Alert Correlation through Triggering Events and Common Resources *

Dingbang Xu and Peng Ning
Cyber Defense Laboratory
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8207
{dxu, pning}@ncsu.edu

Abstract

Complementary security systems are widely deployed in networks to protect digital assets. Alert correlation is essential to understanding the security threats and taking appropriate actions. This paper proposes a novel correlation approach based on triggering events and common resources. One of the key concepts in our approach is triggering events, which are the (low-level) events that trigger alerts. By grouping alerts that share "similar" triggering events, a set of alerts can be partitioned into different clusters such that the alerts in the same cluster may correspond to the same attack. Our approach further examines whether the alerts in each cluster are consistent *with relevant network and host configurations, which help analysts to partially identify the severity of alerts and clusters. The other key concept in our approach is input and output resources. Intuitively,* input resources *are the necessary resources for an attack to succeed, and* output resources *are the resources that an attack supplies if successful. This paper proposes to model each attack through specifying input and output resources. By identifying the "common" resources between output resources of one attack and input resources of another, it discovers causal relationships between alert clusters and builds attack scenarios. The experimental results demonstrate the usefulness of the proposed techniques.*

1. Introduction

As more and more organizations and companies build networked systems to manage their information, network intrusion becomes a serious problem in recent years. At present, there is no single system capable of solving all security concerns. Different types of security systems are deployed into the networks to better protect the digital assets. These systems may comprise firewalls (e.g., ZoneAlarm [24]), intrusion detection systems (IDSs) (e.g., RealSecure Network 10/100 [9], Snort [17] and NIDES [11]), antivirus tools (e.g., Norton AntiVirus [19]), file integrity checkers (e.g., Tripwire [21]), and so forth. They usually serve for different security purposes, or serve for the same purpose through different methods. For example, firewalls focus on accepting, logging, or dropping network traffic, IDSs focus on detecting known attack patterns (signature-based IDSs) or abnormal behaviors (anomaly-based IDSs), antivirus tools focus on scanning viruses based on pre-defined virus signatures, and file integrity checkers monitor the activities on file systems such as file addition and deletion.

Although these security systems are complementary to each other, and combining the reports (i.e., alerts) from them can potentially get more comprehensive result about the threats from outside and inside sources, it is challenging for analysts or analysis tools to analyze these alerts due to the following reasons.

First, a single security system such as a network based IDS may flag thousands of alerts per day [12], and multiple security systems make the situation even worse. Large numbers of alerts may overwhelm the analysts. Second, among a large volume of alerts, a high proportion of them are false positives [12], some of them are low-severity alerts (e.g., an attack to an inactive port), and some others correspond to severe attacks. It is challenging to differentiate these alerts and take appropriate actions. The low level and high volume of the alerts also make extracting the global view of the adversary's attack strategy challenging. Third, different security systems usually run independently and may flag different alerts for a single attack. Discovering that these alerts are actually triggered by the same attack can be time-consuming, though it is critical in assessing the severity of

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

the alerts and the adversary's attack strategy.

To address the challenges, several alert correlation techniques have been proposed in recent years, including approaches based on similarity between alert attributes (e.g, [2, 12, 18, 22]), methods based on pre-defined attack scenarios (e.g., [6,13]), techniques based on pre/post-conditions of attacks (e.g., [3, 15, 20]), and approaches using multiple information sources [14, 16, 23]. Though effective in addressing some challenges, none of them dominates the others. Similarity based approaches group alerts based on the similarity between alert attributes; however, they are not good at discovering steps in a sequence of attacks. Pre-defined attack scenario based approaches work well for known scenarios; however, they cannot discover novel attack scenarios. Pre/post-condition based approaches can discover novel attack scenarios; however, specifying pre/post-conditions is time-consuming and error-prone. Multiple information sources based approaches correlate alerts from multiple information sources such as firewalls and IDSs; however, they are not good at discovering novel attack scenarios.

Our alert correlation techniques proposed in this paper address some limitations of the current correlation techniques. We propose a novel similarity measure based on triggering events, which helps us group alerts into clusters such that one cluster may correspond to one attack. We enhance the pre/post-condition based approaches through using input and output resources to facilitate the specification of pre-conditions and post-conditions. Intuitively, the pre-condition of an attack is the necessary condition for the attack to succeed, and the post-condition is the consequence if the attack does succeed. Accordingly, the *input resources* of an attack are the necessary resources for the attack to succeed, and the *output resources* of the attack are the resources that the attack can supply if successful.

Compared with the approaches in [3, 15] which use predicates to describe pre-conditions and post-conditions, our input and output resource based approach has several advantages. (1) Using predicates to specify pre/post-conditions may introduce too many predicates. Whereas input and output resource types are rather limited compared with the types of predicates and are easy to specify. (2) Since different experts may use different predicates to represent the same condition, or use the same predicate to represent different conditions, it is usually not easy to discover implication relationships between predicates and match post-conditions with pre-conditions. Whereas input and output resource types are rather stable, straightforward to match and easy to accommodate new attacks.

Our approach proposes to correlate alerts in three stages. The key concept in the first stage is *triggering events*, which are the events observed by security systems that trigger an alert. We observe that although different security systems may flag different alerts for the same attack, the events that trigger these alerts must be the same. In this paper, although high-level events are possible, we focus on low-level events (e.g., a TCP connection). Based on this observation, we find triggering events for each alert, and cluster alerts that share "similar" triggering events. The alerts in one cluster may correspond to one attack.

In the second stage, we further identify the severity of some alerts and clusters. This is done through examining whether the alerts are *consistent* with their relevant network and host configurations.

