A Comprehensive Approach to Intrusion Detection Alert Correlation

Fredrik Valeur, Giovanni Vigna, *Member*, *IEEE*, Christopher Kruegel, *Member*, *IEEE*, and Richard A. Kemmerer, *Fellow*, *IEEE*

Abstract—Alert correlation is a process that analyzes the alerts produced by one or more intrusion detection systems and provides a more succinct and high-level view of occurring or attempted intrusions. Even though the correlation process is often presented as a single step, the analysis is actually carried out by a number of components, each of which has a specific goal. Unfortunately, most approaches to correlation concentrate on just a few components of the process, providing formalisms and techniques that address only specific correlation issues. This paper presents a general correlation model that includes a comprehensive set of components and a framework based on this model. A tool using the framework has been applied to a number of well-known intrusion detection data sets to identify how each component contributes to the overall goals of correlation. The results of these experiments show that the correlation components are effective in achieving alert reduction and abstraction. They also show that the effectiveness of a component depends heavily on the nature of the data set analyzed.

Index Terms—Intrusion detection, alert correlation, alert reduction, correlation data sets.

1 Introduction

Recently, networks have evolved into a ubiquitous infrastructure. High-speed backbones and local area networks provide the end-user with bandwidths that are orders of magnitude larger than those available a few years ago. In addition, wireless technology is bringing connectivity to a number of devices, from laptops to cell phones and PDAs, creating a complex, highly dynamic network of systems. Most notably, the Internet has become a mission-critical infrastructure for governments, companies, institutions, and millions of everyday users.

The surveillance and security monitoring of the network infrastructure is mostly performed using Intrusion Detection Systems (IDSs). These systems analyze information about the activities performed in computer systems and networks, looking for evidence of malicious behavior. Attacks against a system manifest themselves in terms of events, which can be of differing nature and level of granularity. For example, events may be represented by network packets, operating system calls, audit records produced by operating system auditing facilities, or log messages produced by applications. The goal of intrusion detection is to analyze one or more event streams and identify manifestations of attacks. When an attack is detected, an alert that describes the type of the attack and the entities involved (e.g., hosts, processes, users) is produced.

Event streams are used by intrusion detection systems in two different ways, according to two different paradigms: *anomaly detection* and *misuse detection*. In anomaly detection systems [9], [14], [20], [23], [25], [52], historical data about a system's activity and/or specifications of the intended behavior of users and applications are used to build a profile of the "normal" operation of the system being monitored. The intrusion detection system then tries to identify patterns of activity that deviate from the defined profile. Misuse detection systems take a complementary approach [17], [19], [34], [38], [42], [50]. These systems are equipped with a number of attack descriptions (or "signatures") that are matched against the stream of audit data looking for evidence that one of the modeled attacks is occurring.

Misuse and anomaly detection both have advantages and disadvantages. Misuse detection systems can perform focused analysis of the audit data, and they usually produce only a few false positives, which are erroneous detections. However, these systems can detect only those attacks that have been modeled. Anomaly detection systems have the advantage of being able to detect abnormal behavior, which may reveal previously unknown attacks. This advantage is paid for in terms of a large number of false positives. Another disadvantage of anomaly detection systems is the difficulty of training a system with respect to a very dynamic environment.

The intrusion detection community has developed a number of different intrusion detection systems that perform intrusion detection in particular domains (e.g., hosts or networks), in specific environments (e.g., Windows NT or Solaris), and at different levels of abstraction (e.g., kernel-level tools or application-level tools). As more IDSs are developed, network security administrators are faced with the task of analyzing an increasing number of alerts resulting from the analysis of different event streams. In addition, IDSs are far from perfect and may produce both false positives and nonrelevant positives. Nonrelevant positives are alerts that correctly identify an attack, but

- *The authors are with the Department of Computer Science, University of California, Santa Barbara, CA, 93106.*
E-mail: {fredrik, vigna, chris, kemm}@cs.ucsb.edu.

Manuscript received 10 June 2004; accepted 10 Aug. 2004.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0082-0604.

1545-5971/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society

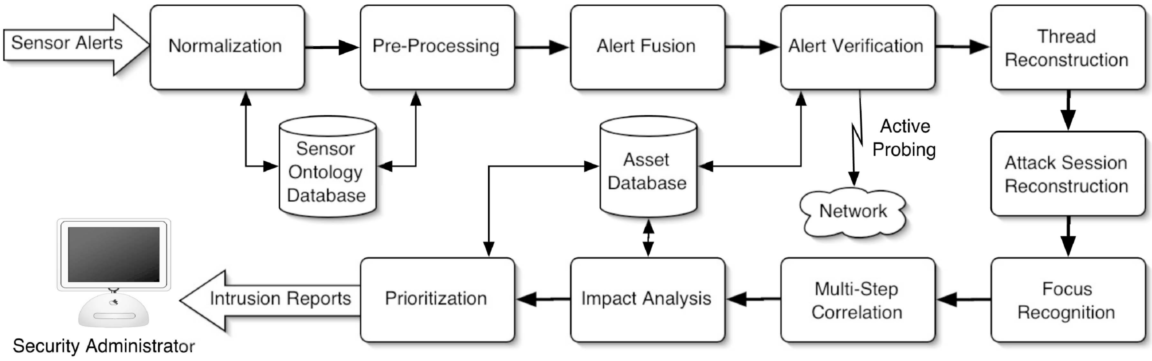

Fig. 1. Correlation process overview.

the attack fails to meet its objective. For instance, the attack may be exercising a vulnerability in a service that is not provided by the victim host. As an example, consider a “Code Red” worm that attacks a Linux Apache server. Although an actual attack is seen on the network, this attack has to fail because Apache is not vulnerable to the exploit utilized by the worm. Clearly, there is a need for tools and techniques that allow the administrator to aggregate and combine the outputs of multiple IDSs, filter out spurious or incorrect alerts (such as “Code Red” attacks against Apache installations), and provide a succinct, high-level view of the security state of the protected network.

To address this issue, researchers and vendors have proposed *alert correlation*, an analysis process that takes the alerts produced by intrusion detection systems and produces compact reports on the security status of the network under surveillance. Although a number of correlation approaches have been suggested, there is no consensus on what this process is or how it should be implemented and evaluated. In particular, existing correlation approaches operate on only a few aspects of the correlation process, such as the fusion of alerts that are generated by different intrusion detection systems in response to a single attack, or the identification of multistep attacks that represent a sequence of actions performed by the same attacker. In addition, correlation tools that do cover multiple aspects of the correlation process are evaluated “as a whole,” without an assessment of the effectiveness of each component of the analysis process. As a result, it is not clear if and how the different parts of the correlation process contribute to the overall goals of correlation.

This paper presents a comprehensive correlation approach and an analysis of its components. The correlation approach has been designed to be as complete as possible and to include all the aspects of correlation discussed in previous research efforts. We built a framework that implements the correlation process and used a tool based on the framework to analyze a number of data sets. Instead of just focusing on the overall effectiveness of correlation, this paper focuses on each aspect of the process separately, to provide insights into how each component of the correlation process contributes to alert reduction and abstraction and what properties of the data being analyzed make one component more effective than another. The results show that the effectiveness of correlation is highly dependent on the nature of the data sets. That is, different parts of the process contribute to correlation in different ways, depending on the nature of the data being analyzed.

The remainder of this paper is structured as follows: Section 2 presents the overall architecture of the correlation process. Section 3 presents the data sets that were analyzed. Section 4 provides a detailed description of the components of the correlation process and the results of applying each component to the sample data sets. Section 5 presents related work on alert correlation. Finally, Section 6 draws conclusions and outlines future work.

2 The Correlation Process

The main objective of the correlation process is to produce a succinct overview of security-related activity on the network. This process consists of a collection of components that transform intrusion detection sensor[1] alerts into intrusion reports. Because alerts can refer to different kinds of attacks at different levels of granularity, the correlation process cannot treat all alerts equally. Instead, it is necessary to provide a set of components that focus on different aspects of the overall correlation task.

Fig. 1 gives a graphical representation of the integrated correlation process that we implemented. The first two tasks are performed on all alerts. In the initial phase, a *normalization* component translates every alert that is received into a standardized format that is understood by all correlation components. This is necessary because alerts from different sensors can be encoded in different formats. Next, a *preprocessing* component augments the normalized alerts so that all required alert attributes (such as start-time, end-time, source, and target of the attack) are assigned meaningful values.

The next four correlation components of our framework all operate on single, or closely related, events. The *fusion* component is responsible for combining alerts that represent the independent detection of the same attack instance by different intrusion detection systems. The task of the *verification* component is to take a single alert and determine the success of the attack that corresponds to this alert. The idea is that alerts that correspond to failed attacks should be

[1] In this paper, we use *intrusion detection system* and *intrusion detection sensor* (or just *sensor*) interchangeably.

appropriately tagged and their influence on the correlation process should be decreased. The task of the *thread reconstruction* component is to combine a series of alerts that refer to attacks launched by a single attacker against a single target. The *attack session reconstruction* component associates network-based alerts with host-based alerts that are related to the same attack.

The next two components in our framework operate on alerts that involve a potentially large number of different hosts. The *focus recognition* component has the task of identifying hosts that are either the source or the target of a substantial number of attacks. This is used to identify denial-of-service (DoS) attacks or port scanning attempts. The *multistep correlation* component has the task of identifying common attack patterns such as island-hopping attacks.[2] These patterns are composed of a sequence of individual attacks, which can occur at different points in the network.

The final components of the correlation process contextualize the alerts with respect to a specific target network. The *impact analysis* component determines the impact of the detected attacks on the operation of the network being monitored and on the assets that are targeted by the malicious activity. Based on this analysis, the *prioritization* component assigns an appropriate priority to every alert. This priority information is important for quickly discarding information that is irrelevant or of less importance to a particular site.

Alerts that are correlated by one component of our framework are used as input by the next component. However, it is not necessary that all alerts pass through the same components sequentially. Some components can operate in parallel, and it is even possible that alerts output by a sequence of components are fed back as input to a previous component of the process.

In this section, we have given just a brief presentation of each of the components of the correlation process. A detailed description of the process and of each of its various components is presented in Section 4.

3 Data Sets

Since their first introduction, intrusion detection systems have been evaluated using various means. The most common approach has been to launch attacks within some background activity and test the ability of the systems to detect the manifestations of the attacks in the event stream.

Over the past six years, DARPA has sponsored a series of intrusion detection evaluation efforts. More precisely, the first evaluation was carried out by the MIT Lincoln Laboratory in 1998 [27], followed by a second, more systematic evaluation in 1999 [11], [12]. In both evaluations, a number of attacks, including portscans, remote compromises, local privilege escalation attempts, and denial-of-service attacks [21], were carried out on a testbed network and the relevant auditing events (network traffic and OS audit records) were collected at a number of critical points.

Even though these evaluations were not perfect [29], they produced data sets that became a reference point in the evaluation of intrusion detection functionality. The success of these evaluation efforts motivated the MIT Lincoln Laboratory group to generate new data sets in 2000. Because of the growing interest in evaluating techniques for the detection of complex attacks, these new data sets were geared toward multistep attack scenarios.

As the focus of many researchers shifted from low-level attack detection to high-level attack analysis and goal recognition, it was necessary to devise new data sets and evaluation procedures. The DARPA-sponsored Cyber Panel Correlation Technology Validation (CTV) effort [16], carried out in 2002, was motivated by the need to assess correlation systems. This evaluation was performed on a testbed network with 59 hosts that were divided into four protected enclaves and an additional part that represented the public Internet. Several runs of 14 different attacks, divided into four classes (denial-of-service, data theft, Web defacement, and backdoor installation), were performed, using only publicly available attack tools. Background traffic (e.g., Web and mail traffic) was simulated using scripted applications. The alerts produced by a number of sensors during these attacks were collected and processed by the correlation tools being evaluated.

