INTRUSION DETECTION SYSTEM ALERT CORRELATION WITH OPERATING SYSTEM LEVEL LOGS

A Thesis Submitted to
The Graduate School of Engineering and Sciences of
İzmir Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Software

by

Mustafa TOPRAK

December 2009

İzmir

We approve the thesis of **Mustafa TOPRAK**

Prof. Dr. Sıtkı AYTAÇ
Supervisor

Assist. Prof. Dr. Tuğkan TUĞLULAR
Committee Member

Prof. Dr. Şaban EREN
Committee Member

11 December 2009

Prof. Dr. Sıtkı AYTAÇ
Head of the Department of
Computer Engineering

Assoc. Prof. Dr. Talat YALÇIN
Dean of the Graduate School of
Engineering and Sciences

ABSTRACT

INTRUSION DETECTION SYSTEM ALERT CORRELATION WITH OPERATING SYSTEM LEVEL LOGS

Internet is a global public network. More and more people are getting connected to the Internet every day to take advantage of the Internetwork connectivity. It also brings in a lot of risk on the Internet because there are both harmless and harmful users on the Internet. While an organization makes its information system available to harmless Internet users, at the same time the information is available to the malicious users as well. Most organizations deploy firewalls to protect their private network from the public network. But, no network can be hundred percent secured. This is because; the connectivity requires some kind of access to be granted on the internal systems to Internet users. The firewall provides security by allowing only specific services through it. The firewall implements defined rules to each packet reaching to its network interface. The IDS complements the firewall security by detected if someone tries to break in through the firewall or manages to break in the firewall security and tried to have access on any system in the trusted site and alerted the system administrator in case there is a breach in security. However, at present, IDSs suffer from several limitations. To address these limitations and learn network security threats, it is necessary to perform alert correlation.

Alert correlation focuses on discovering various relationships between individual alerts. Intrusion alert correlation techniques correlate alerts into meaningful groups or attack scenarios for ease to understand by human analysts. In order to be sure about the alert correlation working properly, this thesis proposed to use attack scenarios by correlating alerts on the basis of prerequisites and consequences of intrusions. The architecture of the experimental environment based on the prerequisites and consequences of different types of attacks, the proposed approach correlates alerts by matching the consequence of some previous alerts and the prerequisite of some later ones with OS-level logs. As a result, the accuracy of the proposed method and its advantage demonstrated to focus on building IDS alert correlation with OS-level logs in information security systems.

ÖZET

SALDIRI TESPİT SİSTEMİ ALARMLARININ İŞLETİM SİSTEMİ LOG KAYITLARI İLE KORELASYONU

İnternet genel bir küresel ağdır. Giderek daha çok insan internete bağlı olmanın avantajlarından faydalanmaktadır. Ancak internet faydanın yanında birçok riski de beraberinde getirmektedir. Çünkü hem zararsız hem de zararlı kullanıcılar internete bağlı durumdadır. Herhangi bir özel ağ internet erişimine açıldığında, zararsız kullanıcılar yanında kötü niyetli kullanıcılara da sistem açılmış olur. Birçok organizasyon kendi özel ağını genel ağdan ayrımak için güvenlik duvarlarını kullanmaktadır. Ancak güvenlik duvarları bir ağı yüzde yüz güvenli kılmaz. Bunun ana nedeni, genel ağ ile özel ağ arasında bazı erişim haklarının verilmesinden kaynaklanmaktadır. Güvenlik duvarı sadece belirli servislere erişim hakları tanımlayarak ağ güvenliğini sağlamaktadır. Güvenlik duvarı üzerinde tanımlı kural setlerine göre ağ paketlerinin erişimine izin verir. Bu noktada saldırı tespit sistemleri güvenlik duvarının oluşturduğu ağ güvenliğini tamamlamaktadır. Eğer bir ağ kullanıcısı güvenlik duvarını kırmaya ya da özel ağdaki bir sistemi kontrol etmeye çalışırsa; saldırı tespit sistemi saldırı ile ilgili olarak sistem yöneticisini uyarır. Bununla beraber saldırı tespit sistemlerinin bazı kısıtları da vardır. Bu kısıtları belirlemek ve ağ güvenliğini tehdit eden faaliyetleri öğrenmek için saldırı tespit sistemi alarmları korelasyona tabi tutulur.

Alarm korelasyonu saldırı tespit sistemi alarmları arasındaki ilişkileri ortaya çıkarmaya odaklanmaktadır. Saldırı alarmları için korelasyon teknikleri alarmları ve saldırı senaryolarını kullanıcıları için anlamlı ve kolay anlaşılır hale getirmektedir. Alarm korelasyonunun doğru çalıştığına emin olmak için bu tez çalışması saldırı senaryolarını kullanıp, üretilen alarmları saldırı önkoşul ve sonuçlarına dayanarak korelasyona tabi tutar. Deney ortamının mimarisi çeşitli saldırıların önkoşul ve sonuçlarına dayanarak, alarmları bir önceki alarmın sonucu ve bir sonraki alarmın da nedeni olarak belirleyip; işletim sistemi kayıtları ile eşleştirmektedir. Sonuç olarak ele alınan yöntemin kesinliği ve faydaları, saldırı tespit sistemi alarmlarının işletim sistemi kayıtları ile korelasyonu sonucunda gösterilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Intrusion detection techniques can be roughly classified into two categories: misuse detection and anomaly detection. To perform misuse detection, a repository of all known attack patterns is necessary. Misuse detection systems examine security events to see whether they match these attack patterns. If they are, the corresponding security events are flagged as attacks. To perform anomaly detection, a repository of normal behaviors for each entity is usually necessary. Anomaly detection systems monitor each entity's activity, and once they find an entity's behavior significantly deviates from the normal profile, an alert is generated (Xu and Ning 2006).

Many IDSs have been designed, implemented and deployed into networks (Xu and Ning 2006). They are a line of defense to protect digital assets. Though many novel designs and improvements have been proposed, at present, IDSs still suffer from a few drawbacks:

- IDSs may flag thousands of alerts every day, thus overwhelming security officers.
- Among all the alerts reported by IDSs, false alerts (false positives) are mixed with true alerts. In addition, it is very possible that a large percentage of alerts are false alerts. This may make the alert investigation very challenging.
- At present, IDSs cannot guarantee the detection of all attacks. In other words, they may miss some attacks, which could be critical for security officers to understand the current security threats.

These limitations of IDSs make security investigation not only time-consuming, but also error-prone. It is very challenging for security officers to fully learn the security threats in their networks as well as over the Internet.

To address these challenges, various alert correlation techniques have been proposed in recent years. To help us better understand these alert correlation methods which roughly classified into four categories:

1. The approaches based on similarity between alert attributes: These approaches can group alerts through computing attribute similarity values.
2. The techniques based on predefined attack scenarios: These techniques construct attack scenarios through matching alerts to predefined scenario templates.
3. The methods based on prerequisites and consequences of attacks: These methods build attack scenarios through matching the consequences of earlier attacks with the prerequisite of later attacks.
4. The approaches based on multiple information sources: These approaches provide frameworks to model different types of information and may further perform reasoning based on IDS alerts and other information.

Several researchers investigated integrating additional information sources into alert correlation to improve its quality (Zhai, et al. 2006). These approaches can improve the performance of correlation by integrating different sources of security-related information. However, the correlation results are still not yet satisfactory (Zhai, et al. 2006). This thesis purposes to harness OS-level event logging and dependency tracking to improve the accuracy of alert correlation. It tracks dependency causing events such as process forking and file operations in the system event log, and spans up a tree of system objects connected by these events from the target object.

The Coral8 complex event processing engine is integrated with IDS alerts and OS-level dependency tracking logs via Coral8 input adapters in comma separated value files format. This integration method based on the following observations: Firstly, most attacks have corresponding operations on specific OS-level objects. Secondly, other than a few exceptions, if one attack prepares for another, the later attack's corresponding operations would be dependent on the earlier one's. Coral8 performs user-defined SQL based correlation methods which based on the above and produces the correlated alerts to security officers.

The reminder of this chapter is organized as follows: In chapter 2, intrusion detection systems (IDSs) are defined, the classification of IDSs is provided. Also it contains the information about the detection methods of IDSs and their response after detecting an intrusion. Chapter 3 defines Coral8 complex event processing tool and their properties and includes some Coral8 queries to execute some correlation examples for some special scenarios. Chapter 4 discusses intrusion alert correlation and its techniques with some examples. Chapter 5 surveys an alert correlation scenario for the technique of prerequisite and consequence approaches. A custom structure has also been built to investigate the correlation method with some metric values in the alert correlation scenario. In Chapter 6, conclusion is processed along with future work.

CHAPTER 2

INTRUSION DETECTION SYSTEM

The Internet is a global public network. With the growth of the Internet and its potential, there has been subsequent change in business model of organizations across the world. More and more people are getting connected to the Internet every day to take advantage of the new business model popularly known as e-business. Internetwork connectivity has therefore become very critical aspect of today's e-business.

There are two sides of business on the Internet. On one side, the Internet brings in tremendous potential to business in terms of reaching the end users. At the same time it also brings in a lot of risk to the business (Sans 2001). There are both harmless and harmful users on the Internet. While an organization makes its information system available to harmless Internet users, at the same time the information is available to the malicious users as well. Complete physical isolation of the information system from all possible suspicious users would be a simple and effective way of denying possible risks, but it may be unacceptable because the physical isolation may render the information system unable to perform its intended function.

Network security technology is very important to avoid security risks. There are several techniques and tools that are used to provide network security. Network security aims is confidentiality, integrity and availability of data so providing this aim firewalls are one of the well-known tools that are very helpful in network security. Basically, a firewall separates the private network from the external and non-secure environment. A firewall that is located between the private network and Internet enforces a predefined network security policy by controlling network traffic passing over it. A network security policy is composed of a set of rules that defines what types of connections are allowed between internal and external networks. Additionally, a security policy defines the action that should be performed when a violation of policy is detected (Erdogan 2008).

Different organizations across the world deploy firewalls to protect their private network from the public network. But, when it comes to securing a private network from the Internet using firewalls, no network can be hundred percent secured. This is because; the business requires some kind of access to be granted on the internal systems to Internet users. The firewall provides security by allowing only specific services through it. The firewall implements a policy for allowing or disallowing connections based on organizational security policy and business needs. The firewall also protects the organization from malicious attack from the Internet by dropping connections from unknown sources.

2.1. The Definition of an IDS and Its Need

Intrusion detection is the process of monitoring and searching networks of computers and systems for security policy violations (Bace 2000). IDSs are software and hardware products that automate this monitoring and analysis process. An IDS inspects all inbound and outbound network activity, system logs and events, and identifies suspicious patterns or events that may indicate a network or system attack from someone attempting to break into or compromise a system (Webopedia 2002).

The question is, where does an IDS fit in the design. Theoretically, IDSs work like a burglar alarm. For example, the lock system in a car protects the car from theft. But if somebody breaks the lock system and tries to steal the car, it is the burglar alarm that detects that the lock has been broken and alerts the owner by raising an alarm.

IDS complements the firewall security in a similar way. The firewall protects an organization from malicious attacks from the Internet and the IDS detects if someone tries to break in through the firewall or manages to break in the firewall security and tries to have access on any system in the trusted side and alerts the system administrator in case there is a breach in security. It is also assumed that most IDSs attempt to detect suspected intrusion and they alert the system administrator that an attack may be taking place so that they can respond accordingly.

IDSs have gained acceptance as a necessary addition to every organization's security infrastructure. The organizations have several compelling reasons to acquire and use IDSs. Some of them are listed below (Bace and Mell 2001):

- To prevent problematic behaviors by increasing the perceived risk of discovery and punishment for those who would attack or otherwise abuse the system,

- To detect attacks and other security violations that are not prevented by other security measures,
- To detect and deal with the preambles to attacks (commonly experienced as network probes and other reconnaissance activities),
- To document the existing threat to an organization,
- To act as quality control tool for security design and administration, especially for large and complex enterprises,
- To provide useful information about intrusions that take place, allowing detailed analysis, recovery, and correction of causative factors.

Therefore, an IDS is a security system that monitors computer systems and network traffic and analyzes that traffic for possible hostile attacks originating from outside the organization and also for system misuse or attacks originating from inside the organization.

2.2. IDS Terminology

Alert/alarm: A signal suggesting a system has been or is being attacked.

Undetected bad traffic (false negative): Failure to identify malicious traffic as an attack. This is the worst thing that can happen, because it means the IDS failed to do its job. Failing to detect an attack can occur when an IDS does not have adequate or comprehensive intrusion detection mechanisms in place. It also occurs when new attacks are created and then missed by poorly implemented detection mechanisms (OneSecure 2001). While it is virtually impossible to detect every attack, the goal of any system should be to minimize the number of undetected attacks.

Detected bad traffic (true negative): Identifying "real attacks" as an attack. This is the ideal result of an IDS. The ability to detect bad traffic with speed and reliability is referred to as intrusion detection accuracy. All other functions of the system hinge on this capability (OneSecure 2001). The more accurate the system, the more trusted its abilities. A system must have proven accuracy before enabling it to take the necessary actions (such as dropping the connection) to the secure network.

Identifying good traffic as an attack (false positive – false alarm): False alarm or false positive. This is the most troublesome and time-consuming aspect of IDS solutions. It occurs when the IDS sees something in legitimate and benign traffic that makes it believe there is an attack (OneSecure 2001). It is detrimental because each and every alarm needs to be investigated in order to determine whether an attack was successful and assess any resulting damage. Every moment spent investigating a false positive reduces the time available to investigate real threats. The result is that false positives can erode trust in the product; sometimes causing real attack alarms to be overlooked (the "crying wolf" effect). Most IDSs can be tuned to try to reduce the occurrence of false positives, however, the tuning process is often long and involved, sometimes taking weeks to accomplish (OneSecure 2001). In addition, because of the management design of current IDSs, tuning is often an all or nothing approach. This means that security managers must choose whether or not to look for a certain attack. If, in the interest of reducing false positives, the detection of certain attacks is turned completely "off", those attacks will be able to go by the IDS completely undetected.

Nevertheless, false positives are not the result of poor software design by IDS vendors. As Stefan Axelsson demonstrated in his 1999 ACM presentation (Axelsson 1999), there are some fundamental mathematical constraints that make false positives endemic to the whole paradigm of real-time signature (pattern) recognition. Deviations from baseline norms can be caused by a variety of factors, many of them innocuous. So, false positives are inherently part of signature-based intrusion detection schemes or any other type of anomaly detection system.

Identifying good traffic as good traffic (true positive): An ideal result of intrusion detection mechanisms, identifying good traffic for what it is good traffic (OneSecure 2001).

Noise: Data or interference that can trigger a false positive.

Site policy: Guidelines within an organization that control the rules and configurations of an IDS.

Site policy awareness: The ability an IDS has to dynamically change its rules and configurations in response to changing environmental activity.

Confidence value: A value an organization places on an IDS based on past performance and analysis to help determine its ability to effectively identify an attack.

Alarm filtering: The process of categorizing attack alerts produced from an IDS in order to distinguish false positives from actual attacks (Whitman, et al. 2009).

2.3. Types of IDSs

There exist various IDS products in the market today. These products are categorized in several ways according to their different characteristics (Debar, et al. 1999):

Detection technique: Behavior based (misuse detection) and knowledge based (anomaly detection).

Audit source location: Host log files (host based) and network packets (network based).

Behavior on detection: Passive and active.

2.3.1. Detection Technique

Intrusion detection is a set of techniques that are used to detect suspicious activity both at the network and host level (Rafeeq 2003). The detection technique describes the characteristics of the analyzer (Debar, et al. 1999). When the IDS uses information about the normal behavior of the system it monitors and it is qualified as misuse detection. When the IDS uses information about the attacks, it is qualified as anomaly detection.

Misuse detection IDS: Knowledge-based intrusion detection techniques apply the knowledge accumulated about specific attacks and system vulnerabilities. The IDS contains information about these vulnerabilities and looks for attempts to exploit them. When such an attempt is detected, an alarm is triggered (Debar, et al. 1999). In other words, any action that is not explicitly recognized as an attack is considered acceptable. Essentially, the IDS looks for a specific attack that has already been documented. Like a virus detection system, misuse detection software is only as good as the database of attack signatures that it uses to compare packet against. Therefore, the accuracy of misuse IDS is considered good. However, their completeness requires that their knowledge of attacks be updated regularly.

Misuse detection provides various benefits. One of the benefits is that the signature definitions are modeled on known intrusive activity. Furthermore, the user can examine the signature database and quickly determine which intrusive activity the misuse detection system is programmed to alert on. Another benefit is that the misuse detection system begins protecting the network immediately upon installation. One final benefit is that the system is easy to understand. When an alarm fires, the user can relate this directly to a specific type of activity occurring on the network.

Along with the numerous benefits, misuse detection systems also have their share of drawbacks. One of the biggest problems is maintaining state information for signatures in which the intrusive activity encompasses multiple discrete events (that is, the complete attack signature occurs in multiple packets on the network) (Carter 2002). Another drawback is that the misuse detection system must have a signature defined for all of the possible attacks that an attacker may launch against the network. This leads to the necessity for frequent signature updates to keep the signature database of the misuse detection system up-to-date. One final problem with misuse detection systems is that someone may set up the misuse detection system in their lab and intentionally try to find ways to launch attacks that bypass detection by the misuse detection system.