In the third stage, we build attack scenarios through input and output resources. We observe that the causal relationships between individual attacks can be discovered through identifying the "common" resources between the output resources of one attack and the input resources of another. For example, *Sadmind_Ping* attack can output the status information of sadmind daemon (service resources), where *sadmind* service is necessary to successfully launch *Sadmind_Amslverify_Overflow* attack. Then we can correlate these two attacks. These causal relationships can help us connect alert clusters and build attack scenarios.

The remainder of this paper is organized as follows. Section 2 presents our alert correlation techniques, including alert clustering through triggering events, alerts and relevant configuration examination, and attack scenario construction based on input and output resources. Section 3 presents experimental results to show the effectiveness of our techniques. Section 4 discusses related work, and Section 5 concludes this paper and points out some future research directions.

2. Alert Correlation based on Triggering Events and Common Resources

We present our major techniques in this section. We start by introducing definitions such as alerts, events, configurations and resources in Subsection 2.1. Given a set of alerts, we are interested in what events trigger each alert, from which we can cluster the alerts that share the "similar" triggering events. These techniques are presented in Subsections 2.2, 2.3 and 2.4. After alert clustering, we use the information about network and host configurations to examine the alerts in each cluster, which provides us opportunities to identify the severity of some alerts and clusters. This technique is presented in Subsection 2.5. The technique on constructing attack scenarios is presented in Subsection 2.6, which focuses on discovering causal relationships based on the input and output resources.

2.1. Alerts, Events, Configurations and Resources

Different security systems may output alerts in various formats (e.g., in a flat text file, in a relational database, or

in a stream of IDMEF [4] messages). We can always extract a set of attributes (i.e., attribute names and values) associated with the alerts. Events are security related occurrences observed by security systems. Configurations encode the information about software and hardware in a host or a network. Resources encode the "system resources" an attack may require to use or can possibly supply if it succeeds. They can all be represented as a set of attributes (attribute names and values). Formally, an *alert type* (or *event type*, or *configuration type*, or *resource type*) is a set S of attribute names, where each attribute name $a_i \in S$ has a domain D_i. A type T *alert* t (or *event* e, or *configuration* c, or *resource* r) is a tuple on attribute name set in T, where each element in the tuple is a value in the domain of the corresponding attribute name. In this paper, for the sake of presentation, we assume each alert and event respectively has at least two attributes: *StartTime* and *EndTime*. If an alert or event only has one timestamp, we assume StartTime and EndTime have the same value. For convenience, we denote the type of alert t, event e and resource r as $Type(t)$, $Type(e)$ and $Type(r)$, respectively. In the following, we may use attributes to denote either attribute names, attribute values, or both if it is not necessary to differentiate between them.

Here we give a series of examples and discussion related to alert types, alerts, event types, events, configuration types, configurations, resource types, and resources. For brevity, we omit the domain of each attribute. As the first example, we define an alert type *Sadmind_Amslverify_Overflow* = {SrcIP, SrcPort, TargetIP, TargetPort, StartTime, EndTime}. A type *Sadmind_Amslverify_Overflow* alert t = {SrcIP = 10.10.1.10, SrcPort = 683, TargetIP = 10.10.1.1, TargetPort = 32773, StartTime = 03-07-2004 18:10:21, EndTime = 03-07-2004 18:10:21} can describe an *Sadmind_Amslverify_Overflow* alert from IP address 10.10.1.10 to IP address 10.10.1.1.

As the second example, we define an event type *malicious sadmind NETMGT_PROC_SERVICE Request* = {SrcIP, SrcPort, TargetIP, TargetPort, StartTime, EndTime} and a corresponding event e = {SrcIP = 10.10.1.10, SrcPort = 683, TargetIP = 10.10.1.1, TargetPort = 32773, StartTime = 03-07-2004 18:10:21, EndTime = 03-07-2004 18:10:21}. Though high-level events are possible, in this paper we are more interested in low-level events. For example, a TCP connection exploiting the vulnerability in a ftp server, and a read operation on a protected file. These low-level events may trigger the security systems to flag alerts. For example, a ftp connection including some special data such as "~^$.—*+()[]{}" [10] in its payload may trigger a *FTP_Glob_Expansion* alert by a network based IDS, and may trigger a *NEW_CLIENT* alert by a network anomaly detector. The *malicious sadmind NETMGT_PROC_SERVICE Request* event may trigger *Sadmind_Amslverify_Overflow* alert if it is captured by a RealSecure network sensor.

As the third example, we define a configuration type *HostFTPConfig* = {HostIP, HostName, OS, FTPSoftware, FTPOpenPort}, and a type *HostFTPConfig* configuration c = {HostIP = 10.10.1.7, HostName = foo, OS = Solaris 8, FTPSoftware = FTP Server, FTPOpenPort = 21}. We are particularly interested in the critical software which may have vulnerabilities, for example, a ftp server program and its open port in a host. We further classify the configurations into two categories: host configuration and network configuration. The aforementioned *HostFTPConfig* is a host configuration listing the ftp software and open port. Network configurations specify the setting about the whole network. For example, the access control list (ACL) in a firewall, which controls the inbound and outbound traffic for the whole network. A type *NetTrafficControlConfig* configuration c' = {Source = any, Destination = 10.10.1.8, DestPort = 80, Protocol = tcp, Action = accept} is an example of network configuration controlling the inbound traffic to 10.10.1.8 at TCP port 80.

As the last example, we define a resource type *file* = {HostIP, Path}, a resource type *network_service* = {HostIP, HostPort, ServiceName}, and a resource type *privilege* = {HostIP, Access}. A type *file* resource r_1 = {HostIP = 10.10.1.9, Path = /home/Bob/doc/info.txt}, a type *network_service* resource r_2 = {HostIP = 10.10.1.9, HostPort = 21, ServiceName = ftp}, and a type *privilege* resource r_3 = {HostIP = 10.10.1.9, Access = AdminAccess}.