The Defcon 9 data set is another data set that is commonly used in IDS evaluations. Defcon is a yearly underground hacking conference. The conference includes a hacker competition, called Capture The Flag (CTF). During the competition, all network traffic was recorded and then made publicly available. The Defcon 9 data set has several properties that make it very different from "real world" network traffic. It contains an artificially high amount of attack traffic, no background traffic, and only a small number of IP addresses. In spite of these shortcomings, this data set is very useful in IDS testing as it represents a worst-case scenario of the amount of attack traffic an IDS will receive and, thus, can be utilized for stress-testing.

Although the aforementioned data sets are a tremendous asset for the intrusion detection community, they have problems that make them unsuitable for the evaluation of the effectiveness of some of the components of the correlation process. First of all, some of the data sets were created to evaluate intrusion detection sensors and not to assess correlation tools. These data sets do not include sensor alerts (i.e., output from the intrusion detection systems). Therefore, the alerts have to be generated by running specific sensors on the collected event streams.

Another problem is that the offline nature of these data sets makes it impossible to perform alert verification because the victim of an attack must be analyzed in real time to determine if the attack is relevant or not. In addition, the impact of attacks on the protected network and on the mission that the network infrastructure supports are impossible to evaluate, due to the lack of a mission model and its relationship to the network assets. Finally, network health monitoring information is usually not provided, which makes it difficult to determine the actual impact of the attacks.

[2] An island-hopping attack refers to the situation in which an attacker breaks into a host and uses it as a launch pad for more attacks.

The data sets presented all suffer from not being representative of real world traffic. The DARPA sponsored data sets are all synthetically generated, and questions have been raised about the realism of the background traffic models used [29]. Also, none of the data sets presented were recorded on a network connected to the Internet. Internet traffic usually contains a fairly large amount of anomalous traffic that is not caused by any malicious behavior [3], and data sets recorded in a network isolated from the Internet might not include these types of anomalies.

To solve some of these problems, we collected three additional data sets. One data set was generated by deploying two honeypot systems, namely, a host running a standard RedHat 7.2 Linux installation and a Microsoft Windows 2000 Server. These systems were exposed to the Internet and, in addition, they were the target of a security evaluation performed by the students of a graduate security class. The Snort intrusion detection system was used to collect alerts, and we developed a tool to perform real-time alert verification. This tool is the verification component used in the correlation process; it is described in detail in Section 4.6.

The second data set was generated during a Cyber Treasure Hunt competition, organized as part of the same graduate class in network security [49]. The Treasure Hunt goal was to break into a simulated payroll system and perform unauthorized money transfer transactions. In this case, the class was divided into two teams. Each team had to perform a number of tasks in sequence (e.g., scan a network or break into a host). Each task had to be completed in a limited amount of time (e.g., 30 minutes) in order to receive the maximum score. If the time elapsed and the team was not able to complete the task, then some helpful hints were provided so that the task could be completed, but no points were given. A new task was disclosed only after the previous one was completed by both teams. The tasks and the execution of this exercise were devised to support the creation of event streams that contain complex and meaningful multistep attacks (i.e., a sequence of attacks that are part of an overall plan to achieve a specific goal).

The last data set was collected by an IDS monitoring the uplink of one of Rome Air Force Research Laboratory's networks. The monitored network was a production network with no artificially generated background traffic or canned attacks. A disadvantage of this data set is that, as far as we know, it contains no successful attacks.

The MIT Lincoln Laboratory data sets can be downloaded from [26]. The Treasure Hunt, Honeypot, and Defcon 9 data sets are available online at [41]. The other data sets are, as far as we know, not available online.

In order to effectively use both the public and the custom data sets that were collected, we first generated alerts from the raw event streams (with the exception of the DARPA Correlation Technology Validation and the Rome AFRL data, which already provided the alerts produced by the sensors). To this end, we ran IDS tools that were readily available (Snort [42], USTAT [18], and LinSTAT [40]), using standard signature sets. These tools all output alerts in the Intrusion Detection Message Exchange Format (IDMEF) [6], which is a standardized way of encoding intrusion alerts. An example IDMEF alert is shown in Fig. 2. This alert represent an ICMP ping from 4.22.161.203 to 26.100.101.1, which was reported by Snort. Table 1 summarizes the characteristics of the data sets in terms of tools used to generate the alerts, the time span, and the number of generated alerts.

```
<IDMEF-Message version="0.3">
  <Alert ident="48562" impact="unknown">
    <Analyzer analyzerid="Snort:1.8.6:26.100.101.3">
      <Node><name>brp-snort</name></Node>
    </Analyzer>
    <CreateTime ntpstamp="0xc12b141a.0xa5baa000"/>
    <Source><Node>
        <Address category="ipv4-addr">
          <address>4.22.161.203</address></Address>
    </Node></Source>
    <Target><Node>
        <Address category="ipv4-addr">
          <address>26.100.101.1</address></Address>
    </Node></Target>
    <Classification origin="vendor-specific">
    <name>ICMP PING NMAP</name></Classification>
  </Alert>
</IDMEF-Message>
```

Fig. 2. Example IDMEF alert.

Note that generating alerts from raw data is dangerous, because it might bias the evaluation of the correlation process. On the other hand, correlation is a new field and there are very few shared data sets, like the Cyberpanel evaluation effort, that can be used to evaluate different correlation tools. In addition, none of the data sets has an associated truth file that can be used in a systematic way to determine if the correlated alerts represent a meaningful grouping of attack detections. Therefore, the assessment of the correctness of the correlation process has to be performed manually.

One contribution of this paper is the lessons learned from analyzing different data sets with very different properties. Even though the results do not necessarily represent a final evaluation of the correlation process in general, they do provide useful findings that can be used by other researchers in the field.

4 Correlation Components and Analysis Results

Alert correlation is a multicomponent process that receives as input a stream of alerts from multiple intrusion detection systems. In each component of the process, alerts are merged into high-level intrusion reports or tagged as nonrelevant if they do not represent successful attacks. Finally, the alerts are prioritized according to the site's security policy, and eventually the results are reported.

4.1 Meta-Alerts

When two or more related alerts are merged as part of the alert correlation process, the result is called a *meta-alert*. The process of "merging" alerts refers to the action of subsuming a set of related alerts as a single meta-alert at a higher level of abstraction. Thus, a meta-alert is an intrusion report at a higher level of abstraction that comprises the information contained in all alerts that

TABLE 1
Data Set Summary Information

Data Set	Sensors	Duration	Alerts
MIT/LL 1999	USTAT, Snort	2 weeks	41,760
MIT/LL 2000	USTAT, Snort	3 hours	36,635
CTV	Snort, EBayes-TCP [46], U-STAT, WinSTAT, Tripwire [22]	2 days	215,190
Defcon 9	Snort	2 days	6,378,096
Rome AFRL	Snort and undisclosed NIDS	4 months	5,299,390
Honeypot	Snort	10 days	260,120
Treasure Hunt	Snort, LinSTAT	4 hours	2,811,169

Snort, USTAT, and LinSTAT were run with their default rule sets.

were used to create the meta-alert. For example, a series of sensor alerts in which each alert refers to an individual network probe packet can be merged into a single portscan meta-alert by the correlation process.

A meta-alert is similar to an alert but its contents (e.g., the victims of an attack) are derived as a function of the attributes of the merged alerts. Each meta-alert also contains references to all of the alerts that were merged to produce the meta-alert. Continuing the example above, the portscan meta-alert contains all the hosts that were scanned by at least one probe packet as attack targets. In addition, the portscan alert holds references to each probe packet alert. The decision of whether or not to merge alerts is dependent on the particular component of the correlation process and on the values of relevant attributes of the alerts. The purpose of meta-alerts is to aggregate information of related attacks and to present a single alert instance that summarizes all the relevant information to a human analyst.

A meta-alert can be further merged with other alerts (either intrusion detection sensor alerts or meta-alerts), resulting in a hierarchical structure, which can be represented as a tree. The most recent meta-alert is the root node in this tree, and all successor nodes can be reached by following the references to the merged alerts. All interior nodes in the tree are meta-alerts, while all leaves are sensor alerts.

Whenever the correlation system considers a meta-alert and a sensor alert as candidates for merging, it first attempts to merge the root of the meta-alert with the sensor alert. If the root alert cannot be merged, all its successor alerts are *independently* considered for merging. The process of finding appropriate alert candidates for merging is repeated recursively until an alert that can be merged with the sensor alert is found, or until all nodes have been considered. The idea behind this approach is that a meta-alert represents the complete attack information of all its successor nodes. Therefore, alerts are considered in a breadth-first-search fashion and merging is performed at the highest level possible.

Consider the following example of merging a meta-alert M, which has two child nodes M_1 and M_2, and a sensor alert S, as shown in Fig. 3. The merging process first checks whether M and S can be merged, but fails. Next, M_1 is considered for merging with S. This operation is successful, and a new meta-alert M' with two children (M_1 and S) is created. The merging process continues to consider M_2 and S, because child nodes are considered independently. Note, however, that no children of M_1 will be considered because a successful merge with a node precludes all of this node's descendants from being candidates.

4.2 Example Attack Scenario

To explain the different components of the correlation process, the following example attack scenario is used throughout the paper. In this example, a victim network is running a vulnerable Apache Web service on a Linux host (IP: 10.0.0.1). This host runs a host-based IDS (H) and an application-based IDS (A) that monitors the Apache Web logs for content that indicates malicious activity. In addition, the network traffic is analyzed by two different network-based IDSs (N1 and N2). Table 2 shows the seven alerts (Alert 1 through Alert 7) that are generated during an attack against the victim host. The *AlertID* and *Name* columns in the table identify an alert instance, the *Start/End* column shows the start-time and end-time of the corresponding attack, and the *Source* and *Target* columns list information about the attack's sources and targets. The *Tag* column is included to show information that the correlation system associates with each alert (e.g., references to successor nodes for meta-alerts or indications of nonrelevant alerts).

The attacker (IP: 31.3.3.7) first launches a port scan against the Linux host and discovers the vulnerable Apache server (Alert 2, Alert 3). While the scan is in progress, a worm (IP: 80.0.0.1) attempts to exploit a Microsoft IIS vulnerability and fails (Alert 1). After the scan finishes, the

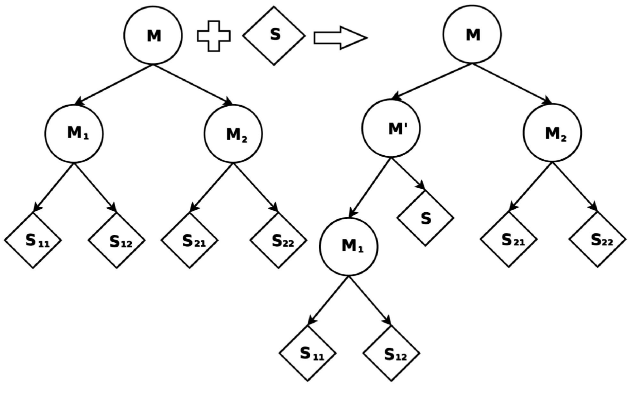

Fig. 3. Alert merging process.

TABLE 2
Example Attack Scenario Alerts

AlertID	Name	Sensor	Start/End	Source	Target	Tag
1	IIS Exploit	N1	12.0 / 12.0	80.0.0.1	10.0.0.1, port:80	
2	Scanning	N2	10.1 / 14.8	31.3.3.7	10.0.0.1	
3	Portscan	N1	10.0 / 15.0	31.3.3.7	10.0.0.1	
4	Apache Exploit	N1	22.0 / 22.0	31.3.3.7	10.0.0.1, port:80	
5	Bad Request	A	22.1 / 22.1		localhost, Apache	
6	Local Exploit	H	24.6 / 24.6		linuxconf	
7	Local Exploit	H	24.7 / 24.7		linuxconf	

attacker launches a successful Apache buffer overflow exploit (Alert 4, Alert 5) and obtains an interactive shell, running as the apache user. Using a local exploit against linuxconf (a system administration tool), the attacker elevates her privileges and becomes root. Note that the same exploit is executed twice with different parameters (Alert 6, Alert 7) before it succeeds.