Anomaly detection IDS: Anomaly detection techniques assume that an intrusion can be detected by observing a deviation from normal or expected behavior of the system or the users (Debar, et al. 1999). The model of normal or valid behavior is extracted from reference information collected by various means. The security manager defines the baseline or normal state of the network's traffic load, breakdown, protocol, and typical packet size. The IDS later compares this model with the current activity. If a deviation is observed, an alarm is generated. In other words, anything that does not correspond to a previously learned behavior is considered intrusive. Therefore, the IDS might be complete, but its accuracy is a difficult issue. The anomaly detection technique is as good as its normal model definition.

Anomaly detection systems offer several benefits. First, they can detect insider attacks or account theft very easily. If a real user or someone using a stolen account starts performing actions that are outside the normal user profile, it generates an alarm. Second, because the system is based on customized profiles, it is very difficult for an attacker to know with certainty what activity he can do without setting off an alarm. Probably the largest benefit, however, is that intrusive activity is not based on a specific traffic that represents known intrusive activity as in a misuse IDS. An anomaly detection system can potentially detect an attack the first time it is used (Debar, et al. 1999). The intrusive activity generates an alarm because it deviates from normal activity, not because someone configured the system to look for a specific stream of traffic.

Like every IDS, anomaly detection systems also suffer from several drawbacks. The first obvious drawback is that the system must be trained to create the appropriate user profiles. During the training period to define what normal traffic looks like on the network, the network is not protected from attack (Debar, et al. 1999). Maintenance of the profiles can also become time-consuming. Nevertheless, the biggest drawback to anomaly detection is probably the complexity of the system and the difficulty of associating an alarm with the specific event that triggered the alarm. Furthermore, there is no guarantee that a specific attack will be generate an alarm. If the intrusive activity is too close to normal user activity, then the attack will go unnoticed. It is also difficult to know which attacks will set off alarms unless actually test the attacks against the network using various user profiles (Carter 2002).

2.3.2. Audit Source Location

The audit source location distinguishes among IDSs based on the kind of input information they analyze. This input information can be audit trails, system logs or network packets.

Host based IDS (HIDS): In a HIDS, activities on each individual computer or host are examined. Host audit sources are the only way to gather information about the activities of the users of a given machine. On the other hand, they are also vulnerable to alterations in the case of a successful attack. This creates an important real-time constraint on host-based IDSs, which have to process the audit trail and generate alarms before an attacker taking over the machine can subvert either the audit trail or the IDS itself. It is advisable to place HIDSs on all mission-critical systems, even those that should not, in theory, allow external access.

The major benefit of a host based monitoring system verifies success or failure of an attack. Since a HIDS uses system logs containing events that have actually occurred, they can determine whether an attack occurred or not with greater accuracy and fewer false positives than a network based system. Second, a host based IDS sensor monitors user and file access activity including file accesses, changes to file permissions, attempts to install new executables etc. A host based IDS sensor can also monitor all user logon and logoff activity, user activities while connected to the network, file system changes, activities that are normally executed only by an administrator. Operating systems log any event where user accounts are added, deleted or modified. The HIDS can detect an improper change as soon as it is executed. Third, host based systems can detect attacks that network based system sensors fail to detect. For example, if an unauthorized user makes changes to system files from the system console, this kind of attack goes unnoticed by the network sensors. Although HIDS does not offer true real-time response, it can come extremely close if implemented correctly. Many current host-based systems receive an interrupt from the operating system when there is a new log file entry. This new entry can be processed immediately, significantly reducing the time between recognition and response (SANS Institute 2001). There remains a delay between when the operating system records the event and the HIDS recognizes it, but in many intruders can be detected and stopped before damage is done.

There are two major drawbacks to a host based IDS. The first difficulty is having incomplete network picture and necessity to support multiple operating systems. The other difficulty lies in the fact that a host based IDS needs to run on every system in the network. This requires verifying support for all of the different operating systems that used on the network.

Network based IDS (NIDS): NIDS analyze the individual packets flowing through the network. NIDSs often consist of a set of single-purpose sensors placed at various points in a network. These units monitor network traffic, performing local analysis of that traffic and reporting attacks to a central management console. NIDSs analyze traffic moving across the network in much greater than a firewall. Therefore, NIDSs can detect malicious packets that are designed to be overlooked by a firewall's simplistic filtering rules. NIDSs also watch for attacks that originate from within the network. That is why, they are complements for firewalls.

A network based monitoring system has the benefit of seeing and coordinating attacks that are occurring across an entire network very easily. Seeing the attacks against the network gives a clear indication of the extent to which the network is being attacked. Furthermore, because the monitoring system is only examining traffic from the network, it does not have to support every type of operating system that used on the network. NIDSs use live network traffic for real-time attack detection. Therefore, an attacker cannot remove the evidence (SANS Institute 2001). Many hackers understand audit logs, they know how to manipulate these files to cover their tracks, frustrating host based systems that need this information to detect an intrusion. NIDSs add valuable data for determining malicious intent. A network based IDS placed outside of a firewall can detect attacks intended for resources behind the firewall, even though the firewall may be rejecting these attempts. Host based systems do not see rejected attacks that never hit a host inside the firewall. This lost information can be critical in evaluating and refining security policies.

One disadvantage of NIDS is that encryption of the network traffic stream can essentially blind the NIDS. Reconstructing fragmented traffic can also be a difficult problem to solve. Probably the biggest drawback to network based monitoring, however, is that as networks become increasingly larger (with respect to bandwidth), it becomes more difficult to place a network based IDS at a single location on the network and capture all traffic successfully (Carter 2002). This then requires the utilization of more sensors throughout the network, which increases the costs of the IDS.

As a final consequence of audit source location categorized IDSs, network based IDS is normally more effective than host based IDS due to the fact that a single system can monitor multiple systems and resources.

2.3.3. Behavior on Detection

Behavior on detection describes the response of the IDS to attacks. When it actively reacts to the attack by taking either corrective closing holes or proactive logging out possible attackers, closing down services actions, then the IDS is said to be active (Debar, et al. 1999). If the IDS merely generates alarms including paging, etc, it is said to be passive.

Passive system: Many IDSs merely log the intrusion and the security administrator is notified by email or pager etc. This is known as passive response intrusion detection, as it does not actively attempts to stop the intrusion. Instead, a system administrator or someone else will have to respond to the alarm, take appropriate action to halt the attack, and possibly identify the intruder. Modern IDSs offer a wide range of options to send notifications of intrusions, including pager, cell phone, email, SNMP trap messages, or simply a message box on the administrator's PC. It is important to make sure that the notifications are send in a secure manner to prevent the attacker from intercepting or altering them (Debar, et al. 1999).

Active system: Active response IDSs automatically take action in response to a detected intrusion. The exact action differs per product and depends on the severity and type of attack. A common active response is increasing the sensitivity level of the IDS to collect additional information about the attack and the attacker. Another possible active response is making changes to the configuration of systems or network devices such as routers and firewalls to stop the intrusion and block the attacker. This could involve blocking the source address of the attacker, restarting a server or service, closing connections or ports, and resetting TCP sessions (Debar, et al. 1999).

2.4. Details of Detection Techniques

As mentioned earlier, there are several types of IDSs which are classified into their detection techniques, audit source location, and behavior on detection. In 2.3.1 Detection Technique, the IDSs categorized as misuse and anomaly detection system according to their using detection techniques. Rafeeq also commented the IDSs working principle that are used to detect suspicious activity at the network and host level by a set of detection techniques (Rafeeq 2003). At this point, the details of detection techniques are discussed below to have an idea about the working principle of IDSs.

The ideal performance on an IDS is to identify as many attacks as possible and limit the number of false alarms (OneSecure 2001). Unfortunately, there is no single detection mechanism available today and an IDS can deploy to detect every type of attack. As a result, only a system that combines a variety of technologies to detect different types of attacks can get the job done. The most common way to implement a combination of detection techniques was to purchase two or more products and run them together. The two most commonly available network intrusion detection mechanisms on the market today are signature based and statistical protocol anomaly based detection (OneSecure 2001). Signature based systems look for known attack patterns (signatures) in traffic. Signature detection finds attacks for which a signature written, but it is unable to detect new attacks or many very complicated attacks. Protocol anomaly detection based systems, on the other hand, do a good job of detecting some of the unknown attacks, but are unable to identify attacks that operate without violating any protocols. All IDSs use the following detection techniques in the market:

- Intrusion detection using signature,
- Intrusion detection using protocol anomalies,
- Intrusion detection using stateful signatures.

2.4.1. Intrusion Detection Using Signature

The evaluation of NIDSs started with the implementation of a non-intrusive packet monitor, called a sniffer because of its ability to “sniff” the packets on the network. Intrusion detection vendors applied to the packet monitoring concept to build systems that performed packet signature detection (OneSecure 2001). Signature based detectors analyze system activity, looking for events or sets of events that match a predefined pattern of events that describe a known attack. They compare events and packets with signatures stored in their database and find out the matching ones. The most common form of signature based detection used in commercial products specifies each pattern of events corresponding to an attack as a separate signature (Bace and Mell 2001). Signature detection finds attacks for which a signature is written and it is very effective at detecting attacks without generating an overwhelming number of false positives.

However, there are many drawbacks to signature based approach to intrusion detection, especially if effort is placed entirely on building up a large repository of attack signatures, without regard to how the traffic is reassembled, decoded, normalized and analyzed. It is a problem when information is transmitted over the network; the information is split into numbered TCP segments that are sent as packets. In an ideal world, the packets would be transmitted in sequence and without loss. But, unfortunately, that is not the case. When a message is actually transmitted, the network will deliver the packets randomly (out of sequence) or as even smaller pieces of data (called fragments), which are broken down by networking devices, such as routers, to facilitate ease of transmission. Even worse, for whatever reason, packets can get “lost” or can be duplicated (OneSecure 2001).

Another disadvantage of this mechanism is that it is unable to detect new attacks. Therefore, it must be constantly updated with signatures of new attacks. However, signature updates may sometimes result in inability of detecting previously detected attacks. Lastly, signature based detectors cannot detect many very complicated attacks.

2.4.2. Intrusion Detection Using Protocol Anomalies

Protocol anomaly detection, which is sometimes called protocol analysis, is the ability to analyze packet flows (the uni-directional communication between two systems) to identify irregularities in the generally accepted Internet rules of communication (OneSecure 2001). Those rules are defined by open protocols and published standards (RFC-Request for Comment), as well as vendor defined specifications for communication between networked devices.

Protocol anomaly detection attempts to save time by first identifying the protocol and looking specifically for anomalous activity or attack patterns relevant to that protocol. By doing so, it can do a much more targeted and thus more effective search (TopLayerNetworks 2002). In other words, protocol anomaly detection identifies traffic that does not meet specifications or violets the relevant standards. Once an irregularity is identified, it can be used to make network security decisions. This is very effective in detecting suspicious activity such as a buffer overflow attack.

The advantages of protocol anomaly detection are that it can detect (OneSecure 2001):

- Unknown and new attacks, based on the fact that these attacks deviate from protocol standards,
- Attacks that bypass systems that implement other detection techniques,
- Slightly modified attacks that change the format of known attack patterns, with no effect on the strength of the attack to evade signature based systems

An example of detecting the FTP bounce attack using protocol anomaly is described below (OneSecure 2001):

The FTP bounce attack exploits a design flaw in the specifications of FTP. To download or upload files, a user must first connect to an FTP server. When this happens, the server requires the client to send the IP address and port number to which the file should be sent to or taken from. This is done via a mechanism called a “Port Command”. However, the port command specification does not limit the IP address to the user’s address. Because of this, an attacker can tell the FTP server to open a connection to an IP address that is different from the user’s address and then use the open port to transfer files containing a Trojan through the FTP server onto the victim.

Since protocol anomaly detection is designed to look network relationships and determine whether both sides are acting within the normal specifications, IDS can parse the requests in a port command whenever seen and compare it to the IP address from which the port command arrived. If they do not match, the IDS needs to send an alarm.

Protocol anomaly based systems do a good job of detecting some of the unknown attacks like the one described. But their main drawback is that they are unable to identify attacks that operate without violating any protocols, such as Trojan or Worm. These attacks install and open up a backdoor on a network resource. This backdoor lays inactive until the attacker activates it and takes control over the resource. Besides that, protocol anomaly detection systems usually produce a large number of false alarms due to the unpredictable behaviors of users and networks.

2.4.3. Intrusion Detection Using Stateful Signatures

Another alternative approach that overcomes the accuracy deficiencies of packet signature detection is stateful signature detection. This advanced detection mechanism identifies attack patterns by utilizing both stateful inspection and protocol analysis, which is performed as part of protocol anomaly detection (OneSecure 2001). As a result, stateful signature detection systems understand the context of each data byte and the state of the client and server at the time of transmission. This means that stateful signatures can be compared to only relevant data bytes, according to the communication state to which each signature is relevant.

Stateful signatures detection mechanism looks at the context and the placement of signature to make smarter decisions about whether it represents an attack. The IDS keeps track of the state of the connection with the outside entity, and considers the broader context of all the transactions initiated during the connection (TopLayerNetworks 2002). In other words, stateful signatures only look for an attack in the state of the communication where that attack can cause damage, thus significantly improving performance and reducing false positives.

The drawback of this system is the same as the signature based detectors; it catches only known attacks and only if the signature database is constantly updated. Unfortunately, the people out there trying to exploit networks are neither lazy nor stupid; they constantly unleash new variations and new attacks. With each new attack, new signatures have to be taught to the IDS. Over time, this led to a need for IDS products to hold literally thousands of attack signatures and constantly scan for them all (TopLayerNetworks 2002). In addition, while the signature database being constantly updated, attackers know exactly how the IDS products work, so they continuously make slight alterations to elude detection. Thus, a more intelligent signature mechanism is needed to bring about more accurate results.

2.5. Response Options for IDSs

Today, IDS products support a wide range of response options, often categorized as active responses, passive responses, or some mixture of both.

2.5.1. Active Responses

Active IDS responses are automated actions taken when certain types of intrusions are detected. There are three categories of active responses:

Collect additional information: The most innocuous, but at times most productive, active response is to collect additional information about a suspected attack. This might involve increasing the level of sensitivity of information sources (for instance turning up the number of events logged by an operating system audit trail, or increasing the sensitivity of a network monitor to capture all packets, not just those targeting a particular port or target system). Collecting additional information is helpful for several reasons. The additional information collected can help resolve the detection of the attack. This option help also allows the organization to gather information that can be used to support investigation and apprehension of the attacker and to support criminal and civil legal remedies (Bace and Mell 2001).

Change the environment: Another active response is to halt an attack in progress and then block subsequent access by the attacker. Typically, IDSs do not have the ability to block a specific person’s access, but instead block IP addresses from which the attacker appears to be coming. It is very difficult to block a determined and knowledgeable attacker, but IDSs can often deter expert attackers or stop novice attackers by taking the following actions (Bace and Mell 2001):

- Injecting TCP reset packets into the attacker's connection to the victim system, thereby terminating the connection,
- Reconfiguring routers and firewalls to block packets from the attacker's apparent location (IP Address or site),
- Reconfiguring routers and firewalls to block the network ports, protocols, or services being used by an attacker, and
- In extreme situations, reconfiguring routers and firewalls to break all connections that use certain network interfaces.

Take action against the intruder: The most aggressive form of this response involves launching attacks against or attempting to actively gain information about the attacker's host or site. The primary reason for approaching this option with a great deal of caution is that it may be illegal. Furthermore, since many attackers use false network addresses when attacking systems, this action has a risk of causing damage to innocent Internet sites and users. Finally, strike back can escalate the attack, provoking an attacker who originally intended only to browse a site to take more aggressive action (Bace and Mell 2001).

2.5.2. Passive Responses

Passive IDS responses provide information to system users, relying on them to take necessary action based on that information. Many IDS products rely solely on passive responses.

Alarms and notifications: Alarms and notifications are generated by IDSs to inform users when attacks are detected. The most common form of alarm is an onscreen alert or popup window. This is displayed on the IDS console or on other systems as specified by the user during the configuration of the IDS. The information provided in the alarm message varies widely, ranging from a notification that an intrusion has taken place to extremely detailed messages outlining the IP addresses of the source and target of the attack, the specific attack tool used to gain access, and the outcome of the attack (Bace and Mell 2001). Another set of options is to configure the IDSs so that they send alert messages to cellular phones and pagers carried by incident response teams or security managers.

SNMP traps and plug-ins: Some commercial IDSs use SNMP traps and messages to send alarms to central network management consoles. This provides the ability to adapt the entire network infrastructure to respond to a detected attack, the ability to shift the processing load, associated with an active response, to a system other than the one being targeted by the attack, and the ability to use communication channels (Bace and Mell 2001).

2.6. Snort

Snort is a free network packet analysis tool written by Martin Roesch in 1998 for his personal use. Snort is now one of the most widely used security tools in the open-source world. Snort is perfect for detecting DoS, fragmentation, known buffer overflows, scanning worms and scripts, cross-site, and injection attacks. Snort can be connected to external databases to ease packet and event logging and analysis, link it to reporting tools, manage it through centralized consoles, and enable it to participate in many types of alert systems.

Snort’s packet capturing routine allows to capture all packets headed to and from a honey pot. In order for Snort to detect malicious packets reliably, it needs to inspect all packets. Captured packets are passed to the packet-decode engine component. The packet-decode engine separates packets into their higher and lower layer components. This is because different attacks occur at different levels, and Snort does not need to apply all its rules to all layers for all attacks. For example, if a buffer overflow will never occur at the frame layer, there is no need to examine that layer for buffer overflow attacks.

