MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION – UGC, GOVT. OF INDIA)

Department of CSE

(Emerging Technologies)

(DATA SCIENCE, CYBER SECURITY, IOT)

INTRUSION DETECTION SYSTEMS

LECTURE NOTES

Prepared by
Dr. I. Nagaraju
Professor

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
Recognized under 2(f) and 12(B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad–500100, Telangana State, India

INTRUSION DETECTION SYSTEMS

LECTURE NOTES

B.TECH (R-20 Regulation)
(III YEAR – II SEM)
(2022-23)

DEPARTMENT OF CSE
(EMERGING TECHNOLOGIES)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

M R C E T CAMPUS | DEPT. OF EMERGING TECHNOLOGIES

Vision of the Department

"To be at the forefront of Emerging Technologies and to evolve as a Centre of Excellence in Research, Learning and Consultancy to foster the students into globally competent professionals useful to the Society."

Mission of the Department

The department of CSE (Emerging Technologies) is committed to:

- To offer highest Professional and Academic Standards in terms of Personal growth and satisfaction.
- Make the society as the hub of emerging technologies and thereby capture opportunities in new age technologies.
- To create a benchmark in the areas of Research, Education and Public Outreach.
- To provide students a platform where independent learning and scientific study are encouraged with emphasis on latest engineering techniques.

QUALITY POLICY

- To pursue continual improvement of teaching learning process of Undergraduate and Post Graduate programs in Engineering & Management vigorously.
- To provide state of art infrastructure and expertise to impart the quality education and research environment to students for a complete learningexperiences.
- Developing students with a disciplined and integrated personality
- To offer quality relevant and cost effective programmes to produce engineers as per requirements of the industry need.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF EMERGING TECHNOLOGIES

SYLLABUS

III Year B. Tech– II Semester **L/T/P/ C**
3/0/0/3

(R20A6206) INTRUSION DETECTION SYSTEMS

Course Objectives:

1. To introduce basic concepts of intrusion detection system.
2. To understand Intrusion Prevention Systems, Network IDs protocol and model for intrusion analysis.
3. To Understand when, where, how, and why to apply Intrusion Detection tools and techniques in order to improve the security posture of an enterprise.
4. To Apply knowledge of the fundamentals and history of Intrusion Detection in order to avoid common pitfalls in the creation and evaluation of new Intrusion Detection Systems.
5. To learn agent development for intrusion detection and architectural models of IDs and IPs.

UNIT-I:
History of Intrusion detection, Audit, Concept and definition , Internal and external threats to data, attacks, Need and types of IDS, Information sources Host based information sources, Network based information sources.

UNIT-II:
Intrusion Prevention Systems, Network IDs protocol based IDs ,Hybrid IDs, Analysis schemes, thinking about intrusion. A model for intrusion analysis , techniques Responses requirement of responses, types of responses mapping responses to policy Vulnerability analysis, credential analysis non credential analysis.

UNIT-III:
Introduction to Snort, Snort Installation Scenarios, Installing Snort, Running Snort on Multiple Network Interfaces, Snort Command Line Options. Step-By-Step Procedure to Compile and Install Snort Location of Snort Files, Snort Modes Snort Alert Modes

UNIT-IV:
Working with Snort Rules, Rule Headers, Rule Options, The Snort Configuration File etc. Plugins, Preprocessors and Output Modules, Using Snort with MySQL.

UNIT-V:
Using ACID and Snort Snarf with Snort, Agent development for intrusion detection, Architecture models of IDs and IPs.

TEXT BOOKS:

1. Rafeeq Rehman : " Intrusion Detection with SNORT, Apache, MySQL, PHP and ACID," 1st Edition, Prentice Hall , 2003.

Reference Books:

1. Christopher Kruegel,Fredrik Valeur, Giovanni Vigna: "Intrusion Detection and Correlation Challenges and Solutions", 1st Edition, Springer, 2005.

2. Carl Endorf, Eugene Schultz and Jim Mellander " Intrusion Detection & Prevention", 1st Edition, Tata McGraw-Hill, 2004.

3. Stephen Northcutt, Judy Novak : "Network Intrusion Detection", 3rd Edition, New Riders Publishing, 2002.

4. T. Fahringer, R. Prodan, "A Text book on Grid Application Development and Computing Environment". 6th Edition, KhannaPublihsers, 2012.

Course Outcomes:

1. Students will be introduced to basic concepts of intrusion detection system.

2. Students will be able to understand Intrusion Prevention Systems, Network IDs protocol and model for intrusion analysis.

3. Students will be able to understand when, where, how, and why to apply Intrusion Detection tools and techniques in order to improve the security posture of an enterprise.

4. Students will be able to apply knowledge of the fundamentals and history of Intrusion Detection in order to avoid common pitfalls in the creation and evaluation of new Intrusion Detection Systems.

5. Students will be able to learn agent development for intrusion detection and architectural models of IDs and IPs.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF EMERGING TECHNOLOGIES

INDEX

UNIT-I

History of Intrusion Detection Systems

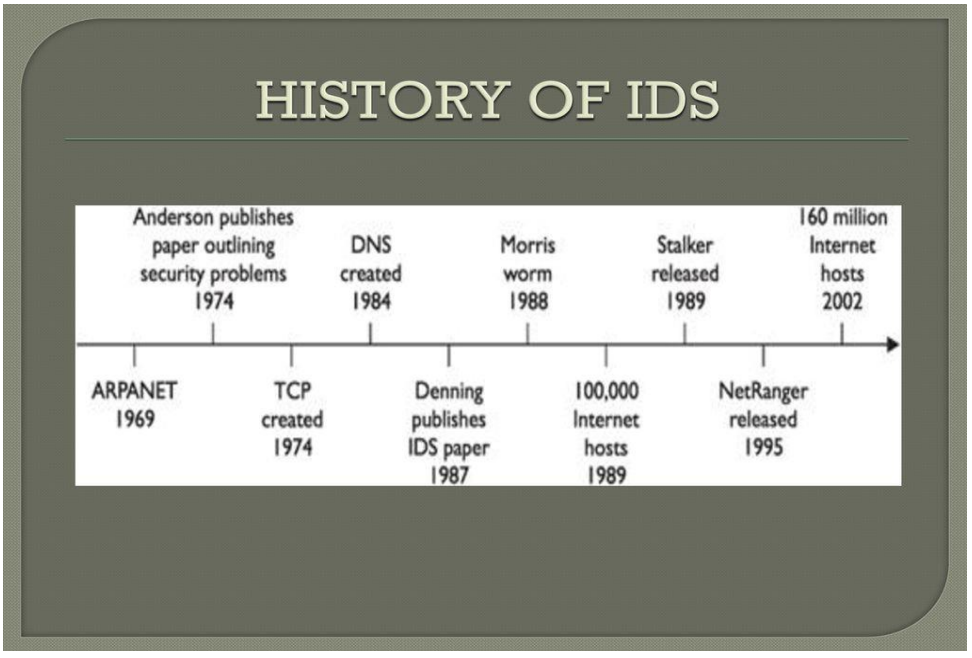
Originally, system administrators performed intrusion detection by sitting in front of a console and monitoring user activities. They might detect intrusions by noticing, for example, that a vacationing user is logged in locally or that a seldom-used printer is unusually active. Although effective enough at the time, this early form of intrusion detection was ad hoc and not scalable.

The next step in intrusion detection involved audit logs, which system administrators reviewed for evidence of unusual or malicious behavior. In the late '70s and early '80s, administrators typically printed audit logs on fan-folded paper, which were often stacked four- to five-feet high by the end of an average week. Searching through such a stack was obviously very time-consuming. With this overabundance of information and only manual analysis, administrators mainly used audit logs as a forensic tool to determine the cause of a particular security incident after the fact. There was little hope of catching an attack in progress.

As storage became cheaper, audit logs moved online and researchers developed programs to analyze the data. However, analysis was slow and often computationally intensive and therefore intrusion detection programs were usually run at night when the system's user load was low. Therefore, most intrusions were still detected after they occurred.

In the early '90s, researchers developed real-time intrusion detection systems that reviewed audit data as it was produced. This enabled the detection of attacks and attempted attacks as they occurred, which in turn allowed for real-time response, and, in some cases, attack preemption.

More recent intrusion detection efforts have centered on developing products that users can effectively deploy in large networks. This is no easy task, given increasing security concerns, countless new attack techniques, and continuous changes in the surrounding computing environment.



Intrusion Detection Overview:

The goal of intrusion detection is seemingly simple: to detect intrusions. However, the task is difficult, and in fact intrusion detection systems do not detect intrusions at all—they only identify evidence of intrusions, either while they're in progress or after the fact.

Such evidence is sometimes referred to as an attack's "manifestation." If there is no manifestation, if the manifestation lacks sufficient information, or if the information it contains is untrustworthy, then the system cannot detect the intrusion.

For example, suppose a house monitoring system is analyzing camera output that shows a person fiddling with the front door. The camera's video data is the manifestation of the occurring intrusion. If the camera lens is dirty or out of focus, the system will be unable to determine whether the person is a burglar or the owner.

Data collection issues

For accurate intrusion detection, we must have reliable and complete data about the target system's activities. Reliable data collection is a complex issue in itself. Most operating systems offer some form of auditing that provides an operations log for different users. These logs might be limited to the security-relevant events (such as failed login attempts) or they might offer a complete report on every system call invoked by every process. Similarly, routers and firewalls provide event logs for network activity. These logs might contain simple information, such as network connection openings and closings, or a complete record of every packet that appeared on the wire.

The amount of system activity information a system collects is a trade-off between overhead and effectiveness. A system that records every action in detail could have substantially degraded performance and require enormous disk storage. For example, collecting a complete log of a 100-Mbit Ethernet link's network packets could require hundreds of Gigabytes per day.

Collecting information is expensive, and collecting the right information is important. Determining what information to log and where to collect it is an open problem. For example, having your house alarm system monitor the water for pollution levels is an expensive activity that doesn't help detect burglars. On the other hand, if the house's threat model includes terrorist attacks, monitoring the pollution level might be reasonable.

Detection techniques

Auditing your system is useless if you don't analyze the resulting information. How intrusion detection systems analyze collected data is an important system characteristic.

There are two basic categories of intrusion detection techniques: anomaly detection and misuse detection.

Anomaly detection: Anomaly detection uses models of the intended behavior of users and applications, interpreting deviations from this "normal" behavior as a problem.

A basic assumption of anomaly detection is that attacks differ from normal behavior. For example, we can model certain users' daily activity (type and amount) quite precisely. Suppose a particular user typically logs in around 10 a.m., reads mail, performs database transactions, takes a break between noon and 1 p.m., has very few file access errors, and so on. If the system notices that this same user logs in at 3 a.m., starts using compilers and debugging tools, and has numerous file access errors, it will flag this activity as suspicious.

The main advantage of anomaly detection systems is that they can detect previously unknown attacks. By defining what's normal, they can identify any violation, whether it is part of the threat model or not. In actual systems, however, the advantage of detecting previously unknown attacks is paid for in terms of high false-positive rates. Anomaly detection systems are also difficult to train in highly dynamic environments.

Misuse detection. Misuse detection systems essentially define what's wrong. They contain attack descriptions (or "signatures") and match them against the audit data stream, looking for evidence of known attacks. One such attack, for example, would occur if someone created a symbolic link to a UNIX system's password file and executed a privileged application that accesses the symbolic link. In this example, the attack exploits the lack of file access checks. The main advantage of misuse detection systems is that they focus analysis on the audit data and typically produce few false positives. The main disadvantage of misuse detection systems is that they can detect only known attacks for which they have a defined signature. As new attacks are discovered, developers must model and add them to the signature database.

Response: After the intrusion

An intrusion detection system's *response* is its output or action upon detecting a problem. A response can take many different forms; the most common is to generate an alert that describes the detected intrusion.

There are also more aggressive responses, such as paging a system administrator, sounding a siren, or even mounting a counter-attack.

A counterattack might include reconfiguring a router to block the attacker's address or even attacking the culprit. Obviously, aggressive responses can be dangerous, since they could be launched against innocent victims. For example, a hacker can attack a network using *spoofed traffic*—traffic that appears to come from a certain address, but that is actually generated elsewhere. If the intrusion detection system detected the attack and reconfigured the network routers to block traffic from that address, it would effectively be executing a denial-of-service attack against the impersonated site.

Internal and external threats to data, attacks:

To protect your systems completely, you must first recognize who or what you're protecting them from. What typically comes to mind when discussing network security is protecting the network from mysterious hackers operating from a dark room full of sophisticated computer systems. This is rarely the case. According to the FBI, up to 80 percent (1999) of all security breaches reported are from internal sources. Internal security threats range from a novice server administrator or user who unknowingly installs software or opens an e-mail attachment to a disgruntled employee who attempts to delete source code from a development server.

To prepare for and defend against threats properly, you must first understand the types of threats to your network security. Four basic network security threats exist.

- Internal threats
- External threats
- Unstructured threats
- Structured threats

Internal Threats:

The term "internal attack" is used to describe an attack being implemented from a person or organization with some level of authorized access on your network. Internal attacks are performed from within the trusted area of the network. This type of threat can be more difficult to defend against because employees already have access to the network and private company data. To compound the internal threat further, most companies only have firewalls at the edge of their networks, and they rely strictly on access control lists (ACL) and server permission to regulate internal security. Server permissions typically protect resources located on the local servers, but provide little or no protection for the network. Internal threats are typically executed by disgruntled employee(s) who want to "get back" at the company.

Many, if not all, of the security measures are logically connected to the perimeter of the network, protecting the inside networks from the external connections, such as the Internet. While the perimeter of the network is secured, the inside or trusted portion of the network tends to be soft. Once an intruder has made it through the hard outer shell of the network, compromising one system after another is usually simple.

Wireless networks introduce a new area of concern for Security Administrators. Unlike cabled networks, wireless networks create a realm of coverage that can be intercepted and used by anyone with the right software and a wireless network adapter. Not only can all network data be viewed and recorded, but network attacks can also be launched from inside the network where the infrastructure is much more vulnerable. Because of the severe security implications, strong encryption should always be used with wireless networks.

External Threats:

External threats are posed by any organization, government, or individual that attempts to gain access from outside the company's network and includes anyone that doesn't have authorized access to the internal network. Typically, external attackers attempt to gain access from dialup servers or Internet connections. External threats are what companies spend the most time and money trying to prevent.

Need and types of IDS:

An Intrusion Detection System is software or a system that monitors network traffic and detects an intrusion or unwanted activities in the network. IDS scan the networks to find out if someone is trying to penetrate the network illegally. In other words, it keeps an eye on the network's traffic to identify intrusion in the network.

Intrusion Detection System if properly configured will help you to:

- Monitor inbound and outbound network traffic.
- Analyze the patterns in the network continuously.
- Send an alarm immediately after detecting unwanted intrusion and activities in the network.

Organizations must properly install IDS into their system. The IDS must analyze the normal traffic on the network. However, if IDS does not analyze the normal traffic properly then it might send a false alarm in case of intrusion.

Different types of Intrusion Detection Systems

Different types of Intrusion Detection systems are classified on the basis of different techniques and methods.

Network Intrusion Detection System (NIDS)

Network Intrusion Detection System sets up across the network at a specific planned point. NIDS monitors the traffic on the network from all devices. Similarly, it examines the traffic passing on the entire subnet and verifies it with the packet metadata and content. If NIDS detects any intrusion in the network, a warning alert is sent to the admin of that network. The best advantage of NIDS is that if it is installed in the same location where the firewall is located, then it will detect if someone is trying to attack the firewall. In other words, with the help of NIDS, the firewall will also be protected from any policy breaching.

Host Intrusion Detection System (HIDS)

Organizations install a Host Intrusion Detection System (HIDS) on independent networked devices. However, HIDS examines the incoming and outgoing traffic of the device only. It detects suspicious activities on the device and alerts the administrator. HIDS also checks whether system files are misplaced or not, for that it takes the screen capture of the current file system and verifies it with the screen capture of the previous file system. This file system stores the analytical information of network traffic. For instance, if the files are misplaced or changed it sends an alert to the administrator.

Protocol-based IDS (PIDS)
Organizations set up a Protocol-based Intrusion Detection System at the front end of the server. It interprets the protocols between the server and the user. PIDS monitors the HTTPS server regularly to secure the web. Similarly, it allows the HTTP server which is related to the protocol.

Application Protocol-based IDS (APIDS)
As we have seen that PIDS is set up at the front end of the server. Similarly, APIDS is set up within a group of servers. It interprets communication with the applications within the server to detect the intrusion.

Hybrid Intrusion Detection System
As the name says Hybrid Intrusion Detection system is a mixture of two different IDS. Hybrid System develops a network system by combining host agents with network information. In conclusion, Hybrid System is more responsive and effective as compared to other IDS.

Types of Intrusion Detection Systems Methods

There are 2 main Intrusion Detection methods to identify malicious attacks or intrusion. However, both these methods serve a different purpose and are not similar.

Signature-based Intrusion Detection Method

The IDS developed the Signature-based intrusion detection method to examine the network traffic and to detect attack patterns. For instance, it verifies the network traffic with the log data to identify the intrusion. If this method detects any intrusion then the IDS solution creates a signature of it and adds it to the list. The patterns which are detected are known as sequences and these sequences are a specific number of bytes or a set of 0's and 1's in the network. However, it is easy to detect the attacks whose patterns are existed in the system in the form of signatures. But to detect new attacks whose signature is not yet created is difficult.

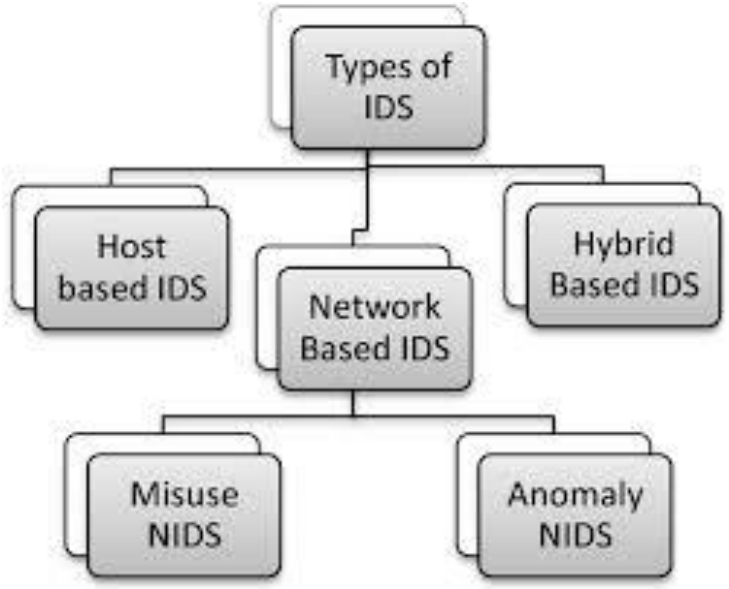
Anomaly-based Intrusion Detection Method

As we have seen that it is difficult to detect unknown or new malware attacks with the help of the Signature-Based Detection method. Therefore, organizations use the anomaly-based intrusion detection method to identify those new and unknown suspicious attacks and policy breaching which the Signature-based detection method cannot identify easily.

However, new intrusion techniques and malware are increasing rapidly. This method uses Machine learning to create an activity model. If this method detects any receiving patterns which are not found in the model, then the method declares these patterns as malicious patterns. In conclusion, the anomaly-based detection system is better in comparison to the Signature-based method.

Hybrid Detection Method

A Hybrid method uses both Signature and Anomaly-based intrusion detection methods together. However, the main reason behind the development of a hybrid detection system is to identify more potential attacks with fewer errors.



Top Intrusion Detection System Tools:

Solar Winds Security Event Manager

SolarWinds designed this tool with the implementation of HIDS and NIDS systems. It collects real-time log data from the network. Similarly, SEM is popular to build customizable intrusion detection methods which can automatically disable accounts and disconnect the devices attached to the network it detects any intrusion.

McAfee

McAfee designed its IDS to identify real-time malicious activities. It uses both signature and anomaly methods to identify the threats with emulation techniques. Therefore, McAfee is a scalable application.

Suricata

Suricata is a free intrusion detection tool. It is an open-source tool based on a network intrusion detection system (NIDS). Therefore, we use Suricata to detect identified threats and malicious activity in real-time. It uses a Signature-based method to identify the known threats or intrusion.

Blumira

Developers designed Blumira to detect threats and malicious activities across cloud services and on-premise devices. It can monitor the IT infrastructure continuously to detect any intrusion. Similarly, it is a SIEM platform that detects an in-progress attack and stops the attack.

Cisco Stealthwatch

Cisco designed Stealthwatch with NIDS and HIDS. In addition, it is compatible with Windows, Linux, and Mac OS operating systems. Cisco Stealthwatch is an intrusion detection system that does not require an agent enabling to grow business requirements. However, it uses machine learning to create baselines of patterns which is acceptable. One of the best features of this tool is that it can also detect intrusion and suspicious activities in the encrypted network without decrypting it.

