.DestIP \wedge n2.DestPort = n3.DestPort$ for ($n2$, $n3$), and two equality constraints $n3.DestIP = n5.SrcIP$ and $n3.DestIP = n5.DestIP$ for ($n3$, $n5$). Take together, these imply $n2.DestIP = n5.SrcIP$ or $n2.DestIP = n5.DestIP$. In other words, if a type *SCAN_NMAP_TCP* alert *indirectly prepares for* a type *Rsh* alert (through a type *IMAP_Authen_Overflow* alert), together they must satisfy one of these two constraints. We obtain the same constraints if we consider nodes $n2$, $n4$, and $n5$ in Figure 4. In general, we can derive constraints for two hyper-alert types when one *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 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 4.6. 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. In the following, we will first present an algorithm to compute indirect equality constraints for *two* hyper-alert types where one *may indirectly prepare for* the other, and then extend it to compute indirect equality constraints for *all pairs* of hyper-alert types at the same time. We will also discuss how to use such indirect equality constraints to reason about missed attacks.

4.3.1 *Computing Indirect Equality Constraints between Two Hyper-Alert Types.* 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'. We assume a type graph TG is already constructed from a set of hyper-alert types, which are specified based on all the attacks known to the IDSs (or equivalently, the set of signatures). 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 relations between attributes of hyper-alert types. Each indirect equality constraint is labeled with the corresponding path that produces the constraint. This provides guidelines for hypothesizing about missed attacks. The usefulness of Algorithm 1 is demonstrated by Lemma 4.10.

LEMMA 4.10. *Consider a type graph TG and two alerts t and t', where $Type(t)$ and $Type(t')$ are in TG. Assume Algorithm 1 outputs a set C of equality constraints for $Type(t)$ and $Type(t')$. If C is non-empty and t indirectly prepares for t', then t and t' must satisfy at least one equality constraint in C.*

PROOF. According to Definition 4.5, if t *indirectly prepares for* t', there must exist a sequence of alerts t_1, ..., t_k, where $k > 0$, such that t *prepares for* t_1, t_i *prepares for*

```
Algorithm 1. Computation of Indirect Equality Constraints for Two Hyper-Alert Types.
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 $ec_1$, $ec_2$, ..., $ec_{k+1}$,
         where $ec_i$ 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 $ec_1$.
  6.         For $j = 1$ to $k + 1$
  7.             For each conjunct $T_{j-1}.a = T_j.b$ in $ec_j$
  8.                 For each $S(T_0.a_i)$ that contains $T_{j-1}.a$
  9.                     Let $S(T_0.a_i) = S(T_0.a_i) \cup \{T_j.b\}$.
  10.            Remove variables of $T_{j-1}$ from each $S(T_0.a_i)$, $i = 1, 2, ..., l$.
  11.        Let $temp = \emptyset$.
  12.        For each non-empty $S(T_0.a_i)$ and each $T_{k+1}.b$ in $S(T_0.a_i)$
  13.            Let $temp = temp \cup \{T_0.a_i = T_{k+1}.b\}$.
  14.        Let $ec$ be the conjunction of all elements in $temp$.
  15.        If $ec$ is in $Result$ Then
  16.            Let $Label(ec) = Label(ec) \cup \{\langle T, T_1, ..., T_k, T' \rangle\}$
  17.        Else Let $Label(ec) = \{\langle T, T_1, ..., T_k, T' \rangle\}$, and $Result = Result \cup \{ec\}$.
  18. Return $Result$.
  End
```

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

t_{i+1} for $i = 1, ..., k-1$, and t_k *prepares for* t'. Thus, we have $Type(t)$ *may prepare for* $Type(t_1)$, $Type(t_i)$ *may prepare for* $Type(t_{i+1})$ for $i = 1, ..., k-1$, and $Type(t_k)$ *may prepare for* $Type(t')$. Following the convention of Algorithm 1, we denote $Type(t)$ as T_0, $Type(t_i)$ as T_i, where $i = 1, ..., k$, and $Type(t')$ 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 t as t_0, and t' as t_{k+1}.

If t_i *prepares for* t_{i+1}, we can conclude t_i and t_{i+1} must satisfy at least one equality constraint for (T_i, T_{i+1}). This is because if t_i and t_{i+1} does not satisfy any equality constraints for (T_i, T_{i+1}), then none of the instantiated predicates in *ExpConseq*(t_i) can match any in *Prereq*(t_{i+1}), which violates the given condition that t_i *prepares for* t_{i+1}. For $i = 0, 1, ..., k$, we denote the constraint t_i and t_{i+1} satisfy as ec_{i+1}. 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 ec_1, ec_2, ..., ec_{k+1} that t_0, t_1, ..., t_{k+1} satisfy (in step 4).

Now consider the process of the above sequence of equality constraints in steps 5 to 10. 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 10 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 10, 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 11 to 14 then transform these equality relations into a conjunctive formula ec. Since the sequence of constraints ec_i, $i = 1, 2, ..., k + 1$, where each ec_i is satisfied by t_{i-1} and t_i, is used in the above process, we can easily conclude that t_0 (t) and t_{k+1} (t') satisfy ec. Thus, if t *indirectly prepares for* t', they must satisfy at least one equality constraint in C. □

Example 4.11. Consider the type graph shown 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 two paths: one path is $\langle$*SCAN_NMAP_TCP*, *IMAP_Authen_Overflow*, *Rsh*$\rangle$, and the other is $\langle$*SCAN_NMAP_TCP*, *FTP_Glob_Expansion*, *Rsh*$\rangle$. □

Given two hyper-alert types T and T' in a type graph, Algorithm 1 derives the indirect equality constraints between them by considering all combinations of (direct) equality constraints between two adjacent hyper-alert types in each path from T to T'. Theoretically, there is a potential problem of combinatorial explosion. However, in practice, because of the limited number of predicates and hyper-alert types, this problem should be tractable. Moreover, Algorithm 1 only needs to be executed once for two given hyper-alert types in a type graph. Thus, the cost of Algorithm 1 does not have significant impact on alert correlation.

4.3.2 *Computing Indirect Equality Constraints for All Pairs of Hyper-Alert Types.* Algorithm 1 focuses on the problem of computing indirect equality constraints for two hyper-alert types. An extension to this problem is: given a set $\mathcal{T}$ of n hyper-alert types, how to calculate indirect equality constraints for all pairs of hyper-alert types where the first one in the pair *may indirectly prepare for* the second one? This is a realistic problem, since we do need to get the equality constraints between all pairs of hyper-alert types to reason about missed attacks.

We can apply Algorithm 1 for up to n^2 times, once for each pair of hyper-alert types (T_i, T_j) where T_i *may indirectly prepare for* T_j ($1 \leq i, j \leq n$). Unfortunately, this is not an efficient solution. To see the inefficiency more clearly, consider a path from T_i to T_j where T_j is further connected to T_k by an edge. If we compute the indirect equality constraints between all pairs of hyper-alert types with Algorithm 1, the computation for T_i and T_k with the path involving T_i, T_j, and T_k will repeat the computation for T_i and T_j with the same path from T_i and T_j. A better approach is to reuse the equality constraints already computed for T_i and T_j to derive those for T_i and T_k.