Snort works by using the libcap capture API (as does WinPcap, tcpdump, and Ethereal) to capture packets for examination and logging. A packet-decode engine examines the packets, hands them to preprocessor plug-ins, and then on the signature-detection engine, which produces actions (allow, deny, and log) and outputs (alert, log, and so on). See Figure 2.1. for an illustration of the Snort packet pathway (Grimes 2005).

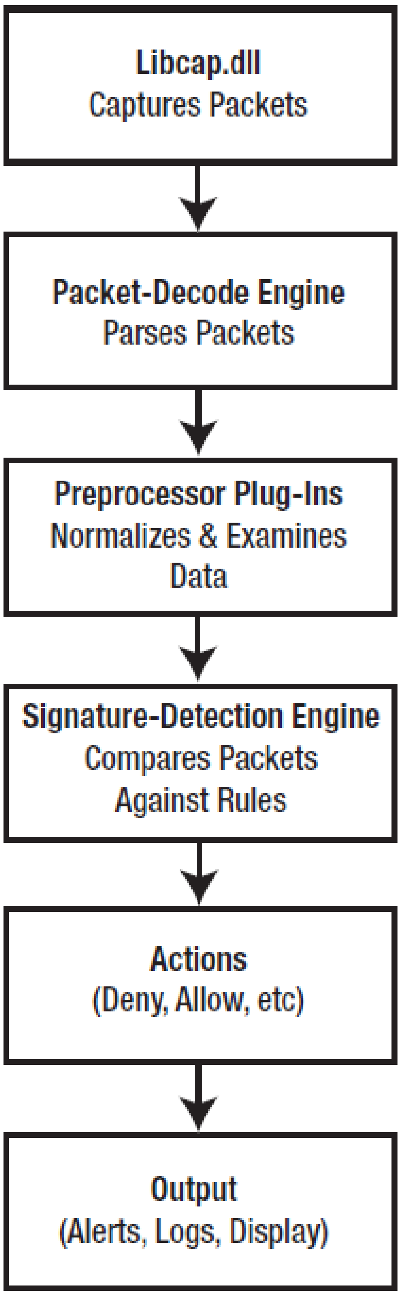


Figure 2.1. Snort packet pathway.

2.6.1. Snort Modes

Snort is often described as a virus scanner for network packets, but it has three modes: packet dump, packet logger, and network IDS. Packet dump mode captures packets and displays them to the screen (the console). Packet logger mode writes packets to a physical file. Snort can be used in both modes at the same time, mimicking a command-line network protocol analyzer. In network IDS mode, Snort will analyze grabbed packets for malicious content. All three modes can be used together in different combinations (Grimes 2005).

Snort is not an install and execute program. It takes a fairly good understanding of what it does and how it operates to configure and use properly. Besides making sure that the Snort host computer is hardened against detection and attack, honey pot administrators should use the following steps when configuring it for the first time:

1. Decide what Snort to do.
2. Configure the Snort configuration file.
3. Configure rule sets.
4. Test the Snort configuration.
5. Create and use a Snort.bat file.

The first decision is which mode to put Snort in. Snort can be configured to be a packet sniffer or network IDS.

Snort packet dump mode: To enable Snort to sniff packets to the console, Snort.exe can be executed with one or more of three command-line switches:

- **-v:** This puts Snort in packet dump (sniffer) mode. With just the –v option enabled, only the network and transport layer header information will be captured and displayed.
- **-e:** If the –e parameter enabled, the layer 2 frame headers added to the information collected. To most people, this means that getting ARP information. This information can be useful to ensure Snort is correctly emulating the MAC addresses in response to queries to its virtual IP addresses.
- **-d:** The –d parameter will display all payload data information from the transport layer.

In packet dump mode with all three parameters set, the following fields (in order) are captured on TCP packets (Grimes 2005):

- Date
- Time
- Source MAC address
- Destination MAC address
- IP protocol ID number (transport protocol type)
- Total packet length (in hex)
- Source IP address
- Source port number
- Destination IP address
- Destination port number
- Transport protocol name
- TTL setting
- Type-of-service setting
- IP packet ID
- IP header length
- Packet length
- TCP flags (represented by their first letter of their name: S for SYN, P for PSH, A for ACK, and so on)
- Sequence number
- Acknowledgement number
- Window size
- Transport protocol header size
- Payload data

Snort network IDS mode: Although Snort can be used purely as a packet sniffer, it is not primary reason to have it as a part of the network security system. Snort's specialty is detecting malicious content in sniffed packets. To put Snort into its network IDS mode, the -c command-line parameter is needed to use to point to the Snort configuration file.

The configuration file contains commands to define the following items during Snort's runtime:

- System variables

- Which preprocessor plug-ins to load
- Which rule sets to load
- Which output plug-ins to use

Snort comes with a default configuration file called Snort.conf, in the etc directory. The different sections are well differentiated in the configuration file. When Snort is executed with the configuration file, Snort will run with those options during its execution (Grimes 2005).

2.6.2. Components of Snort

Snort is logically divided into multiple components. These components work together to detect attacks and to generate output in a required format from the detection system. Snort consists of the following major components (Rafeeq 2003):

- Packet Decoder
- Preprocessors
- Detection Engine
- Logging and Alerting System
- Output Modules

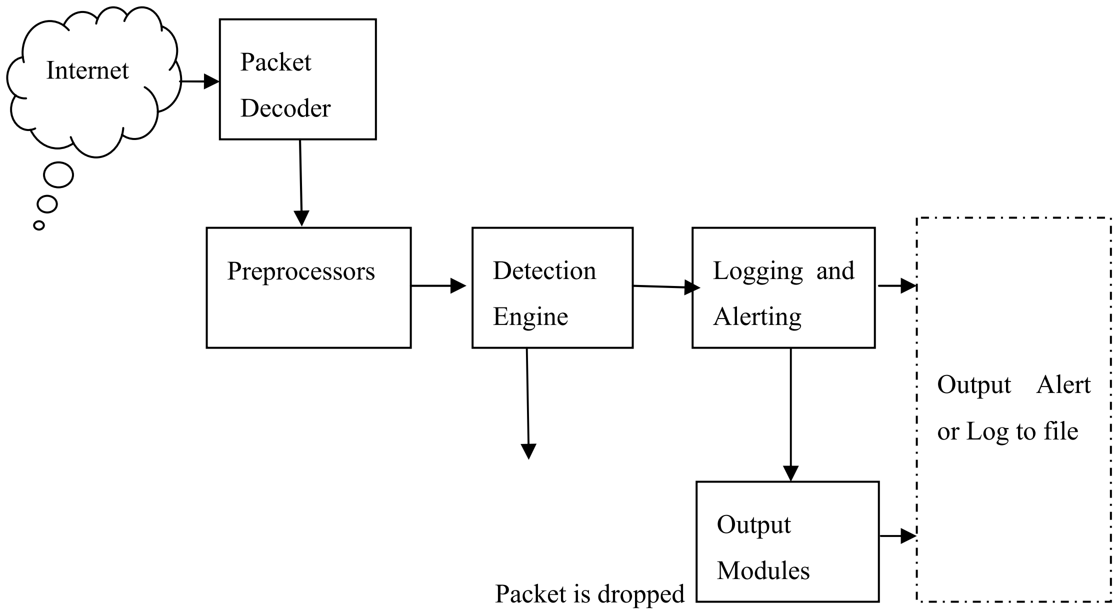


Figure 2.2. Components of Snort
(Source: Rafeeq 2003).

Packet decoder: The packet decoder takes packets from different types of network interfaces and prepares the packets to be preprocessed or to be sent to the detection engine. The interfaces may be Ethernet, SLIP, PPP, WiFi and so on. It parses the packet and decodes the string of bytes into a packet structure that is formed of protocol fields and flags. Each subroutine in the decoder imposes order on the packet data by overlaying data structures on the raw network traffic. These decoding routines are called in order through the protocol stack, from the data link layer up through the transport layer, finally ending at the application layer. During this decoding process, it validates the length and checksum fields. It then forwards the valid packets to the preprocessors (Rafeeq 2003).

Preprocessors: According to the Snort configuration that introduced by Snort Users Manual when a packet is received by Snort, it may not be ready for processing by the main Snort detection engine and application of Snort rules. For example, a packet may be fragmented. Before searching a string within the packet or determine its exact size, defragmentation is required by assembling all fragments of the data packet. On IDS, before applying any rules or try to find a signature, the packets have to be reassembled (Rafeeq 2003). The job of a preprocessor is to make a packet suitable for the detection engine to apply different rules to it. In addition, some preprocessors are used for other tasks such as detection of anomalies and obvious errors in data packets, decoding of HTTP URI. All enabled preprocessors operate on each packet. There is no way to bypass some of the preprocessors based upon some criteria.

Detection engine: The detection engine is the most important part of Snort. Its responsibility is to detect if any intrusion activity exists in a packet. The detection engine employs Snort rules for this purpose (Snort 2009). The rules are read into internal data structures or chains where they are matched against all packets. Snort organizes parts of packets to make the job of matching rules against them faster. It maintains detection rules in a two dimensional linked list of what are termed Chain Headers and Chain Options. The commonalities are condensed into a single Chain Headers and individual detection signatures are kept in Chain Option structures. If a packet matches any rule, appropriate action is taken; otherwise the packet is dropped. Appropriate actions may be logging the packet or generating alerts.

Logging and alerting system: This system is responsible from the generation of alerts and logging of packets and messages. Depending upon what the detection engine finds inside a packet, the packet may be used to log the activity or generate an alert. All of the log files are stored under a preconfigured location by default. This location can be configured using command line options. There are many command line options to modify the type and detail of information that is logged by the logging and alerting system (Snort 2009).

Output modules: Basically, these modules control the type of output generated by the logging and alerting system. Depending on the configuration, output modules can send output messages to a number of other destinations. Commonly used output modules are (Snort 2009):

- The database module is used to store Snort output data in databases, such as MySQL, MSSQL or Oracle,
- The SNMP module can be used to send Snort alert in the form of traps to a management server,
- The Sending Server Message Block (SMB) alerts module can send alerts to Microsoft Windows machines in the form of pop-up SMB alert windows,
- The syslog module logs messages to the syslog utility (using this module logs messages to a centralized logging server),
- XML or CSV modules can be used to save data in XML or comma separated files. The CSV files can then be imported into databases or spreadsheet software for future processing or analysis.

CHAPTER 3

COMPLEX EVENT PROCESSING

Complex event processing (CEP) is primarily an event processing concept that deals with the task of processing multiple events with the goal of identifying the meaningful events within the event cloud. CEP employs techniques such as detection of complex patterns of many events, event correlation and abstraction, event hierarchies, and relationships between events such as causality, membership, and timing, and event driven processes. CEP is to discover information contained in the events happening across all the layers in an organization and then analyze its impact from the macro level as "complex event" and then take subsequent action plan in real time (Wikipedia 2009).

Examples of events include a car, some sensors and various events and reactions. In the situation, the car is moving and the pressure of one of the tires from 45 PSI to 41 PSI in 5 seconds. As the pressure in the tire is reducing, a serious of events containing the tire pressure is generated. In addition, a series of events containing the speed of the car is generated. The car's event processor may detect a situation whereby a loss of tire pressure over relatively long period of time results in the creation of the "lossOfTirePressure" event.

There are many commercial applications of CEP including stock trading, credit card fraud detection, business activity monitoring, and security monitoring. New applications of CEP are emerging as technology vendors find new uses.

Most CEP solutions and concepts can be classified into two main categories. The first one which is a computation oriented CEP solution is focused on executing on-line algorithms as a response to event data entering the system is to continuously calculate an average based in data on the inbound events. The other one is detection oriented CEP is focused on detecting combinations of events called event patterns or situations (Wikipedia 2009).

3.1. Event Driven Systems

An event driven system (Berson 1992) is a system of objects which interact with each other using a message-passing mechanism. This mechanism is controlled by distinct component that is usually called the event dispatcher, and act as an intermediary between objects. The data communicated are called events and they can originate from input devices in an unprocessed form (raw event) or they can be a result of communication between objects. The objects receive events in the form of event messages, typically of a fixed length and made up of an event type identifier and the event parameters. Each object has a designated programming procedure called event procedure that invokes individual procedures called event handlers for each type of event message.

To illustrate how an event driven system works, suppose in a GUI the user clicks the left mouse button in the client area of the window a drawing application. This generates a raw event that contains the mouse position and which mouse buttons were pressed at that moment. The event dispatcher receives the raw event and adds information such as the application it is destined for, creates an event message and places it in a queue for the recipient application to pick it up. The recipient application checks for new messages and finds it. Subsequently, the event procedure is executed and chooses the suitable event handler for the specific mouse event message.

Event driven systems have typically been built around either relational databases or real-time messaging systems, or a combination of both. While these technologies have their advantages, neither is particularly well suited for managing and analyzing events in rapidly changing environments (Coral8 2009):

- Relations database servers can process large amounts of stored data and can analyze the information with relative ease but are not designed to operate in real-time environments, and do not provide an effective way to monitor rapidly changing data.
- Messaging systems permit data to be monitored in real time but are not generally capable of complex computations, correlations, pattern matching or references to historical data.

For these reasons, custom application must often be combined with these technologies to create a viable solution. The use of custom applications to compensate for the limitations of these technologies creates new complications:

- Custom applications become increasingly complex very quickly as an organization's need for progressively more sophisticated analysis grows.
- Custom applications are also costly to modify, and they do not scale well as organizational needs change.

3.2. Real-Time Computing

Real-time computing is the study of hardware and software systems that are subject to a real-time constraint such as operational deadlines from event to system response. By contrast, a non-real-time system is one for which there is no deadline, even if fast response or high performance is desired or preferred. The needs of real-time software are often addressed in the context of real-time operating systems, and synchronous programming languages, which provide frameworks on which to build real-time application software.

A real-time system may be one where its application can be considered (within context) to be mission critical. The anti-lock brakes on a car are a simple example of a real-time computing system; the constraint in this system is the short time in which the brakes must be released to prevent the wheel from locking. Real-time computations can be said to have failed if they are not completed before their deadline, where their deadline is relative to an event. A real-time deadline must be met, regardless of system load (Wikipedia 2009).

Real-time computing is sometimes misunderstood to be high-performance computing, but this is not always the case. For example, a massive supercomputer executing a scientific simulation may offer impressive performance, yet it is not executing a real-time computing. Furthermore, if a network server is highly loaded with network traffic, its response time may be slower but will still succeed. Hence, such a network server would not be considered a real-time system. In a real-time system such as the FTSE 100 Index, a slow-down beyond limits would often be considered catastrophic in its application context. Therefore, the most important requirement of a real-time system is predictability and not performance.

3.3. Coral8 Engine

Coral8 engine is an information processing system designed to run continuous queries. This means that instead of having to submit a query each time to produce results, submit the query to Coral8 server, which runs the query continuously until set it to stop. This gives a power to examine and analyze large volumes of incoming data virtually instantaneously, even if the data is arriving at very high speeds, without having to store the information in a database first.

Today, businesses are operating in environments where the need to monitor events comes at an ever-accelerating pace. These businesses require systems that can process large quantities of fast moving data, monitor events as they occur, detect patterns, and generate needed results immediately. Coral8 engine is a leader in a new generation of tools for processing, analyzing, and managing events in these highly dynamic environments.

Coral8 engine is a powerful new tool in the field of Complex Event Processing (CEP) applications. Coral8 engine can be used to build applications for processing and analyzing large volumes of fast-moving data in highly dynamic environments. Like a real-time messaging system, Coral8 reacts to each new peace of data immediately. Like a relational database system, Coral8 supports the use of a query language similar to SQL to process and analyze incoming data, and to correlate it with historical data. Coral8 also provides sophisticated-matching capabilities without requiring large amounts of custom application code (Coral8 2009).

Coral8 engine includes the following distinguishing features (Coral8 2009):

- A relational model for accessing real-time data streams,
- Queries run continuously, not just when a query submitted,
- A rich language, based on SQL, but adapted for processing real-time event data, and extended to deal with the temporal characteristics of messages,
- The ability to analyze data and event patterns across a wide variety of data sources to locate pertinent information,
- Extensive capabilities for integrating data from external relational databases,

- Adapters that can be used to integrate Coral8 engine with a variety of external systems
- SDKs for a variety of languages (Java, C, C++, Perl, Python, .NET, etc.) that permit customized interfaces and adapters to be easily implemented,
- A modern development environment for developing, monitoring and managing queries,
- Optimized performance that can accommodate hundreds of thousands of messages per second per server, with a processing latency measured in milliseconds,
- Guaranteed delivery, high availability, security, and other important enterprise requirements.

The Coral8 engine consists of the following major components (Morrel and Vidich 2008):

- **Continuous Computing Language (CCL):** CCL is an SQL-like language used for writing continuous queries, which manipulate Coral8 data. CCL is converted into an executable form by the CCL compiler.
- **Coral8 Server:** Coral8 server processes streams of incoming data against registered continuous queries written in CCL.
- **Coral8 Data Model:** The Coral8 data model consists of streams and windows providing a foundation on which CCL queries can process data.
- **Input and Output Adapters:** Adapters translate data from a wide variety of external sources into a format compatible with Coral8's data model, and also translate Coral8 data back into formats that can be sent to external destinations.
- **Coral8 Studio:** Coral8 studio is a GUI, which lets users easily edit CCL queries, view streams and windows, control and monitor Coral8 servers, and debug Coral8 applications in an interactive development environment.