Information sources Host based information sources:

Host-Based IDS

HIDS differ from NIDS in two ways. HIDS protects only the host system on which it resides, and its network card operates in no promiscuous mode. No promiscuous mode of operation can be an advantage in some cases, because not all NICs are capable of promiscuous mode. In addition, promiscuous mode can be CPU intensive for a slow host machine. HIDS can be run directly on the firewall as well, to help keep the firewall secure.

Another advantage of HIDS is the ability to tailor the rule set to a specific need. For example, there is no need to interrogate multiple rules designed to detect DNS exploits on a host that is not running Domain Name Services. Consequently, the reduction in the number of pertinent rules enhances performance and reduces processor overhead.

Host-based intrusion detection systems, commonly called HIDS, are used to analyze the activities on a particular machine. They have many of the same advantages as application level intrusion detection systems do, but on a somewhat reduced scale. A problem with host-based intrusion detection systems is that any information that they might gather needs to be communicated outside of the machine, if a central monitoring system is to be used. If the machine is being actively attacked, particularly in the case of a denial-of-service attack, this may not be possible.

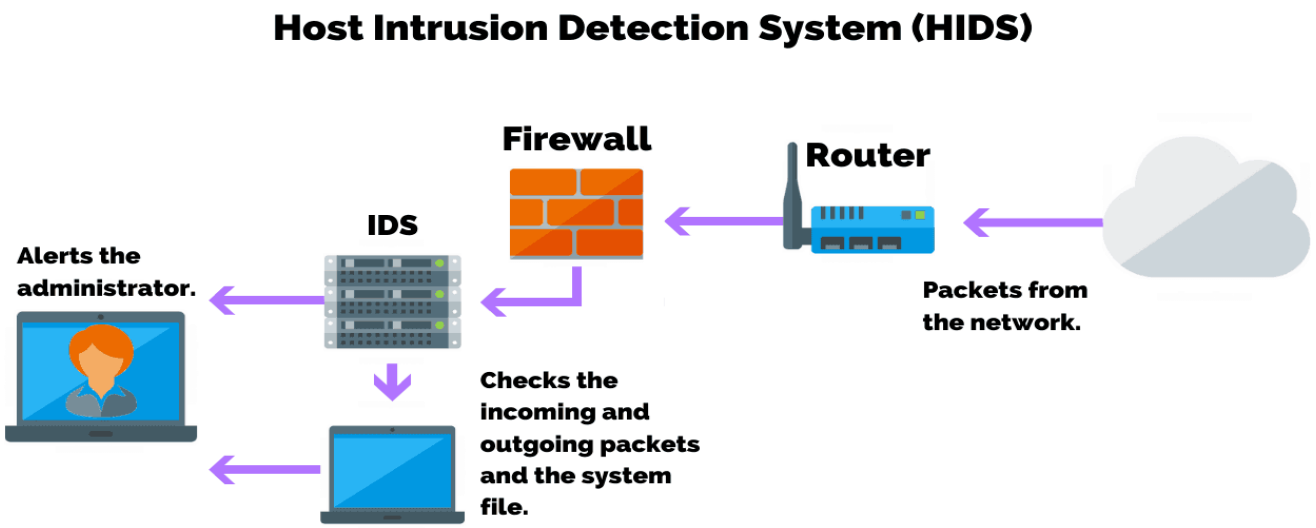
A common implementation of host-based IDS can be found in many of the antimalware products in use today. Many host-based IDSs depend on signature or string matching to detect threats, and they can be defeated by simply changing a tool enough that the signature no longer matches.

Host-Based IDS (HIDS)

Host-based systems were the first type of IDS to be developed and implemented. These systems collect and analyze data that originate on a computer that hosts a service, such as a Web server. Once this data is aggregated for a given computer, it can either be analyzed locally or sent to a separate/central analysis machine. One example of a host-based system is programs that operate on a system and receive application or operating system audit logs. These programs are highly effective for detecting insider abuses. Residing on the trusted network systems themselves, they are close to the network's authenticated users. If one of these users attempts unauthorized activity, host-based systems usually detect and collect the most pertinent information in the quickest possible manner. In addition to detecting unauthorized insider activity, host-based systems are also effective at detecting unauthorized file modification.

On the down side, host-based systems can get unwieldy. With several thousand possible endpoints on a large network, collecting and aggregating separate specific computer information for each individual machine may prove inefficient and ineffective. In addition, if an intruder disables the data collection on any given computer, the IDS on that machine will be rendered useless because there is no backup.

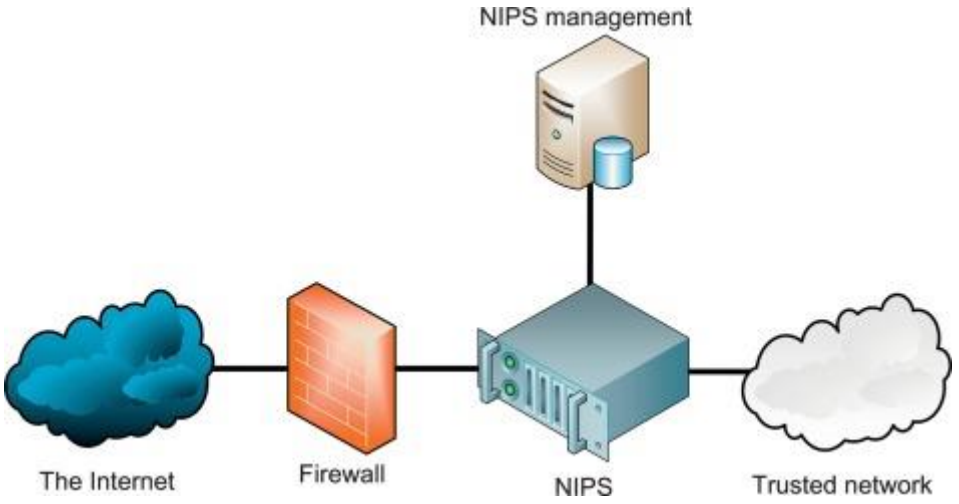
Possible host-based IDS implementations include Windows NT/2000 Security Event Logs, RDMS audit sources, Enterprise Management systems audit data (such as Tivoli), and UNIX Syslog in their raw forms or in their secure forms such as Solaris' BSM; host-based commercial products include RealSecure, ITA, Squire, and Intercept, to name a few.



Network-Based IDS (NIDS)

As opposed to monitoring the activities that take place on a particular network, Network-based intrusion detection analyzes data packets that travel over the actual network. These packets are examined and sometimes compared with empirical data to verify their nature: malicious or benign. Because they are responsible for monitoring a network, rather than a single host, Network-based intrusion detection systems (NIDS) tend to be more distributed than host-based IDS. Software, or appliance hardware in some cases, resides in one or more systems connected to a network, and is used to analyze data such as network packets. Instead of analyzing information that originates and resides on a computer, network-based IDS uses techniques like "packet-sniffing" to pull data from TCP/IP or other protocol packets traveling along the

network. This surveillance of the connections between computers makes network-based IDS great at detecting access attempts from outside the trusted network. In general, network-based systems are best at detecting the following activities:

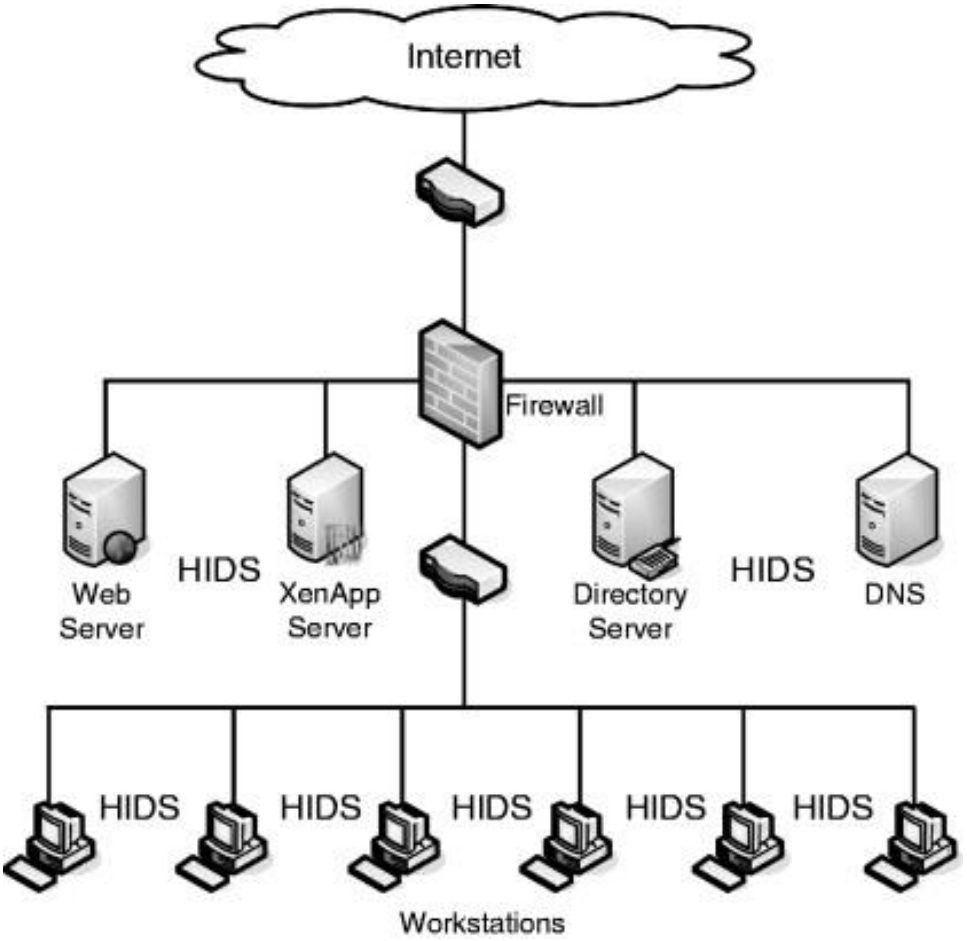


HIDS and NIDS Used in Combination

The two types of intrusion detection systems differ significantly from each other, but complement one another well. The network architecture of host-based is agent-based, which means that a software agent resides on each of the hosts that will be governed by the system. In addition, more efficient host-based intrusion detection systems are capable of monitoring and collecting system audit trails in real time as well as on a scheduled basis, thus distributing both CPU utilization and network overhead and providing for a flexible means of security administration.

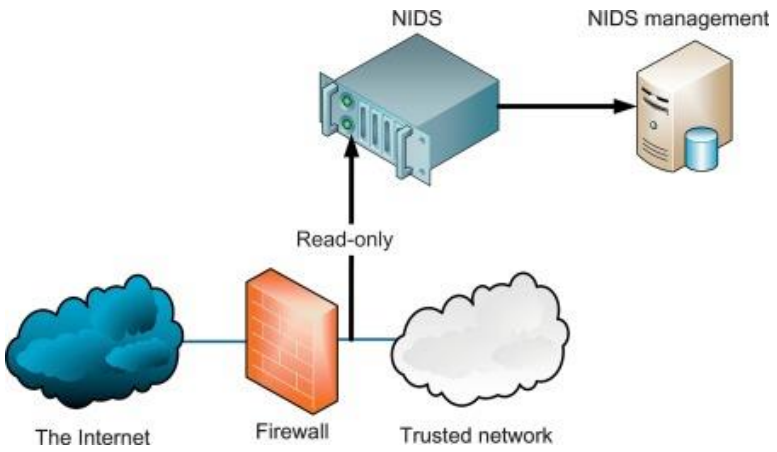
In a proper IDS implementation, it would be advantageous to fully integrate the network intrusion detection system, such that it would filter alerts and notifications in an identical manner to the host-based portion of the system, controlled from the same central location. In doing so, this provides a convenient means of managing and reacting to misuse using both types of intrusion detection.

That said, as an organization introduces an IDS into its network to augment its current information security strategy, the primary focus of the intrusion detection system should be host-based. Although network intrusion detection has its merits and certainly must be incorporated into a proper IDS solution, it has historically been incapable of evolving to comply with the growing technology of data communications. Most NIDS perform miserably, if at all, on switched networks, fast networks of speeds over 100 Mbps, and encrypted networks. Furthermore, somewhere in the range of 80 - 85 percent of security incidents originate from within an organization. Consequently, intrusion detection systems should rely predominantly on host-based components, but should always make use of NIDS to complete the defense. In short, a truly secure environment requires both a network and host-based intrusion detection implementation to provide for a robust system that is the basis for all of the monitoring, response, and detection of computer misuse.



NIDS and NIPS

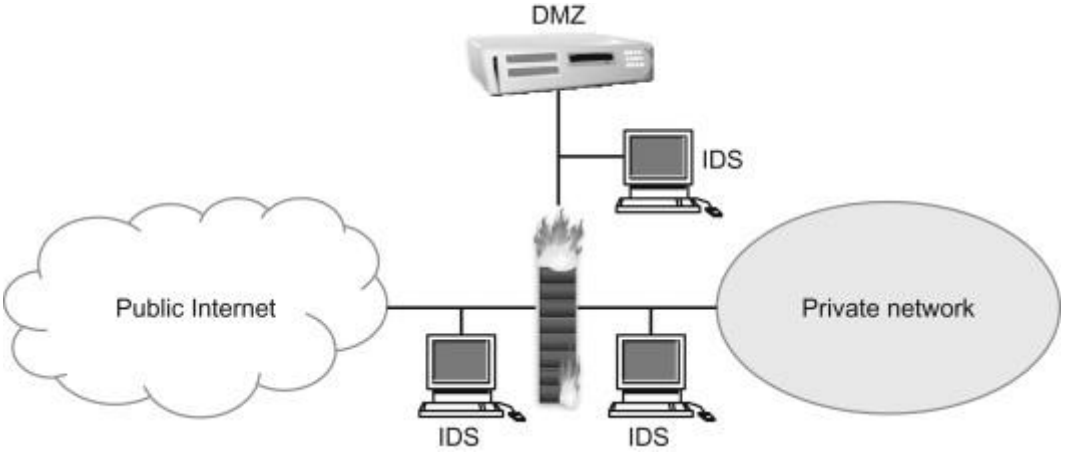
A network-based intrusion detection system (NIDS) detects malicious traffic on a network. NIDS usually require promiscuous network access in order to analyze all traffic, including all uncast traffic. NIDS are passive devices that do not interfere with the traffic they monitor; Fig. 7.2 shows a typical NIDS architecture. The NIDS sniffs the internal interface of the firewall in read-only mode and sends alerts to a NIDS Management server via a different (ie, read/write) network interface.



Network-based intrusion detection systems (NIDS) are devices intelligently distributed within networks that passively inspect traffic traversing the devices on which they sit. NIDS can be hardware or software-based systems and, depending on the manufacturer of the system, can attach to various network mediums such as Ethernet, FDDI, and others. Oftentimes, NIDS have two network interfaces. One is used for listening to network conversations in promiscuous mode and the other is used for control and reporting.

With the advent of switching, which isolates uncast conversations to ingress and egress switch ports, network infrastructure vendors have devised port-mirroring techniques to replicate all network traffic to the NIDS. There are other means of supplying traffic to the IDS such as network taps. Cisco uses Switched Port Analyzer (SPAN) functionality to facilitate this capability on their network devices and, in some network equipment, includes NIDS components directly within the switch. We'll discuss Cisco's IDS products in the next chapter.

While there are many NIDS vendors, all systems tend to function in one of two ways; NIDS are either signature-based or anomaly-based systems. Both are mechanisms that separate benign traffic from its malicious brethren. Potential issues with NIDS include high-speed network data overload, tuning difficulties, encryption, and signature development lag time. We'll cover how IDS work and the difficulties involved with them later in this section.

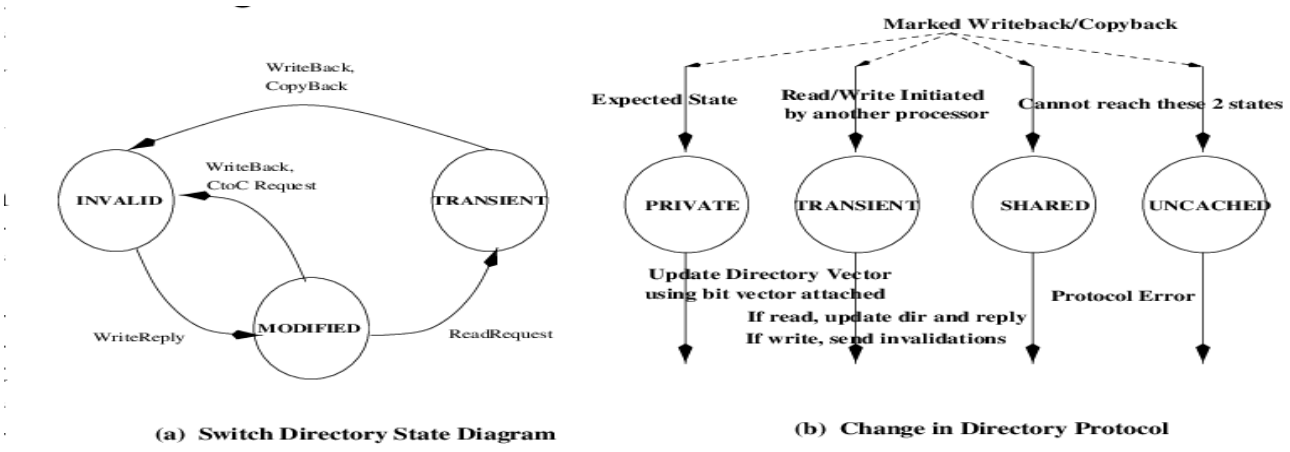


Protocol-based IDS (PIDS):

A **protocol-based intrusion detection system** (**PIDS**) is an intrusion detection system which is typically installed on a web server, and is used in the monitoring and analysis of the protocol in use by the computing system. A PIDS will monitor the dynamic behavior and state of the protocol and will typically consist of a system or agent that would typically sit at the front end of a server, monitoring and analyzing the communication between a connected device and the system it is protecting.

A typical use for a PIDS would be at the front end of a web server monitoring the HTTP (or HTTPS) stream. Because it understands the HTTP relative to the web server/system it is trying to protect it can offer greater protection than less in-depth techniques such as filtering by IP address or port number alone, however this greater protection comes at the cost of increased computing on the web server.

Where HTTPS is in use then this system would need to reside in the "shim" or interface between where HTTPS is un-encrypted and immediately prior to it entering the Web presentation layer.



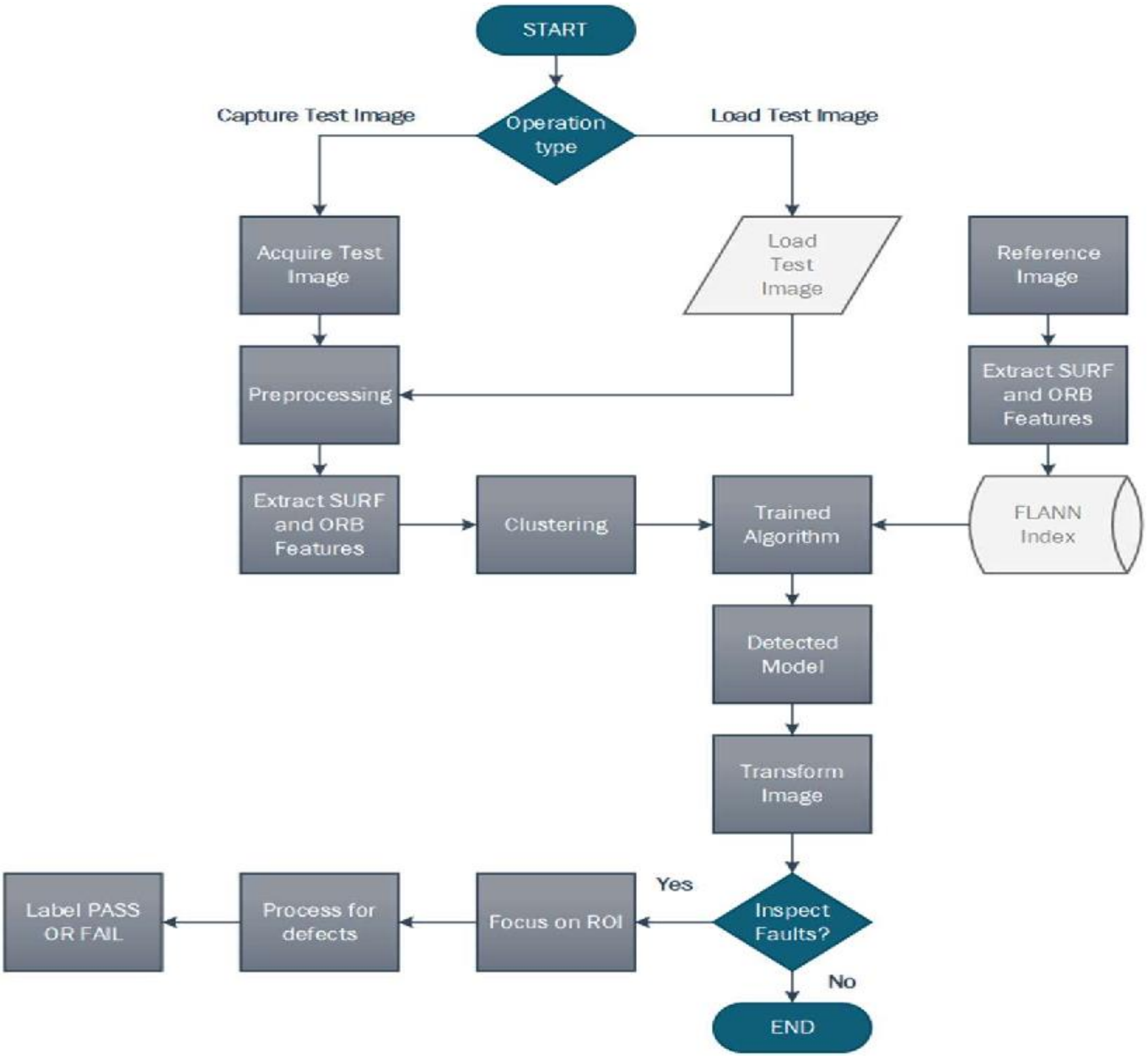
Application Protocol-based IDS (APIDS):

An **application protocol-based intrusion detection system** (**APIDS**) is an intrusion detection system that focuses its monitoring and analysis on a specific application protocol or protocols in use by the computing system.