Given this attack scenario, the correlation system should present the administrator with a single meta-alert that characterizes a multistep attack against the victim host. The multistep attack consists of three steps: the initial scanning, the remote attack against the Web server, and the privilege escalation. The two alerts (Alert 2, Alert 3) describing the scanning incident should be merged and marked as referring to a single incident. The two alerts referring to the Web server attack (Alert 4, Alert 5) should be merged as being part of the same remote exploit. The two alerts associated with the privilege escalation (Alert 6, Alert 7) should be merged because they represent repeated instances of the same local attack. The alert raised because of the worm activity (Alert 1) should be excluded from the scenario and marked as nonrelevant.

The following sections describe the components of the correlation process in detail and show their effect on the example attack scenario. For each of the components, a data analysis section is presented that discusses the effectiveness of the component with respect to each of the data sets considered. Note that even though the process is represented as a "pipeline" for the sake of presentation, some components of the process may be applied multiple times, may be applied in parallel, or may be applied in a different order.

4.3 Alert Normalization

The correlation process may receive alerts from different sensors, and these alerts can be encoded in different formats. The goal of the alert normalization component is to translate all attributes of each sensor alert into a common format. This translation requires that the syntax and semantics of a sensor alert are recognized.

The Internet Engineering Task Force proposed the Intrusion Detection Message Exchange Format (IDMEF) [6] data model as a way to establish a standard representation for intrusion alerts. Although this representation defines the semantics of a few attributes, the IDMEF effort is mostly concerned with syntactic rules. Therefore, it is common for IDS implementors to choose different names for the same attack,[3] provide incomplete information for certain fields, or choose to include additional fields to store relevant data. As a result, similar information can be labeled differently or included in different fields of the alerts.

In our framework, we followed the common practice of implementing wrapper modules to interface with the different intrusion detection sensors. Each module relies on a knowledge base to convert sensor-specific information into attributes and values usable by our framework. The alert names are, whenever possible, taken from the CVE (Common Vulnerabilities and Exposures) [7] list, which is a well-known effort to standardize the names of all publicly known vulnerabilities.

The pseudocode for the normalization process is as follows:

```
global normalization_db, alertname_db

normalize(raw_alert) {
   alert ← new alert
   alert.alertid ← get_unique_id()
   alert.name ←
     get alertname from alertname_db using
     (raw_alert.name, raw_alert.sensortype)
        as key
  mappings ←
   get all m:mapping from normalization_db
     where m.sensor = raw_alert.sensor

  for each m:mapping in mappings:
   alert_attr ← m.alert_attr
   raw_attr ← m.raw_attr
   alert.alert_attr ← raw_alert.raw_attr

  pass alert to next correlation component
}
```

[3] There is no mandatory naming convention and, at the time of writing, a few different competing schemes such as CVE [7] and Bugtraq-IDs [28] exist.

Alert Attribute	Description
alertid	A unique ID identifying the alert
analyzertime	The time when the IDS sent the alert
attackernodes	The set of nodes where the attack originated
attackgraph	A graph showing the progress of complex attacks
consequence	A set of systems that are affected by this attack
createtime	The time when the IDS generated the alert
detecttime	The time when the IDS detected the attack
end_time	The time when the attack ended
name	The name of the attack
priority	A value indicating how important the attack is
receivedtime	The time the alert was received by the correlator
reference	A set of references to other alerts
sensornode	The node at which the IDS that generated the alert runs
start_time	The time when the attack started
type	The attack type (Reconnaissance, Breakin, Escalation, DoS)
verified	If the attack was successful (true, false, unknown)
victimnodes	The set of nodes that were victims of the attack
victimprocess	The full path of the process that was attacked
victimservice	Port number and protocol of the service that was attacked

Fig. 4. Alert attributes.

Each raw alert is translated into a standardized alert format, assigned a standardized name, and its attributes are copied to the appropriate fields of the alert as defined by the attribute mappings in the normalization database. The attributes contained in the standardized alert are shown in Fig. 4.

Given the alerts of the example scenario, the alert normalization component assigns identical names to the two alerts describing the portscan (Alert 2, Alert 3) as shown in Table 3. Note that in this table, as well as in the following ones, the elements in boldface are those affected by the correlation process.

4.3.1 Data Analysis

If only one type of sensor were used, then that sensor's format could be used as the standard format. In that case, there would be no need to have a normalization component. All seven data sets were processed by the normalization component, since the other components in the correlation process depend on the format of the alerts being standardized. The normalization component passes all alerts through and, therefore, it does not provide any reduction in the number of the alerts received as input.

4.4 Alert Preprocessing

Normalized alerts have a standardized name and are in a format that is understood by the other components of the correlation process. However, additional preprocessing may be required since some sensors omit fields that are necessary for other components of the correlation process (i.e., start-time, end-time, source, and target). The goal of the preprocessing component is to supply, as accurately as possible, missing alert attributes that are important for later correlation components.

The IDMEF standard defines three different classes to represent time. These are the *CreateTime* (the time when the alert is created by the IDS), the *DetectTime* (the time when the events producing an alert are detected by the IDS), and the *AnalyzerTime* (the time at the IDS when the alert is sent to the correlation system). If any of these time values are present in the alert, the preprocessing component can use this information as the value for the attack's start-time. More precisely, *DetectTime* is the preferred time value, followed by *CreateTime*, and then *AnalyzerTime*. When no time information is available, the start-time is approximated as the point in time when the correlation system receives the alert from the sensor. When no end-time (or appropriate duration information) is present, the end-time is assumed to be identical to the start-time. The heuristics used could potentially introduce significant errors in the timestamp attributes, which in turn could lead to errors later on in the correlation process. No errors were observed during our evaluation. This is mainly due to the fact that all the clocks of the sensors used in the experiments were synchronized using the Network Time Protocol (NTP) [30], and all alerts processed included at least one of the three different timestamps.

According to the IDMEF standard, the attack source is composed of information about the *node*, the *user*, the *process*, and the *network service* that originated the attack. The attack target is composed of the same information, and it also includes a list of affected *files*. Not all fields have to be defined for both the source and the target, but to operate properly, the correlation process requires at least one nonempty field for each. For host-based alerts, the *node* fields of the attack source and target are set to the address of the host where the sensor is located. For network-based alerts, the *node* information is taken from the source and destination IP addresses.

Finally, an alert can be augmented with additional information on the basis of the standardized alert name that is assigned by the normalization component. Our implementation assigns an attack type to each alert. This attribute describes the type of an attack with respect to a simple scheme that distinguishes between local or remote information gathering and privilege escalation attempts. The type information is useful because it allows one to group attacks with a similar impact together.

The pseudocode for the preprocessing component is as follows:

TABLE 3
Effect of Alert Normalization on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
2	**Portscan**	N2	10.1 / 14.8	31.3.3.7	10.0.0.1	
3	Portscan	N1	10.0 / 15.0	31.3.3.7	10.0.0.1	

Boldface indicates information affected by the correlation process.

TABLE 4
Effect of Alert Preprocessing on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
5	Bad Request	A	22.1 / 22.1	**10.0.0.1**	**10.0.0.1**, Apache	
6	Local Exploit	H	24.6 / 24.6	**10.0.0.1**	**10.0.0.1**, linuxconf	
7	Local Exploit	H	24.7 / 24.7	**10.0.0.1**	**10.0.0.1**, linuxconf	

```
global attack_type_db

preprocess(alert) {
 if alert.start_time is null:
  if alert.detecttime is not null:
   alert.start_time ← alert.detecttime
  else if alert.createtime is not null:
   alert.start_time ← alert.createtime
  else if alert.analyzertime is not null:
   alert.start_time ← alert.analyzertime
  else:
   alert.start_time ← alert.receivedtime

 if alert.end_time is null:
  alert.end_time ← alert.start_time

 if alert is produced by a host-based IDS:
  alert.victimnodes ← alert.sensornode
  alert.attackernodes ← alert.sensornode

 alert.type ←
  get alertype from attack_type_db using
   alert.name as key
 pass alert to next component
}
```

Table 4 shows the alerts of the example attack scenario that are augmented with information provided by the preprocessing component. In this case, source and target information for host and application-level alerts is added.

4.4.1 Data Analysis

Preprocessing is an important step of the correlation process, but it does not provide any reduction in the number of alerts or any abstraction improvement. However, the information added by this component is fundamental for the correct operation of later components. For example, when processing the MIT/LL 2000 data set, the multistep correlation component (see Section 4.10) would not produce any correlated attacks if the alerts were not correctly labeled by the preprocessing component. However, when preprocessing was applied, the system was able to correctly identify six multistep attacks.

4.5 Alert Fusion

The goal of the alert fusion component is to combine alerts that represent the independent detection of the same attack occurrence by different intrusion detection systems. The decision to fuse two alerts is based on the temporal difference between these alerts and the information that they contain. The alert fusion component keeps a sliding time window of alerts. The alerts within the time window are stored in a time-ordered queue. When a new alert arrives, it is compared to the alerts in the queue, starting with the alert with the earliest timestamp. A match is found if all overlapping attributes are equal and the new alert is produced by a sensor different from the sensors already associated with the alert being matched.[4] The latter requirement is reasonable since a sensor is not expected to emit two similar alerts for one single attack occurrence. Upon finding a match, the two alerts are merged, the resulting meta-alert replaces the matched alert in the queue, and the search is terminated. If no match is found after searching through the whole queue, the alert is inserted into the queue, to be considered for matching with future alerts.

In our implementation the merging of alerts is performed by creating a meta-alert. The timestamp of the meta-alert is assigned the earlier of both start-times and end-times. This is done because both alerts are assumed to be related to the same attack, and a later time stamp is likely to be the result of a delay at the sensors. The attribute fields of the fused meta-alert are set to the union of the values of the respective alert attributes. Note that it is possible that more than two sensor alerts may be fused into a single meta-alert. In this case, two alerts are first fused into a meta-alert that is subsequently fused (one at a time) with additional sensor alerts.

When the time difference between the oldest and the newest alert in the queue exceeds the specified window size, the oldest alerts are removed from the queue and passed to the next correlation component. A value for the window size that is too low causes related alerts to escape fusion, while a value that is too high causes unrelated alerts to be fused. For our experiments we used 2 seconds as the window size, which was determined heuristically by choosing a size and analyzing the results of the fusion process.

At a first glance, the constraints that have to be satisfied for fusion to occur seem very restrictive. However, the purpose of the fusion component is to combine duplicate alerts, not to correlate closely related attacks (this is implemented in other components of the analysis).

The pseudocode for the fusion component is as follows:

[4] The alert being matched in the queue could be either another sensor alert or a meta-alert that is the result of an earlier match.

TABLE 5
Effect of Alert Fusion on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
2	Portscan	N2	10.1 / 14.8	31.3.3.7	10.0.0.1	**correlated**
3	Portscan	N1	10.0 / 15.0	31.3.3.7	10.0.0.1	**correlated**
8	**Meta-Alert**	**{N1, N2}**	**10.0 / 14.8**	**31.3.3.7**	**10.0.0.1**	{**2, 3**}

```
parameter window_size
global alert_queue

fuse (alert) {
  remove all a:alert from alert_queue where
    a.start_time <
      (alert.start_time - window_size)
  pass removed alerts to next
    correlation component

  al ← get a:alert with lowest start_time from
    alert_queue where
    alert.analyzer ∩ a.analyzer is empty and
    all overlapping attributes except
    start_time, end_time,
    analyzer, alertid are equal

  if al is not null:
    replace al in alert_queue with
      fuse_merge(alert, al)
  else:
    add alert to alert_queue
}

fuse_merge(alert1, alert2) {
  r ← new alert
  r.alertid ← get_unique_id()
  r.start_time ← min(alert1.start_time,
    alert2.start_time)
  r.end_time ← min(alert1.end_time,
    alert2.end_time)

  for each attr:attribute except start_time,
    end_time, reference, alertid:
      r.attr ← alert1.attr ∪ alert2.attr

  r.reference ← alert1.alertid ∪
    alert2.alertid
  return r
}
```

In the example scenario, the two alerts that refer to the portscan attack (Alert 2, Alert 3) can be fused, because their start-times and end-times are within the 2 second time window and all remaining fields are identical. Note that the two alerts referring to the local exploit (Alert 6, Alert 7) cannot be fused because both are produced by the same sensor. The result of the alert fusion component is shown in Table 5. The generated meta-alert is tagged with references to the two alerts that were fused, while both portscan alerts are tagged as being correlated.