Figure 3.1 shows Coral8 architecture and its major components below:

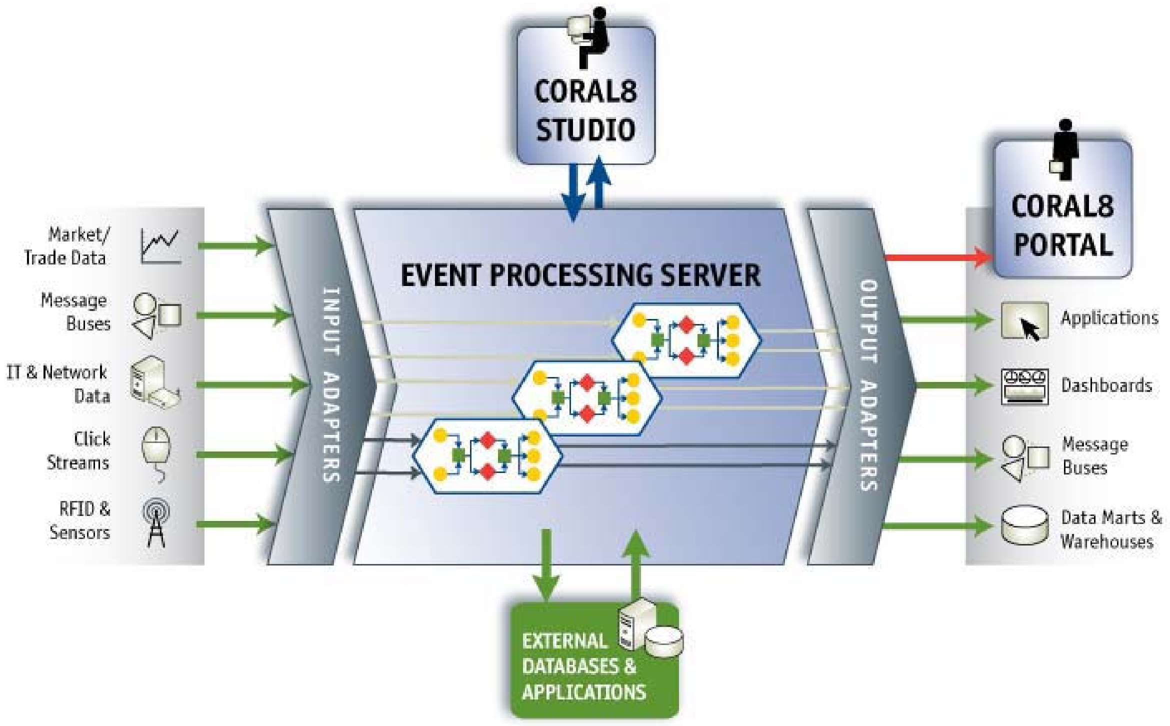


Figure 3.1. Coral8 architecture
(Source: Coral8 2009).

3.4. Using Continuous Computing Language

Data streams are the fundamental way in which data is transmitted throughout Coral8. CCL queries take data from data streams, process that data and send it into other streams. While data streams which are similar to database tables do not retain any state. Therefore, only the most recent row in a data stream is visible to data stream queries.

Time is an essential aspect of Coral8. Data rows arrive continuously and the CCL queries associated with a data stream execute every time a new row arrives. Every row in Coral8 has an associated row timestamp.

External data enters Coral8 engine through input adapters, which translates data from an external format into a format usable by Coral8 engine. A CCL query places its results in a data stream, which may be connected to an output adapter. The output adapter translates the data into the required external format and sends it to the external destination.

A row in a data stream can be processed by a CCL query only when it arrives in the Coral8 engine. However, CCL windows are used to maintain the state of previously arrived rows. The window definition states the policy that determines what rows in a stream should be retained and for how long. Count-based and time-based windows retain rows for a specified maximum number and a specified interval of time respectively. Both count-based and time-based windows may be sliding or jumping.

All data streams and windows must have schemas that specify their structure. A schema specifies the number and names of the columns in the stream or window and indicates each column's datatype. The following datatypes are available in Coral8: Boolean, Integer, Long, Float, String, Timestamp, Interval, XML, Blob and so on.

Data streams and their schemas, adapters, windows, and query modules have discussed above. All of the basic steps listed below must be taken before a CCL query can be successfully executed (Coral8 2009):

1. Create or open a query module,
2. Add uniquely named data streams to the module,
3. Create a schema, or assign an existing schema, for each stream,
4. Add adapters to streams that either subscribe or publish to external sources,
5. Write CCL queries and run the query module.

A basic understanding of Coral8 components and working with CCL queries shown in Figure 3.2 (Coral8 2009):

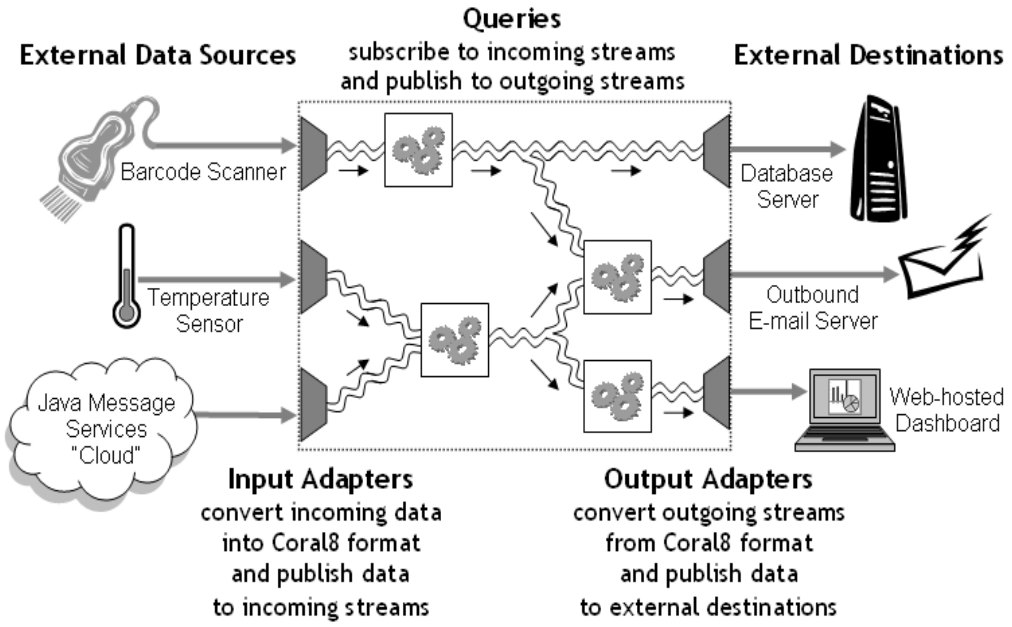

Queries
subscribe to incoming streams
and publish to outgoing streams
External Data Sources
External Destinations
Barcode Scanner
Temperature Sensor
Java Message Services "Cloud"
Database Server
Outbound E-mail Server
Web-hosted Dashboard
Input Adapters
convert incoming data into Coral8 format and publish data to incoming streams
Output Adapters
convert outgoing streams from Coral8 format and publish data to external destinations

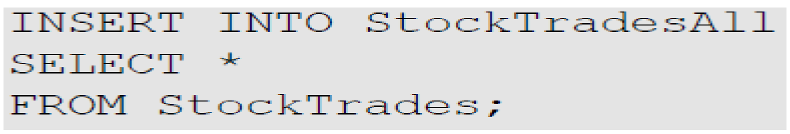
Figure 3.2. Using CCL in Coral8
(Source: Coral8 2009).

3.5. Continuous Computing Language Queries

The Coral8 CCL is a language based on the ANSI SQL Standard. It is used to manipulate, analyze and process data within the Coral8 engine. Like SQL, CCL syntax is structured around commands called statements. One or more CCL statements reside in a query module. When a query module is started, all the CCL statements in it are executed and run continuously until the module is stopped or the Coral8 engine is shut down.

SQL is designed to query data from static tables; CCL is adapted to the needs of processing and analyzing dynamic data. A CCL query statement takes data from one or more data sources, processes it and then publishes it to a destination. This is the most common statement in CCL and can take many complex forms (Coral8 2009).

Figure 3.3 is an example of the simplest possible query statement:



```
INSERT INTO StockTradesAll
SELECT *
FROM StockTrades;
```

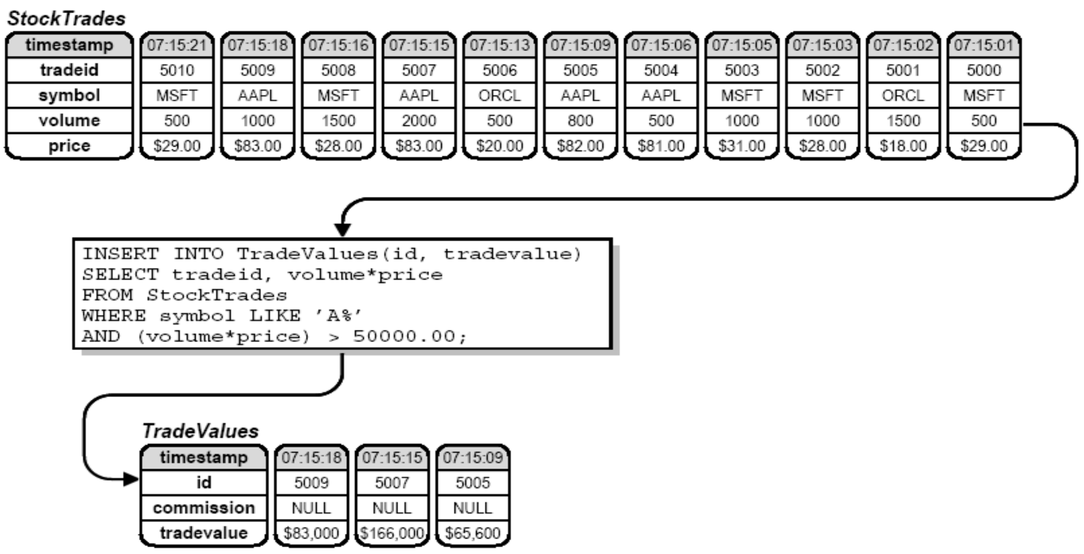
Figure 3.3. The simplest CCL query statement.

The INSERT INTO clause, which is always the first clause in a query statement, indicates the destination of the query output. This query publishes its output to the *StockTradesAll* stream. The SELECT clause defines the contents of the query output. This query uses the "*" syntax which indicates that all columns should be selected from the source data stream. Columns and rows of the source can be selected by using one or more filter or selection conditions. The FROM clause follows the SELECT clause and specifies one or more data sources to which the query subscribes. The semicolon ";" is used to terminate every CCL query statement.

In figure 3.4, CCL expressions are combinations of literals, column names, operators, and functions that evaluate to a single value and Figure 3.5 includes the input and output of execution of the CCL query:

```
INSERT INTO TradeValues(id, tradevalue)
SELECT tradeid, volume*price
FROM StockTrades
WHERE symbol LIKE 'A%' AND (volume*price) > 50000.00
```

Figure 3.4. A CCL expression query example
(Source: Coral8 2009).



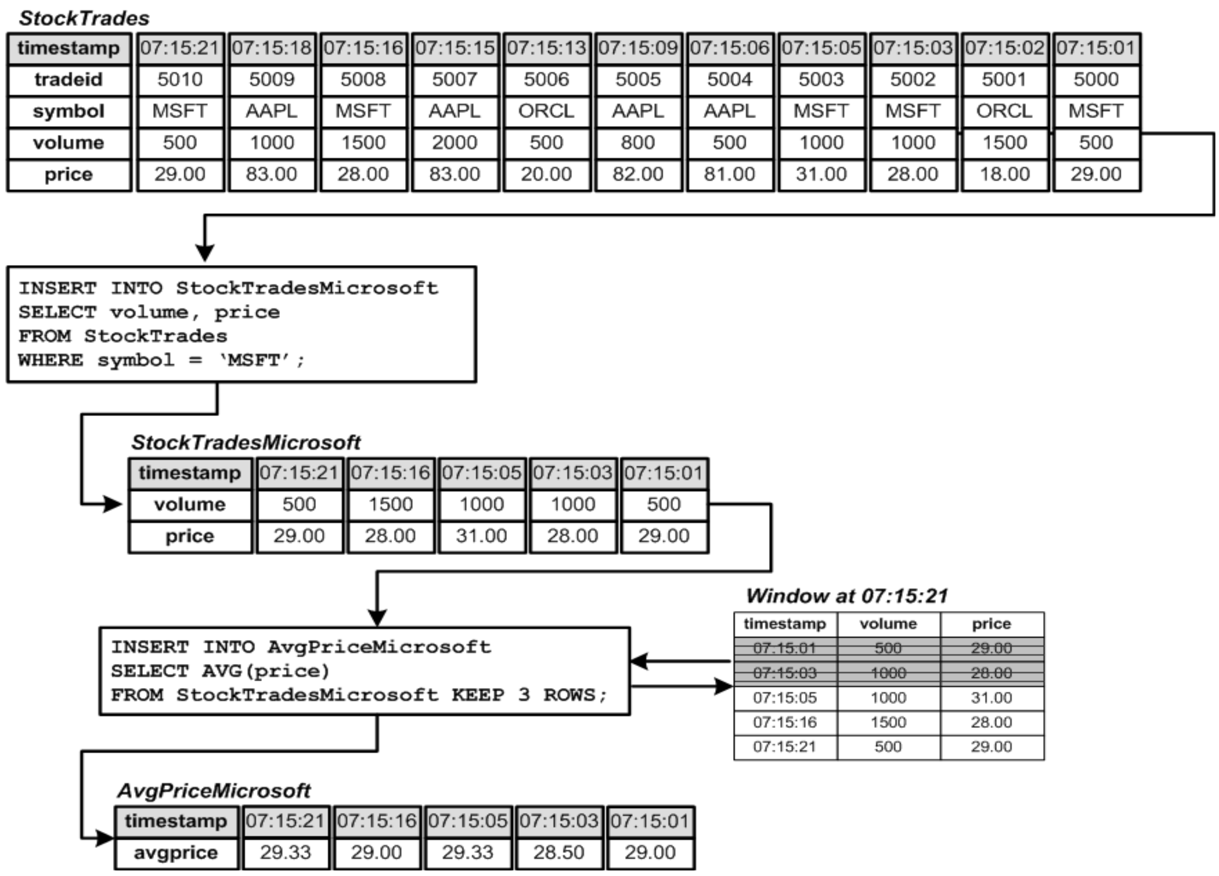
timestamp	07:15:21	07:15:18	07:15:16	07:15:15	07:15:13	07:15:09	07:15:06	07:15:05	07:15:03	07:15:02	07:15:01
tradeid	5010	5009	5008	5007	5006	5005	5004	5003	5002	5001	5000
symbol	MSFT	AAPL	MSFT	AAPL	ORCL	AAPL	AAPL	MSFT	MSFT	ORCL	MSFT
volume	500	1000	1500	2000	500	800	500	1000	1000	1500	500
price	$29.00	$83.00	$28.00	$83.00	$20.00	$82.00	$81.00	$31.00	$28.00	$18.00	$29.00

```
INSERT INTO TradeValues(id, tradevalue)
SELECT tradeid, volume*price
FROM StockTrades
WHERE symbol LIKE 'A%'
AND (volume*price) > 50000.00;
```

timestamp	07:15:18	07:15:15	07:15:09
id	5009	5007	5005
commission	NULL	NULL	NULL
tradevalue	$83,000	$166,000	$65,600

Figure 3.5. The input and output of the CCL expression query
(Source: Coral8 2009).