At a basic level an APIDS would look for, and enforce, the correct (legal) use of the protocol.

However at a more advanced level the APIDS can learn, be taught or even reduce what is often an infinite protocol set, to an acceptable understanding of the subset of that application protocol that is used by the application being monitored / protected.

Thus, an APIDS, correctly configured, will allow an application to be "fingerprinted", thus should that application be subverted or changed, so will the fingerprint change.



UNIT – II

Intrusion Prevention Systems: An intrusion prevention system (IPS) is a form of network security that works to detect and prevent identified threats. Intrusion prevention systems continuously monitor your network, looking for possible malicious incidents and capturing information about them. The IPS reports these events to system administrators and takes preventative action, such as closing access points and configuring firewalls to prevent future attacks. IPS solutions can also be used to identify issues with corporate security policies, deterring employees and network guests from violating the rules these policies contain.

With so many access points present on a typical business network, it is essential that you have a way to monitor for signs of potential violations, incidents and imminent threats. Today's network threats are becoming more and more sophisticated and able to infiltrate even the most robust security solutions.

How Do Intrusion Prevention Systems Work?

Intrusion prevention systems work by scanning all network traffic. There are a number of different threats that an IPS is designed to prevent, including:

- Denial of Service (DoS) attack
- Distributed Denial of Service (DDoS) attack
- Various types of exploits
- Worms
- Viruses

The IPS performs real-time packet inspection, deeply inspecting every packet that travels across the network. If any malicious or suspicious packets are detected, the IPS will carry out one of the following actions:

- Terminate the TCP session that has been exploited and block the offending source IP address or user account from accessing any application, target hosts or other network resources unethically.
- Reprogram or reconfigure the firewall to prevent a similar attack occurring in the future.
- Remove or replace any malicious content that remains on the network following an attack. This is done by repackaging payloads, removing header information and removing any infected attachments from file or email servers.

HOW DO INTRUSION PREVENTION SYSTEMS DETECT MALICIOUS ACTIVITY?

Intrusion prevention systems have various ways of detecting malicious activity, however the two predominant methods are signature-based detection and statistical anomaly-based detection. The signature-based detection method used by intrusion prevention systems involves a dictionary of uniquely identifiable signatures located in the code of each exploit. There are two types of signature-based detection methods for intrusion prevention systems as well: exploit-facing and vulnerability-facing. Exploit-facing methods detect malicious activity based on common attack patterns, whereas vulnerability-facing methods attempt to detect malicious activity by identifying specific vulnerabilities. On the other hand, intrusion prevention systems that rely on statistical anomaly-based detection randomly sample network traffic and then compare the samples to a predetermined baseline performance level.

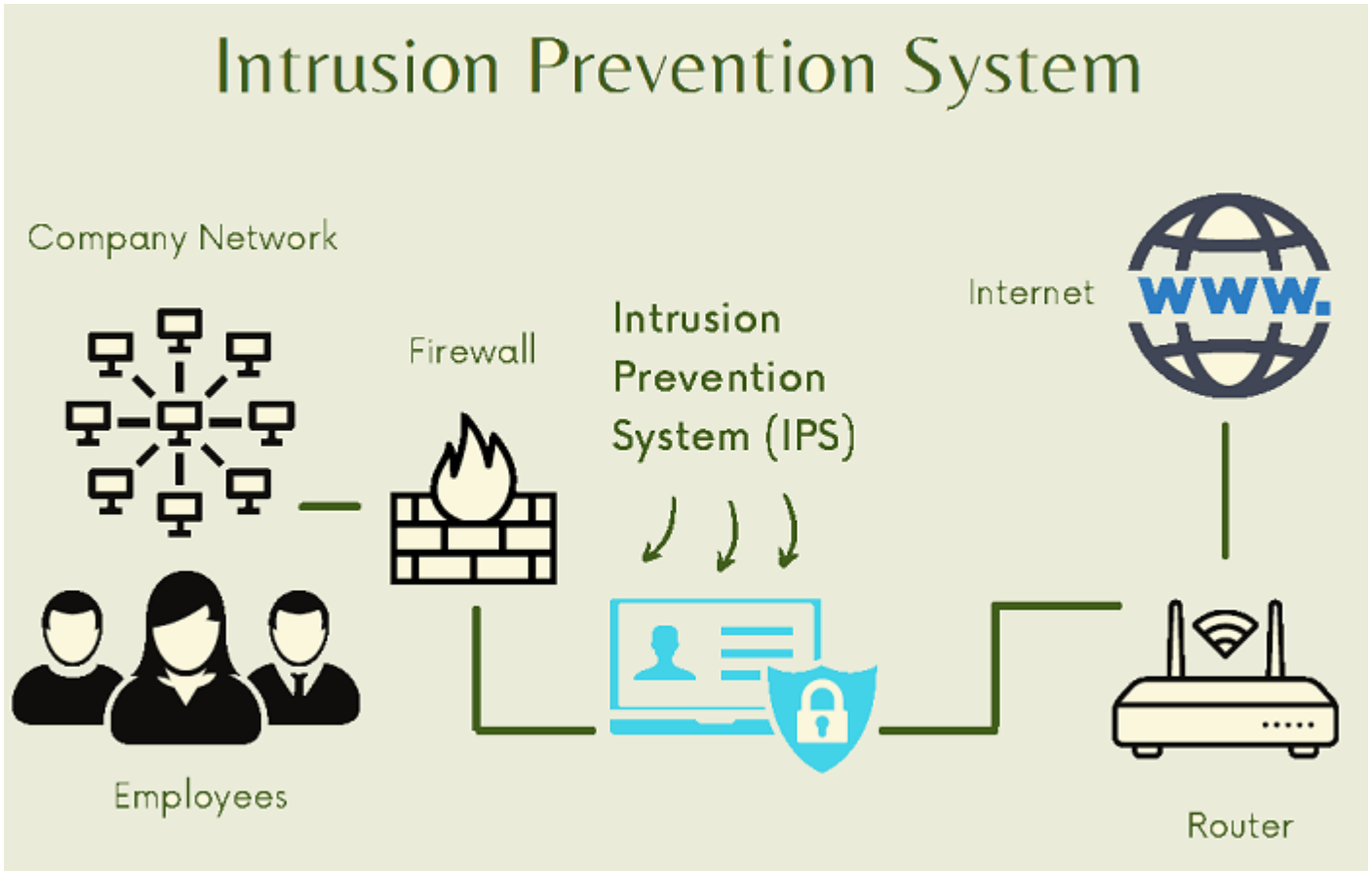
Types of Prevention:

An intrusion prevention system is typically configured to use a number of different approaches to protect the network from unauthorized access. These include:

- Signature-Based - The signature-based approach uses predefined signatures of well-known network threats. When an attack is initiated that matches one of these signatures or patterns, the system takes necessary action.
- Anomaly-Based - The anomaly-based approach monitors for any abnormal or unexpected behavior on the network. If an anomaly is detected, the system blocks access to the target host immediately.
- Policy-Based - This approach requires administrators to configure security policies according to organizational security policies and the network infrastructure. When an activity occurs that violates a security policy, an alert is triggered and sent to the system administrators.

INTRUSION PREVENTION SYSTEM COMPARISON:

There are four common types of intrusion prevention systems. The first type of intrusion prevention system is called a network-based intrusion prevention system (NIPS). This type of intrusion prevention system has the ability to monitor the whole network and look for suspicious traffic by reviewing protocol activity. In contrast, wireless intrusion prevention systems (WIPS) only monitor wireless networks for suspicious activity by reviewing wireless networking protocols. A third type of intrusion prevention system is called network behavior analysis (NBA). Network behavior analysis looks at network traffic in an effort to locate threats that cause unusual traffic flows, including distributed denial of service (DDoS) attacks and policy violations. The last common type of intrusion prevention system is host-based intrusion prevention systems (HIPS). A host-based intrusion prevention system is an installed software package that looks into suspicious activity that occurs within a single host.



Intrusion Detection Systems

Network IDS:

A **Network Intrusion Detection System (NIDS)** is a computer software application that can detect and report network security problems by monitoring network or system activities for malicious or anomalous behavior.

NIDS works by examining a variety of data points from different sources within the network. Packet headers, statistics, and protocol/application data flows are analyzed to determine whether malicious or anomalous activity has taken place. It can be used to identify possible security breaches on a system including sniffers and attacks on services such as HTTP/S, SMB, SSH etc.

There are two types of Network Intrusion Detection Systems:

Network sensors

Network sensors are often dedicated devices or applications that run exclusively as NIDS. The sensor monitors and analyzes network traffic for malicious behavior. The sensor can be located at various points on the network, depending on where it is needed. For example, a router may have a sensor installed to monitor traffic that passes through the router or a switch could have a sensor that monitors traffic as it passes from one port to another.

Host-based intrusion detection systems

For this type of system, the sensor is software that monitors network traffic from within a single host on the network. In most cases, a host-based IDS is used only to monitor traffic within the local host or a particular service or application. However, in some cases, it may monitor packets as they pass through a firewall from one network to another or it could monitor activity on an entire host running multiple services and applications at once.

Technologies that can be monitored by NIDS

Network Intrusion Detection Systems use one or more technologies to analyze for threats on your network. These can include:

Packet headers

Packet headers contain specific information about the packet being transmitted across your network. Information in the header can include source and destination IP addresses, ports, protocol types, etc. The NIDS will analyze this information for suspicious activity or malicious behavior.

Packets/transmissions

The total packets per second are a common technology used by a NIDS to monitor for threats on your network. This may be a configuration option that you specify when installing a traffic monitoring system on your network. IDS can compare normal traffic rates, with those being transmitted at any one time across the network, to detect anything out of the ordinary. For example, if there is no heavy traffic on the network, but packets are still being transmitted at a high rate of speed, this could indicate suspicious activity.

Protocols and applications

Network Intrusion Detection Systems use various types of protocols to monitor for threats on your network. These can include:

- **Packet protocols**. TCP/IP, UDP/IP, ICMP etc.
- **Anomaly-based protocols**. This is where an IDS has been programmed to detect anomalies in protocols that are otherwise benign when working normally. For example, if you had a specific

protocol that was known to have 50% packet loss during normal operation and a packet loss percentage significantly different from the norm is detected, this would trigger an alarm or alert enabling you to investigate the problem further.

- **Data flow analysis**. NIDS can analyze data flow throughout the network to determine where a problem may be taking place. For example, if a user suddenly begins transmitting a large amount of data, an IDS will recognize this and alert you to possible security breaches occurring on your network.

Common types of network intrusion detection systems

There are five common types of NIDS that can be used to monitor traffic on your network. Each has its own benefits and drawbacks depending on your business needs:

Signature-based system

This type of NIDS uses signatures from previously analyzed attacks. It learns which patterns indicate malicious activity so future events with similar characteristics will be detected immediately. Signature-based systems do not need any knowledge about the normal behavior of users or applications to operate.

Stateful protocol analysis system

This type of NIDS is similar to a signature-based system in that it learns which patterns indicate malicious activity. Stateful protocol analysis systems differ because they do not need to know what specific attacks look like before they are detected. Instead, it can maintain temporary information about how your network normally operates and will compare new events against the normal traffic rate of existing connections.

Behavioral-based system

This type of NIDS uses behavioral analysis to determine whether any suspicious activity has occurred. If the behavior being analyzed meets certain conditions set by the administrator, an alert will be triggered so appropriate action can be taken in response to malicious activity.

Anomaly-based system

This type of NIDS is similar to the behavior-based system, except that it learns what typical network behavior looks like by analyzing how real connections are established and used over time. The administrator may also need to provide information about which events should trigger alerts if anomalies are detected. This type of system is configured to learn what the normal traffic on your network looks like, which can reduce false-positive rates, however, changes in user computer activity or changes made by new software installations could also trigger false alarms.

Heuristic-based system

This type of NIDS uses heuristics to look beyond attacks with known signatures and analyze them against a set of rules to determine whether any suspicious activity has occurred. The heuristic-based system is capable of detecting advanced attacks without previously knowing what those attacks look like by looking for a combination of characteristics that indicate a possible security issue.

Advantages and disadvantages of network intrusion detection system

There are several benefits and drawbacks associated with deploying a Network Intrusion Detection System on your organization's network. Some advantages include:

- **Detects known and unknown malware**. A NIDS can be configured to detect common types of malware, as well as new or unknown threats, so you will quickly know when hackers have compromised your systems.

- **Reduces downtime**. Once an intrusion is detected, NIDS immediately shuts down the process and alerts you so you can react quickly to stop further damage.
- **Prevents attacks.** The NIDS constantly monitors network traffic to identify suspicious activity and block it before hackers are able to gain access to your system.
- **Detects compromised devices.** A NIDS can detect when a user's computer has been compromised so, the attacker cannot gain access to other machines on the network or use the compromised machine as an attack vector into other parts of your business's information technology infrastructure.

Disadvantages associated with deploying the Network Intrusion Detection System include:

- **Requires frequent updating**. It is important to update your NIDS regularly so it will continue to recognize known threats and keep up with new ones. There are several ways to perform this update; updates are essential to the success of your NIDS.
- **Requires extensive configuration**. To be most effective, a NIDS must be configured with information about how your network normally operates and what types of activities should trigger an alert. This can require some effort on your part but will ensure that you receive alerts for suspicious behavior or malware only after it has been detected.
- **Requires maintenance**. Many systems require manual updating and configuration and therefore need constant management by IT staff to be most effective. If you do not have dedicated IT resources available to maintain the system, it may need to be removed from your network until these resources become available.

Network intrusion detection system vs. network intrusion prevention system (NIPS)

A NIDS is a passive system that compares the current network traffic against known malware signatures. In contrast, a NIPS actively analyzes the network traffic in real-time and blocks any suspicious activities. It can be configured to prevent an intruder from gaining access to your private information even if it doesn't have a complete understanding of all possible security threats.

Network intrusion detection system vs. firewall

A firewall is a network security system that controls the incoming and outgoing network traffic by monitoring which computer or IP address is allowed to access other computers on your network. A NIDS analyzes the data packets that are transmitted over your business's network to identify possible cyber-attacks or malicious activities.

While both systems monitor your private information networks looking for suspicious activity, they do it in different ways. A NIDS performs continuous analysis of all traffic passing through your business's network looking for known malware signatures, whereas a firewall denies access to specific users and/or IP addresses trying to access your network.

Network intrusion detection system vs. host-based intrusion prevention systems

A host-based intrusion prevention system monitors and blocks suspicious activity that is taking place on a single computer, whereas a NIDS looks for unusual or suspicious activity across all your business's computers, servers, and other devices in real-time to identify potential attacks against the entire network. In addition, a NIDS can be configured to automatically react to an attack by shutting down processes, blocking access from compromised machines, and alerting IT staff to the possible presence of malware.

Network intrusion detection system vs. virus protection

Virus protection software identifies and eliminates computer viruses after they have been downloaded onto your system, whereas NIDS monitor and analyze data packets as they pass through the network to identify suspicious activities that may indicate a security breach.

Network intrusion detection system vs. anti-virus software

Both anti-virus software and NIDS work together to automatically scan all incoming and outgoing data and compare it against known malware signatures. At first glance, you might think that these two products do the same thing, but there are subtle differences between them:

Anti-virus software is designed to protect single host computers from attack by locating specific types of malware on those computers; it scans binaries for known malware signatures and flags them as either safe or infected with malware. A NIDS analyzes all the data packets passing through your business's network to identify signs of an attack; it monitors the network traffic looking for patterns that may indicate suspicious activities such as port scanning or brute force attacks against common services like FTP or Telnet using default usernames and passwords.

A major difference between anti-virus software and NIDS is how they work in practice. Anti-virus software relies on you to update it regularly so it can detect new viruses, whereas most NIDS products are updated automatically overnight without requiring intervention from the user.

Network intrusion detection system vs. anomaly-based intrusion detection system (ABIDS)

An anomaly-based intrusion detection system (ABIDS) works in much the same way that a NIDS does, but it uses statistical analysis to identify unusual activity instead of using signatures to flag suspicious traffic. This form of IDS is most effective against zero-day attacks because it looks for data patterns rather than known malware signatures. ABIDS analyzes all activity taking place on your network and identifies anomalous behavior, whereas NIDS analyzes only network traffic looking for signs of known malicious activities.

ABIDS must process all network data before any activity is flagged as anomalous or suspicious, whereas NIDS only processes the packets that are potentially malicious. Since this form of IDS is more proactive in its monitoring of your network traffic, it can sometimes be more of a drain on your business's resources than a NIDS.

Network intrusion detection system vs. anomaly-based intrusion prevention system

This form of IDS works in the same way that ABIDS do, but instead of generating alerts they automatically react to anomalies by blocking suspicious activities and shutting down compromised processes on your computers much like a host-based intrusion prevention system would. Since it does not rely on signatures to identify malware it is typically more effective at preventing zero-day attacks because it can react to any suspicious activities detected on your network.

The disadvantage of this form of IDS is that if a false positive occurs, legitimate traffic could be blocked, or processes shut down unnecessarily. This means that its accuracy should be carefully monitored and configured by an experienced security specialist so as not to result in too many false positives which would impact the performance of your business's computer systems.

Frequently asked questions about network intrusion detection systems

Can a network intrusion detection system tell if a host is infected?

A NIDS cannot detect whether a host has been infected or not. They can, however, help ensure that any anomaly on the network or system level is caught and reported. For example, an IDS would be able to pick up unusual traffic from a host that is suspected to have been compromised without being affected by it themselves.

Where does a network intrusion detection system send its logs?

A NIDS sends its logs directly to the Security Information and Event Management (SIEM) system, syslog servers, or other data input sources depending on how it is configured.

What type of data does a network intrusion detection system collect?

A nids can typically detect attacks whether they are occurring on the network or not and will store information such as source/destination ip addresses, time stamps, packet details for each event that is detected. This also includes failed logins or other activity that breaches security policy.

Can a network intrusion detection system replace an inline prevention system (IPS)?

No, a NIDS cannot be used as a substitute for an IPS. While both systems perform different functions it's impossible to combine them into one device because each serves its own purpose. A NIDS stays in line with traffic allowing it to inspect every packet on the network, while an IPS acts against detected attacks.

Are network intrusion detection systems easy to manage and deploy?

Yes, most networks are already set up to send logs directly to a SIEM or other data inputs. Installing the additional components necessary for a NIDS is usually straightforward and does not require any major changes to the network's configuration. Although, there are varying degrees of complexity depending on how much security is being implemented. For example, adding IPS capability requires configuring the IDS sensors correctly so that they do not generate too many false positives.

Are network intrusion detection systems hard to install?

No, setting up a NIDS is typically easy even for beginners because it often only requires following simple guidelines provided by the manufacturer during installation. After all mandatory components are installed, you can begin loading your network's packet captures to the appliance.

How many network intrusion detection systems does it take to monitor a network?

Since it may be difficult to determine the actual number of NIDS required for your network, many suggest monitoring as much of the network traffic as possible. This can be done by placing IDS sensors in critical choke points throughout your entire infrastructure or distributing them out evenly across subnets which are then linked up through a SIEM system that centralizes reports generated by each appliance.

Does a network intrusion detection system affect performance?