To take advantage of the above observation, Algorithm 2 (shown in Figure 6) outlines a method to compute equality constraints for all pairs of hyper-alert types at the same time. The output of Algorithm 2 is a constraint matrix. Given n hyper-alert types $T_1, T_2, \cdots, T_n$, a *constraint matrix* M is a $n \times n$ table, where the cell $M(i, j)$ ($1 \leq i, j \leq n$) consists of all and only equality constraints for (T_i, T_j) if T_i *may (indirectly) prepare for* T_j.

As we discussed in Algorithm 1, a path between hyper-alert types T_i and T_j in a type graph represents that T_i *may indirectly prepare for* T_j, and we can derive equality constraints for (T_i, T_j) by reasoning about the equality constraints along the path. The basic idea behind Algorithm 2 is to reuse the equality constraints derived from short paths to compute those for long paths. In a type graph with n hyper-alert types (n nodes), the possible lengths of paths range from 1 to $n - 1$. To compute the indirect equality constraints for a path with length k ($1 < k \leq n - 1$), it is always possible to first carry out the

```
**Algorithm 2. Computation of Equality Constraints for All Pairs of Hyper-alert Types**
Input: A type graph $TG$ over a set of hyper-alert types $\{T_1, T_2, \cdots, T_n\}$.
Output: A $n \times n$ constraint matrix $M$ with each cell $M(i, j)$ containing a set of equality
        constraints for $(T_i, T_j)$.
Method:
  1. Create a $n \times (n-1)$ table $L$, and initialize each cell of $L$ to empty.
     // Each cell $L(i, j)$ is intended to contain the equality constraints (marked with path
     // labels) for the length $j$ paths in $TG$ starting from type $T_i$.
  2. Let $k = 1$. // The variable $k$ represents the possible lengths of the paths in $TG$
  3. For each edge $(T_i, T_j)$ in the type graph $TG$
  4.     For each equality constraint $ec$ for $(T_i, T_j)$
  5.         Put $ec$ into the cell $L(i, 1)$ (with the label $\langle T_i, T_j \rangle$).
  6. For $k = 2$ to $n-1$
  7.     For $i = 1$ to $n$
  8.         For each equality constraint $ec$ in $L(i, k-1)$
  9.             Get the last hyper-alert type $T$ in $Label(ec)$.
  10.            Get the set $\mathcal{T}$ of hyper-alert types where $T$ has edges to each type in $\mathcal{T}$.
  11.            For each hyper-alert type $T'$ in $\mathcal{T}$
  12.                For each equality constraint $ec'$ associated with $(T, T')$
  13.                    Get a constraint $ec''$ via InferredConstraint ($ec$, $ec'$).
  14.                    Let $Label(ec'') = \langle Label(ec), T' \rangle$.
  15.                    Let $L(i, k)$ = $L(i, k) \cup ec''$.
  16. For $i = 1$ to $n$
  17.     For $j = 1$ to $n$
  18.         In row $i$ of $L$, find all equality constraints where the last type in their labels
             is $T_j$, and put these constraints into the cell $M(i, j)$.
  19. Output the matrix $M$.
  End.
```

```
**Subroutine InferredConstraint**
Input: An equality constraints $ec$ for $(T_x, T_y)$ and an equality constraint $ec'$ for $(T_y, T_z)$.
Output: An equality constraint $ec''$ for ($T_x, T_z$) derived from $ec$ and $ec'$.
Method:
  1. Let $ec'' = \{\}$.
  2. For each conjunct $T_x.u = T_y.v$ in $ec$
  3.     If there exists a conjunct $T_y.v = T_z.w$ in $ec'$, then
  4.         Let $T_x.u = T_z.w$ be a conjunct in $ec''$.
  5. Output $ec''$.
  End.
```

Fig. 6. Algorithm to compute indirect equality constraints for all pairs of hyper-alert types

computation of the (indirect) equality constraints for length $k-1$ paths, and then combine the results for such paths with the equality constraints for individual edges to get the constraints for length k paths. Lemma 4.12 ensures that Algorithm 2 can derive all and only equality constraints for T_i and T_j.

LEMMA 4.12. *Given a type graph TG over a set of hyper-alert types $\{T_1, T_2, \cdots, T_n\}$, Algorithm 2 outputs all and only equality constraints for (T_i, T_j) in the cell $M(i, j)$ $(1 \leq i, j \leq n)$.*

PROOF. According to the definition of (indirect) equality constraint, there may be one or more equality constraints for (T_i, T_j) if T_i *may (indirectly) prepare for* T_j. In other words, there may be one or more equality constraints for (T_i, T_j) if there is a path between T_i and T_j in TG. In the following, we first prove by induction that each equality constraint that can be derived for (T_i, T_j) from a length k path p ($k = 1, 2, \cdots, n-1$) is labeled with p and stored in $L(i, k)$.

(1) When $k = 1$, for each length 1 path $p = \langle T_i, T_j \rangle$ (which is an edge in TG), lines 3 through 5 put all equality constraints for (T_i, T_j) into cell $L(i, 1)$, each of which is labeled with the corresponding edge.

(2) Assume for any T_i, T_j in TG, all the equality constraints that can be derived for (T_i, T_j) from a length m path are in $L(i, m)$ with the corresponding path labels. Now we show that for any T_i, T_j in TG, all the equality constraints that can be derived for (T_i, T_j) from a length $m+1$ path are in $L(i, m+1)$ with the corresponding path labels.
Consider lines 7 through 10. For each T_i, these lines find all the edges that can follow each length m path starting with T_i. Thus, they can enumerate all length $m+1$ paths. For convenience, we denote each length $k = m+1$ path $p = \langle T_i, \cdots, T_s, T_j \rangle$ as composed of two connected paths: $p' = \langle T_i, \cdots, T_s \rangle$ and $p'' = \langle T_s, T_j \rangle$, where the length of p' is m and p'' is an edge in TG. According to the induction assumption, the equality constraints that can be derived for (T_i, T_s) from p' are in $L(i, m)$ with the label p'.
Consider lines 11 through 15. For each equality constraint ec in $L(i, m)$ with label $\langle T_i, \cdots, T_s \rangle$ and each equality constraint ec' for (T_s, T_j), the subroutine **Inferred-Constraint** (ec, ec') (line 13) derives the equality ec'' inferred by ec and ec', which is an equality constraint for (T_i, T_j). This equality constraint is then labeled with the path $p = \langle T_i, \cdots, T_s, T_j \rangle$ and then added into $L(i, m+1)$. Since lines 11 through 15 consider all combinations of the equality constraints in $L(i, m)$ and the (direct) equality constraints for each edge that follow a length m path, they can find the equality constraints that can be derived for all length $m+1$ paths starting from T_i. Therefore, for any T_i, T_j in TG, all the equality constraints that can be derived for (T_i, T_j) from a length $m+1$ path are in $L(i, m+1)$ with the corresponding path labels.