2.2. Triggering Events for Alerts

As we mentioned, a single event may trigger different alerts for different security systems. Since security systems may not necessarily tell the analysts what events trigger an alert, it is usually necessary to discover the *triggering events* for alerts. Given an alert, we are interested in the set of event types which *may trigger* an alert type, and the attribute values for each triggering event. Domain knowledge is essential for the discovery of triggering events.

Definition 1 *Given an alert type T_t, the* set of triggering event types *for T_t is a set $\mathcal{T}$ of event types, where for each event type $T_e \in \mathcal{T}$, there is an attribute mapping function f that maps attribute names in T_t to attribute names in T_e. Given a type T_t alert t, the* triggering event set *for t is a set E of events, where for each $T_e \in \mathcal{T}$, there is a type T_e event $e \in E$, and the attribute values of e are instantiated by the attribute values in t through the corresponding attribute mapping function.*

Example 1 *Given an alert type T_t =* Sadmind_Amslverify_Overflow*, the set of triggering*

event types is $\{T_e\}$*, where* T_e = Malicious sadmind NETMGT_PROC_SERVICE Request*, and the attribute mapping function* f *has* $f(T_t.SrcIP) = T_e.SrcIP$, $f(T_t.SrcPort) = T_e.SrcPort$, $f(T_t.TargetIP) = T_e.TargetIP$, $f(T_t.TargetPort) = T_e.TargetPort$, $f(T_t.StartTime) = T_e.StartTime$ *and* $f(T_t.EndTime) = T_e.EndTime$*. Given a type* Sadmind_Amslverify_Overflow *alert* $t = \{$*SrcIP* = 10.10.1.10, *SrcPort* = 683, *TargetIP* = 10.10.1.1, *TargetPort* = 32773, *StartTime = 03-07-2004 18:10:21, EndTime = 03-07-2004 18:10:21*$\}$*, we know the triggering event set has one type* Malicious NETMGT_PROC_SERVICE Request *event* $e = \{$*SrcIP* = 10.10.1.10, *SrcPort* = 683, *TargetIP* = 10.10.1.1, *TargetPort* = 32773, *StartTime = 03-07-2004 18:10:21, EndTime = 03-07-2004 18:10:21*$\}$.

Triggering events provide us an opportunity to find different alerts that may correspond to the same attack. Given a set of alerts, first we can discover the triggering event set for each alert, then we can put individual alerts into clusters if the alerts in the same cluster share the "similar" triggering events. In the following, we simply use the term events instead of triggering events if it is clear from the context. We first discuss the event inference, then define "similarity" between alerts through which our clustering algorithm is introduced.

2.3. Inference between Events

Intuitively, two events are "similar" if they have the same event type, and their attribute names and values are also the same. However, considering the existence of implication relationships between events (the occurrence of one event implies the occurrence of another event), we realize that the concept of "similarity" can be extended beyond this intuition to accommodate event implication.

We first give examples to illustrate the implication relationships. Consider two events: the recursive deletion of directory "/home/Bob/doc" and the deletion of file "/home/Bob/doc/info.txt". The first event can imply the second one because "info.txt" is one of the files in that directory. As another example, we know an event type *restricted_file_write may imply* an event type *filesystem_integrity_violation*. On the other hand, a recursive directory deletion does not necessary imply a file deletion if the file is not in the same directory. For example, the recursive deletion of directory "/home/Bob/doc" cannot imply the deletion of file "/home/Alice/doc/info.txt". This observation tells us when we introduce the implication relationships between events, we not only need to examine the semantics of event types, but also the relationships between attribute names and values. We use *may-imply* to refer to the implication between event types, and use *imply* to refer to the implication between events (including types and their related attributes names and values). For convenience, we denote event e_1 implies event e_2 as $e_1 \rightarrow e_2$.

We introduce a binary *specific-general* relation to help us identify implication relationships. Formally, given two concepts (e.g., two event types, two attribute names, etc) a_1 and a_2, a *specific-general* relation between a_1 and a_2 maps low-level (specific) concept a_1 to high-level (general) concept a_2, and is denoted as $a_1 \leq a_2$ (for convenience, we may also refer to specific-general relation as "$\leq$" relation in this paper). Specific-general relation is reflexive (we have $a \leq a$ for any concept a), antisymmetric and transitive, and it essentially is a partial order over a set of concepts (which is modeled as *concept hierarchy* in data mining [8]).

Specific-general relations can be applied to event types. For example, we can define *file_deletion* $\leq$ *recursive_directory_deletion* and *restricted_file_write* $\leq$ *filesystem_integrity_violation*. Here domain knowledge is necessary to determine whether *EventType1* may imply *EventType2* or *EventType2* may imply *EventType1* even if *EventType1* $\leq$ *EventType2* or *EventType2* $\leq$ *EventType1* is decided. For example, it is straightforward for an expert to decide *recursive_directory_deletion* may imply *file_deletion* and *restricted_file_write* may imply *filesystem_integrity_violation*. Our next job is to decide the relationships between attribute names and values.

Again, the relationships between attributes are determined through specific-general relations. As an example, for specific-general relations between attribute names, we can define $file \leq directory$ and $host \leq network$. In addition, we are interested in whether "$\leq$" relation is satisfied once the attribute names are replaced by their values. For example, under $file \leq directory$ relation, we have "/home/Bob/doc/info.txt" $\leq$ "/home/Bob/doc", and under $host \leq network$ relation, we have $10.10.1.10 \leq 10.10.1.0/24$. In the following, when referring to "$\leq$" relations, we may not distinguish between attribute names and values if it is not necessary.

It is worth mentioning that as a special case, timestamp attributes (StartTime and EndTime) have different characteristics compared with other attributes in that even if two triggering events actually refer to a same event, they may not have the exactly same timestamps due to the clock discrepancy in different systems or event propagation over the network. We propose to use *temporal constraint* to evaluate whether a set of events are "similar" w.r.t. timestamps.