A NIDS does not usually have any effect on network performance unless there is extremely high traffic. If an alert happens at peak traffic times you might notice some slowdown, but if you input the information at a less busy time it should have almost no effect on performance. If this happens at peak traffic times you might notice some slowdown but if you input the information at a less busy time it should have almost no effect on performance.

How does a network intrusion detection system protect my organization?

A NIDS will help protect your organization by analyzing and detecting malicious network activity. Additionally, they are capable of monitoring user accounts, file integrity, firewall logs, database server log files etc. for signs of suspicious activity, and can alert the security administrator when necessary.

IDPS Analysis Schemes:

Analysis

In the context of intrusion detection & prevention, analysis is the organization of the constituent parts of data and their relationships to identify any anomalous activity of interest. Real time analysis is analysis done on the fly as the data travels the path to the network or host. The fundamental goal of intrusion-detection & prevention analysis is to improve an information system's security.

This goal can be further broken down:

- Create records of relevant activity for follow-up.
- Determine flaws in the network by detecting specific activities.
- Record unauthorized activity for use in forensics or criminal prosecution of intrusion attacks.
- Act as a deterrent to malicious activity.
- Increase accountability by linking activities of one individual across system.

Anatomy of Intrusion Analysis:

There are many possible analysis schemes but in order to understand them, the intrusion process can be broken down into following four phases:

- Preprocessing
- Analysis
- Response
- Refinement

Pre-Processing

Preprocessing is the key function once the data is collected from IDPS sensor. The data is organized in some fashion for classification. The preprocessing helps in determining the format the data are put into, which is usually some canonical format or could be a structured database. Once the data are formatted, they are broken down further into classifications.

These classifications can depend on the analysis schemes being used. For example, if rule-based detection is being used, the classification will involve rules and patterns descriptors. If anomaly detection is used, then statistical profile based on different algorithms in which the user behavior is baseline over the time and any behavior that falls outside of that classification is flagged as an anomaly.

Upon completion of the classification process, the data is concatenated and put into a defined version or detection template of some object by replacing variables with values. These detection templates populate the knowledgebase which are stored in the core analysis engine.

Analysis

Once the processing is completed, the analysis stage begins. The data record is compared to the knowledge base, and the data record will either be logged as an intrusion event or it will be dropped. Then the next data record is analyzed. The next phase is *response.*

Response

Once information is logged as an intrusion, a response is initiated. The inline sensor can provide real time prevention through an automated response. Response is specific to the nature of the intrusion or the different analysis schemes used. The response can be set to be automatically performed or it can be done manually after someone has manually analyzed the situation.

Refinement

The final phase is the refinement stage. This is where the fine tuning of the system is done, based on the previous usage and detected intrusions. This gives the opportunity to reduce false-positive levels and to have a more accurate security tool.

Analysis Process By Different Detection Methods

The intrusion analysis process is solely depends on the detection method being used. Following is the information regarding the four phases of intrusion analysis by different detection methods:

Analysis Process By Rule-Based Detection

Rule-based detection, also known as *signature detection, pattern matching and misuse detection*. Rule-based detection uses pattern matching to detect known attack patterns. The four phases of intrusion analysis process applied in rule-based detection system are as under:

- Preprocessing: The data is collected about the intrusions, vulnerabilities and attacks and then it is putted down into classification scheme or pattern descriptors. From the classification scheme a behavior model is built and then into a common format;
- Signature Name: The given name of the signature
- Signature ID: The unique ID for the signature
- Signature Description: The description of the signature & what it does
- Possible False Positive Description: An explanation of any "false positives" that may appear to be an exploit but are actually normal network activity.
- Related Vulnerability Information: This field has any related vulnerability information

The pattern descriptors are typically either content-based signatures, which examine the payload and header of packet, or context-based signatures that evaluate only the packet headers to identify an alert. The pattern descriptors can be atomic (single) or composite (multiple) descriptors. Atomic descriptor requires only one packet to be inspected to identify an alert, while composite descriptor requires multiple packets to be inspected to identify an alert. The pattern descriptors are then put into a knowledge base that contains the criteria for analysis.

- Analysis: The event data are formatted and compared against the knowledge base by using pattern-matching analysis engine. The analysis engine looks for defined patterns that are known as attacks.
- Response: If the event matches the pattern of an attack, the analysis engine sends an alert. If the event is partial match, the next event is examined. Partial matches can only be analyzed with a stateful detector, which has the ability to maintain state, as many IDS systems do. Different responses can be returned depending on the specific event records.
- Refinement: Refinement of pattern-matching analysis comes down to updating signatures, because an IDS is only as good as its signature update.

Analysis Process By Profile-Based Detection (Anomaly Detection)

An anomaly is something that is different from the norm or that cannot be easily classified. Anomaly detection, also referred to as Profile-based detection, creates a profile system that flags any events that strays from a normal pattern and passes this information on to output routines. The analysis process by profile-based detection is as following:

- Preprocessing: The first step in the analysis process is collecting the data in which behavior considered normal on the network is baseline over a period of time. The data are put into a numeric form and then formatted. Then the information is classified into a statistical profile that is based on different algorithms is the knowledge base.
- Analysis: The event data are typically reduced to a profile vector, which is then compared to the knowledge base. The contents of the profile vector are compared to a historical record for that particular user, and any data that fall outside of the baseline of normal activity is labeled as deviation.
- Response: At this point, a response can be triggered either automatically or manually.
- Refinement: The profile vector history is typically deleted after a specific time. In addition, different weighting systems can be used to add more weight to recent behavior than past behaviors.

Types of responses mapping responses to policy Vulnerability analysis:

What is a vulnerability assessment?

A vulnerability assessment is the process of defining, identifying, classifying and prioritizing vulnerabilities in computer systems, applications and network infrastructures. Vulnerability assessments also provide an organization with the necessary knowledge, awareness and risk backgrounds to understand and react to threats to its environment. A vulnerability assessment process is intended to identify threats and the risks they pose. They typically involve the use of automated testing tools, such as network security scanners, whose results are listed in a vulnerability assessment report.

Organizations of any size, or even individuals who face an increased risk of cyber attacks, can benefit from some form of vulnerability assessment, but large enterprises and other types of organizations that are subject to ongoing attacks will benefit most from vulnerability analysis.

Because security vulnerabilities can enable hackers to access IT systems and applications, it is essential for enterprises to identify and remediate weaknesses before they can be exploited. A comprehensive vulnerability assessment, along with a management program, can help companies improve the security of their systems.

Importance of vulnerability assessments

A vulnerability assessment provides an organization with details on any security weaknesses in its environment. It also provides direction on how to assess the risks associated with those weaknesses. This process offers the organization a better understanding of its assets, security flaws and overall risk, reducing the likelihood that a cybercriminal will breach its systems and catch the business off guard.

Types of vulnerability assessments

Vulnerability assessments discover different types of system or network vulnerabilities. This means the assessment process includes using a variety of tools, scanners and methodologies to identify vulnerabilities, threats and risks.

Some of the different types of vulnerability assessment scans include the following:

- **Network-based scans** are used to identify possible network security attacks. This type of scan can also detect vulnerable systems on wired or wireless networks.
- **Host-based scans** are used to locate and identify vulnerabilities in servers, workstations or other network hosts. This type of scan usually examines ports and services that may also be visible to network-

based scans. However, it offers greater visibility into the configuration settings and patch history of scanned systems, even legacy systems.

- **Wireless network scans** of an organization's Wi-Fi networks usually focus on points of attack in the wireless network infrastructure. In addition to identifying rogue access points, a wireless network scan can also validate that a company's network is securely configured.
- **Application scans** test websites to detect known software vulnerabilities and incorrect configurations in network or web applications.
- **Database scans** can identify weak points in a database to prevent malicious attacks, such as SQL injection attacks.

Vulnerability assessments vs. penetration tests

A vulnerability assessment often includes a penetration testing component to identify vulnerabilities in an organization's personnel, procedures or processes. These vulnerabilities might not normally be detectable with network or system scans. The process is sometimes referred to as vulnerability assessment/penetration testing, or VAPT.

Intrusion Detection Systems

Credential analysis non credential analysis:

Vulnerabilities such as the above occur due to either coding errors or security misconfiguration, which the attackers quickly exploit to access system memory, execute insidious commands, steal data, etc. Now, with malicious programs scanning the networks for known vulnerabilities, traditional defenses offered by passwords, SSL and data-encryption, firewalls, and standard scanning programs may not be enough. Only by undertaking a robust regimen of vulnerability assessment (VA) can businesses safeguard their data and networks.

Vulnerability may rise due to myriad factors, from a weak password on a router to an unpatched programming flaw. By undertaking a vulnerability assessment (VA) exercise, organizations can validate security measures from within a network firewall (internal) and check defenses from without (external). It would involve bringing under the scanner all assets such as servers, firewalls, load balancer, network devices, and desktops among others. Experts agree on the need to undertake internal and external VAs every quarter, or whenever a network entity undergoes a critical change. The VA report provides details on the identified vulnerabilities on the network, potential risks, and suggested mitigation measures.

With VA becoming a critical security strategy component, it has become mandatory for organizations to implement it at the risk of losing their PCI compliance certification. Today, organizations run VA scans to assess their network, run patch installation, and re-assess the environment for its durability. This cycle of assess-patch-re-assess is now the standard for organizations to manage their security issues. In fact, organizations have integrated VA into their system rollout process in such a manner that the assessment is triggered whenever a new server is installed.

Credential-based vs. Non-credential Vulnerability Assessment:

There are two kinds of vulnerability assessments: *credentialed* and *non-credentialed* (also known as authenticated and unauthenticated scans).

Credential-based vulnerability assessments, which make use of the admin account, do a more thorough check by looking for problems that cannot be seen from the network. On the other hand, ***non-credentialed scans*** provide a quick view of vulnerabilities by only looking at network services exposed by the host.

Unfortunately, non-credentialed scans do not provide deeper insight into application and operating system vulnerabilities not exposed to the network, or those vulnerabilities behind a firewall. It provides a false hope that the system is safe.

When it comes to the credential-based vulnerability assessment, maintaining an accurate list of all credentials is a big worry. An inaccurate list is one of the main reasons why security teams have a hard time completing credentialed scans. For instance, in large organizations, it is not always possible to track down owners of specific assets; in some cases, even asking the asset owner for their credentials may run in trouble and even be prohibited by company policy.

Nevertheless, a credential test performs a dry run of all credentials and then reports on the successfully authenticated hosts and the unsuccessful ones. This enables security teams to quickly identify and resolve credential issues. In fact, it prevents the security teams from performing VAs that may encounter errors or provide inaccurate or incomplete information due to incorrectly configured credentials.

UNIT – III

Introduction to Snort:

Snort is the foremost Open Source Intrusion Prevention System (IPS) in the world. Snort IPS uses a series of rules that help define malicious network activity and uses those rules to find packets that match against them and generates alerts for users.

Snort can be deployed in line to stop these packets, as well. Snort has three primary uses: As a packet sniffer like tcpdump, as a packet logger — which is useful for network traffic debugging, or it can be used as a full-blown network intrusion prevention system. Snort can be downloaded and configured for personal and business use alike.

SNORT is a network based intrusion detection system which is written in C programming language. It was developed in 1998 by Martin Roesch. Now it is developed by Cisco. It is free open-source software. It can also be used as a packet sniffer to monitor the system in real time. The network admin can use it to watch all the incoming packets and find the ones which are dangerous to the system. It is based on library packet capture tool. The rules are fairly easy to create and implement and it can be deployed in any kind of operating system and any kind of network environment. The main reason of the popularity of this IDS over others is that it is a free-to-use software and also open source because of which any user can be able to use it as the way he wants.

Features:

- Real-time traffic monitor
- Packet logging
- Analysis of protocol
- Content matching
- OS fingerprinting
- Can be installed in any network environment.
- Creates logs
- Open Source
- Rules are easy to implement

What Are the Features of SNORT?

There are various features that make SNORT useful for network admins to monitor their systems and detect malicious activity. These include:

Real-time Traffic Monitor

SNORT can be used to monitor the traffic that goes in and out of a network. It will monitor traffic in real time and issue alerts to users when it discovers potentially malicious packets or threats on Internet Protocol (IP) networks.

Packet Logging

SNORT enables packet logging through its packet logger mode, which means it logs packets to the disk. In this mode, SNORT collects every packet and logs it in a hierarchical directory based on the host network's IP address.

Analysis of Protocol

SNORT can perform protocol analysis, which is a network sniffing process that captures data in protocol layers for additional analysis. This enables the network admin to further examine potentially malicious data packets, which are crucial in, for example, Transmission Control Protocol/IP (TCP/IP) stack protocol specification.

Content Matching

SNORT collates rules by the protocol, such as IP and TCP, then by ports, and then by those with content and those without. Rules that do have content use a multi-pattern matcher that increases performance, especially when it comes to protocols like the Hypertext Transfer Protocol (HTTP). Rules that do not have content are always evaluated, which negatively affects performance.

OS Fingerprinting

Operating system (OS) fingerprinting uses the concept that all platforms have a unique TCP/IP stacks. Through this process, SNORT can be used to determine the OS platform being used by a system that accesses a network.

Installing SNORT:

Preparing your server

Setting up a basic configuration of Snort on Ubuntu is fairly simple but takes a few steps to complete. You will first need to install all the prerequisite software to ready your cloud server for installing Snort itself. Install the required libraries with the following command.

Installing from the source

Setting up Snort on Ubuntu from the source code consists of a couple of steps: downloading the code, configuring it, compiling the code, installing it to an appropriate directory, and lastly configuring the detection rules.

Start by making a temporary download folder to your home directory and then changing into it with the command below.

mkdir ~/snort_src && cd ~/snort_src

Snort itself uses something called Data Acquisition library (DAQ) to make abstract calls to packet capture libraries. Download the latest DAQ source package from the Snort website with the wget command underneath. Replace the version number in the command if a newer source available.

wget https://www.snort.org/downloads/snort/daq-2.0.7.tar.gz

The download will only take a few seconds. When complete, extract the source code and jump into the new directory with the following commands.

tar -xvzf daq-2.0.7.tar.gz
cd daq-2.0.7

Afterwards, run the configuration script using its default values, then compile the program with make and finally install DAQ.

./configure && make && sudo make install

Running Snort on Multiple Network Interfaces:

Multiple Configurations

Snort now supports multiple configurations based on VLAN Id or IP subnet within a single instance of Snort. This will allow administrators to specify multiple snort configuration files and bind each configuration to one or more VLANs or subnets rather than running one Snort for each configuration required. Each unique snort configuration file will create a new configuration instance within snort. VLANs/Subnets not bound to any specific configuration will use the default configuration. Each configuration can have different preprocessor settings and detection rules.

Creating Multiple Configurations

Default configuration for snort is specified using the existing -c option. A default configuration binds multiple vlans or networks to non-default configurations, using the following configuration line:
config binding: <path_to_snort.conf> vlan <vlanIdList>
config binding: <path_to_snort.conf> net <ipList>
config binding: <path_to_snort.conf> policy_id <id>
path_to_snort.conf - Refers to the absolute or relative path to the snort.conf for specific configuration.
vlanIdList - Refers to the comma separated list of vlandIds and vlanId ranges. The format for ranges is two vlanId separated by a "-". Spaces are allowed within ranges. Valid vlanId is any number in 0-4095 range. Negative vland Ids and alphanumeric are not supported.
ipList - Refers to ip subnets. Subnets can be CIDR blocks for IPV6 or IPv4. A maximum of 512 individual IPv4 or IPv6 addresses or CIDRs can be specified.
policy_id - Refers to the specific policyi_id to be applied. Valid policyi_id is any number in 0-4095 range.

Configuration Specific Elements

Generally config options defined within the default configuration are global by default i.e. their value applies to all other configurations. The following config options are specific to each configuration.
policy_id
policy_mode
policy_version

The following config options are specific to each configuration. If not defined in a configuration, the default values of the option (not the default configuration values) take effect.

config checksum_drop
config disable_decode_alerts
config disable_decode_drops
config disable_ipopt_alerts
config disable_ipopt_drops
config disable_tcpopt_alerts
config disable_tcpopt_drops
config disable_tcpopt_experimental_alerts
config disable_tcpopt_experimental_drops
config disable_tcpopt_obsolete_alerts
config disable_tcpopt_obsolete_drops
config disable_ttcp_alerts
config disable_tcpopt_ttcp_alerts
config disable_ttcp_drops

Snort Command Line Options:

Before we go into Snort's basic operational modes, let's first look at a breakdown of the command-line options. This chapter covers each item listed here, but some are not frequently used or may only be used in conjunction with other variables. Some of the options can be specified in the *config* file instead of at the command line. If you are just trying something out, specify the setting at the command line. If you are planning on keeping the setting for a while, set it in the *config* file.

-A *alert-mode*
Generates an alert using one of the specified alert-modes: fast, full, none, and unsock. Rather than specifying the alert mode within a configuration file, you can include it here at the command line.

-b
Logs packets in tcpdump format (i.e., libpcap). Files in tcpdump format are smaller, so this is the best method of recording large amounts of logged data and packets. It is very fast and may be a good option on high-traffic networks.

-B *address-conversion-mask*
Scrambles the networks specified in the -h (or HOME_NET) setting. This helps hide the real internal network addresses inside binary logs.

-c *config-file*
Allows you to specify which configuration file you want to use. If you have different configurations with various rules enabled, you can specify which configuration to use at the command line. This option is required when Snort is run in NIDS mode.

-C
Prints the character data found in the packet payload, rather than displaying it in hexadecimal format. Reading this information is easier than wrestling with Hex output.

-d
Displays the application layer data when in verbose or packet logging mode.

-D
Runs Snort in daemon mode. Alerts are dumped to the *alert* file in the logging directory (*/var/log/snort* by default). Daemon mode is useful if you wish to automate the startup of Snort in the event of a reboot. Passing this option to Snort in a command script starts Snort in the background. No error messages are printed to the console in this mode. Do not use this mode unless you are already familiar with Snort and have a working, viable configuration. (Use the -T option, discussed below, to test your configuration before using daemon mode.)

-e
Displays or logs the link layer packet headers. This is the more verbose method of viewing captured packets when running Snort in sniffing mode.

-F *bpf-file*
Reads *Berkeley Packet Filters* (BPF) from a *bpf* file. These filters are useful when running Snort as a SHADOW replacement or when performing an analysis via a command-line filter. This filter is commonly used to tune out noise or random alerts. (It is not commonly used.) You could use a BPF filter to tell one system to watch only web traffic and another to watch everything else.

-g *group*
Changes the default group ID or GID under which Snort runs after initialization. This is helpful if you want to run Snort in a special group for security reasons.

-h *home-net*
Sets the "home network" to a specific address in CIDR format. With this variable set, all decoded packet logging is done relative to the home network address space. This option is equivalent to setting the HOME_NET variable in the configuration file.

-i *interface*
Specifies which interface Snort should listen on. This option is used on machines that have more than one network interface card or that have different kinds of interfaces, besides Ethernet. Naming conventions for interfaces vary between operating systems.

-I
In alerts, displays the interface on which each packet arrived. Useful when monitoring multiple interfaces; you can see which interface received the suspicious packet. Also very useful when multiple Snort sensors are sending their alerts to a central database (discussed further in Chapter 5).

-k *checksum-mode*
Controls which packet checksums Snort computes and verifies. Valid checksum modes include all, noip, notcp, noudp, noicmp, and none. This can be used to eliminate packets that fail their checksums - caused either by network faults or IDS evasion attempts

-l *logging-directory*
Specifies the logging directory. All alerts and packet logs are placed in this directory. The default logging directory is */var/log/snort*, but that default is only used when Snort is in alert (-A) mode. If you want to use Snort as a simple packet logger, you must use the -l option and specify the logging directory explicitly. Often used when debugging Snort and when logging packets to a temporary directory so that the new logs do not mingle with production logs.

-L *binary-log-file*
Sets the filename of the binary logfile. If this switch is not used, the default name is a timestamp for when the file was created, plus *snort.log*.

-m *umask*
Sets the file mode creation mask to the designated umask variable. This is a simple security measure to prevent others from viewing the logfiles generated during packet capture.

-n *packet-count*
Processes the given number of packets and then exits. Useful when you want to capture a small snapshot network traffic.

-N
Turns off packet logging. Alerts are still generated but are printed to the console only. No records are kept on the system of the generated alerts. This can be useful when testing your configurations.

-o
Changes the order in which the rules are applied to packets. Instead of the rules being applied in the standard Alert → Pass → Log order, this option applies them in Pass → Alert → Log order. Recommended for users running SnortCenter and other web interfaces. This is how the developers of these applications decided to display captured Snort packets. This option is also used to ensure that *pass* rules are applied before detection rules. See Chapter 9 for the caveats with using this option (and *pass* rules).