Further considering lines 16 through 18, which copy all equality constraints derived from paths between T_i and T_j into $M(i, j)$, and that the possible path length in TG is from 1 to $n-1$, we can conclude that all equality constraints for (T_i, T_j) are in $M(i, j)$. Moreover, during Algorithm 2, since we only add the inferred (indirect) equality constraints into L with path labels (lines 14 and 15), and we only move equality constraints derived from paths from T_i to T_j into $M(i, j)$ (lines 16 through 18), $M(i, j)$ only contains equality constraints for (T_i, T_j). □

Example 4.13. To continue Example 4.11, we may use Algorithm 2 to 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. The results are given in Table II, in which each cell contains the equality constraints for the hyper-alert types in the given row and the column. (To save space, we use node names to represent the corresponding hyper-alert types and omit the labels for each equality constraint.) □

Similar to Algorithm 1, Algorithm 2 is also executed only once before alert correlation, and thus does not introduce significant overhead during alert correlation.

Table II. 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 $\wedge$ n2.DestPort = n3.DestPort}	{n2.DestIP = n4.DestIP $\wedge$ n2.DestPort = n4.DestPort}	{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	/	/	/	/	/	/

4.3.3 *Using (Indirect) Equality Constraints.* The equality constraints derived for indirectly related hyper-alert types can be used to determine if two correlation graphs can be integrated. Given two correlation graphs CG_1 and CG_2, we can integrate CG_1 and CG_2 if there exist an alert t_1 in CG_1 and an alert t_2 in CG_2 such that (1) t_1 and t_2 satisfy at least one equality constraint for $(Type(t_1), Type(t_2))$ and (2) t_1's timestamp is before t_2's timestamp.

Moreover, such equality constraints can also facilitate the hypotheses of missed attacks. When integrating two correlation graphs CG and CG', we can hypothesize missed attacks only for such pairs of alerts t and t' that (1) t and t' belong to CG and CG', respectively, and (2) t *indirectly prepare for* t'. Specifically, for each equality constraint ec that t and t' satisfy, we can add the paths in $Label(ec)$ into the integrated correlation graph. Since each path in $Label(ec)$ is in the form of $\langle Type(t), T_1, ..., T_k, Type(t')\rangle$, $Type(t)$ and $Type(t')$ are merged with t and t', respectively, and the rest of the path is added as a virtual path consisting of virtual nodes and edges from t to t'. Note that this may add incorrect hypotheses into the integrated correlation graph. We will present techniques to validate these hypotheses with raw audit data in Section 4.5.

Example 4.14. Let us illustrate how to take advantage of indirect equality constraints to hypothesize missed attacks. Consider two correlation graphs in Figure 1, if an earlier alert *SCAN_NMAP_TCP2* and a later alert *Rsh3* have the same destination IP address, they then satisfy an equality constraint ec: *SCAN_NMAP_TCP.DestIP* $=$ *Rsh.DestIP*, which is an indirect equality constraint for (*SCAN_NMAP_TCP*, *Rsh*) shown in Table II. Thus, we can integrate CG_1 and CG_2 and hypothesize missed attacks based on the label associated with the above equality constraint. For instance, the label associated with the above equality constraint ($Label(ec)$) consists of two paths: $\langle$*SCAN_NMAP_TCP*, *IMAP_Authen_Overflow*, *Rsh*$\rangle$ and $\langle$*SCAN_NMAP_TCP*, *FTP_Glob_Expansion*, *Rsh*$\rangle$. Thus, we can hypothesize two missed attacks *IMAP_Authen_Overflow5* and *FTP_Glob_Expansion6*. The hypothesis process may continue until all such pairs of alerts are examined. $\square$

4.4 Inferring Attribute Values for Hypothesized Attacks

The (direct or indirect) equality constraints not only help hypothesize about missed attacks, but also provide an opportunity to make inferences about the attribute values of hypothesized attacks. In other words, we may further hypothesize about the missed attack *instances*. For example, suppose we hypothesize 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*. From Table II, we know that *SCAN_NMAP_TCP* and *IMAP_Authen_Overflow* have the same destination IP address and destination port, and the destination IP address of *IMAP_Authen_Overflow* is the same as either the source or the destination IP address of *Rsh*.

In general, we can use the equality constraints between the intrusion 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 as a possible range. That is, if an attack t_h is hypothesized as an intermediate step between two intrusion alerts t and t', where t occurs before t', then the possible range of t_h's timestamp is $[t.end\_time, t'.begin\_time]$. Let us first look at an example.

Example 4.15. Consider the integrated correlation graph shown in Figure 3. Let us infer attribute values for the hypothesized attack (instance) *FTP_Glob_Expansion6*. Suppose the IDS reported that the destination IP addresses of alerts *SCAN_NMAP_TCP2* and *Rsh3* were both $152.1.19.5$, and the destination port of *SCAN_NMAP_TCP2* was 21. Following the earlier convention in Figure 4, we use nodes $n2$, $n4$, and $n5$ to denote hyper-alert types *SCAN_NMAP_TCP*, *FTP_Glob_Expansion*, and *Rsh*, respectively. It is easy to see that *SCAN_NMAP_TCP2* and *Rsh3* satisfy the equality constraint *n2.DestIP* $=$ *n5.DestIP*. Based on the constraint matrix in Table II, we can see this equality constraint is derived from the following two equality constraints:

(1) *n2.DestIP* $=$*n4.DestIP* $\wedge$ *n2.DestPort* $=$ *n4.DestPort* for (*SCAN_NMAP_TCP*, *FTP_Glob_Expansion*), and

(2) *n4.DestIP* $=$*n5.DestIP* for (*FTP_Glob_Expansion*, *Rsh*).

Thus, the hypothesized attack *FTP_Glob_Expansion6* should satisfy both of these equality constraints. As a result, we can infer that the destination IP address and the destination port of *FTP_Glob_Expansion6* are $152.1.19.5$ and 21, respectively. $\square$

We generalize Example 4.15 into Algorithm 3, which is shown in Figure 7, to infer attribute values of hypothesized attack instances. Intuitively, given two alerts t and t', for each hypothesized attack T_i along a path of hypothesized attacks, we get the set C_i of equality constraints for $(Type(t), T_i)$ and the set C'_i of equality constraints for $(T_i, Type(t'))$, respectively. Any combination of equality constraints ec_i in C_i and ec'_i in C'_i that result in an equality constraint that t and t' satisfy can be used to infer the attribute values of the hypothesized instance of attack T_i. In other words, we infer the attributes of the hypothesized attack instance to be the same as those of t and t' as indicated by the equality constraints.

```
Algorithm 3. Inferring Attribute Values for Hypothesized Attacks
Input: A type graph $TG$ over a set $\mathcal{T}$ of hyper-alert types, a path $\langle T, T_1, T_2, \cdots, T_k, T' \rangle$
       in $TG$, a type $T$ alert $t$, and a type $T'$ alert $t'$, where $t$ may indirectly prepare for $t'$.
Output: A set $H_i$ of hypothesized attack instances for each type $T_i$, where $i = 1, 2, \cdots, k$.
Method:
   // We assume the constraint matrix $M$ for $\mathcal{T}$ has been computed, in which each equality
   // constraint $ec$ in $M$ is labeled with the corresponding path $Label(ec)$ in $TG$.
   1. Get a set $C$ of equality constraints such that for each $ec \in C$, $t$ and $t'$ satisfy $ec$ and
      $Label(ec) = S$.
   2. For each hypothesized attack $T_i$
   3.     Get the set of equality constraints $C_i$ for $(T, T_i)$ whose label is $\langle T, T_1, ..., T_i \rangle$;
          get the set of equality constraints $C'_i$ for $(T_i, T')$ whose label is $\langle T_i, ..., T_k, T' \rangle$;
          let $H_i = \{\}$.
   4.     For each combination of $ec_i \in C_i$ and $ec'_i \in C'_i$
   5.         If $ec_i$ and $ec'_i$ imply any $ec \in C$ Then
   6.             Generate a type $T_i$ alert $t_i$; set all the attributes of $t_i$ that are equal to some
                  attributes of $T$ in $ec_i$ to the corresponding attribute values of $t$; similarly,
                  set all the attributes of $t_i$ that are equal to some attributes of $T'$ in $ec'_i$ to
                  the corresponding attribute values of $t'$; set the remaining attributes (if any)
                  to Unknown; let the timestamp of $t_i$ be $[t.end\_time, t'.begin\_time]$.
   7.             Let $H_i = H_i \cup \{t_i\}$.
   8.     Output $H_i$.
   End.
```