Definition 2 *Consider a set* E *of events and a time interval* λ*.* E *satisfies the* temporal constraint λ *if and only if for any* $e, e' \in E$ *(*$e \neq e'$*),* $|e.StartTime - e'.StartTime| \leq \lambda$ *and* $|e.EndTime - e'.EndTime| \leq \lambda$*.*

Based on "$\leq$" relations and a temporal constraint λ, we outline an algorithm (shown in Figure 1) to determine

whether event e_1 implies event e_2. The basic idea is that we first identify whether $Type(e_1)$ may imply $Type(e_2)$. If this is the case, we further check "$\leq$" relations between attribute names and values, and examine the temporal constraint to see whether e_1 implies e_2.

Algorithm 1. Determining if event e_1 implies event e_2
Input: Two events e_1 and e_2 and a temporal constraint λ.
Output: *True* if $e_1 \rightarrow e_2$; otherwise *false*.
Method:

Assume the attribute name sets for e_1 and e_2 are A_1 and A_2, respectively. Initialize *result=false*.
1. **If** $Type(e_1)$ may imply $Type(e_2)$
2. **If** $Type(e_1) \leq Type(e_2)$
3. Find a mapping such that $\forall a_1 \in A_1'(A_1' \subseteq A_1)$ and $\forall a_2 \in A_2'(A_2' \subseteq A_2)$, we have $a_1 \leq a_2$
4. **Else** Find a mapping such that $\forall a_2 \in A_2'(A_2' \subseteq A_2)$ and $\forall a_1 \in A_1'(A_1' \subseteq A_1)$, we have $a_2 \leq a_1$
5. Replace names with values for all "$\leq$" relations.
6. **If** all "$\leq$" relations are satisfied in step 5
7. **If** e_1 and e_2 satisfy constraint λ, Let *result=true*
8. **Output** *result*.

Figure 1. An algorithm to discover implication relationship between events.

2.4. Clustering Alerts Using Triggering Events

Intuitively, we intend to group individual alerts into clusters such that all alerts in the same cluster either share the same triggering events, or their triggering events have implication relationships. To formalize this intuition, we first define *similarity* between alerts.

Definition 3 *Consider a set S of alerts $\{t_1, t_2, \cdots, t_n\}$ and a temporal constraint λ. Assume the triggering event sets for $t_1, t_2, \cdots, t_n$ are $E_1, E_2, \cdots, E_n$, respectively. All alerts in S are* similar *if and only if there exist $e_1 \in E_1$, $e_2 \in E_2$, $\cdots$, $e_n \in E_n$ such that for any two events e_i and e_j in $\{e_1, e_2, \cdots, e_n\}$, we have $e_i \rightarrow e_j$ or $e_j \rightarrow e_i$.*

The idea of Definition 3 can be demonstrated by an example. Two alerts are similar if either their triggering events have the same type, attribute names and values, and their timestamps satisfy the given temporal constraint, or their triggering events have implication relationship. Since two events e_1 and e_2 with everything the same is a special case of $e_1 \rightarrow e_2$ or $e_2 \rightarrow e_1$, we can combine these two cases and use implication relationships to define similarity.

Given a set S_u of alerts, we can cluster them based on Definition 3. Intuitively, we can iteratively pick a subset of alerts from S_u such that all the alerts in this subset are similar. However, we have to solve a problem before we can apply this operation. This problem can be demonstrated by an example. Suppose we have three alerts t_1, t_2 and t_3. They are of the same type and have the same attribute names and values except for timestamp values, and their triggering event sets are $\{e_1\}$, $\{e_2\}$, and $\{e_3\}$, respectively. Assume the temporal constraint λ = 1 second, $e_1.StartTime = e_1.EndTime$ = 03-07-2004 18:20:21, $e_2.StartTime = e_2.EndTime$ = 03-07-2004 18:20:22, and $e_3.StartTime = e_3.EndTime$ = 03-07-2004 18:20:23. Based on Definition 3, t_1 and t_2 are similar, and t_2 and t_3 are similar. Thus t_2 can be put into a cluster either with t_1, or with t_3. To solve this ambiguity, we apply a rule named "earlier timestamp first", where the cluster with the earlier (StartTime) alerts will get as many alerts as possible. Applying this rule to the example, we let t_1 and t_2 be in the same cluster. Algorithm 2 (Figure 2) outlines the alert clustering through applying this rule. In Algorithm 2, line 1 prepares the alert set and initializes some variables. Lines 2 through 6 are a loop, which always looks for the alerts that are similar to the first alert in the alert set and puts them into a cluster. This loop will not stop until the alert set is empty. Line 7 finally outputs all clusters.

Algorithm 2. Alert Clustering via Triggering Events
Input: A set S_u of alerts and a temporal constraint λ.
Output: A set C of clusters.
Method:

1. Sort the set S_u of alerts ascendingly on StartTime, and name it S_o. Initialize $C = \emptyset$, and let $i = 1$.
2. **While** S_o is not empty
3. Let the alert with the earliest StartTime in S_o be t.
4. Find set $S' \subseteq S_o$ such that $\{t\} \cup S'$ are similar.
5. Remove $\{t\} \cup S'$ from S_o into a set C_i.
6. Put C_i into C. Let $i = i + 1$.
7. **Output** C.

Figure 2. An algorithm to perform alert clustering based on triggering events.

An interesting observation is that after alert clustering, we may mark some clusters with low severity through examining whether the alerts are consistent with relevant configurations. This will be further discussed in Subsection 2.5.