-O
When in ASCII packet dump mode, replaces the IP addresses printed to the screen or logfile with "xxx.xxx.xxx.xxx". If the home-net address switch is set, -h, only addresses on home-net are obfuscated, while non-home net IPs are left visible. Use this option when capturing sample alerts or packets that need to be posted or shared with other non-trusted users. It is perfect for posting a packet capture to a discussion group or a mailing list.

-p
Turns off promiscuous mode sniffing. When first working with Snort, the usefulness of this option evaded me. The answer came to me in the shower—it can be used to protect only one host. When not in promiscuous mode, an adapter will only accept packets addressed to itself.

-P *snap-length*
Sets the maximum packet capture length to a certain size. Some packets may be very large. While most rules look for characteristics or signatures in the beginning of a packet, setting the maximum packet length may cause you to miss large malicious packets, when the offending string is located at the end.

-q
Tells Snort to run quietly. Does not display banner and initialization information. If you aren't interested in the initialization messages, you can suppress them with this.

-r *tcpdump-file*
Use this option to process a tcpdump-formatted file. The output appears much like it would when capturing data in real-time. This option is used to analyze a packet trace that was collected at an earlier time.

-s
Sends alert messages to a syslog server. This can be either a local or remote server. Use this option when capturing logs and alerts within syslog.

-S *variable=value*
Sets the variable name *variable* to the value *value*. There are a number of variables that Snort uses to define what systems are on your local network (HOME_NET), which are web servers or DNS servers, and which systems are external to your network. It is advised to keep all variables in the *snort.conf* file to limit confusion.

-t *chroot*
Changes Snort's root directory to *chroot* after initialization. Paths for logfiles and alert files are relative to the new root directory.

-T
Starts Snort in self-test mode. Useful for debugging Snort before it is run in daemon mode or before it is launched on a production box. Can be used for testing the correctness of your configuration files.

-u *user*
Changes the default user ID or UID under which Snort runs after initialization.Like the -g option, an added security feature for running Snort as a nondescript user.

-U
Forces the timestamp in all logs to be in UTC (a.k.a. GMT) format. A recommended option when capturing logs from multiple sources on a single syslog server and if sensors are scattered across a large WAN; you won't have to deal with time zone differences.

-v
The verbose option prints all packets to the console. Be careful when using this option, as it may slow Snort and result in dropped packets.

-V
Displays the Snort version number and then exits. Use this to determine which version of Snort is installed on your system.

-X
Displays raw packet data starting at the link layer. With this option you can see the entire packet, including Ethernet headers and trailers.

-y
Includes the year in all alerts and logfiles. Useful when you want to create an archive of logged Snort packets that can be referred to later.

-z
Enables the *stream4* preprocessor. Preprocessors manage incoming packets before passing them off to Snort. They are sometimes used to reconstruct fragmented packets. This option takes advantage of stream4's stateful packet inspection capabilities. It tells Snort to generate alerts only when a packet is part of an established session, foiling some IDS evasion mechanisms.

-?
Lists all switches and options and then exits.

Step-By-Step Procedure to Compile and Install Snort Location of Snort Files:

1. Download and Extract Snort
Download the latest snort free version from snort website. Extract the snort source code to the /usr/src directory as shown below.

```
# cd /usr/src

# wget -O snort-2.8.6.1.tar.gz http://www.snort.org/downloads/116

# tar xvzf snort-2.8.6.1.tar.gz
```

2. **Installing Snort:**

Before installing snort, make sure you have dev packages of libpcap and libpcre.

```
# apt-cache policy libpcap0.8-dev

libpcap0.8-dev:

  Installed: 1.0.0-2ubuntu1

  Candidate: 1.0.0-2ubuntu1

# apt-cache policy libpcre3-dev

libpcre3-dev:

  Installed: 7.8-3  Candidate: 7.8-3
```

Follow the steps below to install snort.

```
# cd snort-2.8.6.1

# ./configure

# make

# make install
```

3. Verify the Snort Installation

Verify the installation as shown below.

```
# snort --version

   -*> Snort! <*- o" ) ~   Version 2.8.6.1 (Build 39)

  ""    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-team

        Copyright (C) 1998-2010 Sourcefire, Inc., et al.

        Using PCRE version: 7.8 2008-09-05
```

4. Create the required files and directory

You have to create the configuration file, rule file and the log directory.

Create the following directories:

```
# mkdir /etc/snort

# mkdir /etc/snort/rules

# mkdir /var/log/snort
```

Create the following snort.conf and icmp.rules files:

```
# cat /etc/snort/snort.conf

include /etc/snort/rules/icmp.rules

# cat /etc/snort/rules/icmp.rules

alert icmp any any -> any any (msg:"ICMP Packet"; sid:477; rev:3;)
```

The above basic rule does alerting when there is an ICMP packet (ping).

Following is the structure of the alert:

```
<Rule Actions> <Protocol> <Source IP Address> <Source Port> <Direction Operator
```

5. Execute snort

Execute snort from command line, as mentioned below.

```
# snort -c /etc/snort/snort.conf -l /var/log/snort/
```

Snort Alert Modes:

Snort alerts are anomalous network traffic and suspicious connections reporting. By default, alerts are stored under the */var/log/snort* directory.

There are 7 available alert modes you can specify when executing Snort, which is listed below:

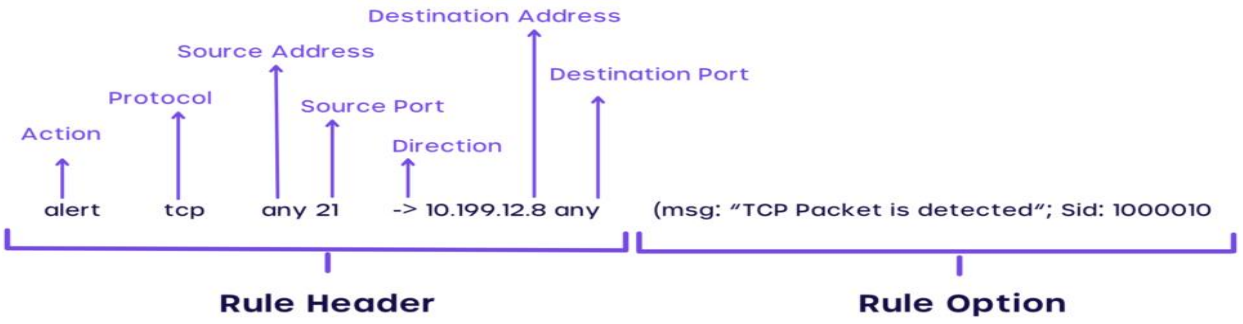
- **Fast:** When in fast mode, Snort alerts report the timestamp, send an alert message, show the source IP address and port, and the destination IP address and port. This mode is instructed using the ***-A fast*** flag.
- **Full:** Additionally to the information printed in the fast mode, the full mode shows the TTL, packet headers and datagram length, service, ICMP type, window size, ACK and sequence number. The full mode is defined with the ***-A full*** flag, but this is the default alerts mode.
- **Console:** prints fast alerts in the console. This mode is implemented with the ***-A console*** flag.
- **Cmg:** This alerts mode was developed by Snort for testing purposes; it prints a full alert on the console without saving logs. The mode is implemented with the ***-A cmg*** flag.
- **Unsock:** This is useful to export alert reports to other programs through Unix sockets. The unsock mode is implemented using the ***-A*** unsock flag.
- **Syslog:** In syslog (System Logging Protocol) mode, Snort sends alert logs remotely; this mode is implemented by adding the ***-s*** flag.
- **None:** With this mode, Snort does not generate alerts.

UNIT – IV

Working with Snort Rules:

In the business world, the Web and Cyber security, Snort refers to IDS– *Intrusion Detection System*. Because such detection helps you get proactive and secure the best interests of your business it is also known as IPS– *Intrusion Prevention System.*

If we drew a real-life parallel, Snort is your security guard. Snort Rules are the directions you give your security personnel. A typical security guard may be a burly man with a bit of a sleepy gait. **With Snort and Snort Rules, it is downright serious *cyber security*.**



Your business is running strong, the future looks great and the investors are happy. All of a sudden, a cyber attack on your system flips everything upside down and now you wonder (/*snort* in anguish) *What, Why, If only!*

Important Features of a Snort Rule:

Trigger Alerts

Alerting a malicious activity that could be a potential threat to your organization, is a natural feature of a snort rule. Crucial information like IP Address, Timestamp, ICPM type, IP Header length, and such are traceable with a snort rule.

There are multiple modes of alert you could generate: *Fast, Full, None, CMG, Unsock, and Console* are a few of the popular ones. Each of which is unique and distinct from one another.

For instance, if you need a *full report* that includes comprehensive details, the rule would look like the following:

```
Output alert_full: alert.full
```

And suppose you need a quick report that doesn't need to be as elaborate as the full report, you could choose to get it with the following rule

```
Output alert_fast: alert.fast
```

Flexible Usage

As may be evident from the above examples, a snort rule is a single line code that helps to identify the nature of traffic. However, modern-day snort rules cater to larger and more dynamic requirements and so could be more elaborate as well.

Also, once you download Snort Rules, it can be used in any Operating system (OS). There is no limitation whatsoever. Be it Linux, Unix, Windows, Ubuntu or whichever for that matter, Snort secures your network just the same.

Identifies Fraud:

Snort is the most popular *IPS*, globally speaking. The open-source IDS – Intrusion Detection System helps to identify and distinguish between regular and contentious activities over your network.

Snort Rules refers to the language that helps one enable such observation. It is a simple language that can be used by just about anyone with basic coding awareness.

It combines 3 methods to detect a potential cyber fraud:

Method #1
Signature: Signature-based IDS refers to the identification of data packets that have previously been a threat. It identifies historic patterns or popular and malefic sequences and detects the same when a similar event is on the cards.

Method #2
Protocol: In this method, Snort detects suspicious behavior from the source of an IP – Internet Protocol. Every computer has a unique IP and the data that is sourced from a distrustful IP is detected and notified in real-time.

Besides high-level protocols like HTTP, Snort detects skeptical user behavior from 3 types of low-level Protocols – TCP, UDP, and ICMP.

Apparently, we may even be able to analyze data packets from different sources like ARP, IGRP, GRP, GPSF, IPX in the future.

Method #3
Anomaly-based Inspection:
There is a palpable difference between Signature/ Protocol-based IDS and Anomaly-based inspection.

While the other 2 rely on previous or historic behavior, Anomaly-based IDS detects and notifies of *any* type of behavior that can be viewed with a veil of suspicion. Suspicious activities and attempts over Operating System (OS) Fingerprints, Server Message Block (SMB) probes, CGI attacks, Stealth Port Scans, Denial of Service (DoS) attacks etc are negated instantly with Snort. We have touched upon the different types of intrusion detection above. It would serve well to be aware that Snort rules can be run in 3 different modes based on the requirements:

3 Modes of Snort: Sniffer, Logging and NIDS

- **Sniffer Mode:** Sniffer mode helps with your IDS objectives in the following instances if:
  1. You only need to print out data: ./snort -v
  2. There is a need to see the data in transit and also check the IP and TCP/ICMP/UDP headers: ./snort -vd
  3. You need slightly elaborate information about data packets: ./snort -vde
  4. To list the command lines exclusively: ./snort -d -v -e

- **Logging Mode:** Just like the term ‘logging’ implies, when you need to log/record the data packets you may designate a logging directory. Understandably, the data packets are recorded in the directory.

  Here’s the line that logs the data in an assumption that you have created a directory called ‘log’ :

  ./snort -dev -l ./log -h 192.168.1.0/24

Network Intrusion Detection System (NIDS) Mode: When you/ or your network administrator is specific about logging a specific kind of data packet/s, you may run Snort in NIDS mode. You may also define the action you want to take upon detection of malicious data packets while you write the rule.

Writing Snort Rules | Examples and Cheat Sheet

We are getting closer to understanding what snort rules are and their examples. So far so good with understanding the essence, features, and the different modes of Snort.

Frankly speaking, the examples and the cheat sheet to write snort rules that we will have later is why we are having this conversation in the first place.

However, doing so without getting familiar with these terms would be somewhat like playing basketball without knowing how to dribble the ball. So here it goes:

Rule Header

1. **Rule Action:** There are 5 rule actions by default while you run a typical Snort rule: Alert. Dynamic, Pass, Log, or/and Activate. The most common rule action is 'alert' which understandably alerts the network administrator upon detecting a potential threat.
2. **Protocol:** As discussed previously, the protocol is the unique address of a computer. While writing the rule, you may key in the Protocol address that you need to be wary of gathered from previous mishaps / historical data.
3. **Source IP Address:** Let's assume that your threat is from Mr Jack. The IP address or in some cases, his network ID is the source IP address. In case you need to get alerts from any and every source, you may just key in 'any' in this particular part of the rule.
4. **Source Port:** This is like the *facilitator* that communicates messages between two or more computer networks. Computers generally have 65,536 TCP Ports and if you wish, you may again key-in 'any' to define all such ports in the rule.
5. **Flow:** In Snort Rules, flow refers to the direction of the traffic. This keyword/symbol helps us apply rules only to the specific parts of the traffic. It is denoted with the arrow (</>) symbol.
6. **Destination IP Address:** Where do the network packets from Source IP address channeled through Source Port and Destination Port go to? No prizes for guesses. They go to the Destination IP Address. So obvious, isn't it?
7. **Destination Port:** Just like the source port which facilitates communication between the source ports, the Destination port does the same with Destination TCP/ UDP.

Rule Header							Options
Rule Action	Protocol	Source IP Address	Source Port	Flow	Destination IP Address	Destination Port	Message
alert	top	any	21	>	10.199.12.8	any	(msg: "TCP Packet Detected" nd: 1000:610)

Snort Rules Examples and Cheat sheet:

Case 1: Securing Email Server with Snort Rules:

alert tcp 192.168.1.0/24 any -> 131.171.127.1 25 (content: “hacking”; msg: ”malicious packet”; sid:2000001;)

Case 2: Detecting TCP SYN Floods

Alert tcp any any -> 192.168.10.5 443 (msg: “TCP SYN flood”; flags:!A; flow: stateless; detection_filter: track by_dst, count 70, seconds 10; sid:2000003;)

Case 3: Securing your Network against Conficker A Worm

alert tcp any any -> any 445 (msg: “conficker.a shellcode”; content: “|e8 ff ff ff ff c1|^|8d|N|10 80|1|c4|Af|81|9EPu|f5 ae c6 9d a0|O|85 ea|O|84 c8|O|84 d8|O|c4|O|9c cc|IrX|c4 c4 c4|,|ed c4 c4 c4 94|&<O8|92|\;|d3|WG|02 c3|,|dc c4 c4 c4 f7 16 96 96|O|08 a2 03 c5 bc ea 95|\;|b3 c0 96 96 95 92 96|\;|f3|\;|24|i| 95 92|QO|8f f8|O|88 cf bc c7 0f f7|2I|d0|w|c7 95 e4|O|d6 c7 17 f7 04 05 04 c3 f6 c6 86|D|fe c4 b1|1|ff 01 b0 c2 82 ff b5 dc b6 1b|O|95 e0 c7 17 cb|s|d0 b6|O|85 d8 c7 07|O|c0|T|c7 07 9a 9d 07 a4|fN|b2 e2|Dh|0c b1 b6 a8 a9 ab aa c4|]|e7 99 1d ac b0 b0 b4 fe eb eb|”; sid: 2000002; rev: 1;)

Case 4: Alerts of Buffer Overflow in BIND

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:”FTP wuftp bad file completion attempt [“;flow:to_server, established; content:”|?|”; content:”[“; distance:1; reference:bugtraq,3581; reference:bugtraq,3707; reference:cve,2001-0550; reference:cve,2001-0886; classtype:misc-attack; sid:1377; rev:14;)

Case 5: Preventing Bug Bear Variants and Content Generalization

alert tcp any any -> any any (msg:Possible BugBear B Attack FuzzRuleId cor(\’||?| 63 e7|\’); content:||?| 63 e7|; regex; dsize:>21;) alert tcp any any -> any any (msg:Possible BugBear B Attack FuzzRuleId cor(\’|3b |?| e7|\’); content:|3b |?| e7|; regex; dsize:>21;) alert tcp any any -> any any (msg:Possible BugBear B Attack FuzzRuleId cor(\’|3b 63 |?||\’); content:|3b 63 |?||; regex; dsize:>21;)

Case 6: Generalization by Rule Inversion

alert udp any any -> any 69 (msg:TFTP GET Admin.dll; content: |0001|; offset:0; depth:2; content:admin.dll; offset:2; nocase; classtype:successful-admin; reference:url, www.cert.org/advisories/CA-2001-26.html; sid:1289; rev:2;)

alert udp any any -> any 69 (msg:TFTP GET Admin.dll; content: |0001|; offset:0; content:admin.dll; offset:2; nocase; classtype:successful-admin; reference:url, www.cert.org/advisories/CA-2001-26.html; sid:1289; rev:2;)

Snort Cheat Sheet

Sniffer Mode

Sniff packets and send to standard output as a dump file

-v (verbose)	Display output on the screen
-e	Display link layer headers
-d	Display packet data payload
-x	Display full packet with headers in HEX format

Packet Logger Mode

Input output to a log file

-r	Use to read back the log file content using snort
-l (directory name)	log to a directory as a tcpdump file format
-k (ASCII)	Display output as ASCII format

Snort Rules Format

Rule Header + (Rule Options)

Action - Protocol - Source/Destination IP's - Source/Destination Ports - Direction of the flow

Alert Example	alert udp !10.1.1.0/24 any -> 10.2.0.0/24 any
Actions	alert, log, pass, activate, dynamic, drop, reject, sdrop
Protocols	TCP, UDP, ICMP, IP

Snort Rules Example

log tcp !10.1.1.0/24 any -> 10.1.1.100 (msg: "ftp access";)

NIDS Mode

Use the specified file as config file and apply rules to process captured packets

-c	Define configuration file path
-T	Use to test the configuration file including rules

Logger Mode command line options

-l logdir	Log packets in tcp dump
-K ASCII	Log in ASCII format

NIDS Mode Options

Define a configuration file	-c (Configuration file name)
Check the rule syntax and format for accuracy	-T -c (Configuration file name)
Alternate alert modes	-A (mode: Full, Fast, None, Console)
Alert to syslog	-s
Print alert info	-v
Send SMB alert to PC	-M (PC name or IP address)
ASCII log mode	-K
No logging	-N
Run in Background	-D
Listen to a specific network interface	-i

Output Default Directory

/var/snort/log

The Snort Configuration File:

The Snort configuration file **contains six basic sections: Variable definitions**. This is where you define different variables that are used in Snort rules as well as for other purposes, such as specifying the location of rule files. Configure dynamic loadable libraries.

Snort uses a configuration file at startup time. A sample configuration file snort.conf is included in the Snort distribution. You can use any name for the configuration file, however snort.conf is the conventional name.

You use the -c command line switch to specify the name of the configuration file.

The following command uses ***/opt/snort/snort.conf*** as the configuration file.

/opt/snort/snort -c /opt/snort/snort.conf

You can also save the configuration file in your home directory as *.snortrc*, but specifying it on the command line is the most widely used method. There are other advantages to using the configuration file name as a command line argument to Snort. For example, it is possible to invoke multiple Snort instances on different network interfaces with different configuration. This file contains six basic sections:

- Variable definitions, where you define different variables. These variables are used in Snort rules as well as for other purposes, like specifying the location of rule files.
- Configure parameters. These parameters specify different Snort configuration options. Some of them can also be used on the command line.
- Preprocessor configuration. Preprocessors are used to perform certain actions before a packet is operated by the main Snort detection engine.
- Output module configuration. Output modules control how Snort data will be logged.
- Defining new action types. If the predefined action types are not sufficient for your environment, you can define custom action types in the Snort configuration file.
- Rules configuration and include files. Although you can add any rules in the main snort.conf file, the convention is to use separate files for rules. These files are then included inside the main configuration file using the *include* keyword. This keyword will be discussed later in this chapter.

Procedure

Click the **SNORT Configuration** tab.

1. In the Import SNORT Configuration File area, use the default configuration file, import a SNORT.conf file, or add supported configuration contents.

Notes:

- If you import a SNORT.conf file, it replaces the default one.
- If you import a SNORT.conf file, delete variable rule paths. Examples of variable rule paths:
  - var PREPROC_RULE_PATH ../preproc_rules
  - var WHITE_LIST_PATH /etc/snort/rules
- If you use the default configuration file, review and adjust its network settings so that it works for your environment.
- The Network IPS appliance does not support the use of third party preprocessors.

In the Interfaces area, set the following options:

- Select the appropriate interfaces to apply the configuration file to.
- Select the **Inspect HA mirrored ports** check box to enable the SNORT systems on appliances in a high availability (HA) pair to analyze packets on mirrored ports.

In the Rule Profiling area, configure the options for gathering performance metrics about SNORT rules.

a. Select the **Enable rule profiling** check box to record SNORT performance statistics.