In figure 3.6 and 3.7, CCL windows permit to maintain the state of rows that have been previously processed from a stream. CCL windows are typically defined by the use of a KEEP clause. Count-based windows have a KEEP clause that defines the maximum number of rows, the window will keep, and whether or not these rows will expire from the window one at a time (a sliding count-based window), or all in a group (a jumping count-based window):



timestamp	07:15:21	07:15:18	07:15:16	07:15:15	07:15:13	07:15:09	07:15:06	07:15:05	07:15:03	07:15:02	07:15:01
tradeid	5010	5009	5008	5007	5006	5005	5004	5003	5002	5001	5000
symbol	MSFT	AAPL	MSFT	AAPL	ORCL	AAPL	AAPL	MSFT	MSFT	ORCL	MSFT
volume	500	1000	1500	2000	500	800	500	1000	1000	1500	500
price	29.00	83.00	28.00	83.00	20.00	82.00	81.00	31.00	28.00	18.00	29.00

```
INSERT INTO StockTradesMicrosoft
SELECT volume, price
FROM StockTrades
WHERE symbol = 'MSFT';
```

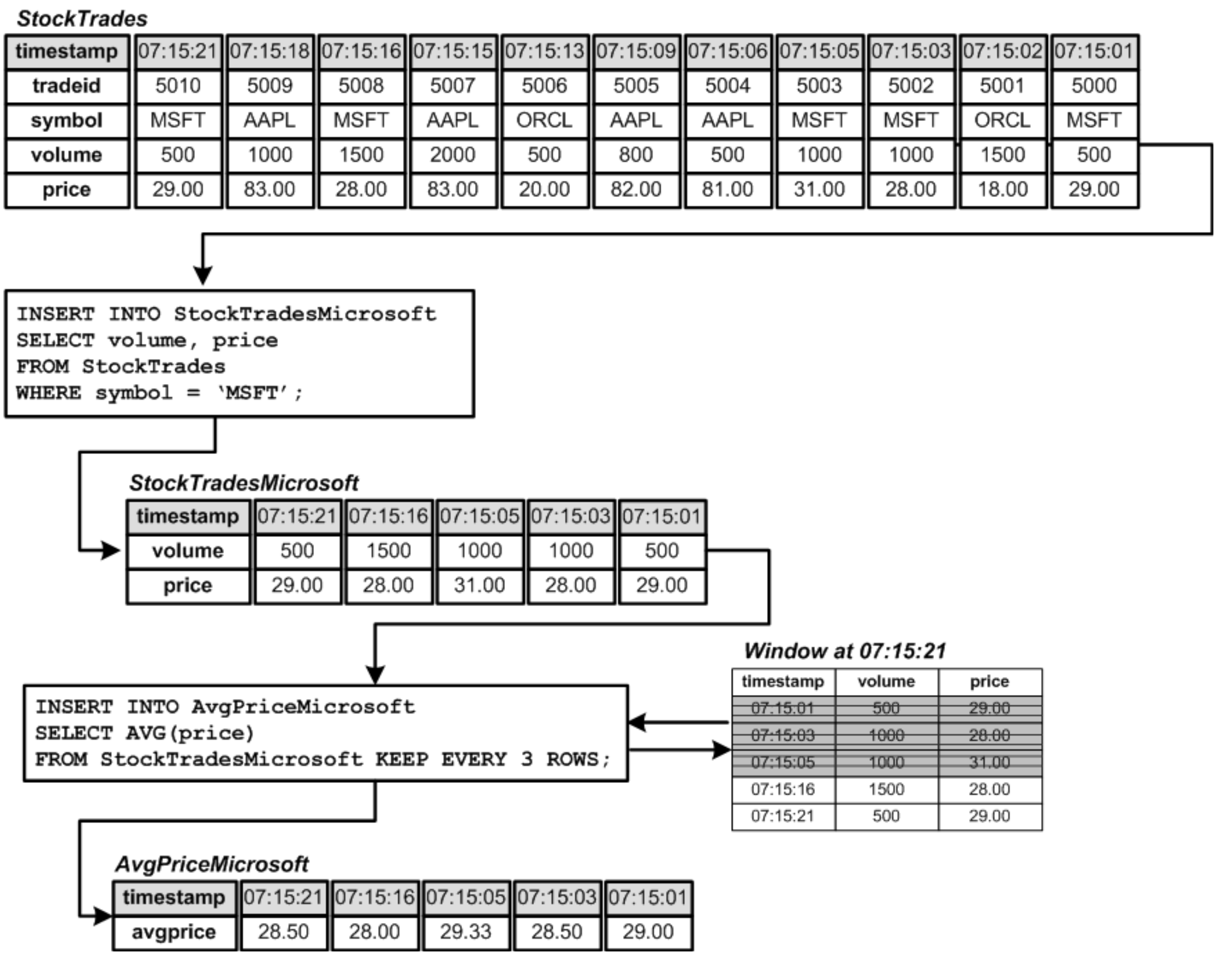
timestamp	07:15:21	07:15:16	07:15:05	07:15:03	07:15:01
volume	500	1500	1000	1000	500
price	29.00	28.00	31.00	28.00	29.00

```
INSERT INTO AvgPriceMicrosoft
SELECT AVG(price)
FROM StockTradesMicrosoft KEEP 3 ROWS;
```

timestamp	volume	price
~~07:15:01~~	~~500~~	~~29.00~~
~~07:15:03~~	~~1000~~	~~28.00~~
07:15:05	1000	31.00
07:15:16	1500	28.00
07:15:21	500	29.00

timestamp	07:15:21	07:15:16	07:15:05	07:15:03	07:15:01
avgprice	29.33	29.00	29.33	28.50	29.00

Figure 3.6. The input and output of the sliding count-based window example (Source: Coral8 2009).


StockTrades

timestamp	07:15:21	07:15:18	07:15:16	07:15:15	07:15:13	07:15:09	07:15:06	07:15:05	07:15:03	07:15:02	07:15:01
tradeid	5010	5009	5008	5007	5006	5005	5004	5003	5002	5001	5000
symbol	MSFT	AAPL	MSFT	AAPL	ORCL	AAPL	AAPL	MSFT	MSFT	ORCL	MSFT
volume	500	1000	1500	2000	500	800	500	1000	1000	1500	500
price	29.00	83.00	28.00	83.00	20.00	82.00	81.00	31.00	28.00	18.00	29.00

```
INSERT INTO StockTradesMicrosoft
SELECT volume, price
FROM StockTrades
WHERE symbol = 'MSFT';
```

StockTradesMicrosoft

timestamp	07:15:21	07:15:16	07:15:05	07:15:03	07:15:01
volume	500	1500	1000	1000	500
price	29.00	28.00	31.00	28.00	29.00

```
INSERT INTO AvgPriceMicrosoft
SELECT AVG(price)
FROM StockTradesMicrosoft KEEP EVERY 3 ROWS;
```

Window at 07:15:21

timestamp	volume	price
~~07:15:01~~	~~500~~	~~29.00~~
~~07:15:03~~	~~1000~~	~~28.00~~
~~07:15:05~~	~~1000~~	~~31.00~~
07:15:16	1500	28.00
07:15:21	500	29.00

AvgPriceMicrosoft

timestamp	07:15:21	07:15:16	07:15:05	07:15:03	07:15:01
avgprice	28.50	28.00	29.33	28.50	29.00

Figure 3.7. The input and output of the jumping count-based window example (Source: Coral8 2009).

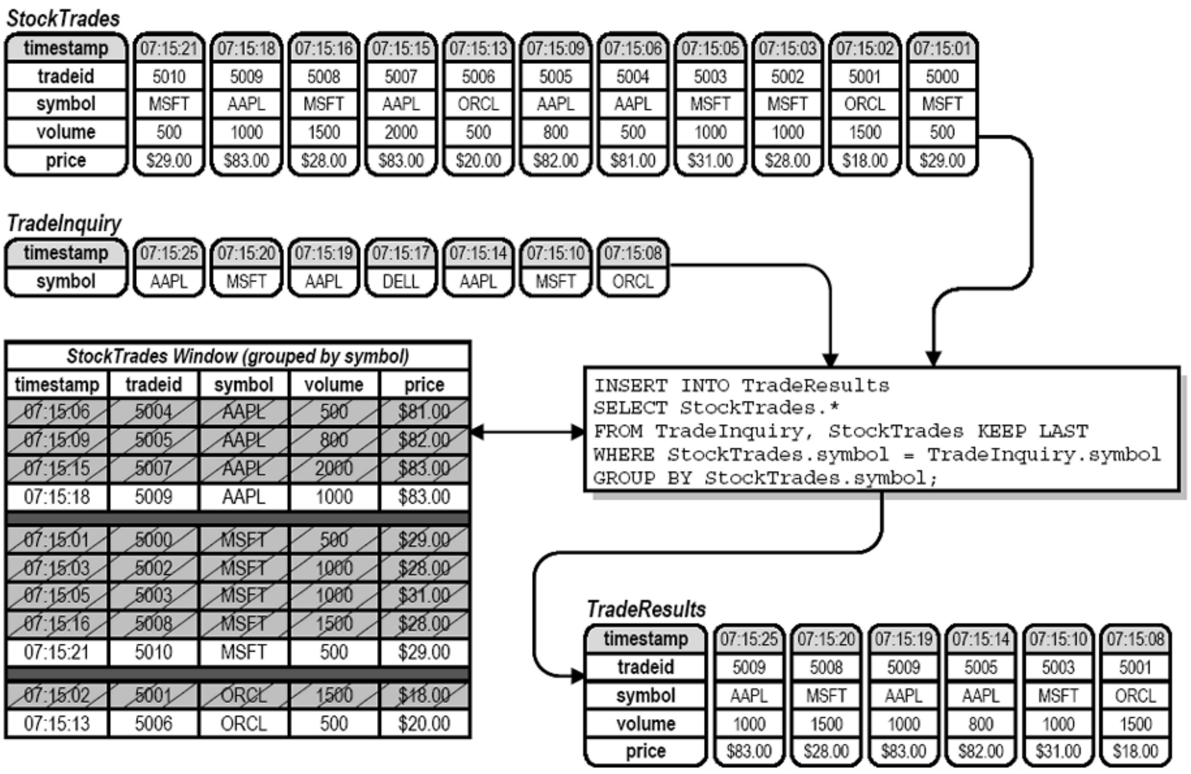
All of the queries seen so far have involved a single data source. However, a query may use data from multiple data sources. Using multiple data sources in a single query is handled by an operator called a join.

Figure3.8 is an example of a simple join query that publishes the most recent trade for a stock whenever an inquiry is made for that stock symbol:

```
INSERT INTO TradeResults
SELECT StockTrades.*
FROM TradeInquiry, StockTrades
KEEP LAST WHERE StockTrades.symbol = TradeInquiry.symbol
GROUP BY StockTrades.symbol;
```

Figure 3.8. The simple join query example.

Multiple data sources in a join (correlation) are separated by commas in the FROM clause. Only one of the sources can be a stream, other sources must be windows. Figure 3.9 shows the simple join multiple data sources query example below:



timestamp	07:15:21	07:15:18	07:15:16	07:15:15	07:15:13	07:15:09	07:15:06	07:15:05	07:15:03	07:15:02	07:15:01
tradeid	5010	5009	5008	5007	5006	5005	5004	5003	5002	5001	5000
symbol	MSFT	AAPL	MSFT	AAPL	ORCL	AAPL	AAPL	MSFT	MSFT	ORCL	MSFT
volume	500	1000	1500	2000	500	800	500	1000	1000	1500	500
price	$29.00	$83.00	$28.00	$83.00	$20.00	$82.00	$81.00	$31.00	$28.00	$18.00	$29.00

timestamp	07:15:25	07:15:20	07:15:19	07:15:17	07:15:14	07:15:10	07:15:08
symbol	AAPL	MSFT	AAPL	DELL	AAPL	MSFT	ORCL

StockTrades Window (grouped by symbol)				
timestamp	tradeid	symbol	volume	price
~~07:15:06~~	~~5004~~	~~AAPL~~	~~500~~	~~$81.00~~
~~07:15:09~~	~~5005~~	~~AAPL~~	~~800~~	~~$82.00~~
~~07:15:15~~	~~5007~~	~~AAPL~~	~~2000~~	~~$83.00~~
07:15:18	5009	AAPL	1000	$83.00
~~07:15:01~~	~~5000~~	~~MSFT~~	~~500~~	~~$29.00~~
~~07:15:03~~	~~5002~~	~~MSFT~~	~~1000~~	~~$28.00~~
~~07:15:05~~	~~5003~~	~~MSFT~~	~~1000~~	~~$31.00~~
~~07:15:16~~	~~5008~~	~~MSFT~~	~~1500~~	~~$28.00~~
07:15:21	5010	MSFT	500	$29.00
~~07:15:02~~	~~5001~~	~~ORCL~~	~~1500~~	~~$18.00~~
07:15:13	5006	ORCL	500	$20.00

```
INSERT INTO TradeResults
SELECT StockTrades.*
FROM TradeInquiry, StockTrades KEEP LAST
WHERE StockTrades.symbol = TradeInquiry.symbol
GROUP BY StockTrades.symbol;
```

timestamp	07:15:25	07:15:20	07:15:19	07:15:14	07:15:10	07:15:08
tradeid	5009	5008	5009	5005	5003	5001
symbol	AAPL	MSFT	AAPL	AAPL	MSFT	ORCL
volume	1000	1500	1000	800	1000	1500
price	$83.00	$28.00	$83.00	$82.00	$31.00	$18.00

Figure 3.9. The input and output of the simple join query example
(Source: Coral8 2009).

The event pattern matching clause MATCHING uses operators “,” (followed by), “&&” (and), “||” (or), and “!” (not) to express highly complex event patterns that would otherwise require multiple joins to detect. Figure 3.10 shows the simple event pattern matching query example below:

```
Insert Into
     StreamAlerts
Select
     StreamA.id
From
     StreamA a, StreamB b, StreamC c, StreamD d
Matching
     [10 seconds: a && b, c, !d]
On
     a.id = b.id = c.id = d.id
```

Figure 3.10. The simple event pattern matching query example.

3.6. Building Custom Complex Event Processing Applications

Custom applications start simple, as the initial requirements are typically very limited. Many begin with simple filtering or aggregation. Problems increase quickly, however, as windows, complex aggregations, correlation, pattern matching, and levels of complexity are added. Despite the promise of a “custom solution”, performance rapidly becomes a problem. Providing enterprise features such as scalability, clustering and high-availability while developing, extending and maintaining custom CEP applications is notoriously difficult and time-consuming. Some custom applications are written on top of a messaging system, or a message bus. Unfortunately, message buses solve mainly transport-level problems, such as asynchronous message delivery, publish/subscribe multicast and guaranteed delivery. Other than performing basic filtering, message buses offer no support for any complex computation, correlation or pattern matching. All of these tasks must still be implemented in a custom application (Coral8 2009).

Coral8 provides a new infrastructure for the world of custom CEP applications. This infrastructure combines the following elements (Coral8 2009):

New programming model for streaming data:

- A relational model for real-time streaming data,
- An execution model for running queries continuously, not only when a query submitted,
- A rich language based on SQL, but suitable for real-time processing, that incorporates extensions to deal with the time dimension, events, event patterns, and so on.

High-performance, scalable, reliable processing platform:

- Performance equaling the very best custom applications (tens of thousands messages per seconds per server, with processing latency measured in milliseconds),
- A clustered and federated model for processing streaming data,
- Guaranteed delivery, failover, high availability and other critical enterprise requirements.

Integration Framework:

- A sophisticated adapter model that supports integration with all kinds of external systems for event input and output,
- Rich capabilities for interacting with external databases and applications, supporting the ability to correlate real-time and historical, or reference data,
- The ability to extend the system by building and registering custom functions, aggregators, adapters, and so on.

Tools:

- Coral8 studio which lets developers create, debug and monitor their CEP applications,
- A portal framework for building business user applications that let business users choose query parameters, presentation options, and notification options,
- Numerous command-line tools and SDKs that support C/C++, Java, .Net, Perl, Python, and other environments.

CHAPTER 4

INTRUSION DETECTION ALERT CORRELATION

Intrusion detection has been an active research field for more than 25 years since Anderson published his seminal work on (Anderson 1980). After Anderson's report, many IDSs have been designed, implemented and deployed into networks. They are a line of defense to protect digital assets. Although many novel designs and improvements have been proposed, at present, IDSs still suffer from a few drawbacks (Xu and Ning 2006):

- IDSs may flag thousands of alerts every day, thus overwhelming security officers. For instance, the experience of the authors shows that 325.968 alerts were reported when a Snort box was deployed for 6 days in a subnet hosting of a teaching lab on a campus network,
- Among all the alerts reported by IDSs, false positives are mixed with true alerts. In addition, it is very possible that a large percentage of alerts are false positives. For example, Julish and other researchers (Julish and Dacier 2002) pointed out that up to 99% of alerts could be false positives. This may make the alert investigation very challenging,
- At present, IDSs cannot guarantee the detection of all attacks. In other words, they may miss some attacks, which could be critical for security officers to understand the current security threats.

These limitations of IDSs make security investigation not only time-consuming, but also error-prone. It is very challenging for security officers to fully learn the security threats in their networks as well as over the Internet. To address these challenges, various alert correlation techniques have been proposed in recent years.

Alert correlation is a process that takes as input the alerts produced by one or more intrusion detection sensors and provides a more succinct and high-level view of occurring or attempted intrusions. The main objective is to produce intrusion reports that capture a high-level view of the activity on the network without losing security-relevant information (Kruegel, et al. 2005).

The notion of security-relevant information cannot be completely objective since it depends on a site's security policy. A security policy defines the desired properties for each part of a secure site's installation. It is a decision that has to take into account the value of the assets that should be protected, the expected threats and the cost of proper protection mechanisms. A sufficient security policy for the data of a normal user at home may not be sufficient for bank applications, as these systems are obviously a more likely target and have to protect more valuable resources. Therefore, it is important to accommodate different requirements of different security policies for a correlation scheme to be adjusted.

The alert correlation process consists of a collection of components that transform sensor alerts into intrusion reports. Since alerts can refer to different kinds of attacks at different levels of granularity, the correlation process cannot equally treat all alerts. Instead, it is necessary to provide a set of components that focus on different aspects of the overall correlation task (Kruegel, et al. 2005).

Some of the components can operate independently on all alerts, according to their type. These components are used in the initial and final phase of the correlation process to implement general functionality that is applicable to all alerts. Other components can only work with certain classes of alerts. These components are responsible for performing specific correlation tasks that cannot be generalized for arbitrary alerts.

Figure 4.1 gives a graphical representation of the alert correlation process (Kruegel, et al. 2005). The core of this process consists of some components implementing the specific functions, which operate on different spatial and temporal properties. For instance, some of the components correlate events occurring at the closing time and space (e.g., alerts generated on one host within a small time window), while others operate on events. These events represent an attack scenario evolving over several hours and including alerts generated on different hosts (e.g., alerts that represent large-scale scanning activity).

In the process shown in Figure 4.1, alerts correlated by one component are used as input by the next component. However, it is not necessary for all alerts passing through the same components sequentially. Some components can operate in parallel, and it is even possible that the output of the alerts occurred as a sequence of components can fed back as an input to a previous component of the process. That is, 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.