2.5. Consistency between Alerts and Configurations

Host and network configurations provide us an opportunity to verify the consistency or discover the in-

consistency between alerts and their relevant configurations. The consistency between an alert and its related configurations can be verified through examining the attributes of the alert and the configurations. For example, consider an *FTP_Glob_Expansion* alert $t = \{$SrcIP = 172.16.1.7, SrcPort = 1042, TargetIP = 10.10.1.7, TargetPort = 21, StartTime = 03-07-2004 18:20:21, EndTime = 03-07-2004 18:20:21$\}$ and a *HostFTPConfig* configuration $c = \{$HostIP = 10.10.1.7, HostName = foo, OS = Solaris 8, FTPSoftware = FTP Server, FTPOpenPort = 21$\}$. Alert t is consistent with configuration c because it exploits a host 10.10.1.7 at port 21 which is an open port listed in this host's configuration. We formalize this relationship in Definition 4.

Definition 4 *Consider an alert type T_t and a configuration type T_c. A* consistent condition *for T_t w.r.t. T_c is a logical formula including attribute names in T_t and T_c. Given a type T_t alert t and a type T_c configuration c, t* is consistent (or inconsistent, resp.) with *c if the formula is evaluated to* True (or False, resp.) *where attribute names in the formula are replaced with the values in t and c.*

Example 2 *Given an alert type* FTP_Glob_Expansion *(T_t) and a configuration type* HostFTPConfig *(T_c), we define $T_t.TargetIP = T_c.HostIP \wedge T_t.TargetPort = T_c.FTPOpenPort$ as a consistent condition for T_t w.r.t. T_c. Given an* FTP_Glob_Expansion *alert $t = \{$SrcIP = 172.16.1.7, SrcPort = 1042, TargetIP = 10.10.1.7, TargetPort = 21, StartTime = 03-07-2004 18:20:21, EndTime = 03-07-2004 18:20:21$\}$ and a* HostFTPConfig *configuration $c = \{$HostIP = 10.10.1.7, HostName = foo, OS = Solaris 8, FTPSoftware = FTP Server, FTPOpenPort = 21$\}$, the consistent condition is evaluated to* True *using attribute values in t and c. Then we know t is consistent with c.*

Consistency and inconsistency relationships between alerts and configurations provide us a way to classify the alerts. We can mark each alert as consistent or inconsistent with the related configurations. A consistent alert tells us the corresponding attack could be possible due to the potential vulnerabilities in the configuration. A special case worth mentioning is that sometimes a consistent alert is a low-severity alert. For example, if a firewall reports a *FWROUTE* alert saying that an inbound packet is blocked, which is consistent with the ACL configuration of the firewall, this alert is less severe because the corresponding connection is blocked. On the other hand, an inconsistent alert may be of low severity because the corresponding attack could not succeed (e.g., an adversary tries to connect to a port which is not open). A special case is that a configuration could be compromised (e.g., an adversary installs malicious programs and opens new ports) without the notice of the legitimate users, and then the corresponding attack may succeed. In this case, the "inconsistent" alert (which actually is not an inconsistent alert because the configuration is changed) deserves more investigation.

We can apply consistency and inconsistency relationships to alert clusters to determine the severity of some clusters. For example, assume a *FWROUTE* alert (reported by a firewall denoting that a connection is blocked) and a *NEW_CLIENT* alert (reported by a network anomaly detector denoting that a new client requests a server) are in the same cluster, and *FWROUTE* is consistent with its configuration. Since *FWROUTE* denies the requested connection, the related attack cannot be successful and this cluster is less severe. Then we could put more efforts on investigating other possibly severe clusters.

2.6. Attack Scenario Construction based on Input and Output Resources

Our approach further determines the causal relationships between alert clusters. We are interested in how individual attacks (represented by alert clusters) are combined to achieve the adversary's goal. The observation tells us that in a sequence of attacks, some attacks have to be performed earlier in order to launch later attacks. For example, an adversary always installs DDoS software before actually launching DDoS attacks. If we are able to capture these causal relationships, it may help us build stepwise attack scenarios and reveal the adversary's attack strategy.

Our approach to modeling causal relationships between attacks is inspired by the pre/post-condition based alert correlation techniques [3, 15, 20]. However, as we mentioned in Section 1, since we use resources to specify pre/post-conditions, compared with the predicates based specification [3, 15], we have several advantages such as the ease of specifying and (partially) matching input and output resources, and the ease of accommodating new attacks. Our approach is based on the observation that the causal relationships between attacks can be captured through examining output resources of one attack with input resources of another. Informally, input resources are the necessary resources for an attack to succeed, and output resources are the resources an attack can supply if successful.

We extend our model for alerts (or alert types, resp.) to accommodate input and out resources (or input and output resource types, resp.). We call them *extended alerts* (or *extended alert types*, resp.) after this extension. Considering that the resource attribute names may not always be the same as the alert attribute names, we further use functions to map the alert attributes to resource attributes. In the following, we formalize extended alert types and extended alerts.

Definition 5 *An* extended alert type *T is a triple ($\mathcal{T}_i$,* attr_names, *$\mathcal{T}_o$), where (1)* attr_names *is a set of attribute*

names (including StartTime and EndTime) where each attribute name a_j has a domain D_j, (2) $\mathcal{T}_i$ and $\mathcal{T}_o$ are a set of resource types, respectively, and (3) for each $T_i \in \mathcal{T}_i$ and $T_o \in \mathcal{T}_o$, there exist attribute mapping functions f_i and f_o that map attribute names in attr_names *to attribute names in T_i and T_o, respectively.*

A type T extended alert *t is a triple (*input, attributes, output*), where (1)* attributes *is a tuple on attr_names, (2)* input *and* output *are a set of resources, respectively, and (3) for each $T_i \in \mathcal{T}_i$ and $T_o \in \mathcal{T}_o$, there exist resources $r_i \in$* input *and $r_o \in$* output*, respectively, where their attribute values are instantiated by the corresponding attribute values in* attributes *through attribute mapping functions.*

Actually, *attr_names* in an extended alert type is an alert type, and *attributes* in an extended alert is an alert that we defined in Subsection 2.1. In the remaining part of this paper, we may simply use alert types (or alerts, resp.) when it is not necessary to differentiate between extended alert types and alert types (or extended alerts and alerts, resp.).