   Note: You must also enable the **SNORT Execution** check box on the **SNORT Execution** tab for this feature to work.

b. Select **Number of rules to display** from the list. The appliance displays the rules with the worst statistics.
c. Select the **Sort option**, which is a list of statistics the system uses to order the rule profile. The statistics are:

Statistic	Description
Checks	The number of times the SNORT engine checks for rule options after the SNORT engine completes an initial analysis to group and pre-screen traffic.
Matches	The number of times the SNORT engine finds traffic matching all rule options.
No Matches	The number of times the SNORT engine finds no traffic matching all rule options.
Average Ticks (Avg/Check)	The average time the SNORT engine takes to check each packet against the listed rule.
Average Ticks Per Match (Avg/Match)	The average time the SNORT engine takes to check each packet that matches all rule options.
Average Ticks Per No Match (Avg/Nonmatch)	The average time the SNORT engine takes to check each packet that did not generate an event. **Note:** This statistic represents wasted time spent checking clean traffic.
Total Ticks	The rules responsible for consuming the most processing time.

Snort Plugins:

Overview:

Snort version 1.5 introduces a major new concept, plugins. There are two types of plugin currently available in Snort: detection plugins and preprocessors. Detection plugins check a single aspect of a packet for a value defined within a rule and determine if the packet data meets their acceptance criteria. For example, the tcp flags detection plugin checks the flags section of TCP packets for matches with flag combinations defined in a particular rule. Detection plugins may be called multiple times per packet with different arguments.

Preprocessors are only called a single time per packet and may perform highly complex functions like TCP stream reassembly, IP defragmentation, or HTTP request normalization. They can directly manipulate packet data and even call the detection engine directly with their modified data. They can perform less complex tasks like statistics gathering or threshold monitoring as well.

Adding New Plugins to Snort as a User:

Right now, adding a new plugin to Snort is straightforward but requires you to edit two files by hand. The plugin should consist of two files, “sp_something.c”/”sp_something.h” for detection plugins, and “spp_something.c”/”spp_something.h” for preprocessors. For detection plugins, there are two steps to integrating it with Snort:

1) Edit plugbase.h and insert the line #include “sp_something.h” into the file with the other “#include” statements. Save and close the file.

2) Edit the plugbase.c file and in the InitPlugins() function, add the name of the setup function to the list with the other Setup functions. Save and close the file.

3) Edit the Makefile.am and add the names of the two files to the list of names on the “snort_SOURCES” line. Save and exit the file. Run “automake”.

All set. Now recompile Snort and the plugin should be ready to use!

Adding preprocessors is equally straightforward. The process is essentially the same as above:

1) Edit plugbase.h and insert the line #include "spp_preproc.h" into the file with the other "#include" statements. Save and close the file.

2) Edit the plugbase.c file and in the InitPreprocessors() function, add the name of the setup function to the list with the other Setup functions.
Save and close the file.

3) Edit the Makefile.am and add the names of the two files to the list of names on the "snort_SOURCES" line. Save and exit the file. Run "automake".

Someday, there will be a nice little program that will do all this work for you!

Writing New Snort Plugins as a Developer:

This process is also pretty straight forward, and the best place to look for information on doing these things is at the files in the "templates" directory. The sp_* files are setup for detection plugins, and the spp_* files refer to the preprocessor files. The main thing to remember once you've got it written is to put the proper includes and function calls into plugbase [.c|.h]. I should probably flesh out this document more, but I think that the best info is in the template files. If you have any questions or comments, don't hesitate to send me an e-mail!

Snort Preprocessors:

Preprocessor Configurations

The Snort preprocessors have changed a lot in recent versions. Old, standby preprocessors (like portscan2) are gone, replaced with new methods born out of development work by the open source community and the commercial incarnation of Snort from Source fire. It seems that the code for some of these deprecated preprocessors is still there—you could still use some of the old functionality if that is what you are used to. We won't talk about it here, however.

The preprocessors serve a few purposes. They normalize traffic for a variety of services, ensuring that the data in the packets Snort is watching will have the best chance of being in a format that the signatures will recognize. Another function of the preprocessors is self-defense. A variety of attacks have been developed that are designed to confuse or overwhelm an NIDS sensor, so an attacker can do her work unnoticed. The frag2 and stream4 preprocessors are primarily defense mechanisms.

The final benefit provided by the preprocessors is that they extend Snort's ability to detect network anomalies that may be signs of intrusion—not just notice things that are contained in the rule sets. Apart from just the raw performance that Snort offers, the preprocessors serve to differentiate Snort from other NIDS solutions on the market today.

1 flow

The flow preprocessor is going to be the central storehouse for state keeping in Snort. Right now, there is only one module for flow: flow-portscan (see below). flow watches all traffic and keeps track of connections between unique systems and between unique ports. When a new unique flow is detected, the information is converted to a hash (smaller and faster to keep track of than tracking IP addresses and port numbers) that is stored in a memory-resident table.

The options for the flow preprocessor are:

Memcap: You can specify a cap for the memory allocated for flow tracking. By default, flow uses about 10 megabytes of memory. For most applications this is more than enough. If you start noticing dropped packets in the statistic summary when Snort is terminated, or low memory problems on the Snort system itself, you can reduce this number.

Rows: You can specify the number of rows in the hash table (the number of unique flows). By default, this is set to 4,099 rows, which should be plenty.

stats_interval: You can dump the statistics of the flow preprocessor to stdout. Its value is an integer that represents the time (in seconds) between dumps. This is useful for testing purposes, but can be disabled by supplying 0 as the value. This information is dumped at shutdown, in any case.

Hash: The method used for hashing the information in the table. Can be set to 1 for "hash by byte" or 2 for "hash by integer." Use 2—it's much faster.

Here's a recommended configuration for flow: preprocessor flow: stats_interval 0 hash 2

2 frag2

When a packet is travelling from one network to another, it occasionally needs to be broken (fragmented) into a series of smaller packets, because the second network has a limit on packet size that's smaller than the first network. The pieces are reassembled when they reach their destination. A number of network attacks have been based on using very small packets to sneak past firewalls or intrusion detection systems. For example, consider a Snort rule that is looking for the string */users.pwd* in the data section of a packet. An attacker might create a fragmented sequence of small packets in which each fragment only contains a few bytes of data: the first fragment might contain */user*, and the second might contain *s.pwd*. These packets won't trigger alert by themselves, because they don't match the rule. The frag2 preprocessor, by reassembling the fragments into a whole, allows snort to see the "big picture" and detect the string */users.pwd*.

An attacker can also use fragmentation to cause problems. Some devices do not handle fragmented packets well and can crash as a result. Too many very small packets can cause resource starvation on a system and overlapping packets can confuse others. There's an attack tool called Fragroute that fragments a network stream. frag2 has several options that defend against these attacks.

The placement of the Snort sensor has a bearing on whether or not to use the frag2 preprocessor. Some network devices perform fragment reassembly before passing the traffic on. The Cisco PIX firewall is one such device. If the Snort sensor is behind one of these devices, the frag2 preprocessor has almost nothing to do and can likely be disabled altogether, freeing up the resources for other preprocessors.

frag2 has several options:

timeout: The number of seconds before an inactive session is flushed from memory. It defaults to 60 seconds.

Memcap: The number of bytes of memory to set aside. The default of 4 Megabytes (4,194,304 bytes) is plenty for most applications.

detect_state_problems:Turns on alerts for some unnatural fragmentation conditions (like overlapping fragments). Since such conditions are very often sign of malicious activity, this should be enabled for most environments.

min_ttl: Sets the minimum time to live (ttl) accepted by the preprocessor. Set to 0 by default.

ttl_limit: Sets the maximum variance of ttl between packets of a network stream. This should not vary too much and a wide variance may be a sign of nefarious activity on the network. Defaults to a value of 5.

stream4

Stream4 was initially written to protect Snort from a new class of attacks that attacked an environment's NIDS sensors by overwhelming them with packets containing strings likely to trigger alerts. These attacks came to light with the release of two tools called stick and snot, previously mentioned in our discussion of IDS evasion techniques. Stick actually uses the Snort rules to create packets that match the characteristics that Snort is watching for (it will also probably generate alerts for other IDS solutions), thus flooding it with alerts. Since a network conversation (at least a TCP session) is started by a three-way handshake with a server, if there was a way to track the active conversations that were set up correctly, it would be possible to eliminate these stick packets since they are not part of an active conversation.

Stream4 has two goals: stateful inspection and awareness and session reassembly. Session reassembly is handled by the stream4_reassemble preprocessor. For accurate function of stream4, stream4_reassemble should be enabled.

Along with providing a defense against tools such as stick and snot, tracking the state of a network conversation provides additional benefit. We will discuss this in much more detail in but stream4 gives us the ability to build rules that watch a particular side of a conversation using the flow keyword (not to be confused with the flow preprocessor). This helps reduce false positives. To activate this stateful inspection by rules, use the -z option at the command line. It used to be necessary to use -z est to enable this, but the "est" is deprecated and no longer needed (in fact, it causes an error).

The default settings for stream4 do a decent job, but consider tuning things further to suit your environment (and to further reduce false positives). The options for the stream4 preprocessor are:

detect_scans: Disabled by default. This directive tells stream4 to generate an alert when a portscan is detected. stream4 detects portscans that use nonstandard methods to identify listening ports. For instance, if a packet is sent to a port on a host with the FIN flag set, the host responds with a RST packet if the port is closed and replies with nothing if the port is open and listening. These are referred to as *stealth* scans and can take a variety of forms. Include this parameter if you are interested in detecting stealth scans.

detect_state_problems: Disabled by default. This generates an alert when a problem with the state of a conversation is detected. This option generates a large number of alerts and generally remains disabled in most environments. Networks with Windows systems generate a large number of these alerts. High-latency environments experience a large number of false positives, since a system that transmits packets after an ACK is sent causes alerts to be generated.

disable_evasion_alerts: Disabled by default, too. If enabled, this option may detect attackers trying to confuse the IDS by sending retransmissions of packets, thinking that the second one would make it through. It is also possible to send a data payload with a SYN packet. Since a SYN packet is often used to initiate a TCP connection, it does not carry a data payload by design. Generally, it is best to keep this option enabled.

min_ttl: Sets a minimum time to live (ttl) setting for packets accepted into the stream4 preprocessor. Some attacks may have an artificially low ttl so that they make it to the IDS (overwhelming it), while the packet does not make it to the actual server. This situation is fairly rare, and it's usually acceptable not to set a value for this option. This option defaults to 1.

ttl_limit: Sets the maximum amount the time to live setting can vary in a network conversation. Every time a packet travels through a router, the ttl setting is decremented by one. During a network conversation, the packets traveling between the two hosts follow generally the same path across the network and the ttl should be fairly constant in each direction. If an attacker is trying to insert packets into a network conversation, the ttl may change, indicating an attack is underway. Since routing is dynamic (thus, a change in ttl might not be something nefarious), it is difficult to decide on a value for this option (a flapping route might cause false positive alerts, too). Most sources indicate that a value between 10 and 15 is reasonable. This option defaults to 5.

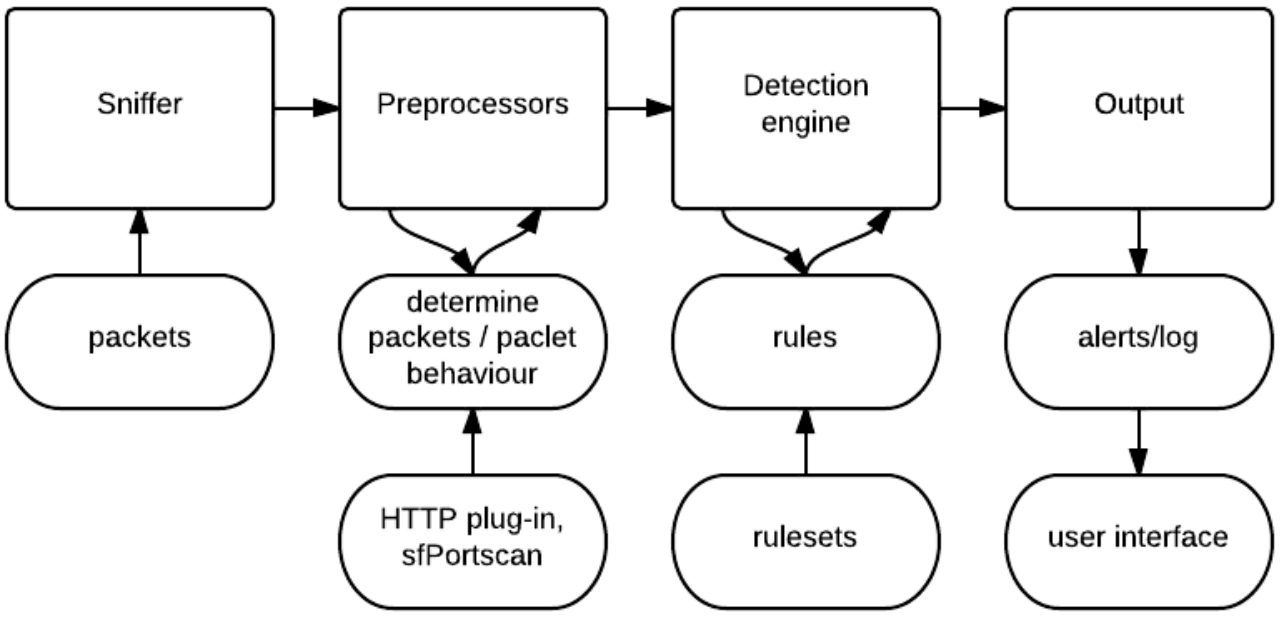
keepstats [machine, binary]: Disabled by default. If enabled, it outputs statistics on each session tracked in one of two modes: machine dumps to a text file; binary outputs a unified binary format useable by tools like Barnyard.

Noinspect: Normally defaults to off. Tells stream4 to disable stateful inspection of all packets that are destined for ports not included in the stream4_reassemble's ports option. Below for details on the ports option). You might not be concerned with traffic destined for ports that you are not listening on. In general, you can leave this option off.

Timeout: Normally, when a TCP connection is torn down, it is pruned from the session table. If the connection is not torn down or is improperly torn down, this timeout setting prunes the session from the table after an indicated period of time has passed without activity. Essentially, it acts as an idle timer. This option defaults to a value of 30 seconds—kind of short, for most environments. Since the memory requirements are fairly low (on the order of a few kilobytes) for the session table, setting this to 60 seconds is not a bad idea.

log_flushed_streams: If a packet in a monitored stream causes an alert, you can dump the session information from stream4 to disk. This only works if you are logging in pcap mode. This option does carry some overhead and is not commonly enabled. The packet logs it generates are difficult to work with.

Memcap: This looks for a number of bytes sets a limit on the amount of memory allocated to stream4. Normally, the default value works fine, but if you have the memory or are operating in an enterprise-class environment with a dedicated sensor, you could ratchet this value up. Keep in mind that the option is looking for a number of bytes, not Megabytes. This option defaults to 8 Megabytes (8,388,608 bytes).



UNIT – V

Analysis Console for Intrusion Databases (ACID) is a tool used to analyze and present Snort data using a web interface. It is written in PHP. It works with Snort and databases like MySQL, as you have learned in the last chapter, and makes information available in the database to the user through a web server. In addition to Snort, the tool can be used with other security-related products like firewalls and networking monitoring.

What is ACID? ACID consists of many Pretty Home Page (PHP) scripts and configuration files that work together to collect and analyze information from a database and present it through a web interface. A user will use a web browser to interact with ACID. You have to have a web server, database server, PHP and some other tools installed on your system to make it work. For the sake of this book, I am using a RedHat Linux 7.1 machine. I have installed Apache web server, PHP, and MySQL, which are part of the RedHat distribution. The database is configured to work with Snort as explained in Chapter 5. The latest version of ACID is available from http://www.cert.org/kb/acid/

ACID (the Analysis Console for Intrusion Detection) is an open source project developed by Roman Danyliw at the CERT coordination center, as part of the AIRCERT project. It uses a PHP-based web application that can act as the frontend for several tools—we will only discuss using ACID with Snort in this chapter. ACID interfaces with the database that Snort uses to log alerts.

ACID should be considered beta software and may be vulnerable to user input validation problems. Care should be taken to secure access to the ACID console (discussed further below). The current version is 0.9.6b23; it has not been updated since January of 2003. It still does an outstanding job in acting as a Snort alert console, but the recent changes in Snort (namely the move from the portscan2 and conversation preprocessors to flow-portscan) have exposed some problems. I still prefer ACID over almost any other open source solution (there are some commercial products that can act as a management console for Snort, too).

ACID was designed to help a security administrator manage the alerts generated by multiple IDS sensors. ACID can generate trending information and allow searches based upon time, address, alert, priority, classification

ACID consists of many PHP scripts and configuration files that work together to collect and analyze information from a database and present it through a Web interface. You have to have a Web server, database server, PHP, and some other tools installed on your system to make it work. I am using a Red Hat Linux 7.1 machine with the Apache Web server, PHP, and MySQL, which are part of the Red Hat distribution.

ACID offers many features:

- You can search on a large number of criteria like source and destination addresses, time, and ports.
- You can view different parts of packet — header parts as well as the payload.
- You can manage alerts by creating alert classes and sending them to an email address.
- Graphical representation includes charts based upon time, protocol, IP addresses, port numbers, and classifications.
- You can take snapshots of the alerts database; for example, you can view alerts for the last 24 hours, unique alerts, or frequent alerts.
- You can go to different who is a database on the Internet to find out who owns a particular IP address that is attacking your network.
-

All of these facilities are available through the Web browser. Support packages like GD library and PHPLOT are used to print graphs on the Web pages. PHP connects to the backend MySQL database to get and update data. For this purpose, you have to provide the database user name and password.

Installation and configuration:

Since ACID needs additional packages like PHPLOT and GD library to work, you need to make sure that everything is installed properly. Fortunately you can install components independently from each other in no particular order. The following step-by-step process makes it easy to put everything in place.

- Install and test Snort.
- Install and test MySQL. Create a database and tables so that Snort can log its activity into the database. After that you have to configure Snort using snort.conf file so that it logs its data to the database server.
- Install Apache.
- Download ACID and uncompress it under the directory where Apache looks for HTML files. (The Apache package that is part of the Red Hat distribution has its HTML files under /var/www/html directory.
- Install PHP. (If you are using a precompiled or RPM version of Apache, PHP may already have been built into it as a module.) Set display_errors variable in /etc/php.ini to Off.
- Install GD library as /usr/lib/libgd.so.
- Uncompress PHPLOT in the directory where Apache looks for HTML files. This software is used to create graphics in the Web pages.
- Download ADODB and install it in the directory where Apache looks for HTML files. ADODB is an object-oriented library written in PHP used to connect to the database.
- If you want to archive old data using ACID, create a MySQL database snort_archive using “create database snort_archive;” command and grant permissions to a user (in our case username rr) to manage the database using the command grant CREATE,INSERT,DELETE,UPDATE,SELECT on snort_archive.* to rr@localhost;.
- Create tables in this database using the command mysql -u rr -p snort_archive <CONTRIB/CREATE_MYSQL.
- Set display_errors variable in /etc/php.ini to Off.

Now configure ACID so that it can interact with the MySQL database. The configuration process also enables Snort to use the PHPLOT package. The configuration process is simple and includes setting up different parameters in the acid_conf.php configuration file which is located in the same directory where you uncompressed the ACID files. In our case, the file is located in the /var/www/html/acid directory. You have to put information about the following items in this file:

- Location of ADODB files. In our case this path is ./adodb, which is the adodb directory under the directory where ACID files are located.
- Type of database server. For the example in this book the type of server is “mysql”.
- MySQL database name for Snort log data.
- MySQL database server name or IP address.
- MySQL database user name and password.
- Name of the archive database if you are using one.
- Database server name where archive database is located. In our case both snort and snort_archive databases are located on localhost.
- Database user name and password to access snort_archive database.
- Location of PHPLOT files. In our case this is ./phplot-4.4.6, which is the phplot-4.4.6 directory under the directory where ACID files are located.