4.5.1 Data Analysis

The alert fusion component is particularly relevant if the existing intrusion detection infrastructure contains a substantial level of redundancy. For example, when intrusion detection systems operate on both sides of a firewall, or when they are run both at the network gateway and on individual hosts, redundant alerts are likely to be generated.

The results obtained for the data sets analyzed are shown in Table 6. The Defcon 9 data set shows a 28 percent reduction rate, since all subnetworks had an IDS installed. Because of this, all traffic between two teams was seen by two sensors. The Honeypot and Rome AFRL data sets were both produced by a single sensor; therefore, no fusion was possible.

For the remaining data sets, only a small fraction of the alerts were fused. In all these cases, most of the attack-related events were seen by only one sensor. For instance, in the Treasure Hunt data set, a majority of the alerts was caused by attacks against hosts in the demilitarized zone (DMZ) of the target network. These attacks were only seen by the IDS in the DMZ, not by the IDSs installed on the inside network.

4.6 Alert Verification

An important task of alert correlation is the aggregation of alerts to provide a high-level view (i.e., the "big picture") of malicious activity on the network. Unfortunately, when the correlation process receives false positives as input, the quality of the results can degrade significantly. Correlating alerts that refer to failed attacks can easily result in the

TABLE 6
Impact of Alert Fusion

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	41,760	36,635	215,190	6,378,096	5,299,390	260,120	2,811,169
Output Alerts	39,094	36,631	215,113	4,565,029	5,299,390	260,120	2,808,595
Reduction	6.38%	0.01%	0.04%	28.43%	0.00%	0.00%	0.09%

detection of whole attack scenarios that are nonexistent. The goal of the alert verification component is to remove (or tag) alerts that do not represent successful attacks.

When a sensor outputs an alert, there are three possibilities:

1. The sensor has correctly identified a successful attack. This alert is most likely relevant (i.e., a true positive).
2. The sensor has correctly identified an attack, but the attack failed to meet its objectives. This kind of alert is called a nonrelevant positive or noncontextual (reflecting the missing contextual information that the IDS would require to determine a failed attack). Because some sites might be interested in failed attack attempts, the alert should be differentiated from a successful instance.
3. The sensor incorrectly identified an event as an attack. This alert represents incorrect information (i.e., a false positive).

The idea of alert verification is to discriminate between successful and failed intrusion attempts (both false and nonrelevant positives). This is important for the correlation process because, although a failed attack indicates malicious intent, it does not provide increased privileges or any additional information (other than that an attacker learned that the particular attack is ineffective). Identifying failed intrusion attempts allows other correlation components to reduce the influence of these alerts on their decision process.

The verification of the result of an attack can be accomplished by extending intrusion detection signatures with an expected "outcome" of the attack. The "outcome" describes the visible and verifiable traces that a certain attack leaves at a host or on the network (e.g., a temporary file or an outgoing network connection). Verification can be performed using both passive and active techniques, and each approach requires different support from the intrusion detection infrastructure.

Passive techniques depend on a priori information gathered about the hosts, the network topology, and the installed services. By relying on a network model, it is possible to verify whether the target of the attack exists and whether a (potentially vulnerable) service is running. For remote attacks, it is also possible to check whether malicious packets can possibly reach the target, given the network topology and the firewall rule configuration, or whether the target host reassembles the packets as expected by the intruder (e.g., using the tool presented in [43]). Passive monitoring also allows one to check for outbound network connections that are initiated from the victim machine, possibly connecting back to the attacking host.

The advantage of passive techniques is that they do not interfere with the normal operation of the network. In addition, it is not necessary to perform additional tests that delay the notification of administrators or the start of active countermeasures. A disadvantage of passive mechanisms is the potential difference between the state stored in the knowledge base and the actual security status of the network. New services might have been installed or the firewall rules might have been changed without updating the knowledge base. This can lead to attacks being tagged as nonrelevant, even though the target is vulnerable. Another disadvantage is the limitation of the type of information that can be gathered in advance. For example, when the signature of an attack is matched against a packet sent to a vulnerable target, the attack could fail for a number of other reasons (e.g., an incorrect offset for a buffer overflow exploit).

Active verification techniques need to look for evidence of the success of an attack by checking information at the victim machine. These techniques usually require that a network connection be established to the victim. The connection enables one to scan the attack target and to assess whether a target service is still responding or has become unresponsive. It also enables the alert verification subsystem to check whether unknown ports accept connections, which could be evidence that a backdoor has been installed.

One possible approach to alert verification is to leverage vulnerability scanners. When an attack has been detected, a scanner can be used to check for the vulnerability that this attack attempts to exploit.

Active alert verification has the advantage, compared to passive mechanisms, that the information is current. Unfortunately, active actions are visible on the network and scanning could possibly have an adverse effect on one's own machines. Port scanning also consumes network bandwidth and resources at the scanned host, and tests run by a vulnerability scanner could crash a service. One also has to make sure that the alerts generated in response to the activity of the vulnerability scanner are excluded from the correlation process. The scope of these scans is also limited; the identification of some evidence associated with an attack might require local access to the victim machine.

To address the limitations of vulnerability scanning, it is possible to rely on techniques that gather evidence about the result of an attack using authenticated access to the victim host. The difference with respect to the previous group of techniques is that the alert verification subsystem presents authentication credentials to the target host. The alert verification system then can remotely log in and execute scripts or system commands. This allows one to monitor the integrity of system files (e.g., the password file or system specific binaries) or to check for well-known files that are created by attacks (e.g., worms usually leave an executable copy of themselves on the file system). In addition, programs to collect interesting forensic data such as open network connections (e.g., `netstat`), open files (e.g., `lsof`), or running processes (e.g., `ps`) can be invoked.

The advantage of mechanisms in this group is the access to high-quality data gathered directly from a target machine. The downside is the need to configure the target machine for authenticated remote access. This could be cumbersome in large network installations or when hosts with many different operating systems are used.

We use an active scanning approach in the verification component. More specifically, we have implemented an

TABLE 7
Effect of Alert Verification on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
1	IIS Exploit	N1	12.0 / 12.0	80.0.0.1	10.0.0.1, port:80	**nonrelevant**

extension for Snort [42], [44] that uses NASL [2] scripts written for the Nessus vulnerability scanner [33]. When a Snort rule is triggered, the suspect packet and associated event data is queued for verification. The appropriate NASL script is determined using a match on the CVE identifiers [7], and then the script is run against the target machine. If Nessus detects that the target is vulnerable to the particular attack, the Snort alert is tagged as verified. Otherwise, it is assumed to be unsuccessful. An example of when this component can tag an attack as unsuccessful is when a Windows DCOM RPC buffer overrun attack is detected and the NASL script that checks for this particular vulnerability ("Microsoft RPC Interface Buffer Overrun KB824146") reports that the host that was attacked does not run the Windows RPC service.

The pseudocode for the verification component is as follows:

```
global nasl_scripts

verify(alert) {
 scripts ← get all s:script from nasl_scripts
  where s.name = alert.name

 if scripts is null:
  alert.verified ← unknown
 else:
  for each s:script in scripts:
   run s on alert.victimhosts
   if script reported successful exploit:
    alert.verified ← true
    break
   else:
    alert.verified ← false
 pass alert to next correlation component
}
```

In our example scenario, the alert corresponding to the worm attack (Alert 1) is identified as being unsuccessful because it is an exploit for Microsoft IIS that targets a Linux service. Thus, the alert is tagged as nonrelevant and excluded from further correlation, as shown in Table 7.

4.6.1 *Data Analysis*

The alert verification process requires that the protected assets be available for real-time verification of the actual exposure of the system and/or that a detailed model of the installed network services be available. Unfortunately, this information is not available for most of the data sets that we analyzed. Therefore, the only meaningful results reported here are those collected for our honeypot system.

During a period of 10 days, Snort reported 185,185 alerts referring to attacks against the RedHat Linux machine and 74,935 alerts referring to attacks against the Windows machine. Given these alerts, our alert verification component, which utilized the active scanning approach, was able to tag 180,229 attacks against the Linux host (97.3 percent) and 72,333 attacks against the Windows host (96.5 percent) as unsuccessful. Out of the 642 different types of attacks we saw, we had NASL verification scripts for 553 of the corresponding vulnerabilities. These results are summarized in Table 8.

4.7 Attack Thread Reconstruction

An attack thread combines a series of alerts that refer to attacks launched by one attacker against a single target. The goal of this component is to correlate alerts that are caused by an attacker who tests different exploits against a certain program or that runs the same exploit multiple times to guess correct values for certain parameters (e.g., the offsets and memory addresses for a buffer overflow). Attack threads are constructed by merging alerts with equivalent source and target attributes that occur in a certain temporal proximity. Similar to the fusion component, this component keeps a sliding time window of alerts. The matching criteria, however, is more relaxed. Two alerts are considered a match if the source and target attributes are equivalent. Different from fusion, the sensor alerts need not be produced by different sensors. Merging is done by creating a meta-alert containing only the attributes that are equal in both sensor alerts. The timestamp of the meta-alert is assigned the earlier of the two start-times and the later of the two end-times.

The pseudocode for the thread reconstruction component is as follows:

TABLE 8
Impact of Alert Verification

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	39,094	36,631	215,113	4,565,029	5,299,390	260,120	2,808,595
Output Alerts	39,094	36,631	215,113	4,565,029	5,299,390	7,558	2,808,595
Reduction	0.00%	0.00%	0.00%	0.00%	0.00%	97.09%	0.00%

TABLE 9
Effect of Attack Thread Reconstruction on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
4	Apache Exploit	N1	22.0 / 22.0	31.3.3.7	10.0.0.1, port:80	**correlated**
6	Local Exploit	H	24.6 / 24.6	10.0.0.1	10.0.0.1, linuxconf	**correlated**
7	Local Exploit	H	24.7 / 24.7	10.0.0.1	10.0.0.1, linuxconf	**correlated**
8	Meta-Alert	{N1, N2}	10.0 / 14.8	31.3.3.7	10.0.0.1	{2, 3},**correlated**
9	**Meta-Alert**	**{N1, N2}**	**10.0 / 22.0**	**31.3.3.7**	**10.0.0.1, port:80**	**{4, 8}**
10	**Meta-Alert**	**H**	**24.6 / 24.7**	**10.0.0.1**	**10.0.0.1, linuxconf**	**{6, 7}**

```
parameter window_size
global alert_queue

attack_thread(alert) {
 remove all a:alert from alert_queue where
  a.start_time <
   (alert.start_time - window_size)
 pass removed alerts to next correlation
  component

 al ← get a:alert with lowest start_time from
  alert_queue where
  alert.victimhosts = a.victimhosts and
  alert.attackerhosts = a.attackerhosts

 if al is not null:
  replace al in alert_queue with
   thread_merge(alert, al)
 else:
  add alert to alert_queue
}

thread_merge(alert1, alert2) {
 r ← new alert
 r.alertid ← get_unique_id()
 r.start_time ← min(alert1.start_time,
  alert2.start_time)
 r.end_time ← max(alert1.end_time,
  alert2.end_time)
 r.analyzer = alert1.analyzer ∪
  alert2.analyzer
 r.reference ← alert1.alertid ∪
  alert2.alertid

 if alert1.name = alert2.name:
  r.name ← alert1.name
 else:
  r.name ← "Attack Thread"

 for each attr:attribute except start_time,
  end_time, reference, analyzer, alertid:
   if alert1.attr = alert2.attr:
    r.attr ← alert1.attr
   else:
    r.attr ← null
 return r
}
```

The value of the time window in our experiments was set to 120 seconds. This value was found to be a good trade-off between a small value, which would cause several attack threads to go undetected, and a larger value, which would slow down the system by requiring the component to keep a large number of alerts in the queue.