Example 3 *Define an* Sadmind_Amslverify_Overflow *(T) extend alert type as* { {*network_service*} , {*SrcIP, SrcPort, TargetIP, TargetPort, StartTime, EndTime*}, {*privilege*} }*, where network_service (T_i) =* {*HostIP, HostPort, ServiceName*} *and privilege (T_o) =* {*HostIP, Access*}*. For attribute mapping, we have $f_i(T.TargetIP) = T_i.HostIP$, $f_i(T.TargetPort) = T_i.HostPort$, and $f_o(T.TargetIP) = T_o.HostIP$.*

Given a type Sadmind_Amslverify_Overflow *alert* {*SrcIP* = 10.10.1.10*, SrcPort* = 683*, TargetIP* = 10.10.1.1*, TargetPort* = 32773*, StartTime = 03-07-2004 18:10:21, EndTime = 03-07-2004 18:10:21*}*, we can get their input and output resources as* input = { {*HostIP* = 10.10.1.1*, HostPort = 32773, ServiceName = sadmind*} }*, and* output = { {*HostIP* = 10.10.1.1*, Access = AdminAccess*}}*. These three parts combined together are an extended alert.*

Please note in Definition 5, when performing attribute mapping from *attr_names* to $T_i \in \mathcal{T}_i$ and $T_o \in \mathcal{T}_o$, based on domain knowledge, we can mark some attributes in T_i and T_o as special attributes, where they have pre-determined values once the attribute values of resources in *input* and *output* are instantiated. For example, as shown in Example 3, *Access* attribute in *privilege* resource has a predetermined *AdminAccess* value.

Similar to the implication relationship between events, one resource r_1 can imply another resource r_2 (we use $r_1 \rightarrow r_2$ to represent r_1 implies r_2). For example, a *privilege* resource {HostIP = 10.10.1.9, Access = AdminAccess} implies another *privilege* resource {HostIP = 10.10.1.9, Access = UserAccess}. Please note two resources r_1 and r_2 have their types, attribute names and values all the same is a special case of $r_1 \rightarrow r_2$ or $r_2 \rightarrow r_1$. The implication relationships between resources can be determined through specific-general relations and a similar procedure described in Subsection 2.3 (The difference is that in this paper we do not associate resources with timestamps). For space reasons, we do not repeat it here.

We can identify causal relationships between attacks through discovering "common" resources between input and output resources. Intuitively, if one attack's output resources include one resource in another attack's input resources, we can correlate these two attacks together. We formalize this intuition as follows.

Definition 6 *Given two extended alerts t =(*input, attributes, output*) and t' = (*input', attributes', output'*), t and t' are* causally correlated *if there exist $r_o \in$* output *and $r'_i \in$* input' *such that r_o implies r'_i and $t.EndTime < t'.StartTime$.*

Example 4 *Consider two alerts t and t' ($Type(t) =$ SCAN_NMAP_TCP and $Type(t') =$ FTP_Glob_Expansion). Suppose the output resource of t is a* network_service *resource* {*HostIP* = 10.10.1.7*, HostPort = 21, ServiceName = ftp*}*, the input resource of t' is a* network_service *resource* {*HostIP* = 10.10.1.7*, HostPort = 21, ServiceName = ftp*}*, and $t.EndTime < t'.StartTime$. Since the output resource of t and the input resource of t' are the same, we know that t and t' are causally correlated.*

We also refer to "causally-correlate" relationships introduced in Definition 6 as causal relationships, which provide us opportunities to build attack scenarios. Consider a set of alerts reported by different security systems. We can group alerts into clusters using triggering events. Each cluster may correspond to one attack. Through discovering causal relationships between alerts in different clusters, we can link different clusters and construct the attack scenarios. Definition 7 further formalizes this intuition.

Definition 7 *Consider a set C of clusters where each cluster is a set C_i of alerts. A* scenario graph *$SG = (V, A)$ is a directed acyclic graph, where (1) V is the vertex set, and A is the edge set, (2) each vertex $v \in V$ is a cluster in C, and (3) there is an edge $(v_1, v_2) \in A$ if and only if there exist $t_1 \in v_1$ and $t_2 \in v_2$ such that t_1 and t_2 are* causally correlated.

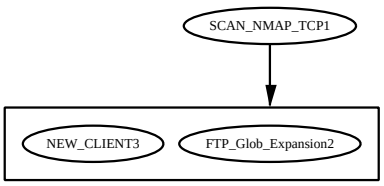

Figure 3. An example scenario graph

Example 5 *An example of scenario graph is shown in Figure 3. The string inside each node is the alert type followed by an ID (we will follow this convention in our experiments). This scenario has two clusters:* C_1 ={SCAN_NMAP_TCP1} *and* C_2 ={NEW_CLIENT3, FTP_Glob_Expansion2}, *where* SCAN_NMAP_TCP1 *is reported by Snort,* FTP_Glob_Expansion2 *is reported by a RealSecure network sensor, and* NEW_CLIENT3 *is reported by a network anomaly detector. Assume* SCAN_NMAP_TCP1 *and* FTP_Glob_Expansion2 *are causally correlated. Then we can correlate* C_1 *and* C_2 *together as shown in Figure 3. Such graph clearly discloses an adversary's attack strategy.*

3. Experimental Results

To evaluate the effectiveness of our techniques, we performed experiments through DARPA Cyber Panel Program Grand Challenge Problem Release 3.2 (GCP) [5, 7], which is an attack scenario simulator. GCP simulator can simulate the behavior of sensors and generate alert streams. There are totally 10 types of sensors in the networks. All the sensors generate the alerts in IDMEF [4] messages.