This information is present in the start of the acid_conf.php file. The typical opening lines of this file in my installation are as follows:

```
<?php

$ACID_VERSION = "0.9.6b21";

/* Path to the DB abstraction library
* (Note: DO NOT include a trailing backslash after the
* directory)
* e.g. $foo = "/tmp" [OK]
* $foo = "/tmp/" [OK]
* $foo = "c:tmp" [OK]
* $foo = "c:tmp" [WRONG] */
$DBlib_path = "./adodb";
/* The type of underlying alert database*
* MySQL : "mysql"
* PostgresSQL : "postgres"
* MS SQL Server : "mssql"*/
$DBtype = "mysql";
/* Alert DB connection parameters
* - $alert_dbname : MySQL database name of Snort: alert DB
* - $alert_host : host on which the DB is stored
* - $alert_port : port on which to access the DB
* - $alert_user : login to the database with: this user
* - $alert_password : password of the DB user*
* This information can be gleaned from the Snort database
* output plugin configuration.*/
$alert_dbname = "snort";
$alert_host = "localhost";
$alert_port = "";
```

```
$alert_user = "rr";
$alert_password = "rr78x";
/* Archive DB connection parameters */
$archive_dbname = "snort_archive";
$archive_host = "localhost";
$archive_port = "";
$archive_user = "rr";
$archive_password = "rr78x";
/* Type of DB connection to use
* 1 : use a persistant connection (pconnect)
* 2 : use a normal connection (connect)*/
$db_connect_method = 1;
/* Path to the graphing library
* (Note: DO NOT include a trailing backslash after the directory)*/
$ChartLib_path = "./phplot-4.4.6";
```

Use the same user name, password, and database name as you use in snort.conf file.

Using ACID

If you have installed everything right, you should now be able to access ACID by going to URL http://<your_web_server>/acid/. The first time you visit this URL, ACID needs to perform some setup tasks. Click the Setup page link to move to the DB Setup page. Click the “Create ACID AG” link so that ACID can create its own table to support Snort. ACID creates these tables in the main Snort database and uses them for its own housekeeping data. You can now click the “Main Page” link towards the bottom of the page to go to the main ACID page.

The ACID main page provides an overview of currently available data. It has different sections to display information in groups. You can view traffic profiles by different protocols, get a snapshot of sensors, search data and see:

- A list of sensors that are logging data to the database.
- The number of unique alerts and their detail.
- The total number of alerts and their detail.
- Source IP addresses for the captured data. By following the subsequent links, you can find the owner of the source IP address by looking up who is databases.
- Destination IP addresses for captured data.
- Source and destination ports.
- Alerts related to a particular protocol, like TCP alerts, UDP alerts, and ICMP alerts.
- Search alert and log data for particular entries.
- Most frequent alerts.
- Plot alert data, which is still experimental.

ACID can search the captured log and alert data using parameters such as:

- A particular sensor, when you are using a central database to log data from many Snort sensors.
- Time of alert using start and ending time.

- Source and destination addresses.
- Different fields in the IP packet header.
- Transport layer protocols.
- String of data in the payload area of the IP packet.

Searching for data in the database is easy. All the criteria that you specify in this screen are translated to a SQL statement that is passed to the MySQL database server. Results of your query are displayed when you click the "Query DB" button. You can then click a particular alert line to find out more information about that alert.

Snort can also be used to find fully qualified names for source and destination addresses found in captured data. For example, to create a list of unique destination IP addresses and hostnames, you can write a rule that creates an alert for all outgoing HTTP requests, though of course that is not intrusion activity.

To get whois information about a particular address, you can click on any address and select a who is database, like American Registry for Internet Numbers (ARIN). This information is usually the first step to finding out the owner of the attacking IP address and his contact information. Once you have it, you can contact the owner and ask him to stop bad guys from probing your network.

Generating graphs and archiving data:

Generating graphs is still experimental in ACID. You can go to the ACID main page where a link is provided to generate graphs. When generating graphs, you can select data and type of graph. For example, you can generate a line or bar graph for alerts in the last five days.

A sample bar graph

ACID uses the PHPLOT package on the backend side to generate these graphs. You can use JpGraph in place of PHPLOT. JpGraph has a different licensing scheme and there may be some restrictions for using it in commercial environment. In addition to the tasks presented here, you can also use ACID to archive data and delete data from the database.

Earlier, you created a new database called snort_archive to archive the data from the main Snort database. Using ACID, you can move alerts from the main database to the archive database or just copy them. For example, if you want to move all alerts from the main database to the archive database, click the number next to "Total Number of Alerts" on the main ACID page. The next page displays all of the alerts in the database. If the number of alerts is more than 50, then only the first 50 alerts are displayed. Then you can use the bottom part of the screen to archive the alerts.

SnortSnarf and Barnyard

In addition to ACID, the article also provides basic information about SnortSnarf, another tool to display Snort data using a Web interface. SnortSnarf is able to parse Snort log files and generate HTML pages that can be viewed using a Web browser.

SnortSnarf is a Perl script; you can run it after downloading without going through any compilation process. You can run SnortSnarf from a cron script on a periodic basis.
It can parse Snort log files as well as extract data from MySQL database. The following command parses

var/log/snort/alert file and places the newly generated HTML files in the /var/www/html/snortsnarf directory where they can be viewed later using a Web browser.

```
snortsnarf.pl /var/log/snort/alert -d /var/www/html/snortsnarf
```

The following command extracts data from MySQL database running on the localhost. It uses a user name rr and password rr78x to login to the database.

```
snortsnarf.pl rr:rr78x@snort@localhost -d /var/www/html/snortsnarf
```

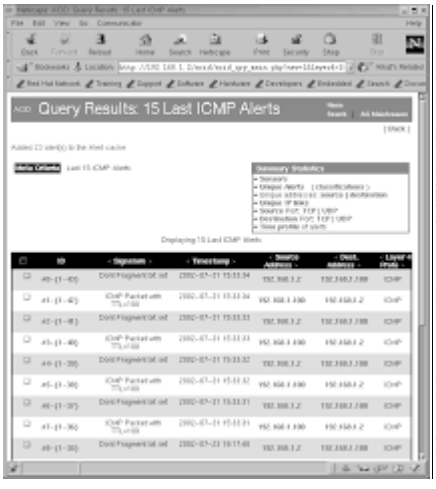
To get data from a database, you have to define the following parameters on the command line:

- Database user name
- Password
- Database name
- Host where database server is running
- Port number for the database server. By default the port number is 3306 and this parameter is optional.
-

The general format of defining these parameters is user:passwd@dbname@host:port.

The SnortSnarf main page provides basic information about alert data.

You may also want to try Barnyard, a new tool intended to parse binary log files generated by Snort when you use the unified logging module. Download the package, decompress it, and run the configure script with a prefix command line parameter to define the directory where you intend to install it. A typical command line may be `configure --prefix=/opt/barnyard`. Run the `make` command, then run `make install` to install it. You also need to edit the barnyard.conf file before using the tool.



Agent development for intrusion detection systems: When Snort is running in the Network Intrusion Detection (NID) mode, it generates alerts when a captured packet matches a rule. Snort can send alerts in many modes.
These modes are configurable through the command line as well as through snort.conf file. Common alert modes are explained in this section. To explain the alert modes, I have used a rule that creates an alert when Snort detects an ICMP packet with TTL 100. This rule is listed below.

alert icmp any any -> any any (msg: "Ping with TTL=100"; \ttl:100;)

Rules will be explained in the next chapter in detail. For this discussion, it is sufficient to understand that this rule will create an alert with the text message "Ping with TTL=100" whenever such an ICMP packet is captured. The rule does not care about source or destination address in the packet. I have used the following command on my Windows PC to send one ICMP echo packet with TTL=100.

C:\rrehman>**ping -n 1 -i 100 192.168.1.3**
Pinging 192.168.1.3 with 32 bytes of data:
Reply from 192.168.1.3: bytes=32 time=3ms TTL=255
Ping statistics for 192.168.1.3:
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Snort Alert Modes 67
Approximate round trip times in milli-seconds:
Minimum = 3ms, Maximum = 3ms, Average = 3ms
C:\rrehman>

The "-n 1" command line option is used to send only one ICMP packet. The "-i 100" option is used to set the TTL value equal to 100 in the ICMP packet. For details on the format of ICMP packet headers, refer to RFC 792 at ftp://ftp.isi.edu/innotes/ rfc792.txt or Appendix C. Whenever this command is executed, Snort captures the ICMP packet and creates an alert. The amount of information logged with the alert depends on the particular alerting mode. Now let us see how different alerting modes work on a packet.

Fast Mode: The fast alert mode logs the alert with following information:
• Timestamp
• Alert message (configurable through rules)
• Source and destination IP addresses
• Source and destination ports

To configure fast alert mode, you have to use "`-A fast`" command line option.
This alert mode causes less overhead for the system. The following command starts Snort in fast alert mode:

```
/opt/snort/bin/snort -c /opt/snort/etc/snort.conf -q -A fast
```

The `–q` option used on the command line stops the initial messages and final statistical summary from being displayed on the screen. Now when you create an alert, it will be logged in `/var/log/snort/alert` file. However, you can change the location of this file using `-l` command line option. The alert message is similar to the following:

```
05/28-22:16:25.126150 [**] [1:0:0] Ping with TTL=100 [**]
{ICMP} 192.168.1.100 -> 192.168.1.3
```

This alert message shows the following information:
• Date and time the alert occurred.
• Message present in the rule that generated this alert. In this example, the message is "Ping with TTL=100".

• Source address which is 192.168.1.100
• Destination address which is 192.168.1.3.
• Type of packet; in the above example, type of packet is ICMP.
Note that the actual packet is not logged in this file when using this alert mode.

Full Mode:

This is the default alert mode. It prints the alert message in addition to the packet header. Let us start Snort with full alerting enabled with the following command:

/opt/snort/bin/snort -c /opt/snort/etc/snort.conf -q -A full

When Snort generates an alert in this mode, the message logged in /var/log/snort/alert file is similar to the following:

[**] [1:0:0] Ping with TTL=100 [**]
05/28-22:14:37.766150 192.168.1.100 -> 192.168.1.3
ICMP TTL:100 TOS:0x0 ID:40172 IpLen:20 DgmLen:60
Type:8 Code:0 ID:768 Seq:20224 ECHO

As you can see, additional information is logged with the alert message. This additional information shows different values in the packet header, including:

• Time to Live (TTL) value in the IP packet header. For details on TTL value, refer to RFC 791 at ftp://ftp.isi.edu/in-notes/rfc791.txt
• The Type Of Service (TOS) value in the IP packet header. For details on TOS value, refer to RFC 791 at at ftp://ftp.isi.edu/in-notes/rfc791.txt and Appendix C.
• Length of IP packet header shown as IpLen:20.
• Total length of IP packet shown as DgmLen:60.
• ICMP Type field. For details on ICMP type field refer to RFC 792.
• ICMP code value. For details on ICMP type field refer to RFC 792.
• IP packet ID.
• Sequence number.
• ICMP packet type which is ECHO

UNIX Socket Mode:
If you use "-a unsock" command line option with Snort, you can send alerts to another program through UNIX sockets. This is useful when you want to process alerts using a custom application with Snort. For more information on socket, use the "man socket" command.

No Alert Mode:
You can also completely disable Snort alerts using "-A none" command line option. This option is very useful for high speed intrusion detection using unified logging. You can disable normal logging using this option while using the unified option.

Sending Alerts to Syslog:
This command allows Snort to send alerts to Syslog daemon. Syslog is a system logger daemon and it generates log files for system events. It reads its configuration file /etc/syslog.conf where the location of these log files is configured. The usual location of syslog files is /var/log directory. On Linux systems, usually /var/ log/messages is the main logging file. For more information, use the "man syslog" command. The "man syslog.conf" command shows the format of the syslog. conf file. Depending on the configuration of the Syslog using /etc/syslog.conf file, the alerts can be saved into a particular file. The following command enables Snort to log to the Syslog daemon: /opt/snort/bin/snort -c /opt/snort/etc/snort.conf –s.

Using the default configuration on my RedHat 7.1 computer, the messages are logged to /var/log/messages file. When you cause an alert message by sending the special ICMP packet with TTL=100, the following line will be logged to the /var/ log/messages file.

snort snort[1750]: [1:0:0] Ping with TTL=100 {ICMP} 192.168.1.100 -> 192.168.1.3

Using Syslog facility will be discussed in Chapter 4 later on in this book. You will also learn how to enable logging to Syslog using the output plug-in.

Sending Alerts to SNMP:
One very useful feature of Snort is SNMP traps. You can configure an output plug-in to send messages in the form of SNMP traps to a network management system. Using this feature you can integrate your intrusion detection sensors into any centralized NMS like HP OpenView, OpenNMS, MRTG and so on. Snort can generate SNMP version 2 and version 3 traps. The configuration process for SNMP traps will be discussed later on in detail.

Sending Alerts to Windows:
Snort can send alerts to Microsoft Windows machines in the form of pop-up windows. These pop-up windows are controlled by Windows Messenger Service. Windows Messenger Service must be running on your Windows machine for pop-up windows to work. You can go to Control Panel and start the *Services* applet to find out if Windows Messenger Service is running. The *Services* applet is found in the Administrative Tools menu on your Windows system. Depending on your version of Microsoft Windows, it may be found in Control Panel or some other place.

The SAMBA client package must be installed on your UNIX machine. SAMBA is an open source software suite that allows UNIX file and printer sharing with Microsoft Windows machines. SAMBA software runs on UNIX platforms. It can work with any other operating system that understands Common Internet File System (CIFS) or Server Message Block (SMB) protocol. More information about SAMBA is available from http://www.samba.org.

The Snort alert mechanism uses smbclient program on the UNIX machine to connect to the Windows machines and send the alerts. Make sure that the SAMBA client is working properly before trying to use this service. SAMBA operations are dependent upon its configuration file /etc/samba/smb.conf on a RedHat system. This file may be located at a different place on other UNIX systems. Although detailed discussion on SAMBA is beyond the scope of this book, a sample SAMBA configuration file is listed below. This file can be used to jump start SAMBA. The file creates a workgroup REHMAN which you can view from "Network Neighborhood" part of your Windows machines.

Sample Samba Configuration File:
A sample /etc/samba/smb.conf file is as follows:

```
[global]
workgroup = REHMAN
server string = REHMAN file server
log file = /var/log/samba/log.%m
max log size = 50
security = user
encrypt passwords = yes
socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
dns proxy = no
domain logons = no
unix password sync = no
map to guest = never
password level = 0
null passwords = no
os level = 0
preferred master = yes
```

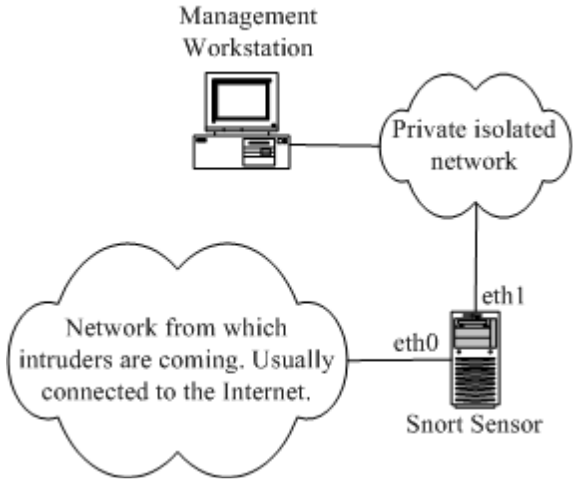
```
domain master = yes
wins support = yes
dead time = 0
debug level = 0
load printers = yes
[homes]
comment = Home Directories
browseable = yes
writable = yes
available = yes
public = yes
only user = no
[htmldir]
comment = html stuff
path = /home/httpd/html
public = yes
writable = yes
printable = no
write list = rehman
[virtualhosting]
comment = html stuff
path = /usr/virt_web
public = yes
writable = yes
printable = no
write list = rehman
[printers]
[netlogon]
available = no
```

Running Snort in Stealth Mode:

Sometimes you may want to run Snort in stealth mode. In stealth mode, other hosts are not able to detect the presence of the Snort machine. In other words, the Snort machine is not visible to intruders or other people. There are multiple ways to run Snort in stealth mode. One of these methods is to run Snort on a network interface where no IP address is assigned. Running Snort on a network interface without an IP address is feasible in the following two cases:

1. A stand-alone Snort sensor with only one network adapter.

2. A Snort sensor with two network adapters: one to access the sensor from an isolated network and the other one connected to the public network and running in stealth mode. This arrangement is shown in Figure where network interface eth1 is connected to a private isolated network and eth0 is connected to a public network:



Running Snort in stealth mode on a system with two network adapters

When you want to access the sensor itself, you go through network interface eth1 which has an IP address configured to it. The management workstation shown in the figure may be used to connect to the sensor either to collect data or to log information to a centralized database. If many sensors are present in an organization, all of these are connected to this isolated network so that they can log information to the central database running on the management workstation or to some other database server connected to this isolated network.

No IP address is configured on network interface eth0 which has connectivity to the Internet. Interface eth0 remains in stealth mode but can still listen to the network traffic from this side of the network.

Before starting Snort on eth0, you have to bring it up. On Linux systems, you can do it by using the following command: ifconfig eth0 up The command makes the interface usable without allocating an IP address. After that, you can start Snort on this interface by using "-i eth0" command line option as follows: snort -c /opt/snort/etc/snort.conf -i eth0 –D.

Architecture models of IDs and IPs:

The architecture is one of the most critical considerations in intrusion detection and prevention. An effective architecture is one in which each machine, device, component, and process performs its role in an effective and (often) coordinated manner, resulting in efficient information processing and output, and also appropriate preventive responses that meet the business and operational needs of an organization. A poorly designed or implemented architecture, on the other hand, can produce a variety of undesirable consequences, such as data not being available or not being available when needed, networking slowdowns, or a lack of appropriate and timely responses.

Tiered Architectures

At least three types of tiered architectures can be used: single-tiered, multi-tiered, and peer-to-peer architectures.

Single Tiered Architecture:

A *single-tiered architecture*, the most basic of the architectures discussed here, is one in which components in an IDS or IPS collect and process data themselves, rather than passing the output they collect to another set of components. An example of a single-tiered architecture is a host-based intrusion-detection tool that takes the output of system logs (such as the utmp and wtmp files on Unix systems) and compares it to known patterns of attack.

A single tier offers advantages, such as simplicity, low cost (at least in the case of freeware tools running on individual hosts), and independence from other components (an advantage if they should become compromised or disabled). At the same time, however, a single-tiered architecture usually has components that are not aware of each other, reducing considerably the potential for efficiency and sophisticated functionality.

Multi Tiered Architecture:

As the name implies, a multi-tiered architecture involves multiple components that pass information to each other. Many of today's IDSs, for example, consist of three primary components: sensors, analyzers or agents, and a manager.

Sensors perform data collection. For example, network sensors are often programs that capture data from network interfaces. Sensors can also collect data from system logs and other sources, such as personal firewalls and TCP wrappers.

Sensors pass information to *agents* (sometimes also known as *analyzers*), which monitor intrusive activity on their individual hosts. Each sensor and agent is configured to run on the particular operating environment in which it is placed. Agents are normally specialized to perform one and only one function. One agent might, for example, examine nothing but TCP traffic, whereas another might examine only FTP (File Transfer Protocol) connections and connection attempts. Additionally, third-party tools, such as network-monitoring tools, neural networks and connection-tracing tools can be used if expanding the scope of analysis is advantageous.

When an agent has determined that an attack has occurred or is occurring, it sends information to the *manager component*, which can perform a variety of functions including (but not limited to) the following:

- Collecting and displaying alerts on a console
- Triggering a pager or calling a cellular phone number
- Storing information regarding an incident in a database
- Retrieving additional information relevant to the incident
- Sending information to a host that stops it from executing certain instructions in memory
- Sending commands to a firewall or router that change access control lists
- Providing a management console—a user interface to the manager component

A central collection point allows for greater ease in analyzing logs because all the log information is available at one location. Additionally, writing log data to a different system (the one on which the manager component resides) from the one that produced them is advisable; if an attacker tampers with or destroys log data on the original system (by installing a rootkit tool that masquerades the attacker's presence on the system, for instance), the data will still be available on the central server—the manager component. Finally, management consoles can enable intrusion-detection and intrusion-prevention staff to remotely change policies and parameters, erase log files after they are archived, and perform other important functions without having to individually authenticate to sensors, agents, and remote systems. Figure 6-1 outlines the multi-tier architecture.

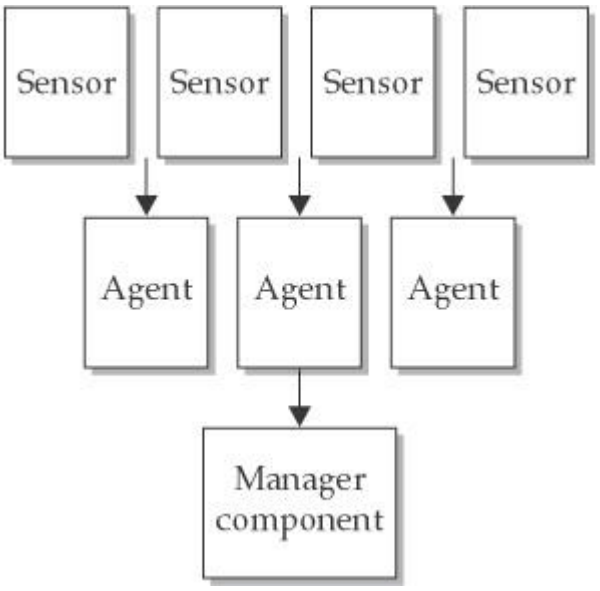


Figure: A multi-tiered architecture

Advantages of a multi-tiered architecture include greater efficiency and depth of analysis. With each component of the architecture performing the function it is designed to do, often mostly independent of the other components, a properly designed multi-tiered architecture can provide a degree of efficiency not possible with the simpler single- tiered architecture. It can also provide a much more complete picture of the security condition of an organization's entire network and the hosts therein, compared to a single-tiered architecture. The main downsides include increased cost and complexity. The multiple components, interfaces, and communications methods translate to greater difficulty in setting up this architecture and more day-to-day maintenance and troubleshooting challenges.