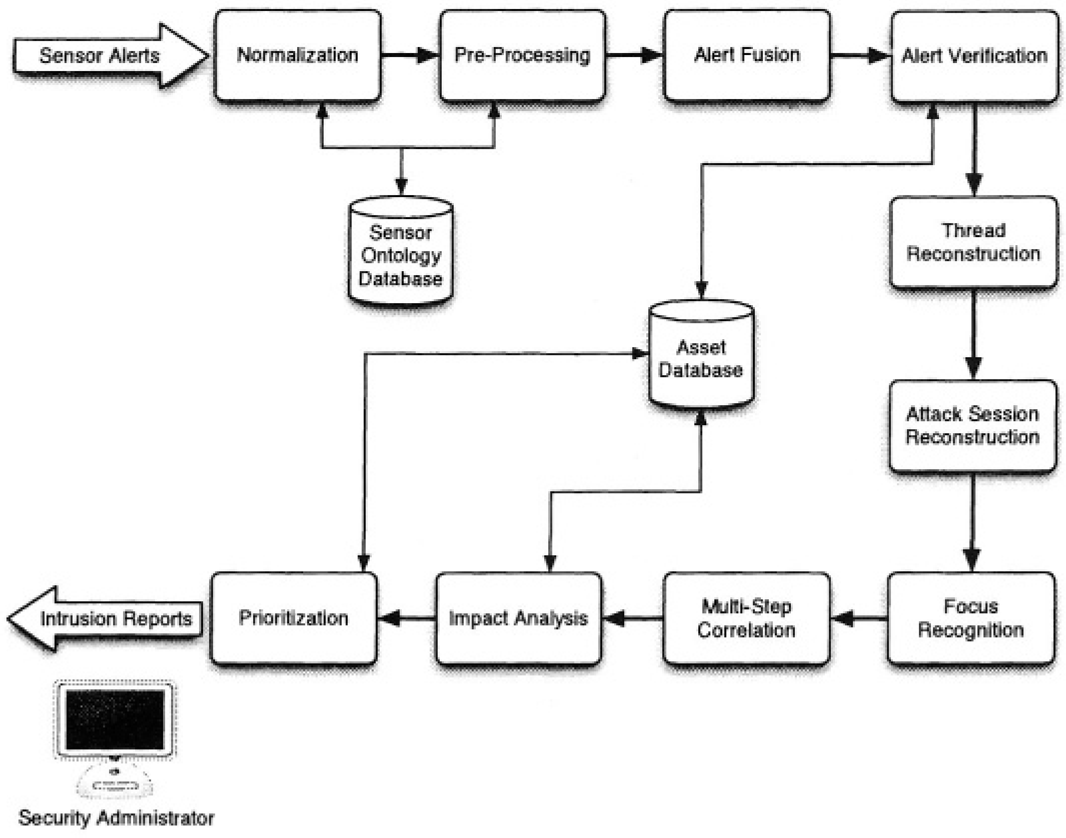


Figure 4.1. Correlation process overview
(Source: Kruegel, et al. 2005).

4.1. Alert Correlation Techniques

The alert correlation process should be carried out by implementing alert correlation techniques. These alert correlation techniques can be classified into three categories (Zhai, et al. 2006): similarity-based correlation, correlation by matching with pre-defined attack scenarios, and correlation based on the prerequisite and consequence of individual attacks. Xu and Ning also add a correlation technique which based on multiple information sources (Xu and Ning 2006). This provides frameworks to model different types of information and may further perform reasoning based on IDS alerts and other information. Each technique has its advantages and disadvantages. However, the correctness of correlation results is strongly affected by the false positives and false negatives among IDS alerts.

4.1.1. Similarity-based Correlation

IDSs may flag alerts when suspicious events are observed. Each alert usually has several attributes associated with it. For example, NIDSs report the suspicious event's source IP address, source port number, destination IP address, destination port number, and timestamps information. Based on these attribute values, some similarity based alert correlation approaches first compute how similar two or more alerts are, and then group alerts together based on these computed similarity values. Approaches in this category can potentially reduce the number of alerts reported to the security officers, because a group of similar alerts may correspond to the same attack or attack trend (Xu and Ning 2006).

To understand this idea, assume that there are two network based IDSs: Snort and RealSecure network sensor. Furthermore, there is an FTP attack in the network, and this attack is detected by both Snort and RealSecure network sensors. For the alerts reported by Snort and RealSecure, it is very likely that they have the same attribute values (IP Addresses, port numbers, timestamps, etc.). Through identifying these similar attribute values, security officers may realize that these alerts correspond to the same attack.

At this point, how to define similarity measures usually is one of the major focuses. There are several different similarity measures being purposed.

A probabilistic approach to performing alert correlation specifies *expectation of similarity* and *minimum similarity* for common features of alerts. Each alert reported by heterogeneous IDSs' sensors is assumed to have several features, for example, target hosts and ports, and timestamps. The main purpose of this approach is to compute the similarity values among the alerts (Valdes and Skinner 2001). For each feature of the alerts, a similarity function is defined, which will be used to calculate the similarity value for the same feature among different alerts. Notice that feature similarity functions may be defined through various criteria. For example, the similarity between IP addresses may consider if they are identical or from the same subnet; if a feature has a list of values (e.g., all open ports reported by a scanning attack), the overlapping values among multiple lists can be considered when calculating similarity. The similarity value is between 0 and 1.

For each feature, if their similarity value is less than the corresponding predefined minimum similarity, then the overall similarity is 0; if the minimum similarity is satisfied, then the overall similarity is the weighted average of similarity values for those common features, where the weights are the expected similarity values for the corresponding features. The formula for overall similarity computation is defined as following:

$$SIM(A,B) = \frac{\sum_{i=1}^{n} SIM(A_i,B_i) \times E_i}{\sum_{i=1}^{n} E_i},$$

Figure 4.2. The formula for overall similarity computation.

Where A and B are two alerts have n features in common, A_i and B_i are values for the common feature *i* in A and B, respectively, SIM(A_i, B_i) is the similarity between A_i and B_i, and E_i is the expected similarity value for feature *i*.

Second approach focuses on port scanning detection to define the similarity of the measure; it can be extended to correlate other security events (Saniford, Hoagland and McAlerney 2002). In this approach, network packets are the primary information to be dealt with. To detect port scanning, feature data such as source IP addresses, destination IP addresses, and destination ports are extracted from network packets. The combinations of these features are also called events. The detection of port scanning can be performed into two steps. In the first step, an anomaly score A(x) is computed as *$A(x) = -log(P(x))$*, where x is an event, and P(x) is x's probability value based on network traffic distribution. When the anomaly scores of events (network packets) are greater than certain thresholds, these events are passed to the second step. The general idea of the second step is to correlate events together, and the groups of events may be identified as portscans. The evaluation function computes the strength of connections between events. Given two events e_1 and e_2, the evaluation function is defined as *$f(e1, e2) = c1h1(e1, e2)+c2h2(e1, e2)+\cdot\cdot\cdot+cjhj(e1, e2)$,* after computing the strengths of connections, a set of events may be grouped if the strengths of connections between the events greater than a certain threshold. In addition, the anomaly score of each group is the summation of the anomaly scores of all events in the group. If the anomaly score of a group is greater than a threshold, a port scanning alert is reported to a security officer.

The other proposed one is an alert clustering approach for performing root cause analysis, where the root cause is the reason why the alerts are triggered (Julisch 2003). The root cause is an HTTP server with a broken TCP/IP stack which may trigger many fragmented IP alerts. The rationale of this approach is based on the observation. Although IDSs flag thousand of alerts every day, it is not uncommon that a few dominant root causes may trigger 90% of all alerts. Thus if security officers can identify these root causes (with the corresponding alerts), and remove these root causes, they can dramatically reduce the number of potential alerts in the future.

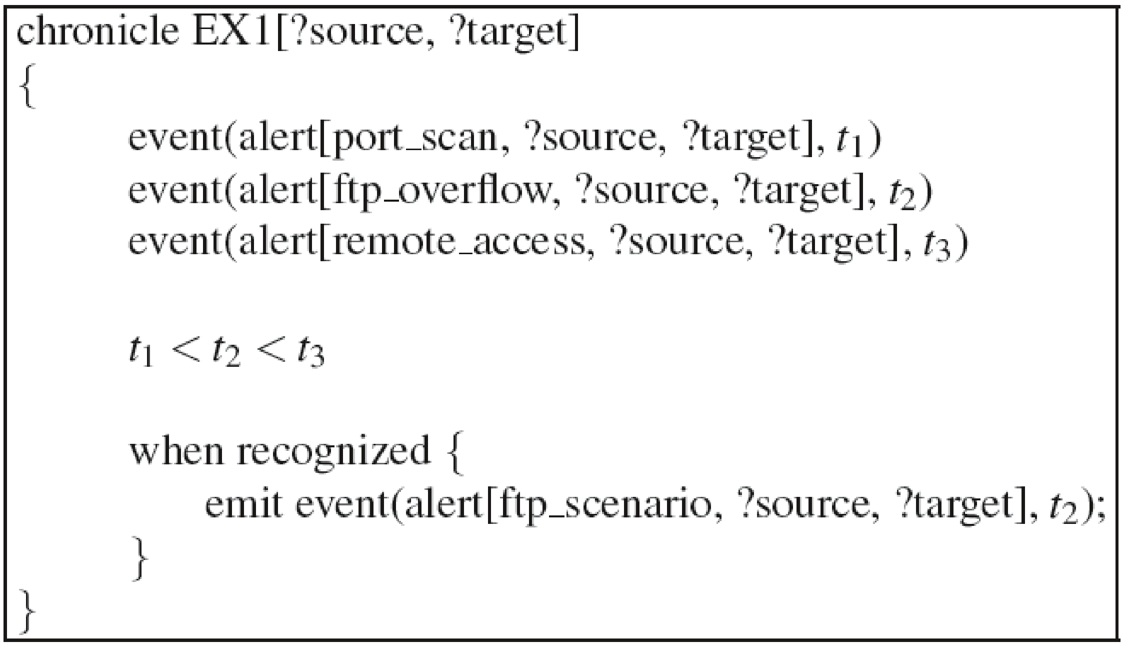
In 2003, Qin and Lee proposed an alert correlation approach to performing statistical causality analysis. The focus of this approach is to conduct time series and statistical analysis to get attack scenarios, though Qin and Lee also propose clustering techniques to aggregate certain lower level alerts to a hyper alert (i.e., a group of alerts ordered by timestamps) thus potentially reduce the alert volume, and perform alert prioritization to identify important alerts (Qin and Lee 2003).

Finally, there is one more approach to manage alerts in an environment with multiple IDSs. It is very possible to flag different alerts by using different IDSs, even for the same attack. Alert processing, which is related to MIRADOR project funded by the French DGA/CASSI, is divided into three steps in this approach: alert (base) management, alert clustering, and alert merging. In first step, alerts reported by different IDSs are transformed into records and then saved into relational databases. The second step is alert clustering. In this step, alerts reported by different IDSs are grouped into clusters so that the alerts in one cluster correspond to one attack. The critical part in this step is to identify similarity between alerts. The similarity is specified by an expert system which performs expert rules and these rules include classification, source, target, and time similarity actions. The third step of alert processing is alert merging, where the alerts in each cluster are merged, and a global alert is created with merged data. One of the critical points here is how to merge different attributes such as classification, source, target, and time information (Cuppens 2001).

4.1.2. Correlation Based on Predefined Attack Scenarios

An attack scenario usually is a sequence of individual attack steps linked together to show an aggregated or global view of security threats. To build these attack sequences, a straightforward way is to first predefine some attack scenario templates. For example, an attack sequence template may be specified where IP_Scan is followed by TCP_Port_Scan and then by FTP_Buffer_Overflow. Next, individual alerts reported by IDSs are matched to these scenario templates to construct attack scenarios. These approaches can help security officers to discover all scenarios where their corresponding patterns are aware of and predefined. However, sometimes it is not easy to exhaustively list all attack sequence templates. Another limitation of these methods is that once some novel attack patterns are created by attackers, the corresponding attack scenarios may not be recognized (Xu and Ning 2006).

In a dynamic system, chronicles provide a mechanism to model event temporal patterns and monitor the system's evolution. To specify a chronicle model, event patterns, related timestamp information, time constraints among events, as well as other patterns and actions will be used for modeling. Figure 4.2 shows an example of chronicles.



```
chronicle EX1[?source, ?target]
{
        event(alert[port_scan, ?source, ?target], t1)
        event(alert[ftp_overflow, ?source, ?target], t2)
        event(alert[remote_access, ?source, ?target], t3)

        t1 < t2 < t3

        when recognized {
              emit event(alert[ftp_scenario, ?source, ?target], t2);
        }
}
```

Figure 4.3. An example of chronicles
(Source: Xu and Ning 2006).

In this chronicle, there events are considered *port_scan*, *ftp_overflow*, and *remote_access*, where their corresponding timestamps should be in increasing order (i.e., t1 <t2 <t3), and their corresponding domain attributes (i.e., source and target) should be equal. If all these patterns as well as constraints are satisfied, then this chronicle is recognized and a synthetic alert is generated. Notice that usually only

synthetic alerts will be reported to security officers. So this may significantly reduce the workload for security officers (Morin and Debar 2003).

4.1.3. Correlation Based on Prerequisites and Consequence of Attacks

This method correlates intrusion alerts using the prerequisites and consequences of attacks. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful. For example, the existence of a vulnerable service is the prerequisite of a remote buffer overflow attack against the service. The consequence of an attack is the possible outcome of the attack, for instance, gaining certain privilege on a remote machine. In a series of attacks, there are usually connections between the consequences of the earlier attacks and the prerequisites of the later ones. Accordingly, the prerequisites and the consequences of attacks are identified and correlated the detected attacks (i.e., alerts) by matching the consequences of previous alerts to the prerequisites of later ones (Zhai, et al 2006).

The correlation method uses logical formulas, which are logical combinations of predicates, to represent the prerequisites and consequences of attacks. The correlation model represents the attributes, prerequisites, and consequences of known attacks as alert types. The correlation process is to identify the preparation for relations between alerts, which is done with the help of prerequisite sets and expanded consequence sets of alerts. Given an alert, its prerequisite set is the set of all predicates in its own prerequisite, and its expanded consequence set is the set of all predicates in or implied by its consequence. An earlier alert t_1 prepares for a later alert t_2 if the expanded consequence set of t_1 and the prerequisite set of t_2 share some common predicates. An alert correlation graph is used to represent a set of correlated alerts. An alert correlation graph CG = (N, E) is a connected directed acyclic graph, where N is a set of alerts, and for each pair n_1, n_2 € N, there is a directed edge from n_1 to n_2 in E if and only if n_1 prepares for n_2.

The advantage of this method is that the correlation result is easy to understand and directly reflects the possible attack scenarios. However, as the correlation is solely based on IDS alerts, the result highly depends on the quality of the IDS alerts. For example, the result may contain false correlations when there are false alerts.

4.2. Integrating Alert Correlation and OS-Level Dependency Tracking

A different aspect of alert correlation which based on multiple information sources to identify the relevancy between the relationships among IDS alerts and the dependencies among OS-level objects, and then use the OS-level objects, and then the OS-level dependencies to verify or discover the relationships among IDS alerts. To identify such relationships, first the attacks' OS-level behaviors are looked into.

From the operating system's point of view, an attack is a set of OS-level events that access or modify a set of system objects. The OS-level objects and operations corresponding to an attack can be derived from the semantics of the attack. For example, such semantics consist of two parts: one is the prerequisites and consequences of attacks, and the other is the correspondence between the predicates in attacks prerequisites or consequences and the OS-level objects on the host. With such information, the OS-level objects can be identified the corresponding to the attacks on the host. In other words, given an attack that exploits a vulnerable service as its prerequisite and yields a shell as its consequence, the corresponding service process and shell process can be identified.

Accordingly, the OS-level objects corresponding to an attack can be divided into two sets: the *prerequisite object set*, which are the objects derived from the attack's prerequisite, and the *consequence object set*, which are the objects derived from the attack's consequence. These two sets may overlap, because some attacks' consequences may affect their prerequisite objects. By monitoring among the OS-level objects, the dependencies among those objects at the OS-level can also be found. Though different from the preparation for relation used in alert correlation, such OS-level dependencies can be utilized to verify or discover the preparation for relations among the alerts (Zhai, et al. 2006).

In essence, first necessary information from the alerts is extracted to identify the corresponding OS-level objects. Then the dependencies among alerts are verified by using the OS-level dependencies among their corresponding objects, and thus the alert correlation based on the causal relationship is improved. Moreover, by identifying the OS-level objects corresponding to the evidence of possibly are missed attacks, and tracking back from those objects, the performance of existing methods can be improved for hypothesizing about possibly missed attacks.

An attack has to have impacts on the local system in order to be observable in the OS-level log. Thus, OS-level dependency tracking guarantees improvement of alert correlation for the alerts of successful attacks, though it may provide positive results from some failed attack attempts.