Fig. 7. Algorithm to infer attribute values for hypothesized attacks

In Algorithm 3, line 1 gets the set C of equality constraints that alerts t and t' satisfy and that are associated with the given path in the type graph. Lines 2 through 7 are a loop to infer attribute values for each hypothesized attack in given path in TG, which is a possible sequence of attacks that have happened. Line 3 obtains the set of equality constraints C_i for $(Type(h), T_i)$ and C'_i for $(T_i, Type(h'))$ that are associated with the two halves of the given path (through the constraint matrix). In the following steps, the algorithm tries all combinations of equality constraints in C_i and C'_i, and infers the attribute values of hypothesized type T_i attacks . Each hypothesized attack instance of Type T_i is derived through two equality constraints ($ec_i \in C_i$ and $ec'_i \in C'_i$). The condition checking in line 5 guarantees that the inferred attribute values do not conflict (otherwise, the corresponding combination of equality constraints could not lead to an equality constraint that t and t' satisfy). Finally, line 8 outputs the hypothesized attack instances for each attack type (hyper-alert type) T_i, where $i = 1, 2, ..., k$.

We make several observations about Algorithm 3. First, given two alerts t and t', the hypotheses of missed attack instances between t and t' are *specific* to the paths between t and t'. In other words, the hypothesis of each missed attack is supported by the possibility that an attacker has launched a sequence of attacks (or, more precisely, attack instances), including the hypothesized one, that leads from t to t'. This also implies that Algorithm 3 should be performed multiple times when hypothesizing about missed attack instances based on a given pair of alerts. Second, the two alerts t and t' and a given path in the type graph may lead to multiple instances of each attack type, since there may be multiple (direct or indirect) equality constraints for each pair of hyper-alert types. Third, the hypothesized attack instances are usually not as specific as regular alerts. That is, a hypothesized attack may have unknown values on some attributes, which cannot be inferred from the available alerts.

Algorithm 3 is essentially a best-effort "guess" of what could have been missed by IDSs. The hypothesized attack instances can certainly be wrong. In the next subsection, we investigate how to prune those incorrectly "guessed" attack instances using a complementary information source, the raw audit data.

4.5 Pruning Hypothesized Attacks with Raw Audit Data

The hypothesized attack instances can be further validated using raw audit data. For example, we may hypothesize that 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 further narrow the hypothesized attacks down to the meaningful ones.

To take advantage of this observation, we extend our model to associate a "filtering condition" with each hyper-alert type. Assuming that the raw audit data set consists of a sequence of audit records, we can extract attribute values 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*.

Definition 4.16. Given a hyper-alert type T and a set A of audit attributes, 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.

Example 4.17. Consider the following set of audit attributes: A = {*SrcIP, SrcPort, DestIP, DestPort, Protocol, FrameArrivalTime*}. Given a hyper-alert type *FTP_Glob_Expansion*, we may have a simple logical formula "*Protocol* = *ftp*" as a filtering condition for type *FTP_Glob_Expansion* w.r.t. A. □

Intuitively, a filtering condition for a hyper-alert type is a necessary condition for the corresponding attack or its variations. We can simply evaluate the condition to prune some incorrect hypotheses. If a filtering condition is evaluated to True, the corresponding attack may have happened; if it is evaluated to False, the corresponding hypothesized attack could not have happened, and should be ruled out.

The above filtering condition is essentially prior knowledge of known attacks. There is an additional opportunity to prune incorrect hypotheses if we further consider the inferred attribute values of the hypothesized attacks. For example, if we can infer that the destination IP address of a hypothesized *FTP_Glob_Expansion* attack is $152.1.19.5$, we may further check whether there is ftp activities destined to $152.1.19.5$ in Example 4.17. In other words, we can revise the filtering condition in Example 4.17 to "*Protocol* = *ftp* $\wedge$ *DestIP* = $152.1.19.5$" for the hypothesized attack instance.

To formalize this idea, we introduce the notion of *filtering template*.

Definition 4.18. Given a hyper-alert type $T = (fact, prerequisite, consequence)$ and a set A of audit attributes, a *filtering template for T w.r.t. A* is a logical formula involving variables in $fact$ and A, which evaluates to *True* or *False* if these variables are replaced with specific values. Given a hypothesized attack t of type T with a set f_s of inferred attributes, where $f_s \subseteq fact$, a filtering template $Temp_f$ is *instantiatable by* t if all the variables in $Temp_f$ are either in f_s or in A. If a filtering template $Temp_f$ is instantiatable by t, we can then get an *instantiated filtering condition for* t by replacing the variables in $Temp_f$ with the inferred attribute values of t.

Example 4.19. Consider the set of audit attributes: A = {*SrcIP, SrcPort, DestIP, DestPort, Protocol, FrameArrivalTime*}. Given a hyper-alert type *FTP_Glob_Expansion* (See Table I), we may have a filtering template "*A.DestIP = FTP_Glob_Expansion.DestIP*" as a filtering template for type *FTP_Glob_Expansion* w.r.t. A. Assume there is a hypothesized attack *FTP_Glob_Expansion6* with an inferred attribute *DestIP* = $152.1.19.5$. The above filtering template is then instantiatable by *FTP_Glob_Expansion6*, and can be instantiated to "*A.DestIP* = $152.1.19.5$". □

Intuitively, a filtering template is a template of filtering condition for a type of attack. Given a hypothesized attack with a set of inferred attributes, we may convert a filtering template into a filtering condition if all the attack attributes that appear in the filtering template have specific inferred values. To distinguish from the filtering condition defined in Definition 4.16, we call those defined for hyper-alert types the *predefined filtering conditions*, and those instantiated from hypothesized attacks the *instantiated filtering conditions*. We can then use such an instantiated filtering condition in the same way as the predefined filtering conditions.

Pruning incorrectly hypothesized attacks with predefined and/or instantiated filtering conditions is pretty straightforward. Before correlating alerts, we specify filtering conditions and filtering templates for each hyper-alert type. When hypothesizing and reasoning about missed attacks, for each hypothesized attack with a possible range of its timestamp and a set of inferred attributes, we first determine whether each filtering template corresponding to the hypothesized attack is instantiatable by this hypothesis w.r.t. the raw audit attributes. If the answer is positive, we derive an instantiated filtering condition for this filtering template. We then compute the actual filtering condition as the conjunction of the predefined filtering condition and all the instantiated filtering conditions.