In the example scenario, the meta-alert that was created by fusing the two portscan alerts (Alert 8), and the remote Apache exploit (Alert 4) are merged into one attack thread because both have the same source and target attributes and are close in time. Note that Alert 4 is not merged with the individual portscan alerts (Alert 2 or Alert 3). This is because Alert 8, which is the meta-alert summarizing Alert 2 and Alert 3, has already been used in the merging process (as explained in Section 4). The two alerts referring to the local exploit (Alert 6 and Alert 7) are also combined into one attack thread. Table 9 shows the alerts that are created or modified by the attack thread reconstruction component.

4.7.1 Data Analysis

Attack thread reconstruction can contribute significantly to alert reduction when a very large number of alerts has been generated as the result of a single attack. This is the case for the Treasure Hunt data set in which several password brute-force attacks and a number of buffer overflow attempts created a large number of alerts. This particular data set is reduced by 99 percent using thread reconstruction. The MIT/LL 2000 data set, on the other hand, did not achieve a comparable reduction rate because the attacks were more focused and did not generate a large number of alerts. The results for all of the data sets are shown in Table 10.

4.8 Attack Session Reconstruction

The goal of the attack session reconstruction component is to link network-based alerts to related host-based alerts. Identifying relations between these two types of alerts is difficult because the information that is present in the alerts differs significantly. Network-based sensors usually provide the source and destination IP addresses and ports of packet(s) that contain evidence of attacks. Host-based sensors, on the other hand, include information about the process that is attacked and the user on whose behalf the process is executed. Because it is not straightforward to relate network-based information (IP addresses and ports) to host-based information (process and user identifiers), it is not obvious how a connection between alerts from intrusion detection sensors operating on different event streams can be established.

TABLE 10
Impact of Attack Thread Reconstruction

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	39,094	36,631	215,113	4,565,029	5,299,390	7,558	2,808,595
Output Alerts	8,966	34,211	147,352	1,814,656	1,599,476	2,126	2,286
Reduction	77.07%	6.61%	31.50%	60.25%	69.82%	71.87%	99.91%

One approach is to require a rough spatial and temporal correspondence between the alerts. The idea is that a host-based alert should be linked to a network-based alert when the host-based attack occurs a short time after the network-based attack and the network-based attack targets the host where the host-based attack is observed. This approach is simple, but has the obvious drawback that it is very imprecise and might correlate independent alerts. It can, however, be improved by utilizing a priori information about the port(s) used by certain network services. For example, by assuming that a Web server process listens on port 80, host-based alerts referring to this process are correlated only to network-based alerts with destination port 80 on the host where the Web server is running. Another possibility is to specify that a certain attack is known to *prepare for* or *to be a precondition for* another attack. This allows network traffic containing a known Web-based attack to be linked to an alert raised by a sensor monitoring the victim Web server. To the best of our knowledge, all current alert correlation systems require such information, which is manually encoded in a knowledge base (called pre and postconditions in [5] and prerequisites and consequences in [36]).

We argue that it is useful to extend these manually defined links between network-based and host-based alerts with a more general mechanism. This mechanism is based on information about the network port(s) that processes are bound to. Given this information, it is possible to determine if it is likely that the data triggering a network-based alert reaches the victim process reported in the host-based alert. The attack session reconstruction component automatically links a network-based alert to a host-based alert when the victim process listens to the destination port mentioned in the network-based alert.

For our implementation, we created a database of mappings between the network ports that accept connections and the name and user id of the process listening to that port. We were able to obtain this information only for the Treasure Hunt data set, because this was the only experiment in which real-time access to the monitored systems was possible and both network-based and host-based alerts were generated.

Instead of using a simple sliding time window, as was the case for several of the previous components, the session reconstruction algorithm uses a sliding time window with extended timeout model. That is, the window is initially set to the timeout value, but if an alert arrives that is determined to be part of the same session, then the window is extended by timeout units from the time of the latest correlated alert. If no other alert that is part of the same session arrives within timeout seconds from the last alert in the session, then the session is determined to be over, and the correlated alert information is passed on to the next component. The timeout value in our experiments was set to 125 seconds.

The pseudocode for the attack session reconstruction component is as follows:

```
parameter timeout
global session_list, service_db

attack_session(alert) {
  remove all e:element from session_list where
    e.timestamp < (alert.start_time - timeout)

  for each removed element e:
    if e.state = "Host":
      pass session_merge(e.alerts)
        to next correlation component
    else:
      pass e.alerts
        to next correlation component

  if alert is produced by a network-based IDS:
    matching_session ← get s:session with
        lowest timestamp
      from session_list where
        alert.victimhosts = s.victimhosts and
        alert.victimservice = s.victimservice and
        s.state = "Network"

    if matching_session is null:
      matching_session ← new session
      matching_session.victimhosts ←
        alert.victimhosts
      matching_session.victimservice ←
        alert.victimservice
      matching_session.state ← "Network"

    matching_session.timestamp ←
      alert.start_time
    add alert to matching_session.alerts
  else if alert is produced by a host-based IDS:
    alert.victimservice ← get s:service from
        service_db where
      s.host = alert.victimhosts and
      s.process = alert.victimprocess

  matching_session ← get s:session with lowest
      timestamp
    from session_list where
```

TABLE 11
Effect of Attack Session Reconstruction on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
5	Bad Request	A	22.1 / 22.1	10.0.0.1	10.0.0.1, Apache	**correlated**
9	Meta-Alert	{N1, N2}	10.0 / 22.0	31.3.3.7	10.0.0.1, port:80	{4, 8}, **correlated**
11	**Meta-Alert**	**{N1, N2, A}**	**10.0 / 22.1**	**{31.3.3.7, 10.0.0.1}**	**10.0.0.1, port:80, Apache**	**{5, 9}**

```
    alert.victimhosts = s.victimhosts and
    alert.victimservice = s.victimservice

  if matching_session is null:
    pass alert to next correlation component
  else:
    matching_session.timestamp ←
        alert.start_time
    add alert to matching_session.alerts
    matching_session.state ← "Host"
}

session_merge(alert_list) {
  r ← new alert
  r.alertid ← get_unique_id()
  r.start_time ← min start_time of all alerts in
      alert_list
  r.end_time ← max end_time of all alerts in
      alert_list
  r.analyzer ← union of all analyzer in
      alert_list
  r.reference ← union of all alertid in
      alert_list

  if name is equal for all alerts in alert_list:
    r.name ← alert_list[1].name
  else:
    r.name ← "Attack Session"

  for each attr:attribute except start_time,
      end_time, reference, analyzer, name,
      alertid:
    if attr is equal for all alerts in alert_list:
      r.attr ← alert[1].attr
    else:
      r.attr ← null
  return r
}
```

In the example attack scenario, the host-based sensor on the Linux host detects that Apache listens on port 80. This allows the session reconstruction to link the application-based alert (Alert 5) to the meta-alert describing the remote exploit (Alert 9), creating a new meta-alert as shown in Table 11.

4.8.1 Data Analysis

Attack session reconstruction requires either real-time access to the systems being protected or very detailed auditing information in order to map network traffic to host activity. For most of the data sets analyzed, this detailed information was not provided.

Table 12 shows the results of the attack session reconstruction component on all of the data sets. Even though attack session reconstruction achieved only a limited reduction rate, we believe it provides very valuable information to the network administrator since it allows one to track the evolution of an attack from the network domain to the host domain.

4.9 Attack Focus Recognition

The goal of the attack focus recognition component is to identify hosts that are either the source or the target of a substantial number of attacks. More specifically, this component aggregates the alerts associated with single hosts attacking multiple victims (called a *one2many* scenario) and single victims that are targeted by multiple attackers (called a *many2one* scenario). This correlation component is effective in reducing the number of alerts caused by distributed denial-of-service (DDoS) attacks and large-scale scans. Alerts related to a DDoS attempt can be merged into a single *many2one* meta-alert, while alerts related to individual scans can be combined into a single *one2many* meta-alert.

Both the *one2many* and the *many2one* scenarios are based on a sliding time window with extended timeout model. In the *one2many* scenario, the number of different targets is recorded for each attack source during the extended time window. If this number exceeds an a priori specified

TABLE 12
Impact of Attack Session Reconstruction

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	8,966	34,211	147,352	1,814,656	1,599,476	2,126	2,286
Output Alerts	8,966	34,211	147,352	1,814,656	1,599,476	2,126	2,234
Reduction	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	2.27%

TABLE 13
Impact of Focus Recognition

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	8,966	34,211	147,352	1,814,656	1,599,476	2,126	2,234
Output Alerts	7,985	17,247	14,832	205,856	465,831	2,078	1,104
Reduction	10.93%	49.58%	89.93%	88.65%	70.87%	2.26%	50.58%

threshold, an appropriate meta-alert is generated. The source of the meta-alert is set to be the attacker's host, while the target is the union of the targets of the individual attacks. The *many2one* scenario operates in a similar way, the only difference being that the roles of the attacker (source) and victim (target) are reversed.

When possible, the alerts generated by the *one2many* and *many2one* scenarios are further classified as either scans or denial-of-service attacks. More precisely, a *many2one* attack is classified as a DDoS attack when the total number of attacks against a victim exceeds a user-defined limit. A *one2many* alert is classified as a horizontal scan when a single port is scanned on all victim machines, or as a horizontal multiscan when more than one port is scanned.

The pseudocode for the one2many scenario is as follows:

```
parameter: timeout, report_threshold
global scenarios

one2many(alert) {
 remove all s:scenario from scenarios where
  s.timestamp < (alert.start_time - timeout)
 for each removed scenario s:
  if number of distinct victimhosts in
   s.alerts > report_threshold:
    send onetomany_merge(s.alerts) to
     next correlation component
  else:
   send s.alerts to next correlation component

 scenario ← get s:scenario from scenarios
  where s.hosts = alert.attackerhosts

 if scenario is null:
  scenario ← new scenario
  scenario.hosts ← alert.attackerhosts
  add scenario to scenarios

 scenario.timestamp ← alert.start_time
 add alert to scenario.alerts
}

one2many_merge(alert_list) {
 r ← new alert
 r.alertid ← get_unique_id()
 r.start_time ← min start_time of all alerts in
  alert_list
 r.end_time ← max end_time of all alerts in
  alert_list
 r.analyzer ← union of all analyzer in
  alert_list
 r.reference ← union of all alertid in
  alert_list

 if all alerts in alert_list refer to same port
  number:
  r.name ← "Horizontal Scan"
 else:
  r.name ← "Horizontal Multiscan"

 for each attr:attribute except start_time,
  end_time, reference, analyzer, name,
  alertid:
   if attr is equal for all alerts in
    alert_list: r.attr ← alert[1].attr
   else:
    r.attr ← null
 return r
}
```

The one2many scenario has two tunable parameters: the size of the timeout, which is used for the initial window size, and the minimum number of sensor alerts for a meta-alert to be generated. For the experiments we used a timeout of 120 seconds, and the minimum number of sensor alerts in a meta-alert was two.

The pseudocode for the many2one scenario is similar, but it is not presented due to space limitations. The many2one scenario has three tunable parameters: the first two are the same as for the one2many scenario. The third parameter is the number of meta-alerts required before a *many2one* alert is labeled as a denial-of-service attack. For the experiments the threshold for a denial-of-service attack was 500.