The current implementation is a proof-of-concept system. We use Java as the programming language, and Microsoft SQL Server 2000 as the DBMS to save the alert data set and domain knowledge. Alert processing in our system can be divided into four stages. In the first stage, we concentrate on data preparation. We extract the attributes from the IDMEF messages generated by GCP simulator and put them into the database. All the necessary domain knowledge such as triggering event types and resource types are all put into the database. The second stage is the alert clustering stage. We group alerts into different clusters based on Algorithm 2 shown in Figure 2. The third stage is to examine the consistency or inconsistency between alerts and configurations. In the last stage, we use input and output resource based correlation techniques to discover causal relationships and build attack scenarios. To save our development effort, we use GraphViz [1] to draw scenario graphs.

The experiments were performed using Attack 1 scenario in GCP attack simulator. We chose 4 network enclaves, namely HQ enclave, APC enclave, Ship enclave and ATH enclave, to play this scenario. Attack 1 is a (agent-based) worm related attack. After the agent being activated, it performs a series of malicious actions such as communicating with an external host, getting malicious code and instructions, spreading from one network enclave to another, compromising hosts in the network enclaves, sniffing the network traffic, reading and modifying the sensitive files, sending the sensitive data to the external host, getting new malicious instructions, and so forth. For this scenario, we totally got 529 alerts with 16 different types.

Cluster ID	Alerts
2	NEW_CLIENT10, FWROUTE7
4	NEW_CLIENT25, FWROUTE27
54	NEW_CLIENT132, FTP_Globbing_Attack135
102	NEW_CLIENT6, FWROUTE5
116	NEW_CLIENT124, ServiceUnavailable125
132	NEW_CLIENT33, FWROUTE21
136	NEW_CLIENT122, ServiceUnavailable121
184	NEW_CLIENT24, FWROUTE34
236	NEW_CLIENT32, FWROUTE20
238	NEW_CLIENT49, FWROUTE57
242	NEW_CLIENT153, FTP_Globbing_Attack154
281	NEW_CLIENT29, FWROUTE26
333	NEW_CLIENT8, FWROUTE12
335	NEW_CLIENT30, FWROUTE39
340	NEW_CLIENT54, FWROUTE53
342	NEW_CLIENT50, FWROUTE56
385	NEW_CLIENT134, FTP_Globbing_Attack133

Table 1. All 2-alert clusters.

Our first goal is to evaluate the effectiveness of alert clustering proposed in this paper. For space reasons, we omit listing the set of triggering event types for each alert type. We set the temporal constraint $\lambda = 1$ second.

Totally we get 512 clusters from alert clustering. Among them there are 17 clusters, each of them comprises 2 alerts, and all other clusters are single-alert clusters. Table 1 lists all 2-alert clusters. From Table 1, we observe every cluster has a *NEW_CLIENT* alert, which is reported by network anomaly sensors denoting a new client requests a server (service). This is normal because the connection requests trigger these alerts. Both alerts in each cluster in Table 1 actually refer to the same network connection, which triggers different alerts for different systems.

Our next goal is to evaluate the effectiveness of consistent conditions in identifying the severity of some alerts and clusters. Among all alerts, we find 4 alerts inconsistent with their configurations. These 4 alerts are *NEW_CLIENT122*, *NEW_CLIENT124*, *ServiceUnavailable121* and *ServiceUnavailable125*. *NEW_CLIENT122* and *ServiceUnavailable121* target at port 111 on host 10.1.2.2, and *NEW_CLIENT124* and *ServiceUnavailable125* target at port 21 on host 10.1.2.2. They are also in Table 1 (Cluster ID = 136 and 116), which means these 4 alerts actually represent two attacks. Our investigation shows that both attacks are failed attempts (one is through sadmind exploit, and the other is through ftp globbing exploit) because the ports 111 and 21 are not open at host 10.1.2.2.

We also investigate the 2-alert clusters where one alert in the cluster is *FWROUTE*. In Table 1, there are 12 clusters that include *FWROUTE* alerts. These *FWROUTE* alerts are consistent with their configurations. Since *FWROUTE* represents connections being blocked, their impact to the network may not be severe. Thus the corresponding 2-alert