To validate a hypothesized attack, we can search through the raw audit records during the time period when the hypothesized attack may have happened, and evaluate the filtering condition using the values of the attributes of each raw audit record. To continue Examples 4.17 and 4.19, we can generate the final filtering condition as "*Protocol* = *ftp* $\wedge$ *DestIP* = $152.1.19.5$" to validate (or deny) *FTP_Glob_Expansion6*. If there is no ftp traffic associated with the destination IP address $152.1.19.5$ between alerts *SCAN_NMAP_TCP2* and *Rsh3*, i.e., the above filtering condition evaluates to False for all audit records, we can conclude that the *FTP_Glob_Expansion6* attack is falsely hypothesized. As a result, the integrated correlation graph in Figure 3 can be refined to the one in Figure 8.

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

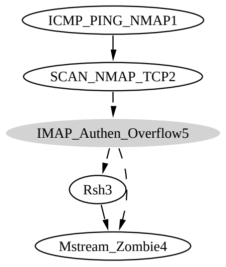


Fig. 8. Integration of CG_1 and CG_2 after refinement with raw audit data

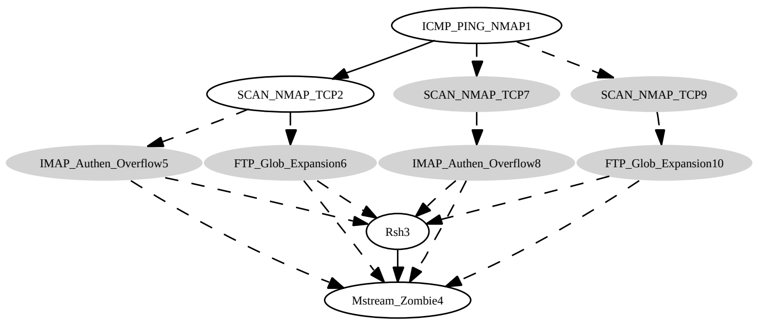


Fig. 9. Hypothesized attacks when integrating CG_1 and CG_2

carried out and the inferred attribute values as filtering conditions. It is interesting to study how to get the "right" way to specify filtering conditions. We will investigate it in detail 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 happen. 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.

4.6 Consolidating Hypothesized Attacks

In the earlier subsections, we investigated various techniques to hypothesize and reason about missed attacks. However, our method has not considered the possibility that the same attack may be hypothesized multiple times in different contexts. As a result, an integrated correlation graph may include too many hypothesized attacks. Though it is possible that the same attack are repeated multiple times (as hypothesized), having too many uncertain details reduces the usability of the integrated correlation graph.

Let us look at an example to see this problem more clearly. Consider Figure 9, which shows some hypothesized attacks resulting from the integration of CG_1 and CG_2 in Figure 1. Assume *ICMP_PING_NMAP1*, *SCAN_NMAP_TCP2* and *Rsh3* all have the same destination IP address $152.1.19.5$. Since *SCAN_NMAP_TCP2* and *Rsh3* satisfy the equality constraint *SCAN_NMAP_TCP.DestIP* $=$ *Rsh.DestIP*, based on the type graph in Figure 4, we hypothesize two attacks: *IMAP_Authen_Overflow5* and *FTP_Glob_Expansion6*, which both have the same destination IP address $152.1.19.5$ derived through attribute value inference. Similarly, alerts *ICMP_PING_NMAP1* and *Rsh3* satisfy the equality constraint *ICMP_PING_NMAP.DestIP* $=$ *Rsh.DestIP*. Thus, we may hypothesize the following four attacks: *SCAN_NMAP_TCP7*, *IMAP_Authen_Overflow8*, *SCAN_NMAP_TCP9* and *FTP_Glob_Expansion10*, all with the same destination IP address $152.1.19.5$.

This example leads to two observations. First, it is possible that the hypothesized attack instance *SCAN_NMAP_TCP7* is the same attack as reflected by the existing alert *SCAN_NMAP_TCP2*, but it is also possible that the attacker launched two separate attacks. Similarly, it is equally possible for *IMAP_Authen_Overflow5* and *IMAP_Authen_Overflow8* to be the same attack or two separate attacks. Second, having all these hypothesized attacks makes the integrated correlation graph complex and difficult to understand. Since the hypothesized attacks are all uncertain, having multiple hypotheses for one attack does not give more information. Indeed, if we consider the typical goal of attack hypothesis during intrusion analysis, it is not critical to know how many times an attack has been used in one step of attacks; instead, it is usually more important to know whether an attack has been used or not.

Based on the above observations, we propose to consolidate the hypothesized attacks. Specifically, we remove a hypothesized attack if it may have been detected (as an existing alert), or aggregate a set of hypothesized attacks if they may be the same attack. Our approach is based on the "consistency" between a hypothesized attack and an alert, or the "consistency" among hypothesized attacks. Informally, a set of hypothesized attacks are *consistent* if they could be the same attack, and a hypothesized attack is *consistent* with an alert if this hypothesized attack could have been detected and reflected as the alert.

We first look at the consistency between a hypothesized attack and an alert by examining their attack types, attribute values, and timestamp information. Once a hypothesized attack t_h is identified as consistent with an alert t, we subsume t_h into t by merging t_h and t (as well as the duplicated edges resulting from this merge).

In the following, we first clarify the consistency relations.

Definition 4.20. A hypothesized attack t_h is *consistent* with an alert t if (1) t_h and t are of the same type, (2) if t_h and t both have specific values on the same attribute, these two values are the same, and (3) the timestamp of t_h includes the timestamp of t (i.e., $t_h.begin\_time \leq t.begin\_time \wedge t_h.end\_time \geq t.end\_time$).

The consistency among a set of hypothesized attacks can be defined in a similar way.

Definition 4.21. A set $H_h = \{t_1, t_2, \cdots, t_n\}$ of hypothesized attacks is *consistent* if (1) all hypothesized attacks in H_h are of the same type, (2) if more than one hypothesized attacks in H_h have specific values on an attribute, then all these values must be the same, and (3) $\min\{t_i.end\_time | i = 1, 2, ..., n\} > \max\{t_i.begin\_time | i = 1, 2, ..., n\}$ (i.e., the intersection of all the interval-based timestamps is not empty).

The intuition behind Definition 4.21 is that a set of hypothesized attacks are consistent if they have the same type, their attribute values do not conflict, and the possible ranges of their interval-based timestamps overlap.