How to find the OS-level objects accessed by the attacks which trigger IDS alerts is discussed now. This process is called the mapping of IDS alerts to OS-level objects. The semantics carried by an alert that can be used to identify the corresponding OS-level objects are summarized below. Firstly, an IDS alert comes with a timestamp, which indicates when the attack happens. Secondly, given alert has the knowledge about how the attack works and how the system should behave in response to it. For example, given a Snort alert "FTP EXPLOIT wu-ftpd 2.6.0" and the corresponding attack exploits a vulnerable wu-ftpd server and forks a root shell. Finally, given local system's configuration can be identified that the OS-level objects correspond to each predicate in attacks' prerequisites and consequences (Zhai, et al. 2006). For example, a predicate "Samba server" may correspond to "/usr/sbin/smbd" process on a given computer. How each type of knowledge used to map the alerts is discussed below.

Though the number of logged events and objects is large in system logs, the timestamp of each alert can be used to easily narrow down the potentially relevant system objects. Given the timestamp of an alert, an approximate time window can be estimated during which all the relevant OS-level activities occur, and then narrow down the scope of OS-level objects that need to be examined.

Given the name of an alert, the prerequisite of the corresponding attack and the knowledge of the consequence predicates from experts is known. Each of those predicates is associated with some OS-level objects such as services, processes, and files. Thus, for each predicate in attacks' prerequisites and consequences, the corresponding file or process can be identified on the host computer, and represent them as (*predicate, OS-level object*) pairs in the knowledge base. For example, given a pair (*Samba_service(host_IP),"/usr/sbin/smbd"*) in the knowledge base, whenever there is predicate of *Samba_service(host_IP)*, its corresponding process of *"/usr/sbin/smbd"*can be located. Thus, after identifying the predicates in an attack's prerequisite and consequence, the OS-level objects can be identified from the corresponding to those predicates on the host computer.

4.3. Alert Correlation Graphs

Alert correlation and analysis is a critical task in security management. Several techniques and approaches have been proposed to correlate and analyze security alerts, most of them focus on the aggregation and analysis of raw security alerts, and build attack scenarios (Al-Mamory and Zhang 2007).

The correlation between alerts can be represented as a directed acyclic graph *ACG (Alert Correlation Graph) = (N, E)* where N is a set of nodes representing alerts. For each pair of nodes n1 and n2 € N, there is an edge from n1 to n2 if n1 prepares for n2. In other words, the ACG provides an intuitive representation of correlated alerts, reveals the intrusion strategies behind the attacks, and leads to better understanding of the attacker's intention (Al-Mamory and Zhang 2007).

An interesting method is the work of Ning et al. An alert correlation model proposed which based on the observation that most intrusions consist of many stages, with the early stages preparing for the later ones. Alerts collected from NIDS, correlated off-line, and tried to draw big picture (through ACGs) of what happens in the network. The resulted ACGs show that the proposed system can correlate related alerts, uncover the attack strategies, and can effectively simplify the analysis of large amounts of alerts (Ning, et al. 2004).

The above method generates ACGs depending on pre and post conditions of individual alerts. The correlation model is build upon two aspects of intrusions that are, *Prerequisites* (the necessary conditions for an intrusion to be successful) and *Consequences* (the possible outcome of an intrusion). With knowledge of prerequisites and consequences, the correlation model can correlate related alerts by finding causal relationships between them. They used hyper alert correlation graphs to visually represent the alerts, where each node represents a hyper alert and the edges represents preparation for relation (Al-Mamory and Zhang 2007).

A hyper alert type T is a triple (*fact, prerequisite, consequence*), where *fact* denotes the kind of information reported with the alert, *prerequisite* specifies what must be true for the attack to be successful, consequence describes what is true if the attack indeed succeeds. An hyper alert instantiates its *prerequisite* and *consequence* by replacing the free variables with their specific values and adds an *interval based timestamp: [begin_time, end_time]*. Given a hyper alert type, a hyper alert instance (hyper alert) can be generated if the corresponding attack is detected and reported by an IDS. For example, we can generate a hyper alert instance of type *SadminBufferOverflow* from a corresponding alert. The notion of hyper alert instance is formally defined as follows. An hyper alert h1 prepares for hyper alert h2, if the end time of all the prerequisite events of h2 found in the consequences of h1 occurred earlier → (h1.end_time < h2.begin_time) (Ning, et al. 2002):

Hyper-alert Type *SadmindBufferOverflow* =({*VictimIP, VictimPort*}, *ExistHost(VictimIP)^VulnerableSadmind(VictimIP), GainRootAccess(VictimIP)*)
Hyper-alert $h_{SadmindBOF}$ ={(*VictimIP* = 152.1.19.5, *VictimPort* = 1235*),*
(*VictimIP* = 152.1.19.7, *VictimPort* = 1235*)}*

The preparation for relation between hyper alerts provides a natural way to represent the causal relationship between correlated hyper alerts. A hyper alert correlation graph represents attack scenarios on the basis of the preparation for relation. For example, if an IDS detects a DDOS daemon running on a host, it would be helpful to inform the administrator how this happened, that is, report all the alerts that directly or indirectly preparation for the DDOS daemon (Ning, et al. 2002).

The pseudo code of the proposed algorithm that builds ACG is shown in Figure 4.3:

```
Input:  Stream of meta-alerts in time order
Output: Correlation Graphs
Begin
1:     Initialize Queue Q and Graph G;
2:     Do until All MA ∈ RM visited {
3:         Get new unvisited MA and put it in Q and G;
4:         While Q not empty{
5:             ActiveMA ← Q.dequeue;
6:             S ← set of ActiveMA's children and not visited yet;
7:             Q.enqueue all S elements;
8:             Set ActiveMA as visited;
9:              G ← G ∪ ActiveMA (Connect ActiveMA to
                appropriate MAs in G using Attach_Threshold);
10:        }
11:        Output G as detected Correlation graph;
12:    }
End
```

Figure 4.4. Pseudo-code of graph generation algorithm
(Source: Ning, et al. 2002).

In Figure 4.4, an intrusion alert correlator using the preparation for correlation model. The implementation of the correlator assumes the alerts reported by IDSs are stored in the database. Using the information in the knowledge base, the alert preprocessor generates hyper alerts as well as auxiliary data from the original alerts. The correlation engine then performs the actual correlation task using the hyper alerts and the auxiliary data. After alert correlation, the hyper alert correlation graph generator extracts the correlated alerts from database, and generates the graph files in the format accepted by GraphViz. As the final step of alert correlation, GraphViz is used to visualize the hyper alert correlation graphs.

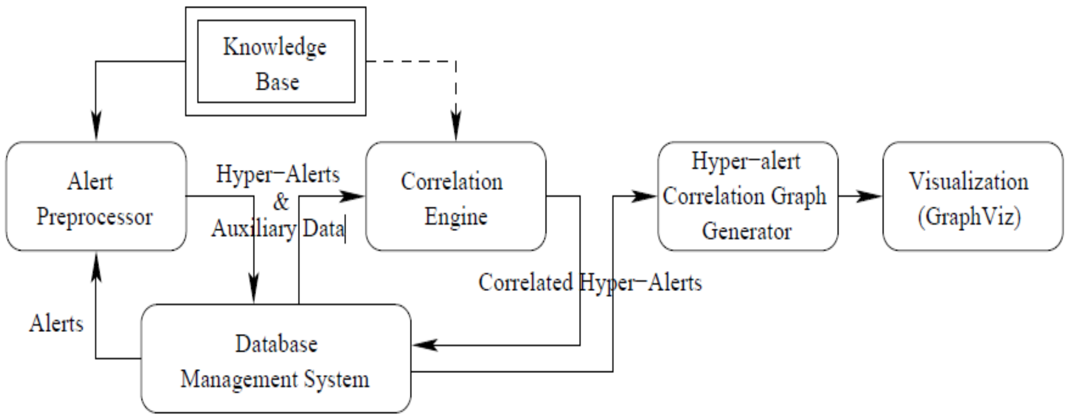


Figure 4.5. The architecture of the intrusion alert correlator (Source: Ning, et al. 2002).

Figure 4.5 shows one of the hyper alert correlation graphs discovered from the 2000 DARPA intrusion detection evaluation datasets (Ning and Cui 2002). Each node in Figure 4.5 represents a hyper alert. The numbers inside the nodes are the alert IDs generated by the IDS. This hyper alert correlation graph clearly shows the strategy behind the sequence of attacks.

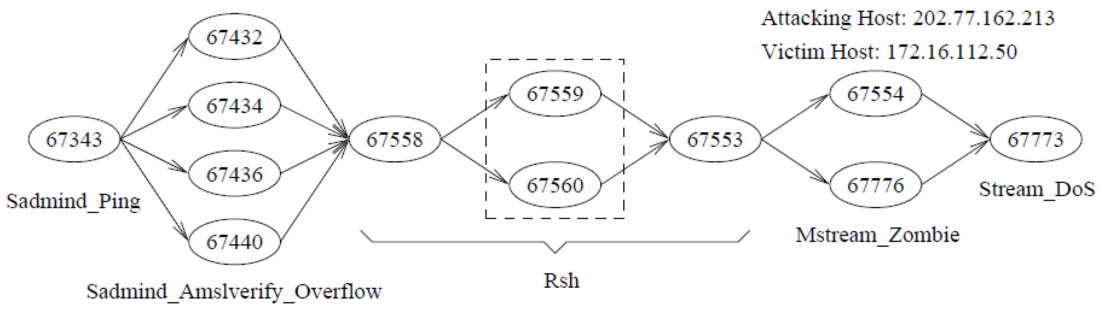


Figure 4.6. A hyper alert correlation graph discovered in the 2000 DARPA intrusion detection evaluation datasets (Source: Ning and Cui 2002).

CHAPTER 5

PROPOSED ARCHITECTURE AND EXPERIMENTAL ENVIRONMENT

5.1. The Architecture of the Intrusion Alert Correlation

As discussed earlier, our goal is to improve alert analysis by integrating OS-level event logging and object dependency tracking into IDS alert correlation. In this chapter, a brief introduction is given to the alert correlation and OS-level dependency tracking techniques to be used in our method. For alert correlation, the method used is based on the prerequisite of the attack and consequence, due to the ease to make connections between alert correlation and OS-level objects in this method.

In this section, an explanation of the proposed architecture is given. The proposed architecture (Figure 5.1) contains an Ubuntu 9.04 server with an IP number 10.1.1.164 which hosts a Snort 2.8.4 IDS and Coral8 5.6.0 Correlation Engine and acts as an IDS Alert Correlation Server in TurkDex network. Ubuntu 9.04 monitors specific types of OS-level objects, i.e., processes and files. The objects are kept in a log with their properties of the objects. It also monitors specific dependency-causing system calls like process forking, file reading, and memory sharing, which together are called "high-control events". One vulnerable service running on the server: Samba 3.0 as files and printers sharing service. Snort 2.8.4 was installed on the server to monitor the network traffic as an IDS sensor. To detect attacks, the rules set used with Snort. Both Ubuntu 9.04 OS-level objects and Snort alerts are integrated to Coral8 Correlation Engine via "/var/log/snort/syslogFormatted.csv" and "/var/log/snort/csv_alert.csv" respectively to correlate the alerts and OS-level objects.

The background traffic is injected during the experiments to mimic an operational network. The background traffic was collected on the target machine when it was connected to the TurkDex network, and was manually verified to contain no attacks toward the target machine. Some attack attempts are also injected by using Nessus 4.0 vulnerability scanning server.

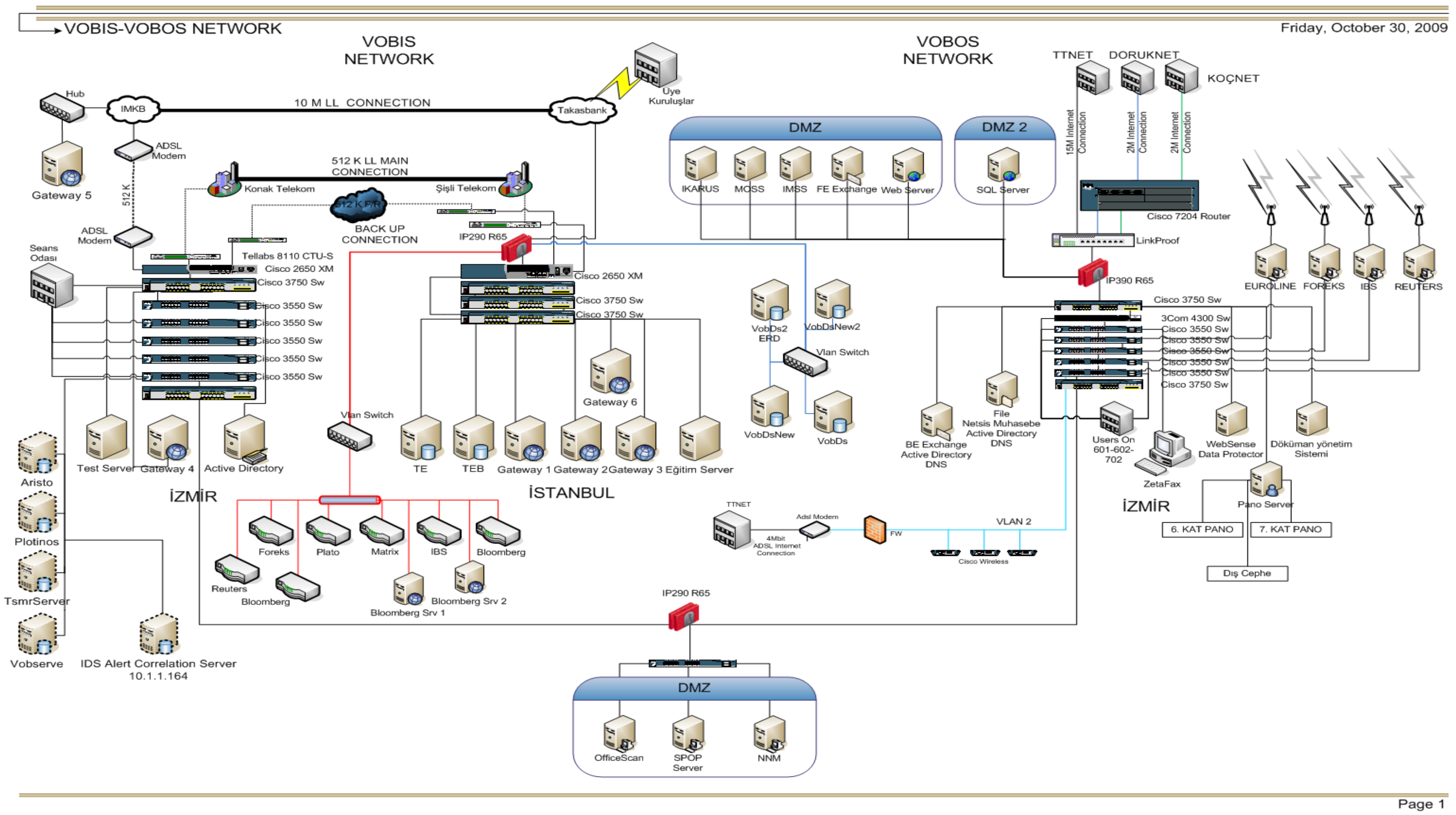


Figure 5.1. TurkDex network structure
(Source: TurkDex 2009).

The operating system of IDS alert correlation server is Ubuntu 9.04. Snort 2.8.4 IDS, Coral8 5.6.0 complex event processing engine server and studio, Samba 3.0 files and printers sharing service, and Nessus 4.0 vulnerability scanning server need to be installed.

First, the Ubuntu 9.04 Desktop ISO image that corresponds to our hardware architecture i386 is needed and which can be downloaded from www.ubuntu.com. When the download is over, burn the ISO image with a CD/DVD burning application on a blank CD at 8x speed. Leave and reinsert the CD in your CD/DVD-ROM device and reboot the server in order to boot from the CD. Follow the default installation steps and then the Ubuntu 9.04 operating system will be installed.

The second step is to set up Snort 2.8.4 IDS on Ubuntu 9.04 OS.

# sudo –s

Type "1234" root password to install service or programs on the Ubuntu.

# apt-get install snort **2.8.4**

At the end of Snort's installation routine, you will be prompted if you wish to set up a database for use with Snort. Choose no. The Snort manually configured. The Snort output logs located into "/var/log/snort/csv_alert.csv" so that to integrate the output logs with Coral8 CEP engine. To set the output log file edit the snort.conf file.

# vi /etc/snort/snort.conf

At the line **number** 791 within Snort.conf file, write the output type and fields of the snort output logs like the following line comment:

output alert_csv: csv_alert.csv timestamp, src, srcport, dst, dstport, msg

# /etc/init.d/snort start

When you start the Snort IDS by the above command and which starts logging the alerts into "/var/log/snort/csv_alert.csv" file.

Installation instructions are listed below for Coral8 CEP engine server and studio.