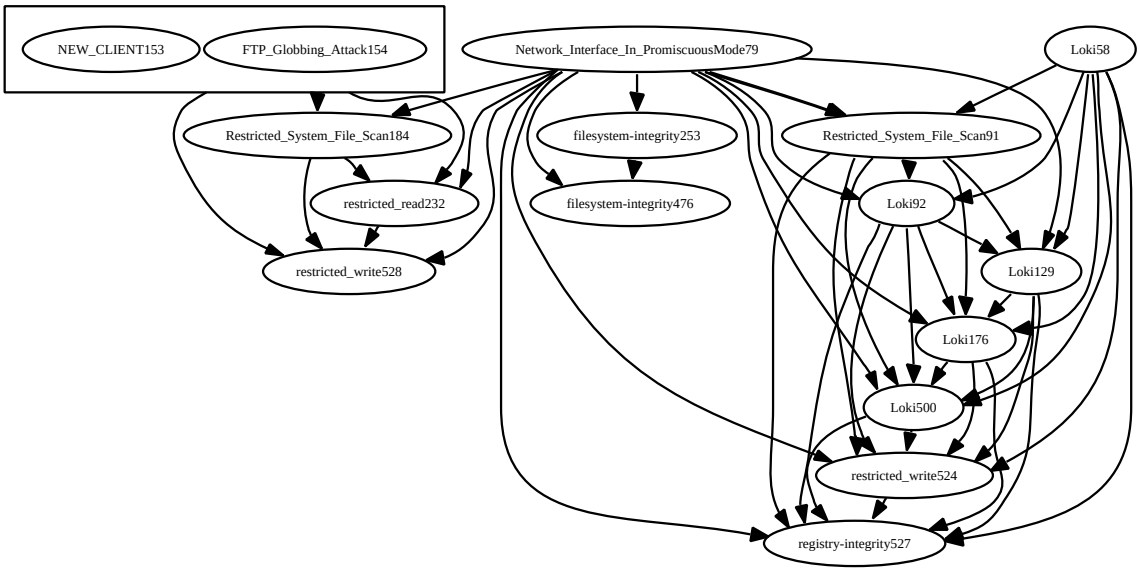

Figure 4. One Scenario Graph in HQ Enclave

clusters are low-severity clusters.

Our last goal is to evaluate the effectiveness of our techniques in building attack scenarios. We performed the experiments on the above data set and got 10 scenario graphs. For space reasons, we only show one of them in Figure 4.

Figure 4 is a scenario graph in HQ enclave. The alerts in this figure can be roughly divided into two parts: the right side part and the left side part. The right side part reveals that the adversaries iteratively read (*Restricted_System_File_Scan*), write (*restricted_write*) and sniff (*Network_Interface_In_PromiscuousMode*) sensitive data in HQ enclave, and use tunneling techniques such as *Loki* to secretly transmit data to the external host. The adversaries also modify critical files and keys (*filesystem-integrity* and *registry-integrity*) to disrupt the operation of the network. The left side part reveals that the adversaries use *FTP_Globbing_attack* to compromise the victim hosts, and also read and write sensitive data in the enclave. The attackers' strategy disclosed in this scenario graph is consistent with the description of GCP attack scenarios.

4. Related Work

In recent years, several alert correlation approaches have been proposed. They can be roughly classified into four categories.

The first category consists of the similarity based approaches [2,12,18,22]. These approaches group alerts based on the similarity between alert attributes. They are essentially clustering analysis. Our techniques also include alert clustering, which uses a novel similarity measure: triggering events. Triggering events are a similar concept to *root cause* [12] in that they represent the reason why the alerts are flagged. However, triggering events focus on low-level events (though high-level events are possible) and we assume security systems or domain knowledge can tell us triggering event types for each alert type, while root cause analysis concentrates on high-level events and clustering techniques are used to discover root causes. In [2], Cuppens proposes to use alert clustering to identify "the same attack occurrence", where expert rules are used to specify the similarity requirement between alerts.

The second category of alert correlation techniques is the pre-defined attack scenario based approaches [6, 13]. They work well for well-defined attack sequences, but they cannot discover novel attack scenarios.

The third category is the pre/post-condition based approaches [3, 15, 20]. Through (partially) matching the post-condition of one attack with the pre-condition of another, these approaches can discover novel attack scenarios. However, specifying pre-conditions and post-conditions for each attack is time-consuming and error-prone. Our techniques on building attack scenarios fall into this pre/post-condition based category. However, since our approach uses resources to specify pre/post-conditions, compared with the predicate based specification [3, 15], it is easy to specify and (partially) match input and output resources, and easy to accommodate new attacks.

The fourth category is the multiple information sources based approaches [14, 16, 23]. The mission-impact-based approach [16] ranks the alerts based on the overall impact

to the mission of the networks. M2D2 [14] proposes a formal model to describe the concepts and relations about various security systems. DOMINO [23] is a distributed intrusion detection architecture targeting at coordinated attack detection with potentially less false positives. We consider these techniques are complementary to ours.

5. Conclusion and Future Work

In this paper we proposed a correlation approach based on triggering events and input and output resources. One key concept in our approach is triggering events, which captures the (low-level) events that trigger alerts. We proposed to group different alerts into clusters if they share "similar" triggering events, through which we can identify the alerts that may correspond to the same attack. We further introduced network and host configurations into our model, and identified consistent and inconsistent alerts, which help us mark the severity of some alerts and clusters. The other key concept in our approach is input and output resources. We proposed to model each attack through specifying input and output resources, and discover causal relationships between attacks through identifying "common" resources between output resources of one attack and the input resources of another. This approach helps us identify logical connections between alert clusters and build attack scenarios. Our preliminary experimental results demonstrated the effectiveness of our techniques.

There are several future research directions. In this paper we mainly focus on low-level events as triggering events. An alternative way is to use high-level events, or combine low-level and high-level events. Another future direction is quantitatively evaluating different correlation approaches.

References

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

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

[3] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

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

[5] DARPA Cyber Panel Program. DARPA cyber panel program grand challenge problem. `http://www.grandchallengeproblem.net/`, 2003.

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

[7] J. Haines, D. Ryder, L. Tinnel, and S. Taylor. Validation of sensor alert correlators. *IEEE Security & Privacy Magazine*, 1(1):46–56, 2003.

[8] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

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

[10] Internet Security Systems, Inc. REALSECURE signatures reference guide. `http://www.iss.net/`.

[11] H. S. Javitz and A. Valdes. The NIDES statistical component: Description and justification. Technical report, SRI International, Mar. 1994.

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

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

[14] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.

[15] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002.

[16] P. Porras, M. Fong, and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 95–114, 2002.

[17] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA conference*, 1999.

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

[19] Symantec Corporation. Symantec's norton antivirus. `http://www.symantec.com`.

[20] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*, pages 31 – 38. ACM Press, September 2000.

[21] Tripwire, Inc. Tripwire changing monitoring and reporting solutions. `http://www.tripwire.com`.

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

[23] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the domino overlay system. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, February 2004.

[24] Zone Labs. Zonealarm pro. `http://www.zonelabs.com`.