Figure 10 outlines an algorithm to consolidate hypothesized attacks. Step 1 groups all hypothesized attacks based on their types. Step 2 starts to process each group. This processing can be divided into two stages. The first stage (steps 3 through 5) reduces the

```
Algorithm 4. Consolidating Hypothesized Attacks
Input: A set $S$ of alerts and a set $S_h$ of hypothesized attacks.
Output: A set $S'_h$ of hypothesized attacks after consolidation.
Method:
  1. Partition $S_h$ into groups such that the hypothesized attacks in each group have the
     same type.
  2. For each group $G_h$ in $S_h$
  3.     For each hypothesized attack $t_h$ in $G_h$
  4.         If $t_h$ is consistent with an alert $t$ in $S$ then
  5.             Remove $t_h$ from $G_h$, and merge $t_h$ with $t$.
  6.     If $G_h$ is not empty then
  7.         Partition $G_h$ into maximal subgroups such that the hypothesized attacks in
             each subgroup are consistent.
  8.         Replace each subgroup $G_s$ with a hypothesized attack $t_h$ with the same type.
  9.         For each attribute $a_i$ of $t_h$
  10.            If there exists a hypothesized attack $t'_h \in G_s$ that has a specific value on $a_i$
  11.                let $t_h.a_i = t'_h.a_i$,
  12.            else let $t_h.a_i = Unknown$.
  13.        Add $t_h$ into $S'_h$.
  14.Output $S'_h$.
  End.
```

Fig. 10. Algorithm to consolidate hypothesized attacks

hypothesized attacks based on the consistency relations between hypothesized attacks and alerts. The second stage (steps 6 to 13) partitions each group of hypothesized attacks into subgroups so that all attacks in each subgroup are consistent, consolidates each subgroup into one hypothesized attack, and instantiates the attributes of the hypothesized attack if they are inferable. Step 14 finally outputs the consolidated version of hypothesized attacks.

Consolidating hypothesized attacks helps reduce the number of virtual nodes in an integrated correlation graph. To get a concise attack scenario, the following job is to merge the virtual edges associated with those hypothesized attacks being consolidated. This is trivial: If a hypothesized attack t_h is consolidated based on an alert t, then all virtual edges related to t_h should be re-directed to t. Likewise, given a set S_h of hypothesized attacks, if all attacks in S_h can be consolidated into a hypothesized attack t_h, then all virtual edges related to the hypothesized attacks in S_h should be re-directed to t_h.

Our consolidation technique is effective in reducing the size of integrated correlation graphs. For example, in one of our experiments, we have consolidated 137 hypothesized attacks into 5 ones. However, we shall point out that, after consolidation, each hypothesized attack may correspond to multiple instances of missed attacks. In other words, each hypothesized attack in an integrated correlation graph is indeed a place-holder for one or several possible attacks.

5. EXPERIMENTAL RESULTS

We have implemented all the techniques we discussed in this paper. In our implementation, we used Java as the programming language, and Microsoft SQL Server 2000 as the database to store the hyper-alert types, the alert data sets, and the analysis results. We assume the NCSU Intrusion Alert Correlator version 0.2 [Cui 2002] is used to correlate IDS alerts into correlation graphs. To validate the hypothesized attacks using raw audit data, our implementation uses Ethereal (version $0.9.14$) to extract audit attribute values from the raw tcpdump file (i.e., the network audit data). Finally, we use GraphViz [AT & T Research Labs] to visualize the integrated correlation graphs.

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 [MIT Lincoln Lab 2000]. 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 [Internet Security Systems] as the IDS sensor to generate alerts, which are then correlated by the NCSU Intrusion Alert Correlator into correlation graphs.

On constructing the type graph for the experiments, we consider all attacks (represented as hyper-alert types) in the data sets that can be detected by RealSecure Network Sensor 6.0. The specification of these hyper-alert types is given in Table III, the implication relationships between predicates are shown in Table IV, and the type graph is given in Figure 11. For space reasons, we did not put the isolated nodes (the nodes which do not have edges connecting to them) into the type graph.

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 data set. 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. In our experiment with inside traffic of LLDOS 1.0 data set, before dropping *Sadmind_Amslverify_Overflow* alerts, we only got one correlation graph. After dropping, however, this correlation graph was divided into four parts. Figure 12 shows all these four correlation graphs.

Now let us focus on the correlation graphs in Figure 12. What we should do first is to determine if two correlation graphs can be integrated. The second step is to perform hypotheses, inference, validation and consolidation. For the sake of presentation, we first consider integrating two correlation graphs CG_c (Figure 12(c)) and CG_d (Figure 12(d)).

As mentioned earlier, if two alerts in two different correlation graphs satisfy at least one equality constraint associated with their types, we can combine these correlation graphs together. Since the destination IP addresses of both *Sadmind_Ping67343* (in CG_c) and *Rsh67553* (in CG_d) are $172.16.112.50$, they satisfy the constraint *Sadmind_Ping.DestIP* = *Rsh.DestIP*. Thus, it is easy to see CG_c and CG_d can be integrated together.

Based on the type graph, we can easily hypothesize that variations of *HTTP_Shells*, *FTP_Put* and *Sadmind_Amslverify_Overflow* could have been missed by the IDS sensor. 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 can reduce the hypotheses of missed 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 it is easy to see *Sadmind_Ping67343* cannot *indirectly prepare for Rsh67543* through a variation of attack *Sadmind_Amslverify_Overflow*. After missed attack hypotheses and reasoning, we perform attribute value inference. For example, a hypothesized *Sadmind_Amslverify_Overflow* attack