In the example scenario, attack focus recognition cannot be applied to the alerts. Note, however, that if numerous portscan alerts against multiple targets were received, these alerts would have been merged into a *one2many* meta-alert.

4.9.1 Data Analysis

The analysis of the data sets, presented in Table 13, shows that this component is very effective in achieving alert reduction for most of the data sets. The reduction is especially high when DDoS or large-scale scanning attempts are present in the data sets. The Honeypot data set achieved a lower reduction rate because only two hosts were monitored and no large-scale sweep of IP address blocks could be detected by monitoring such a small network.

4.10 Multistep Correlation

The goal of the multistep correlation component is to identify high-level attack patterns that are composed of several individual attacks. For example, consider an intruder who first scans a victim host, then breaks into a user account on that host and, finally, escalates privileges to become the `root` user. These three steps should be identified as belonging to one attack scenario performed by a single intruder. The high-level patterns are usually specified by using some form of expert knowledge.

In our system, multistep correlation is achieved by specifying attack scenarios using STATL [13], which is an extensible language that allows one to express attack patterns using states and transitions. We chose STATL to model multistep scenarios because a well-tested intrusion detection system capable of processing STATL scenarios was readily available [51].

Our correlation system supports two multistep scenarios: *recon-breakin-escalate* and *island-hopping*. The recon-breakin-escalate scenario models an attacker who scans for vulnerabilities in a network or host, breaks into a vulnerable host, and escalates her privilege after breaking in. The island-hopping scenario models an attacker who breaks into a host and then uses that host as a platform to break into other hosts. This scenario is able to identify chains of alerts in which the victim in one alert becomes the attacker in the following one. Both of these scenarios use the sliding time window with extended timeout model. The timeout value for the recon-breakin-escalate scenario is 20 minutes.

The pseudocode for the recon-breakin-escalate scenario is as follows:

```
parameter timeout
global attack_senarios

recon_breakin_escalate(alert) {
 remove all s:scenario from attack_scenarios
  where s.timestamp <
   (alert.start_time - timeout)
 for each removed scenario s:
  if s.state is "Reconnaissance":
   pass s.alerts to next correlation component
  else:
   pass recon_breakin_escalate_merge
    (s.alerts) to next correlation component

 scenarios ← get all s:scenario from
  attack_senarios where
  alert.victimnodes ∩ s.nodes is nonempty

 if scenarios is null and alert.type is
   "Reconnaissance":
  s ← new scenario
  add s to attack_senarios
  s.state ← "Reconnaissance"
  s.timestamp ← alert.start_time
  scenarios ← s

 for each s:scenario in scenarios:
  s.timestamp ← alert.start_time
  s.nodes ← s.nodes ∪ alert.victimhosts
  if alert.type is "Breakin" and s.state is
   "Reconnaissance":
   s.state ← "Breakin"
  else if alert.type is "Escalation" and
    s.state is "Breakin":
     s.state ← "Escalation"

 if scenarios is null:
  pass alert to next correlation component
}

recon_breakin_escalate_merge(alert_list) {
 r ← new alert
 r.alertid ← get_unique_id()
 r.start_time ← min start_time of all alerts in
  alert_list
 r.end_time ← max end_time of all alerts in
  alert_list
 r.name ← "Recon_Breakin_Escalate"
 r.analyzer ← union of all analyzer in
  alert_list
 r.reference ← union of all alertid in
  alert_list
 r.attackerhosts ← union of all attackerhosts
  in alert_list
 r.victimhosts ← union of all victimhosts in
  alert_list

 for each attr:attribute except start_time,
  end_time, name, attackerhost, victimhost,
  reference, analyzer, alertid:
   if attr is equal for all alerts in
    alert_list: r.attr ← alert_list[1].attr
   else:
    r.attr ← null
 return r
}
```

The pseudocode for the island-hopping scenario is as follows:

```
parameter windowsize
global victim_graphs

islandhopping(alert) {
 remove all g:graph from victim_graphs where
  g.timestamp <
   (alert.start_time - windowsize)

 for each removed graph g:
  if g.alerts contains only one alert a:
   pass a to next correlation component
  else:
   pass islandhop_merge(g) to next correlation
    component
```

TABLE 14
Effect of Multistep Correlation on the Alerts for the Example Attack

AlertID	Name	Sensor	Start/End	Source	Target	Tag
10	Meta-Alert	H	24.6 / 24.7	10.0.0.1	10.0.0.1, linuxconf	{6, 7}, **correlated**
11	Meta-Alert	{N1, N2, A}	10.0 / 22.1	{31.3.3.7, 10.0.0.1}	10.0.0.1, port:80, Apache	{5, 9}, **correlated**
12	**Meta-Alert**	**{N1, N2, H, A}**	**10.0 / 24.7**	**{31.3.3.7, 10.0.0.1}**	**10.0.0.1, port:80, Apache, linuxconf**	**{10, 11}**

```
  if alert.type = breakin:
    for each g:graph in victim_graphs where
      g.nodes ∩ alert.attackerhosts is nonempty:
        add alert.victimhosts to g.nodes
        g.timestamp ← alert.start_time
        add alert to g.alerts
        for each host m in alert.attackerhosts:
          for each host n in alert.victimhosts:
            add an edge (m,n)
    if no graph g is found in previous step:
      g ← new graph
      add g to victim_graphs
      g.nodes ← alert.victimhosts
      g.timestamp ← alert.start_time
      g.alerts ← alert
  else:
    pass alert to next correlation component
}

islandhop_merge(graph) {
  r ← new alert
  r.alertid ← get_unique_id()
  r.start_time ← min start_time of all alerts in
    graph.alerts
  r.end_time ← max end_time of all alerts in
    graph.alerts
  r.name ← "Islandhop"
  r.analyzer ← union of all analyzer in
    graph.alerts
  r.attackgraph ← graph
  r.reference ← union of all alertid in
    graph.alerts
  r.attackerhosts ← union of all attackerhosts
    in graph.alerts
  r.victimhosts ← union of all victimhosts in
    graph.alerts
  for each attr:attribute except start_time,
   end_time, name, attackerhost, victimhost,
   reference, analyzer, attackgraph, alertid:
    if attr is equal for all alerts in
      graph.alerts:
      r.attr ← graph.alerts[1].attr
    else:
      r.attr ← null
  return r
}
```

When building up the island-hopping meta-alert, a graph containing the compromised nodes is generated. This graph has an edge from each attacking host to all targets compromised from this host. These graphs can be used in many ways to display the advancing stages of an attack to the site security officer. The timeout value for the island-hopping scenario is 20 minutes.

A multistep pattern that includes scanning, break-in, and escalation of privileges can be applied to our example attack scenario. The meta-alert that describes the scan and the remote attack against the Web server can be merged with the local exploit. The results of this operation are shown in Table 14.

4.10.1 Data Analysis

While multistep attack analysis may not generate the same level of alert reduction achieved by other components, it often provides a substantial improvement in the abstraction level of the produced meta-alerts. The result of applying the multistep component to our data sets is shown in Table 15. In three of the data sets, we were not able to detect any multistep attacks. The MIT/LL 1999 data set is not believed to contain any multistep attacks since the data set was not created with these types of attacks in mind. The Rome AFRL and Defcon 9 data sets are likely to contain numerous multistep attacks. However, we were not able to detect any in the Rome AFRL data set because all alerts classified as scans did not contain the destination IP of the scanned

TABLE 15
Impact of Multistep Correlation

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	7,985	17,247	14,832	205,856	465,831	2,078	1,104
Output Alerts	7,985	17,220	14,738	203,303	465,831	2,057	1,080
Reduction	0.00%	0.16%	0.63%	1.24%	0.00%	1.01%	2.17%

hosts. This is believed to be caused by human error when exporting the alerts from the Air Force database. Note that the MIT/LL 2000 and Treasure Hunt data sets were created explicitly to contain evidence of multistep attacks. These attacks were correctly identified, and even though the resulting reduction rate is minimal, the correlated meta-alerts provide a high-level, abstract view of the complex attacks.

4.11 Impact Analysis

The goal of the impact analysis component is to determine the impact of an attack on the proper operation of the protected network. This allows one to link the failure of a network service to an attack that may at first seem unrelated. Consider, for example, an attacker that crashes the RPC (remote procedure call) daemon that is used by the NFS server. Using impact analysis, the failure of the NFS (network file system) service can be linked to the RPC attack, although the NFS server has not been attacked directly.

The impact analysis process combines the alerts from the previous correlation components with data from an asset database and a number of heartbeat monitors. The asset database stores information about installed network services, dependencies among these services, and their importance to the overall operation of a network installation. An example of a dependency between two services is a mail server that requires an operational domain name server (DNS) to work properly or the NFS server that requires the RPC services. The purpose of the heartbeat monitors is to assure that certain services are alive and functional.

Whenever an alert affects certain network services, the asset database is consulted to determine all other services that are dependent on this target (and subsequently, other services that depend on those services). The heartbeat monitor then checks whether all dependent services are still operational. If any service is found to have failed, this information can be added to the alert as a likely consequence of the attack.

Consider, for example, an alert that refers to a bandwidth denial-of-service attack against a network link. Using the asset database, the correlation system can determine that the proper operation of the domain name service is dependent on the availability of this particular link. The database also contains knowledge that the mail service requires an operating DNS server. When the heartbeat monitor detects a failure of the mail system (e.g., by sending messages to a test account), the failure of both the mail and DNS servers are linked to the alert as additional consequences of the attack. In our example attack scenario, only a single service (i.e., the Web server) is present and, therefore, no dependencies are modeled in the asset database.

The pseudocode for the impact analysis component is as follows:

```
parameter window_size
global asset_database, attack_table

impact_processalert(alert) {
 remove all a:alert from attack_table where
  a.start_time <
   (alert.start_times - window_size)
 if alert.type is not ″Reconnaissance″:
  add alert to attack_table
 pass alert to next correlation component
}

impact_processheartbeat(heartbeat) {
 if heartbeat.status = ″Down″:
  dependencies ← get all a:asset from
   asset_database where
    (heartbeat.victimhost,
     heartbeat.victimservice) is
    dependent on a

 for each dependecy:asset in dependencies:
  attacks ← get all a:alert from attack_table
   where dependency.host ∈ a.victimhosts and
   dependency.service = a.victimservice
  for each alert in attacks:
   alert.consequence ← alert.consequence ∪
    (heartbeat.victimhost,
     heartbeat.victimservice)
}
```

4.11.1 Data Analysis

Impact analysis requires a precise modeling of the relationships among assets in a protected network and requires the constant monitoring of the health of those assets. Therefore, the data sets used in our experiments cannot be used to evaluate the effectiveness of this analysis technique.

4.12 Alert Prioritization

Priorities are important to classify alerts and quickly discard information that is irrelevant or of less importance to a particular site. The goal of the alert prioritization component is to assign appropriate priorities to alerts. This component has to take into account the security policy and the security requirements of the site where the correlation system is deployed. Therefore, there is no absolute priority for an attack. For example, port scans have become so common on the Internet that most administrators consider them as mere nuisances and do not spend any time or resources in tracking down their sources. This suggests that an alert referring to a scan should be tagged with a low priority. However, the situation is different when the scan originates from one's own network. Also, there are high-security sites that do not expect any scans at all. In these cases (and under the corresponding security policies), a scan would be marked with a high priority.