Download and unpack the installation tar-files using the tar command:

# tar xvfz coral8-server-5.6.0.tar.gz

# tar xvfz coral8-studio.5.6.0.tar.gz

1. Configure the Coral8 server by install-server.sh script file:

# sh ./coral8/install-server.sh

2. Start the Coral8 server using the following command:

# ./coral8/server/coral8-server.rc start

3. Start the Coral8 studio using the following script command:

```
# ./coral8/studio/coral8-studio.sh
```

The next step is to setup a script file which writes the Ubuntu 9.04 system logs into "/var/log/snort/syslogFormatted.csv" simultaneously. The script file "startSnort.sh" located in "/usr/local/bin" and which includes the following linux commands:

```
#/bin/bash
SNORT_PATH=/var/log/snort
touch $SNORT_PATH/syslogFormatted.csv
echo "syslogFormatted.csv created."
echo "month, day, time, hostname, message" > $SNORT_PATH/syslogFormatted.csv
echo "waiting for syslog..."
tail -f /var/log/syslog | sed 's/\([a-z]*\) \([0-9]*\) \([0-9]*:[0-9]*:[0-9]*\) \([a-zA-Z]*\) /\1,\2,\3,\4,/' >> $SNORT_PATH/syslogFormatted.csv
```

Samba 3.0 files and printers sharing service should be installed on the server to provide a vulnerable service to log system calls into syslogFormatted.csv and to detect attacks by the Snort IDS.

```
# sudo -s
# apt-get install samba
# /etc/init.d/samba start
```

At the last step of the server installation, the Nessus 4.0 installed which is one of the best vulnerability scanners provided by www.nesus.org. The Nessus 4.0 can be installed on Ubuntu 9.04 by following linux commands:

```
# sudo –s
# apt-get install nessusd nessus nessus-plugins
# /etc/init.d/nessusd start
```

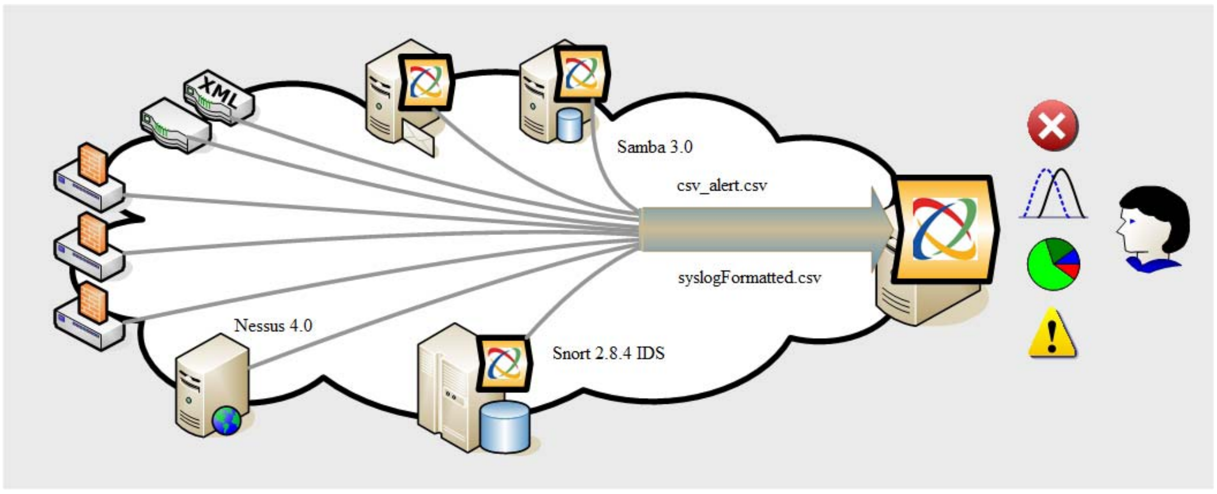


Figure 5.2. Security monitoring by using IDS alert correlation server.

5.2. Details of Alert Correlation Scenario

The attack scenario exploits a vulnerable Samba server. It includes the following attack steps: First, two remote buffer overflow attack attempts are exploited the vulnerable Samba server. Second, a server daemon of a DDoS is uploaded and started by Nessus vulnerability scanner on the target host. Lastly, Nessus client program is used to direct the Nessus server daemon on the victim server to start SYN flood and UDP flood attacks against another computer (Zhai, et al. 2006).

The above attacks took about 5 minutes. During the period, Ubuntu 9.04 logged 81.613 events. Moreover, the Snort sensor raised 9 “NETBIOS SMB trans2open buffer overflow attempt” (“smb-bof”) alerts, 15 “DDOS tfn2k icmp possible communication” (“tfn2k-icmp”) alerts, and 2 “ATTACK-RESPONSES id check returned root” (“id-root”) alerts. The background traffic triggered 32 alerts related to the target server, which include 8 “SCAN nmap TCP” (“nmap-tcp”) alerts, 23 “SNMP public access udp” (“snmp-udp”) alerts, and 1 “FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux” (“wuftp-fs”) alert. Among the 3 types of alerts, the third one is triggered by the failed attempt of wu-ftpd buffer overflow attack injected into the background traffic.

5.3. Experimental Results

Using the alert correlation method proposed in (Ning, Cui and Reeves 2002), the correlation graph generated shown in Figure 5.3.

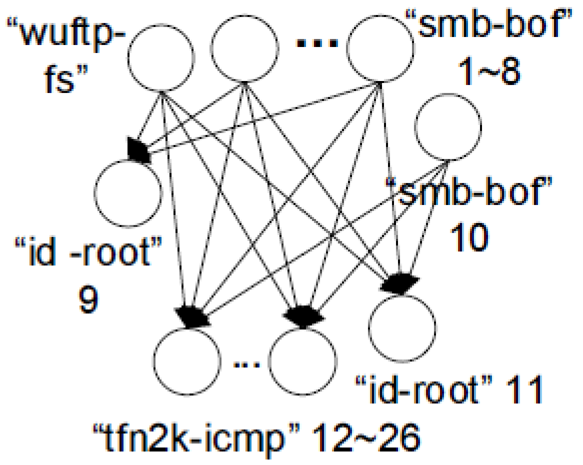


Figure 5.3. The correlation graph.

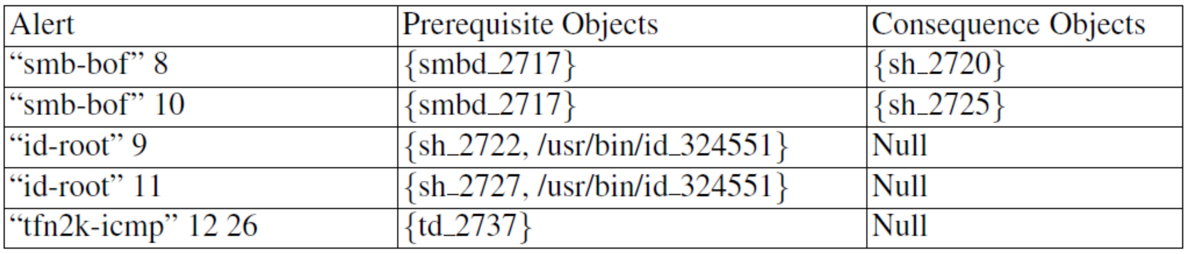
In this alert correlation method, the encoded prerequisite and expanded consequence sets in two Coral8 input adapters are stored, *PrereqSet* and *ExpandedConseqSet*, along with the corresponding hyper-alert ID and timestamp, assuming that each hyper-alert is uniquely identified by its ID. Both input adapters have attributes *HyperAlertID*, *EncodedPredicate*, *begin_time*, and e*nd_time*, with meanings as indicated by their names. As a result, alert correlation can be performed using the following SQL statement on the Coral8 studio which is executed by the Coral8 engine.

```
SELECT DISTINCT c.HyperAlertID, p.HyperAlertID
FROM PrereqSet p, ExpandedConseqSet c
WHERE p.EncodedPredicate = c.EncodedPredicate
AND c.end_time < p.begin_time
```

The correctness of our implementation method is guaranteed by the following theorem:

Theorem 1. Under assumptions 1 and 2, the implementation method discovers all and only hyper-alert pairs such that the first one of the pair preparation for the second one (Ning, et al. 2002).

Obviously, the result of the correlation contains many false correlations due to the false positives within the alerts. Using the operating system's log and the semantics of these alerts, mapped these alerts to a number of OS-level objects, as listed in Figure 5.4.



Alert	Prerequisite Objects	Consequence Objects
"smb-bof" 8	{smbd_2717}	{sh_2720}
"smb-bof" 10	{smbd_2717}	{sh_2725}
"id-root" 9	{sh_2722, /usr/bin/id_324551}	Null
"id-root" 11	{sh_2727, /usr/bin/id_324551}	Null
"tfn2k-icmp" 12 26	{td_2737}	Null

Figure 5.4. OS-level objects corresponding to the alerts in the alert correlation scenario (Source: Ning, et al. 2002).

For each alert prepared by other alerts, the generated Ubuntu 9.04 dependency graphs by tracking back from their prerequisite objects are shown in Figure 5.3. In these graphs, it is looked for paths from the earlier alerts' consequence objects to a later alert's prerequisite objects if the former prepares for the alert. For example, when such a path exists, the two alerts are strongly connected and thus the correlations between them are verified at the OS-level. Otherwise, the correlation would be considered false. In this way, each of the correlations can be verified in the original graph, remove those that are verified to be false and finally come up with a new correlation graph. The new correlation graph for the scenario is shown in Figure 5.5, it can be seen it is the correct correlation graph of the reported Snort alerts based on the actual attack scenario.

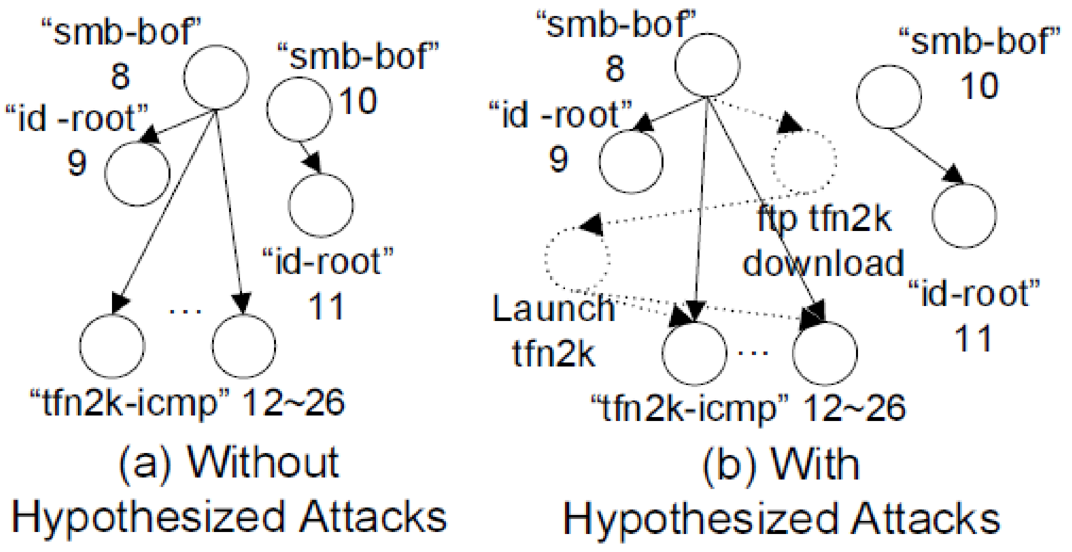


Figure 5.5. The alert correlation graphs (Source: Ning, et al. 2002).

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, a series of techniques defined to integrate the alert correlation method and examined the technique which based on prerequisites and consequences of attacks and OS-level dependency tracking. A critical step in this integration is to establish input adapters to Coral8 complex event processing engine with IDS alerts and OS-level objects and to perform the SQL statement of the alert correlation method into Coral8 engine via Coral8 studio. A number of constraints identified that the OS-level objects should satisfy if they are relevant to the IDS alerts (or attacks) that are correlated. By using these constraints, the IDS alerts verified as well as the correlation between IDS alerts and reduced false correlations. Moreover, the dependency between OS-level objects can also facilitate the hypotheses of attacks possibly missed by the IDSs. The experimental evaluation gave favorable results, showing that OS-level dependency tracking can significantly reduce false correlations when integrated with the alert correlation method.

Several issues are worth for future research. In particular, self-learning alert correlation systems would be the most applicable products for the information security industry. The alert correlation will certainly help the system analyst in identifying the intrusion by concentrating on the groups of alerts that are relevant with each other. By using artificial intelligence techniques for clustering the alerts automatically by giving inputs of such features which are extracted from generated alerts.

REFERENCES

Al-Mamory, S.O. and H. L. Zhang. 2007. Scenario discovery using abstracted correlation graph. *Computational Intelligence and Security, 2007 International Conference on* 702-706.

Ammann, P., D. Wijesekera, and S. Kaushik. 2002. Scalable, graph-based network vulnerability analysis. *In Proceedings of the 9th ACM Conference on Computer and Communications Security.*

Axelsson, S. 1999. The base-rate fallacy and its implications for the difficulty of intrusion detection. *In Proceedings of the 6th ACM Conference on Computer and Communications Security* 1-7.

Bace, R. 2000. Intrusion detection. *Macmillan Technical Publishing.*

Bace, R. and P. Mell. 2001. Intrusion detection systems. *NIST Special Publication.*

Carter, E. 2002. Intrusion detection systems. http://www.informit.com (accessed September 17, 2009).

Coral8 Inc. Coral8 getting started. http://www.coral8.com.

Coral8 Inc. Coral8 programmer's guide. http://www.coral8.com

Coral8 Inc. Coral8 technology overview. http://www.coral8.com.

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

Debar, H. and A. Wespi. 2001. Aggregation and correlation of intrusion detection alerts. *In Recent Advances in Intrusion Detection* 85-103.

Debar, H., M. Dacier, and A. Wespi. 1999. Towards a taxanomy of intrusion detection systems. *Computer Networks.* 31 1999 805-822.

Erdogan, Y. 2008. Development of a distributed firewall administration tool. *Izmir Institute of Technology thesis of Master.*

Julisch, K. 2003. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security* 443-471.

Kruegel, C., F. Valeur, and G. Vigna. 2005. Intrusion detection and correlation challenges and solutions. *Advances In Information Security, Springer US* 29-33.

Morin, B. and H. Debar. 2003. Correlation of intrusion symptoms: an application of chronicles. *In Proceedings of the 6th International Conference on Recent Advances in Intrusion Detection.*

Morrel, J. and S. D. Vidich. 2008. Complex event processing with Coral8. http://www.coral8.com/developers/documentation.html.

Ning, P., D. Xu, C. Healey, and R. St. Amant. 2004. Building attack scenarios through integration of complementary alert correlation methods. *In Proceedings of the 11th Annual Network and Distributed System Security Symposium* 97-111.

Ning, P., Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. *Proceedings of the 9th ACM Conference on Computer and Communications Security* Session: Intrusion Detection, 245-254.

OneSecure 2001. Intrusion detection and prevention protecting your network from attacks allowed by the firewall.

Qin, X. and W. Lee. 2003. Statistical causality analysis of infosec alert data. *In Proceedings of the 6th International Symposium on Recent Advances in Inrusion Detection.* Pittsburg, PA.

Rafeeq, R. U. 2003. Intrusion detection systems with Snort. *Prentice Hall Ptr.* ISBN 0-13-140733-3.

Sans Institute. 2001. Sans: Intrusion detection FAQ. http://www.sans.org.

Snort. 2009. http://www.snort.org (accessed October 9, 2009).

TopLayerNetworks. 2002. Beyond IDS: Essentials of network intrusion prevention. *Top Layer Networks.*

Valdes, A. and K. Skinner. 2001. Probabilistic alert correlation. *In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection* 54-68.

Valeur, F., G. Vigna, C. Kruegel, and R. A. Kemmerer. 2004. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing* 146-169.

Vigna, G. and R. A. Kemmerer. 1999. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security* 37-71.

Webopedia. 2009. Intrusion detection system. http://www.webopedia.com.

Wikipedia. 2009. Complex event processing. http://en.wikipedia.org.

Wikipedia. 2009. Real-time computing. http://en.wikipedia.org.

Xu, D. and P. Ning. 2006. Correlation analysis of intrusion alerts. *North Carolina State University.*

Zhai, Y., P. Ning, and J. Xu. 2006. Integrating IDS alert correlation and OS-level dependency tracking. *Lecture Notes in Computer Science, Springer Berlin / Heidelberg* 272-284.

Zhai, Y., P. Ning, P. Iyer, and D. Reeves. 2004. Reasoning about complementary intrusion evidence. *In Proceedings of the 20th Annual Computer Security Applications Conference.*

APPENDIX A

CSV ALERT LOGS

Table A.1. Alert Logs

10/06-14:13:16.813679	SMB Server daemon access	TCP	10.1.1.100	1600	10.1.1.49	1633
10/06-14:13:18.180317	SMB Server daemon access	TCP	10.1.1.100	1600	10.1.1.127	1093
10/06-14:13:18.685073	SMB Server daemon access	TCP	10.1.1.4	52324	10.1.1.18	445
10/06-14:13:18.685585	SMB Server daemon access	TCP	10.1.1.4	52324	10.1.1.18	445
10/06-14:13:18.747273	SMB Server daemon access	TCP	10.1.1.100	1600	10.1.1.125	1699
10/06-14:13:20.317846	SNMP public access udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.317846	SNMP request udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.323481	SNMP public access udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.323481	SNMP request udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.340528	SNMP public access udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.340528	SNMP request udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.354223	SNMP public access udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:20.354223	SNMP request udp	UDP	10.1.1.67	64186	10.1.1.13	161
10/06-14:13:21.565185	SMB Server daemon access	TCP	213.139.216.130	443	10.1.1.27	3050
10/06-14:13:28.893088	SMB Server daemon access	TCP	192.168.240.19	80	10.1.1.191	52770
10/06-14:13:28.893092	SMB Server daemon access	TCP	192.168.240.19	80	10.1.1.191	52770
10/06-14:13:28.893806	SMB Server daemon access	TCP	192.168.240.19	80	10.1.1.191	52771
10/06-14:13:28.894589	SMB Server daemon access	TCP	192.168.240.19	80	10.1.1.191	52771