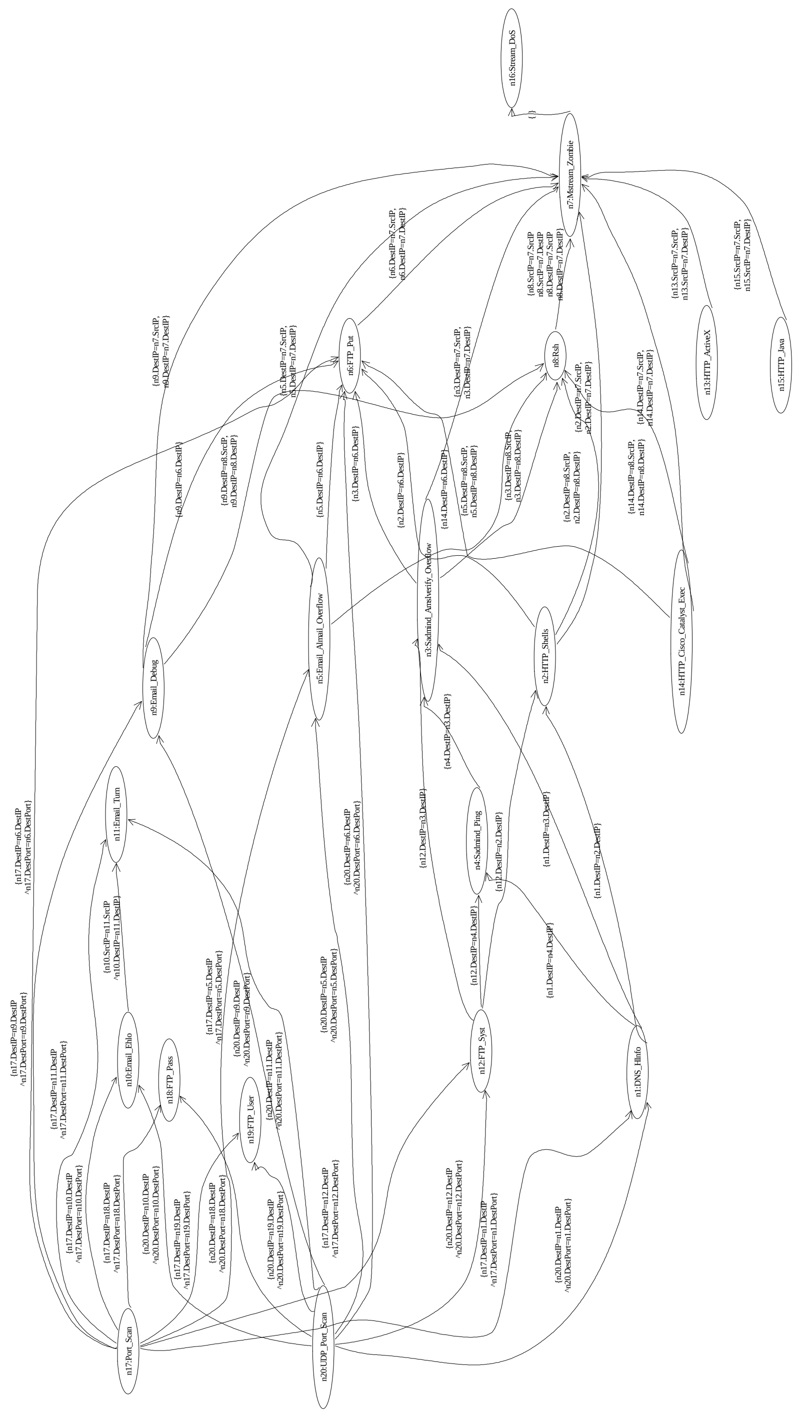


Fig. 11. The type graph used in our experiments

Table III. Hyper-alert types used in our experiments (The set of *fact* attributes for each hyper-alert type is {*SrcIP,SrcPort,DestIP,DestPort*}).

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)}

between *Sadmind_Ping67343* and *Rsh67553* has the destination IP address $172.16.112.50$.

The hypothesized attacks are further validated using the raw audit data. For example, in our experiments, 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 extracted values of audit attributes from audit records between the alerts that result in the corresponding hypothesized attacks. For example, we search all the pre-fetched packet information between *Sadmind_Ping67343* and *Rsh67553* for *Sadmind* packets (re-

Table IV. 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

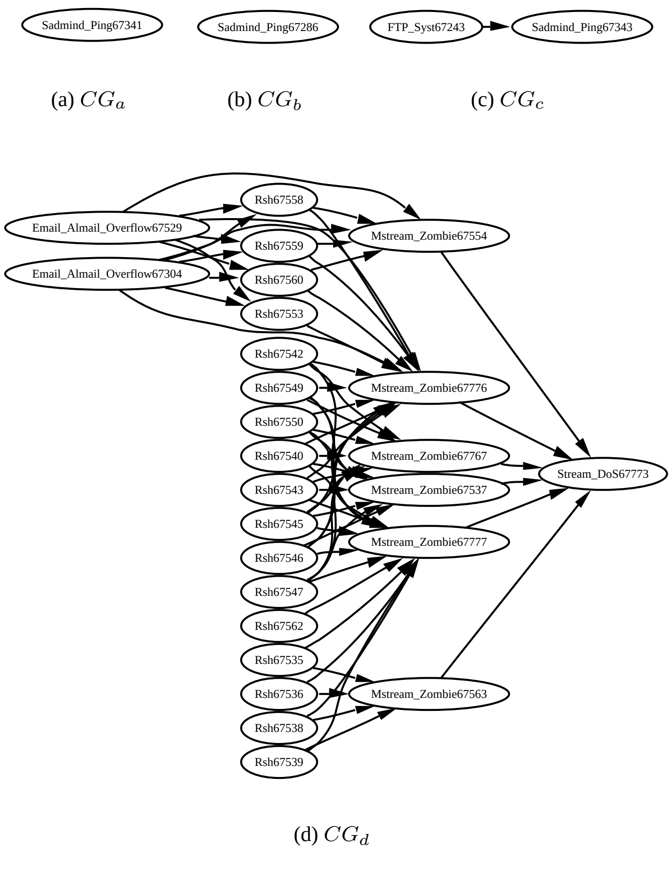


Fig. 12. Four correlation graphs constructed from LLDOS 1.0 inside traffic

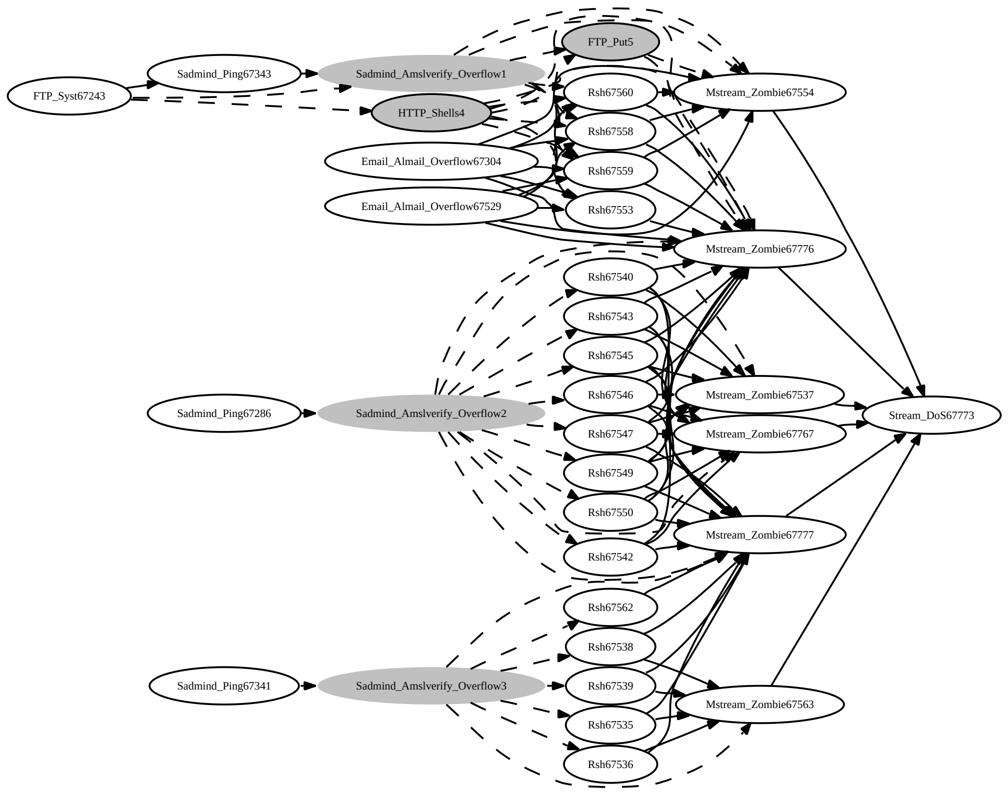
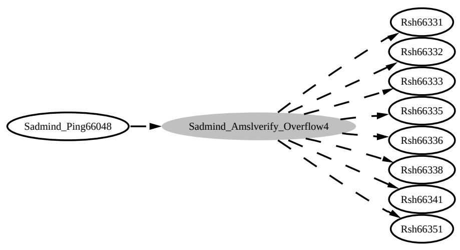


Fig. 13. The integrated correlation graph constructed from LLDOS 1.0 inside traffic

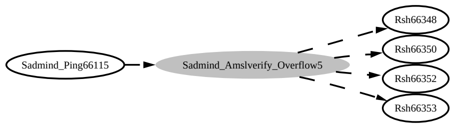
lated to the host $172.16.112.50$) in order to validate a hypothesized (variation of) *Sadmind_Amslverify_Overflow* attack. Finally we can get the integration result (without consolidation) on correlation graphs CG_c and CG_d.