The alert prioritization component can use the information from the impact analysis and the asset database to determine the importance of network services to the overall operation of the network. For each network resource, the asset database contains values that characterize its *secrecy*, *integrity*, and *availability* properties. For example, a public Web service owned by a company may have a low *secrecy* value, because the information published through the Web

TABLE 16
Correlated Output with Priorities for the Example Attack Scenario

Priority	AlertID	Description	Tag	Reference
High	12	Multistep Attack Scenario		{11, 10}
Low	11	Remote Attack Session	correlated	{9, 5}
Low	9	Remote Attack Thread	correlated	{8, 4}
Low	8	Fused Portscan	correlated	{2, 3}
Low	2	Portscan	correlated	
Low	3	Portscan	correlated	
Low	4	Apache Exploit	correlated	
Low	5	Bad Request	correlated	
Low	10	Local Attack Thread	correlated	{6, 7}
Low	6	Local Exploit	correlated	
Low	7	Local Exploit	correlated	
Low	1	IIS Exploit	nonrelevant	

service is freely available. However, the *integrity* and *availability* values may be high if it is vital for the company to keep the information accurate and accessible. As another example, a mail service might have a high *secrecy* value if it is used to exchange confidential email messages. If the mail system relies on a number of backup mail servers, the *availability* value for a single server could be low. Note that the different values are manually entered into the database and do not represent an absolute measure of the importance of any asset, but rather reflect the subjective view of the security administrator.

The pseudocode for the alert prioritization component is as follows:

```
global asset_database, dos_attacks,
 read_attacks, write_attacks

prioritize(alert) {
 asset_list ← get all a:asset from
   asset_database where:
  a.host ∈ alert.victimhosts and
  a.service = alert.victimservice

 if alert.name ∈ dos_attacks:
  alert.priority ← max availability_score of
    all assets in asset_list
 else if alert.name ∈ read_attacks:
  alert.priority ← max secrecy_score of all
   assets in asset_list
 else if alert.name ∈ write_attacks:
  alert.priority ← max integrity_score of all
   assets in asset_list
}
```

In our example scenario, the Web service is considered to be important for the operation of the site. Therefore, the meta-alert that summarizes the multistep attack against the Linux machine receives a high priority. All correlated or failed attacks are, as a default action, assigned a low priority. The final output of our correlation system for the sample scenario is shown in Table 16. As desired, only a single, high-priority alert is presented to the administrator. This meta-alert summarizes the complete attack scenario and combines all alerts that are related to this attack. The remaining alerts are intermediate alerts that are either part of the attack scenario or indications of nonrelevant intrusion attempts.

4.12.1 Data Analysis

Alert prioritizing relies on meta-information about the nature of the attack to determine if one alert or meta-alert should be considered more critical than another. Thus, unless a precise description of how to prioritize alerts is provided (in addition to the data sets), it is difficult to evaluate the effectiveness of this component. We developed a very naive alert prioritization policy just to give a rough idea of how much reduction this step could achieve. In our policy, Web servers are the only critical assets. This policy could be suitable for a Web-based business, but in most cases a much more complex policy is needed. The results are shown in Table 17.

TABLE 17
Impact of Prioritization

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	7,985	17,220	14,738	203,303	465,831	2,078	1,080
High Priority Alerts	2,571	85	424	3,705	32,608	1,356	546
Reduction	67.80%	99.51%	97.12%	98.18%	93.00%	34.74%	49.44%

TABLE 18
Total Alert Reduction

	MIT/LL 1999	MIT/LL 2000	CTV	Defcon 9	Rome AFRL	Honeypot	Treasure Hunt
Input Alerts	41,760	36,635	215,190	6,378,096	5,299,390	260,120	2,811,169
Output Alerts	7,985	17,220	14,738	203,303	465,831	2,078	1,080
Reduction	80.87%	53.00%	93.15%	96.81%	91.21%	99.20%	99.96%

4.13 Summary of Experimental Results

Table 18 shows the total reduction rates for each of the data sets. In this table, the reduction caused by the prioritization step is not included, since the policy used for prioritization in our experiments was too simplistic and would have distorted the results.

None of the data sets have a truth file suitable for evaluating the correctness of each of the correlation components. The Lincoln Lab data and the Cyber Technology Validation data do have truth files, but the data contained in them is not complete enough to be used to determine the correct results for each individual correlation component. For instance, to evaluate the correctness of the fusion component a list of alerts representing the same attack occurrence is needed. This information is not available from any of the truth files. However, we have validated the correctness of our approach by manually analyzing the output of the correlation system. For the data sets that have a textual description of the contained attacks (Lincoln Lab, Cyber Technology Validation, and Treasure Hunt), the output of the correlation system is consistent with the description provided.

As shown in the data analysis part of the previous sections, the reduction rate for each step of the correlation process is heavily dependent on the data set. Several properties of a data set play a role in determining how much reduction is achieved at each step. These properties can roughly be divided into three different groups.

The first group is the topology of the defended network. For instance, in the honeypot data set the defended network was very small (two hosts), which made it impossible to detect any large-scale IP sweeps. These large-scale sweeps were very common in all the other data sets. The network topology also includes the type of intrusion detection sensors that are deployed as well as their configuration and placement. For instance, in the Defcon 9 data set, the sensors were placed such that most network traffic was observed by two sensors. This resulted in a higher fusion rate than for the other data sets.

The second group of properties is the characteristics of the attacks. For instance, a brute-force password guessing attack caused a very high reduction rate in the thread reconstruction step of the Treasure Hunt data set. Also, we did not find any multistep attacks in the MIT/LL 1999 data since the data set was comprised of individual attacks, and no high-level attack patterns were included when the data was generated.

Finally, the meta-data available for the correlation system plays an important role in how well the correlation is performed. If information about alert classification is not provided, no multistep attacks are detected in any of the data sets. Similarly, the session reconstruction step needs information about which processes are listening to which ports. Without this information, session reconstruction cannot be performed reliably. The alert verification step is also dependent on meta-data. In particular, this component requires up-to-date information about which hosts are running services with known vulnerabilities.

5 Related Work

Alert correlation has been advocated by researchers and vendors as both a means to reduce the number of alerts that need to be manually analyzed and a way to increase the relevance and abstraction level of the resulting reports. Even though the goals of correlation seem to be well-defined, the correlation approaches proposed so far emphasize different aspects of the correlation process, making it difficult to compare the results of each solution [16].

A number of the proposed approaches include a multiphase analysis of the alert stream. For example, the model proposed by Andersson et al. [1] and Valdes and Skinner [47], [48] present a correlation process in two phases. The first phase aggregates low-level events using the concept of attack threads. The second phase uses a similarity metric to fuse alerts into meta-alerts, in an attempt to provide a higher-level view of the security state of the system. These phases correspond to the *thread reconstruction* and *focus recognition* components in our model, respectively. Our experience with the correlation process shows that these phases are important, but not sufficient to reduce both false positives and the number of relevant alerts presented to the security officer. In particular, an *alert verification* component is key to reducing the number of relevant alerts to be considered in the correlation process and a *multistep attack* component allows a security expert to model higher-level attack scenarios.

The model initially proposed in [47] has been extended to take into account the impact of alerts on the overall mission that a network infrastructure supports [39]. This approach relies on a knowledge base that describes the security-relevant characteristics of a protected network to prioritize the alerts and to perform a simple form of passive alert verification. The information about network assets is gathered using Nmap and contains only information that is gathered by this specific tool, such as IP addresses, installed operating systems, and open ports. Our system is different in two ways. First, it can use runtime information to verify if an alert is related to a successful intrusion. Second, it supports a more comprehensive set of mechanisms to gather information about the security posture of the attack target.

Another extension recently proposed by the same group is the use of an attack modeling language, called CAML, to specify the pre and postconditions of an attack [4]. Pre and postconditions are defined for individual attacks, and alerts are connected (or correlated) when the postcondition of one alert matches the precondition of a later one. This allows the specification of complex chains of attacks without explicitly modeling complex scenarios. This approach is similar to the *multistep attack* component in our model, but it appears that the integration with the other phases in the correlation process has not been completed, and there are no available results to evaluate the effectiveness of the integration of this component in the correlation process.

Several other researchers have proposed mechanisms similar to pre and postconditions to model dependencies between attacks. Ning et al. propose a model that identifies causal relationships between alerts using prerequisites and consequences [35], [36]. Graphs of connected alerts, called hyper-alerts, are created and then graph manipulation techniques are applied to reduce these hyper-alerts to a manageable size. The approach proposed by Cuppens and Miège in [5] also uses pre and postconditions. In addition, it includes a number of phases including alert clustering, alert merging, and intention recognition. In the first two phases, alerts are clustered and merged using a similarity function. The intention recognition phase is referenced in their model, but has not been implemented. An interesting aspect of this approach is the attempt to generate correlation rules automatically. While it may seem appealing, this technique could generate a number of spurious correlation rules that, instead of reducing the number of alerts and increasing the abstraction level of the reports, could introduce the correlation of alerts that are "close" or "similar" by pure chance, in this way increasing the noise in the alert stream.

Another example that uses pre and postconditions to identify causal relationships between alerts is JIGSAW [45]. In JIGSAW, attack conditions are expressed using *capabilities* and *concepts*. Capabilities are used to describe the information that the attacker must know to perform a certain attack (e.g., a user name and password for a valid account) or a condition that represents a necessary context for an attack (e.g., a particular configuration of the network). Concepts are used to model fragments of complex attacks (e.g., a denial-of-service attack against a specific host) and both their requisites and their impact on the security of the protected network are expressed in terms of capabilities. By composing the capability provided by a concept with the capability required by another concept, it is possible to recognize complex attack scenarios (e.g., a remote shell connection spoofing that relies on a denial-of-service attack). Unfortunately, no empirical evaluation of the effectiveness of the system has been published and, therefore, it is not possible to compare it to other systems. In addition, JIGSAW does not provide any additional support for other components of the correlation process other than multistep attack detection.

Most of the approaches based on pre and postconditions are focused on the modeling and detection of multistep attacks to provide a high-level view of the "attack history" associated with a security compromise. It is assumed that the analyzed event stream is composed only of well-defined, relevant alerts, and that real attacks trigger more than a single alert. As a result, these systems can focus on clusters of related alerts and discard all alerts that have not been correlated. Unfortunately, this assumption has not been substantiated by experimental data or supported by a rigorous analysis. Instead, it is often necessary to filter out nonrelevant alerts that may generate spurious attack histories. This view is supported by a recent paper [37] on alert correlation, which states that "false alerts generated by IDSs have a negative impact [on correlation]."

A limitation of approaches that are based on pre and postconditions is the need to manually define these conditions for all alerts. No automatic correlation operations such as *thread reconstruction* or *session reconstruction* take place. Also, when only dependencies between alerts are modeled (as opposed to complete scenarios), it is not possible to monitor the evolution of a particular scenario instance from state to state in real-time, possibly anticipating the further progress of an intrusion.

In [8], Debar and Wespi propose a system that performs both aggregation and correlation of intrusion detection alerts produced by a number of different sensors. This approach acknowledges that an alert stream may contain a large number of false positives, but it does not provide any specific technique to eliminate these spurious alerts. An attempt to address this limitation is described as part of the M2D2 model [31], [32], which relies on a formal description of sensor capabilities in terms of scope and positioning to determine if an alert is a false positive. More precisely, the model is used to verify if all sensors that could have been able to detect an attack agreed during the detection process, making the assumption that inconsistent detections denote the presence of a false alarm. While this approach benefits from a sound formal basis, it suffers from the limitation that false alerts can only be detected for those cases in which multiple sensors are able to detect the same attack and can participate in the voting process. Unfortunately, many real-world intrusion detection systems do not provide enough detection redundancy to make this approach applicable.

Alert verification using vulnerability analysis information has been advocated as an important tool to reduce the noise in the alert stream produced by intrusion detection sensors in [10], [15], [24]. Unfortunately, these approaches are preliminary and have not been integrated in the overall correlation process. In addition, most alert verification systems rely on information about the security configuration of the protected network that was collected at an earlier time using vulnerability scanning tools, and most do not support dynamic mechanisms for alert verification.

6 Conclusions

In this paper, we described a multicomponent correlation process and a framework that performs the correlation analysis. To the best of our knowledge, the approach described in this paper integrates the most complete set of components in the correlation process. We applied a tool, based on our framework, to a large number of diverse data sets, to analyze if and how each component contributes to the overall correlation process.