We continue the above process to integrate the resulting correlation graph with additional ones (CG_a in Figure 12(a) and CG_b in Figure 12(b)). The alerts in these two graphs are *Sadmind_Ping67341* and *Sadmind_Ping67286*, respectively, which are both uncorrelated alerts. As a slight difference, several instances of *FTP_Put* are hypothesized during both integration processes, but all of them are invalidated later using the extracted audit information. In other words, we find no *ftp* activities involving the corresponding host during the time frame when the hypothesized attacks might happen. Figure 13 shows the integrated correlation graph after the hypothesized attacks are consolidated. The consolidation reduced the number of hypothesized attacks from about 137 to 5. In the integrated correlation graph shown in Figure 13, 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.

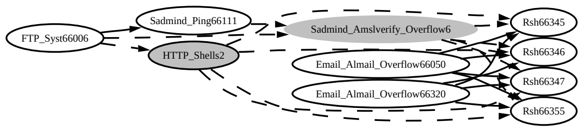
Now let us examine the integrated correlation graph in Figure 13. According to the description of the data sets [MIT Lincoln Lab 2000], the three *Sadmind_Amslverify_Overflow* attacks and the *prepare-for* relations between these attacks and the other alerts are hy-



(a) Integrated Correlation Graph ICG_a



(b) Integrated Correlation Graph ICG_b


FTP_Syst66006
Sadmind_Ping66111
HTTP_Shells2
Sadmind_Amslverify_Overflow6
Email_Almail_Overflow66050
Email_Almail_Overflow66320
Rsh66345
Rsh66346
Rsh66347
Rsh66355

(c) Integrated Correlation Graph ICG_c

Fig. 14. Experimental results using the DMZ dataset in LLDOS 1.0

pothesized correctly. However, the *FTP_Put* and *HTTP_Shells* attacks are hypothesized incorrectly.

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 11, we generated three integrated correlation graphs in Figure 14, in which hypothesized attacks are shown in gray. Based on the attribute value inference, 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 data sets [MIT Lincoln Lab 2000], 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 the underlying IDSs.

The experimental evaluation reported in this paper is still preliminary, though it has demonstrated the potential of the proposed techniques. To further understand the capability of these techniques, a more detailed, quantitative evaluation is required.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a series of techniques to construct high-level attack scenarios to facilitate the analysis of intrusion alerts. Our approach is based on a key concept: equality constraint, which captures the intrinsic relationships between possibly related attacks.

Moreover, to reason about hypothesized attacks, we developed techniques to compute equality constraints that indirectly related attacks must satisfy. We proposed to further infer attribute values for hypothesized attacks and validate hypothesized attacks through raw audit data. Finally, we presented a technique to consolidate hypothesized attacks to generate concise representations of attack scenarios. Our experimental results demonstrated the potential of these techniques.

Though the proposed techniques are aimed at improving IDSs' detection results, the actual performance is still limited by the performance of IDSs. In the worst case, if the IDSs miss all attacks, or all alerts are false ones, the proposed techniques will not perform well. Fortunately, our preliminary experiment has shown some promising results for the current generation of IDSs. We expect the proposed techniques will generate better results as the performance of IDSs is improved.

This paper is a starting point for improving intrusion detection through alert correlation. There are still a number of problems that are worth additional investigation. One such problem is the granularity in which the attacks are modeled. If the representation of attacks is too specific, the type graph may not be general enough to allow the hypotheses about variations of missed attacks. If the representation is too general, some causal relationships may not be captured in the type graph at all. More research is necessary to understand the best way to model attacks in the proposed framework.

Hypothesizing attacks is essentially making guesses about what could be missed by the IDSs. Obviously, even a good solution may lead to incorrect hypotheses. It is critical to have other means to validate the hypotheses. We will investigate additional information sources (other than the raw audit data) and techniques to validate and reason about hypothesized attacks.

The evaluation of the proposed techniques in this paper is still preliminary. Besides the lack of data sets for evaluation, another technical problem is how to quantitatively evaluate the quality of hypothesized attack instances, which may have specific values on *some* attributes, as well as the hypothesized causal (i.e., *prepare-for*) relations. We will seek solutions to this problem in our future research.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

AMMANN, P., WIJESEKERA, D., AND KAUSHIK, S. 2002. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 217–224.

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

AXELSSON, S. 2000. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security 3,* 3 (August), 186–205.

BURCH, J. R., CLARKE, E. M., MCMILLAN, K. L., DILL, D. L., AND HWANG, L. J. 1992. Symbolic model checking: 10^{20} states and beyond. *Information and Computation 98,* 2 (June), 142–170.

CERT COORDINATE CENTER. 2002. Overview of attack trends.

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

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

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

DACIER, M., DESWARTE, Y., AND KAâNICHE, M. 1996. Quantitative assessment of operational security: Models and tools. Tech. Rep. LAAS Research Report 96493. May.

DAIN, O. AND CUNNINGHAM, R. 2001a. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*. 231–235.

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

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

ECKMANN, S., VIGNA, G., AND KEMMERER, R. 2002. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security 10*, 1/2, 71–104.

FYODOR. 2003. Nmap free security scanner. `http://www.insecure.org/nmap`.

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

HAN, J. AND KAMBER, M. 2001. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers.

INTERNET SECURITY SYSTEMS. RealSecure intrusion detection system. `http://www.iss.net`.

JHA, S., SHEYNER, O., AND WING, J. 2002. Two formal analyses of attack graphs. In *Proceedings of the 15th Computer Security Foundation Workshop*.

JULISCH, K. 2000. Dealing with false positives in intrusion detection. In *The 3th Workshop on Recent Advances in Intrusion Detection*.

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

JULISCH, K. 2003. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security 6,* 4 (Nov), 443–471.

JULISCH, K. AND DACIER, M. 2002. Mining intrusion detection alarms for actionable knowledge. In *The 8th ACM International Conference on Knowledge Discovery and Data Mining*.

KAUFMAN, L. AND ROUSSEEUW, P. J. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons.

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

MORIN, B. AND DEBAR, H. 2003. Correlation of intrusion symptoms: an application of chronicles. In *Proceedings of the 6th International Conference on Recent Advances in Intrusion Detection (RAID'03)*.

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

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

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

NING, P. AND XU, D. 2003. Learning attack stratagies from intrusion alerts. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*. 200–209.

NUSMV. NuSMV: A new symbolic model checker. `http://nusmv.irst.itc.it/`.

PHILLIPS, C. AND SWILER, L. 1998. A graph-based system for network vulnerability analysis. In *Proceedings of New Security Paradigms Workshop*. 71–79.

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

QIN, X. AND LEE, W. 2003. Statistical causality analysis of infosec alert data. In *Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*. Pittsburgh, PA.

RAMAKRISHNAN, C. AND SEKAR, R. 1998. Model-based vulnerability analysis of computer systems. In *Proceedings of the 2nd International Workshop on Verification, Model Checking and Abstract Interpretation*.

RAMAKRISHNAN, C. AND SEKAR, R. 2002. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security 10,* 1/2, 189–209.

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

SMV. The SMV system. `http://www.cs.cmu.edu/˜modelcheck/smv.html`.

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

SWILER, L., PHILLIPS, C., ELLIS, D., AND CHAKERIAN, S. 2001. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference andn Exposition*. 307–321.

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

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

Received Month Year; revised Month Year; accepted Month Year