Our experiments demonstrate that the effectiveness of each component is dependent on the data sets being analyzed. As discussed in Section 4.13, the topology of the network, the characteristics of the attack, and the available meta-data all significantly influence the performance of the correlation process. Thus, one cannot, in general, determine a ranking among components with respect to their effectiveness. Each component can contribute to the overall analysis. Therefore, the most complete set of components should be used.

A preliminary version of our correlation tool was one of five systems that were independently evaluated in DARPA's Correlation Technology Validation effort [16]. Even though, at that time, the prototype included only a subset of the components described in this paper, it was ranked as one of the top correlators. The prototype produced two orders of magnitude reduction in the number of alerts and operated in real-time.

The current implementation of our framework uses a pipeline architecture. Part of our future work will be to investigate other architectures to see if we can achieve better performance or more effective alert correlation.

Other future work will be to more thoroughly investigate the algorithms currently being used in each of the components to determine if any improvements can be made. For instance, the fusion component currently matches a new alert with the earliest occurrence of the same alert from a different sensor that is within a specified time window. Another possibility would be to fuse the new alert with the temporally closest occurrence of the same alert from a different sensor.

The verification component in our correlator was limited in that there were some alerts that could not be verified by using the active scanning approach. Therefore, we could not determine if these alerts were true positives. Future work will investigate the feasibility of using authenticated access to check the success of an attack, and some of the passive techniques that were discussed in Section 4.6 will also be investigated.

Acknowledgments

This research was supported by the Army Research Laboratory and the Army Research Office, under agreement DAAD19-01-1-0484. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The authors would also like to thank the anonymous reviewers. Their insightful comments and suggestions helped to make this a much better paper.

References

[1] D. Andersson, M. Fong, and A. Valdes, "Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis," *Proc. Third Ann. IEEE Information Assurance Workshop,* June 2002.

[2] M. Arboi, *The Nessus Attack Scripting Language Reference Guide,* 2002, http://www.nessus.org/doc/nasl2_reference.pdf.

[3] S.M. Bellovin, "Packets Found on an Internet," technical report, AT&T Bell Laboratories, May 1992.

[4] S. Cheung, U. Lindqvist, and M. Fong, "Modeling Multistep Cyber Attacks for Scenario Recognition," *Proc. DARPA Information Survivability Conf. and Exposition (DISCEX III),* pp. 284-292, Apr. 2003.

[5] F. Cuppens and A. Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework," *Proc. IEEE Symp. Security and Privacy,* May 2002.

[6] D. Curry and H. Debar, "Intrusion Detection Message Exchange Format: Extensible Markup Language (XML) Document Type Definition," draft-ietf-idwg-idmef-xml-10.txt+, Jan. 2003.

[7] Common Vulnerabilities and Exposures, http://www.cve.mitre.org/, 2003.

[8] H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts," *Proc. Int'l Symp. Recent Advances in Intrusion Detection,* pp. 85-103, Oct. 2001.

[9] D.E. Denning, "An Intrusion Detection Model," *IEEE Trans. Software Eng.,* vol. 13, no. 2, pp. 222-232, Feb. 1987.

[10] N. Desai, "IDS Correlation of VA Data and IDS Alerts," http://www.securityfocus.com/infocus/1708, June 2003.

[11] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo, "Addendum to Testing and Evaluating Computer Intrusion Detection Systems," *Comm. ACM,* vol. 42, no. 9, p. 15, Sept. 1999.

[12] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo, "Testing and Evaluating Computer Intrusion Detection Systems," *Comm. ACM,* vol. 42, no. 7, pp. 53-61, July 1999.

[13] S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-Based Intrusion Detection," *J. Computer Security,* vol. 10, nos. 1-2, pp. 71-104, 2002.

[14] A.K. Ghosh, J. Wanken, and F. Charron, "Detecting Anomalous and Unknown Intrusions against Programs," *Proc. Ann. Computer Security Application Conf. (ACSAC '98),* pp. 259-267, Dec. 1998.

[15] R. Gula, "Correlating IDS Alerts with Vulnerability Information," technical report, Tenable Network Security, Dec. 2002.

[16] J. Haines, D.K. Ryder, L. Tinnel, and S. Taylor, "Validation of Sensor Alert Correlators," *IEEE Security and Privacy Magazine,* vol. 1, no. 1, pp. 46-56, Jan./Feb. 2003.

[17] L.T. Heberlein, G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *Proc. IEEE Symp. Research in Security and Privacy,* pp. 296-304, May 1990.

[18] K. Ilgun, "USTAT: A Real-Time Intrusion Detection System for UNIX," *Proc. IEEE Symp. Research on Security and Privacy,* May 1993.

[19] ISS, Realsecure, http://www.iss.net/, 2004.

[20] H.S. Javitz and A. Valdes, "The NIDES Statistical Component Description and Justification," technical report, SRI Int'l, Mar. 1994.

[21] K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems," master's thesis, MIT, June 1999.

[22] G.H. Kim and E.H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," technical report, Purdue Univ., Nov. 1993.

[23] C. Ko, M. Ruschitzka, and K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-Based Approach," *Proc. 1997 IEEE Symp. Security and Privacy,* pp. 175-187, May 1997.

[24] C. Kruegel and W. Robertson, "Alert Verification: Determining the Success of Intrusion Attempts," *Proc. First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004),* July 2004.

[25] C. Kruegel and G. Vigna, "Anomaly Detection of Web-Based Attacks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* pp. 251-261, Oct. 2003.

[26] MIT Lincoln Laboratory, Lincoln Lab Data Sets, http://www.ll.mit.edu/IST/ideval/data/data_index.html, 2000.

[27] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation," *Proc. DARPA Information Survivability Conf. and Exposition,* vol. 2, Jan. 2000.

[28] BugTraq Mailing List, Vulnerabilities by Bugtraq ID, http://www.securityfocus.com/bid/bugtraqid/, 2004.

[29] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evalautions as Performed by Lincoln Laboratory," *ACM Trans. Information and System Security,* vol. 3, no. 4, Nov. 2000.

[30] D.L. Mills Network Time Protocol (Version 3), RFC 1305, 1992.

[31] B. Morin and H. Debar, "Correlation of Intrusion Symptoms: An Application of Chronicles," *Proc. Int'l Symp. Recent Advances in Intrusion Detection,* Sept. 2003.

[32] B. Morin, L. Me, H. Debar, and M. Ducasse, "M2D2: A Formal Data Model for IDS Alert Correlation," *Proc. Recent Advances in Intrusion Detection,* pp. 115-137, 2002.

[33] Nessus Vulnerabilty Scanner, http://www.nessus.org/, 2004.

[34] P.G. Neumann and P.A. Porras, "Experience with EMERALD to Date," *Proc. First USENIX Workshop Intrusion Detection and Network Monitoring,* pp. 73-80, Apr. 1999.

[35] P. Ning, Y. Cui, and D.S. Reeves, "Analyzing Intensive Intrusion Alerts Via Correlation," *Proc. Int'l Symp. the Recent Advances in Intrusion Detection,* pp. 74-94, Oct. 2002.

[36] P. Ning, Y. Cui, and D.S. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," *Proc. ACM Conf. Computer and Comm. Security,* pp. 245-254, Nov. 2002.

[37] P. Ning and D. Xu, "Learning Attack Strategies from Intrusion Alert," *Proc. ACM Conf. Computer and Comm. Security (CCS '03),* Oct. 2003.

[38] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Proc. Seventh USENIX Security Symp.,* Jan. 1998

[39] P. Porras, M. Fong, and A. Valdes, "A Mission-Impact-Based Approach to INFOSEC Alarm Correlation," *Proc. Int'l Symp. the Recent Advances in Intrusion Detection,* pp. 95-114, Oct. 2002.

[40] UCSB Reliable Software Group, LinSTAT Webpage, http://www.cs.ucsb.edu/rsg/STAT/software/linstat.html, 2003.

[41] UCSB Reliable Software Group, collection of ntrusion detection data sets, http://www.cs.ucsb.edu/rsg/datasets/, 2004.

[42] M. Roesch, "Snort—Lightweight Intrusion Detection for Networks," *Proc. USENIX LISA '99 Conf.,* Nov. 1999

[43] U. Shankar and V. Paxson, "Active Mapping: Resisting NIDS Evasion Without Altering Traffic," *Proc. IEEE Symp. Security and Privacy,* 2003.

[44] Snort—The Open Source Network Intrusion Detection System, http://www.snort.org, 2004.

[45] S.J. Templeton and K. Levitt, "A Requires/Provides Model for Computer Attacks," *Proc. New Security Paradigms Workshop,* pp. 31-38, Sept. 2000.

[46] A. Valdes and K. Skinner, "Adaptive, Model-Based Monitoring for Cyber Attack Detection," *Proc. RAID 2000 Conf.,* Oct. 2000.

[47] A. Valdes and K. Skinner, "An Approach to Sensor Correlation," *Proc. Int'l Symp. Recent Advances in Intrusion Detection,* Oct. 2000.

[48] A. Valdes and K. Skinner, "Probabilistic Alert Correlation," *Proc. Int'l Symp. Recent Advances in Intrusion Detection,* pp. 54-68, Oct. 2001.

[49] G. Vigna, "Teaching Hands-On Network Security: Testbeds and Live Exercises," *J. Information Warfare,* vol. 3, no. 2, pp. 8-25, 2003.

[50] G. Vigna and R.A. Kemmerer, "NetSTAT: A Network-Based Intrusion Detection System," *J. Computer Security,* vol. 7, no. 1, pp. 37-71, 1999.

[51] G. Vigna, F. Valeur, and R.A. Kemmerer, "Designing and Implementing a Family of Intrusion Detection Systems," *Proc. European Software Eng. Conf. and ACM SIGSOFT Symp. the Foundations of Software Eng. (ESEC/FSE 2003),* Sept. 2003.

[52] C. Warrender, S. Forrest, and B.A. Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *Proc. IEEE Symp. Security and Privacy,* pp. 133-145, 1999.

Fredrik Valeur is currently a PhD student at the University of California, Santa Barbara. He holds a Sivilingeniør degree in computer science from the Norwegian University of Science and Technology in Trondheim. His research interests include intrusion detection, network security, penetration testing, and network scanning techniques.

Giovanni Vigna received the MS degree with honors and the PhD degree from Politecnico di Milano, Italy, in 1994 and 1998, respectively. He is an associate professor in the Department of Computer Science at the University of California in Santa Barbara. His current research interests include network and computer security, intrusion detection, security of mobile code systems, penetration testing, and wireless systems. In particular, he worked on STAT, a framework for the modular development of intrusion detection systems. He also published a book on security and mobile agents and he has been the program chair of the International Symposium on Recent Advances in Intrusion Detection (RAID 2003). He is a member of the IEEE and the IEEE Computer Society.

Christopher Kruegel received the PhD degree with honors in computer science from the Technical University Vienna while working as a research assistant for the Distributed Systems Group. He is an assistant professor with the Automation Systems Group at the Technical University Vienna. Before that, he was working as a research postdoc for the Reliable Software Group at the University of California, Santa Barbara. His research interests include most aspects of computer security, with an emphasis on network security, intrusion detection, and vulnerability analysis. He is a member of the IEEE.

Richard A. Kemmerer received the BS degree in mathematics from the Pennsylvania State University in 1966, and the MS and PhD degrees in computer science from the University of California, Los Angeles, in 1976 and 1979, respectively. He is a professor and a past chair of the Department of Computer Science at the University of California, Santa Barbara. His research interests include formal specification and verification of systems, computer system security and reliability, programming and specification language design, and software engineering. He is author of the book *Formal Specification and Verification of an Operating System Security Kernel*. He is a fellow of the IEEE and IEEE Computer Society, a fellow of the Association for Computing Machinery, a member of the IFIP Working Group 11.3 on Database Security, and a member of the International Association for Cryptologic Research. He is a past editor-in-chief of *IEEE Transactions on Software Engineering* and has served on the editorial boards of the *ACM Computing Surveys* and *IEEE Security and Privacy.*