Peer to Peer Architecture:

Whereas the multi-tiered architecture generally takes raw information, processes it, then sends the output to a higher-order component, the peer-to-peer architecture involves exchanging intrusion-detection and intrusion-prevention information between peer components, each of which performs the same kinds of functions. This peer-to-peer architecture is often used by cooperating firewalls (and, to a lesser degree, by cooperating routers or switches). As one firewall obtains information about events that are occurring, it passes this information to another, which may cause a change in an access control list or addition of restrictions on proxies connections. The second firewall can also send information that causes changes in the first. Neither firewall acts as the central server or master repository of information.

The main advantage of a peer-to-peer architecture is simplicity. Any peer can participate in what is effectively a group of peer machines, each of which can benefit from the information the others glean. The main downside is a lack of sophisticated functionality due to the absence of specialized components (although the functionality is better than what is possible in a single-tiered architecture because the latter does not even have cooperating components).

Critical Need for Architectures:

Architectures make a critical difference in terms of the quantity and quality of the dividends that intrusion-detection and intrusion-prevention technology produce. At the same time, however, it is important to remember that "Rome was not built in a day." If you lack financial resources, starting with a simple single-tiered architecture is often the logical first step in developing an intrusion-detection and intrusion-prevention architecture. As more resources become available, and as the exact business and operational objectives of intrusion-detection and intrusion-prevention efforts become clearer, a migration to a more sophisticated architecture can occur.

A peer-to-peer architecture is well suited to organizations that have invested enough to obtain and deploy firewalls capable of cooperating with each other, but that have not invested much (if anything) in IDSs and IPSs. As discussed earlier, firewalls are, all things considered, the best single source of intrusion-detection data. Using them in a peer-to-peer manner to distribute information they have obtained and to make adaptive changes in access control lists or proxy rules can, at least to some degree, compensate for the absence of IDSs and IPSs.

Sensors:

Sensors are critical in intrusion-detection and intrusion-prevention architectures—they are the beginning point of intrusion detection and prevention because they supply the initial data about potentially malicious activity. A deeper look at sensor functionality, deployment, and security will provide insight into exactly what sensors are and how they work.

Sensor Functions:

Considering all the possible intrusion-detection and intrusion-prevention components within a particular architecture, sensors are usually (but not always) the lowest end components. In other words, sensors typically do not have very sophisticated functionality. They are usually designed only to obtain certain data and pass them on. There are two basic types of sensors: network-based and host-based sensors.

Network-Based Sensors:

Network-based sensors, the more frequently deployed of the two types, are programs or network devices (such as physical devices) that capture data in packets traversing a local Ethernet or token ring or a network switching point. One of the greatest advantages of network-based sensors is the sheer number of hosts for which they can provide data. In an extreme case, one sensor might be used to monitor all traffic coming into and out of a network. If the network has a thousand hosts, the sensor can, in theory, gather data about misuse and anomalies in all thousand hosts. The cost-effectiveness of this approach is huge (although critics

justifiably point out that a single sensor is also likely to miss a considerable amount of data that may be critical to an intrusion-detection and intrusion-prevention effort if the sensor does not happen to be recording traffic on the particular network route over which packets containing the data are sent). Additionally, if configured properly, sensors do not burden the network with much additional traffic, especially if two network interfaces—one for monitoring and the other for management—are used. A monitoring interface has no TCP/IP stack whatsoever, nor does it have any linkage to any IP address, both of which make it an almost entirely transparent entity on the network.

Host-Based Sensors:

Host-based sensors, like network-based sensors, could possibly also receive packet data captured by network interfaces and then send the data somewhere. Instead of being set to promiscuous mode, the network interface on each host would have to be set to capture only data sent to that particular host. However, doing so would not make much sense, given the amount of processing of data that would have to occur on each host. Instead, most host-based sensors are programs that produce log data, such as Unix daemons or the Event Logger in Windows NT, 2000, XP, and Windows Server 2003. The output of these programs is sent (often through a utility such as scp, secure copy, which runs as a cron job, or through the Windows Task Scheduler) to an analysis program that either runs on the same host or on a central host. The program might look for events indicating that someone has obtained root privileges on a Unix system without entering the su (substitute user) command and the root password—a possible indication that an attacker has exploited a vulnerability to gain root privileges.

Sensor Deployment Considerations:

Many sensors require that a host be running one or more network interfaces in promiscuous mode. In many current Unix systems, entering this command will produce standard output that displays the IP address, the MAC address, the net mask, and other important parameters, including "promise" if the interface is in promiscuous mode. Note that if there is only one network interface, it is not necessary to enter the name of the interface in question. "ifconfig ".

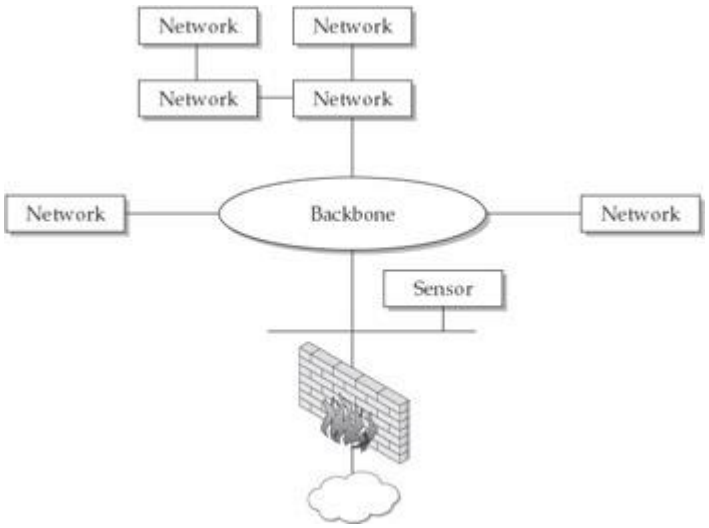


Figure: Placement of a sensor within a DMZ

More Considerations for Network-Based Sensor Functionality: Ensuring that a network interface is in promiscuous mode is critical for network-based sensor functionality, but there are other important technical considerations, too. For one thing, programs that capture data or read these data need certain privileges or access rights. In most versions of Linux, for example, root privileges are necessary for accessing captured data in files such as/var/log/messages. Running windump, the Windows version of tcpdump, requires Administrator-level privileges on systems such as Windows 2000 or XP. Running tcpdump (or, in the case of Windows systems, windump) sets the network interface in promiscuous mode. Disk space management is another important consideration. Capturing raw packet data off the network can quickly eat up precious disk space. Having a huge amount of disk capacity, regularly checking how full the disk is, and archiving and then purging the contents of files that hold raw packet data is usually a necessary part of operations associated with intrusion detection and prevention.

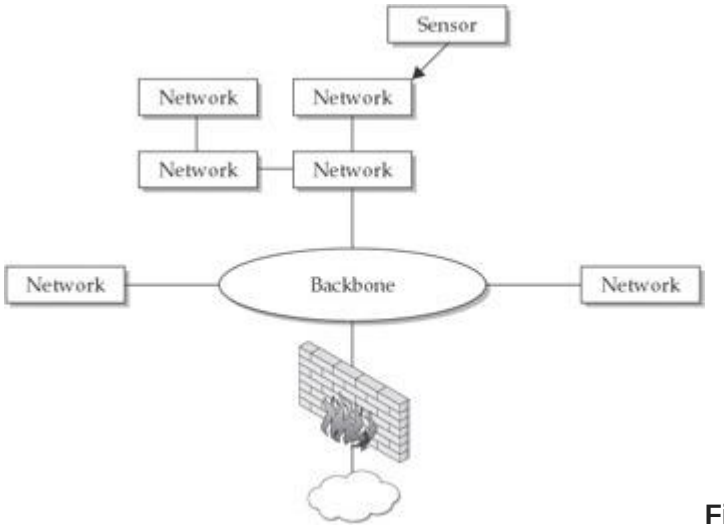


Figure: Deploying sensors in the network periphery

Host-based sensors can be placed at only one point—on a host—so the point within the network where this type of sensor is deployed is not nearly as much of an issue. As always, the benefits should outweigh the costs. The costs of deploying host-based sensors generally include greater financial cost (because of the narrower scope of host-based as opposed to network-based sensors), greater utilization of system resources on each system on which they are deployed, and the consequences of being blind to what is happening on a host due to unauthorized disabling of the sensor on the host (especially if that host is a sensitive or valuable system). Although network-based sensors are generally used in DMZs, for example, deploying a host-based sensor on a particularly critical public web server within a DMZ would be reasonable.

A hybrid approach—deploying network-based sensors both at external gateways as well as at gateways to subnets or within virtual local area networks (VLANs) and using host-based sensors where most needed—is in many cases the best approach to deploying sensors. This kind of sensor deployment ensures that packet data for traffic going in and out of the network, as well as at least some of the internal traffic, will be captured. If a sensor at an external gateway becomes overwhelmed with data, data capture within the network itself can still occur. Furthermore, although the network-based sensors at external gateways are unlikely to glean information about insider attacks, the internal network-based sensors are much more likely to do so. At the same time, deploying host-based sensors on especially sensitive and valuable servers is likely to yield the information necessary to determine whether inside or outside attacks have occurred and, in the case of IPSs, may possibly stop malicious code from executing or unauthorized commands from being run in the first place. Finally, if host-based sensors fail, there will at least be some redundancy—network-based sensors (especially the internally deployed network-based sensors) can provide some information about attacks directed at individual systems.

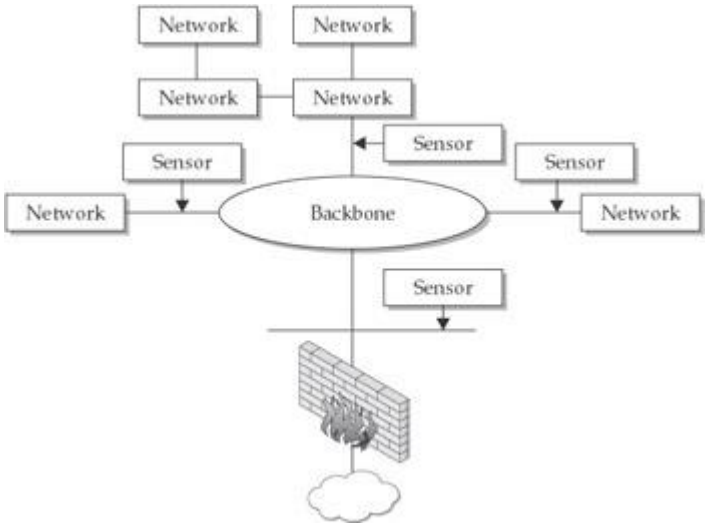


Figure: A hybrid approach to deploying sensors

Agents:

Agents are the next consideration in intrusion-detection and intrusion-prevention architectures. This section discusses the functions, deployment considerations, and security considerations associated with agents.

Agent Functions:

Agents are relatively new in intrusion detection and prevention, having been developed in the mid-1990s. As mentioned previously in this chapter, their primary function is to analyze input provided by sensors. Although many definitions exist, we'll define an *agent* as a group of processes that run independently and that are programmed to analyze system behavior or network events or both to detect anomalous events and violations of an organization's security policy. Each agent should ideally be a bare-bones implementation of a specialized function. Some agents may, for example, examine network traffic and host-based events rather generically, such as checking whether normal TCP connections have occurred, their start and stop times, and the amount of data transmitted or whether certain services have crashed. Having agents that examine UDP and ICMP traffic is also desirable, but the UDP and ICMP protocols are stateless and connectionless. Other agents might look at specific aspects of application layer protocols such as FTP, TFTP, HTTP, and SMTP as well as authentication sessions to determine whether data in packets or system behavior is consistent with known attack patterns. Still others may do nothing more than monitor the performance of systems.

Our definition of agent states that agents run independently. This means that if one agent crashes or is impaired in some manner, the others will continue to run normally (although they may not be provided with as much data as before). It also means that agents can be added to or deleted from the IDS or IPS as needed. In fact, in a small intrusion-detection or intrusion-prevention effort, perhaps only a few of two dozen or so agents may be deployed. In a much larger effort, perhaps all of the agents may be deployed.

Although each agent runs independently on the particular host on which it resides, agents often cooperate with each other. Each agent may receive and analyze only one part of the data regarding a particular system, network, or device. Agents normally share information they have obtained with each other by using a particular communication protocol over the network, however. When an agent detects an anomaly or policy violation (such as a brute force attempt to su to root, or a massive flood of packets over the network), in most cases, the agent will immediately notify the other agents of what it has found. This new information, combined with the information another agent already has, may cause that agent to report that an attack on another host has also occurred.

Agents sometimes generate false alarms, too, thereby misleading other agents, at least to some degree. The problem of false alarms is one of the proverbial vultures hovering over the entire intrusion-detection and intrusion-prevention arena, and cooperating but false-alarm-generating agents can compound this problem. However, a good IDS or IPS will allow the data that agents generate to be inspected on a management console, allowing humans to spot false alarms and to intervene by weeding them out.

The Advantages and Disadvantages of Agents:

The use of agents in intrusion detection and prevention has proven to be one of the greatest breakthroughs. Advantages include:

- **Adaptability** Having a number of small agents means that any of them can potentially be modified to meet the needs of the moment; agents can even be programmed to be self-learning, enabling them to be able to deal with novel threats.
- **Efficiency** The simplicity of most agent implementations makes them more efficient than if each agent were to support many functions and to embody a great deal of code.
- **Resilience** Agents can and do maintain state information even if they fail or their data source fails.
- **Independence** Agents are implemented to run independently, so if you lose one or two, the others will not be affected.
- **Scalability** Agents can readily be adapted to both large- and small-scale intrusion-detection and intrusion-prevention deployments.

- **Mobility** Some agents (believe it or not) may actually move from one system to another; agents might even migrate around networks to monitor network traffic for anomalies and policy violations.

  There are some drawbacks to using agents, too:

- **Resource allocation** Agents cause system overhead in terms of memory consumption and CPU allocation.
- **False alarms** False alarms from agents can cause a variety of problems.
- **Time, effort, and resources needed** Agents need to be modified according to an organization's requirements, they must be tuned to minimize false alarms, and they must be able to run in the environment in which they are deployed—this requires time, effort, and financial and other resources.
- **Potential for subversion** A compromised agent is generally a far greater problem than a compromised sensor.

At a bare minimum, an agent needs to incorporate three functions or components:

- A *communications interface* to communicate with other components of IDSs and IPSs
- A *listener* that waits in the background for data from sensors and messages from other agents and then receives them
- A *sender* that transmits data and messages to other components, such as other agents and the manager component, using established means of communication, such as network protocols

  Agents can also provide a variety of additional functions. Agents can, for example, perform correlation analyses on input received from a wide range of sensors. In some agent implementations, the agents themselves generate alerts and alarms. In still other implementations, agents access large databases to launch queries to obtain more information about specific source and destination IP addresses associated with certain types of attacks, times at which known attacks have occurred, frequencies of scans and other types of malicious activity, and so forth. From this kind of additional information, agents can perform functions such as tracking the specific phases of attacks and estimating the threat that each attack constitutes.

  Although the types of additional functions that agents can perform may sound impressive, "beefing up" agents to do more than simple analysis is not necessarily advantageous. These additional functions can instead be performed by the manager component (to be discussed shortly), leaving agents free to do what they do best. Simplicity—in computer science jargon, *Occam's razor*—should be the overwhelming consideration with agents, provided, of course, that each agent implementation embodies the required functionality. Additionally, if resource utilization is already a problem with simple agents, think of the amount of resources multifunctional agents will use!

Agent Deployment Considerations:

Decisions about deployment of agents are generally easier to make than decisions concerning where to deploy sensors. Each agent can and should be configured to the operating environment in which it runs. In host-based intrusion detection, each agent generally monitors one host, although, as mentioned before, sometimes sensors on multiple hosts send data to one or more central agents. Choosing the particular hosts to monitor is thus the major dilemma in deciding on the placement of host-based agents. Most organizations that use host-based intrusion detection select "crown jewel" hosts, such as servers that are part of billing and financial transaction systems, more than any other. A few organizations also choose a few widely dispersed hosts throughout the network to supplement network-based intrusion detection.

In network-based intrusion detection, agents are generally placed in two locations:

- **Where they are most *efficient*** Efficiency is related to the particular part of a network where connections to sensors and other components are placed. The more locally co resident the sensors

and agents are, the better the efficiency. Having an agent in one network and the sensors that feed the agent in another is an example of inefficiency.

- **Where they will be sufficiently *secure*** Security of agents is our next topic, so suffice it to say here that placing agents in secure zones within networks, or at least behind one or more firewalls, is essential.

Agent Security Considerations:

The threat of subversion of agents is a major issue. Agents are typically much smarter than sensors; if an agent is successfully attacked, not only will the attacker be able to stop or subvert the type of analysis that the agent performs, but this person will also be able to glean information that is likely to prove useful in attacking the other components of the IDS or IPS. Compromised agents thus can rapidly become a security liability.

Fortunately, the way agents are typically deployed provides at least some level of defense against attacks that are directed at them. Agents (especially in network-based IDSs and IPSs) are generally distributed throughout a network or networks. Each agent must therefore be individually discovered and then attacked. This substantially increases the work involved in attacking agents, something that is very desirable from a security perspective. The diversity of functionality within agents also provides some inherent security—each is to some degree unique, and attacking each presents unique challenges to the attacker. Additionally, mobile agents are becoming increasingly popular, and agent mobility makes discovery of agents by attackers considerably more difficult.

Nevertheless, agents need to be secured by doing many of the same things that must be done to protect sensors—hardening the platform on which they run, ensuring that they can be accessed only by authorized persons, and so on. Here are a few guidelines:

- **Dedicate the hardware platform** Dedicating the hardware platform on which agents run to agent functionality is essential. If other applications run on the same platform as one that houses one or more agents, attackers may be able to access the platform via the other applications and then escalate privileges to the point where they gain control over all the agents.
- **Encrypt traffic** Because of the high importance of agent security, encrypting all traffic between agents and other agents and possibly also between agents and other components is also advisable. Including a digital signature that
  must be validated before any message is processed is another good measure.
- **Filter input** Additionally, to guard against denial-of-service attacks, filters that prevent excessive and repetitive input from being received should be deployed. Many vendor agent implementations have this kind of filtering capability built in.

Other interesting approaches to agent security include using APIs (application programming interfaces) to control data transfer between agents. In this approach, one of the most important considerations is sanitizing the data transferred between agents to guard against the possibility of exploiting vulnerabilities to gain control of agents and the platforms on which they run by passing specially crafted data.

Manager Component:

The final component in a multi-tiered architecture is the *manager* (sometimes also known as the *server*) component. The fundamental purpose of this component is to provide an executive or master control capability for an IDS or IPS.

Manager Functions:

We've seen that sensors are normally fairly low-level components and that agents are usually more sophisticated components that, at a minimum, analyze the data they receive from sensors and possibly from each other. Although sensors and agents are capable of functioning without a master control component, having such a component is extremely advantageous in helping all components work in a coordinated

manner. Additionally, the manager component can perform other valuable functions, which we'll explore next.

Data Management:

IDSs and IPSs can gather massive amounts of data. One way to deal with this amount of data is to compress (to help conserve disk space), archive it, and then periodically purge it. This strategy, however, is in many cases flawed, because having online rather than archived data on storage media is often necessary to perform the necessary ongoing analyses. For example, you might notice suspicious activity from a particular internal host and wonder if there has been similar activity over the last few months. Going to a central repository of data is preferable to having to find the media on which old data reside and restoring the data to one or more systems.

Having sufficient disk space for management purposes is, of course, a major consideration. One good solution is RAID (Redundant Array of Inexpensive Disks), which writes data to multiple disks and provides redundancy in case of any disk failing. Another option is optical media, such as worm drives (although performance is an issue).

Ideally, the manager component of an IDS or IPS will also organize the stored data. A relational database, such as an Oracle or Sybase database, is well suited for this purpose. Once a database is designed and implemented, new data can be added on the fly, and queries against database entries can be made.

Alerting:

Another important function that the manager component can perform is generating alerts whenever events that constitute high levels of threat occur (such as a compromise of a Windows domain controller or a network information service (NIS) master server, or of a critical network device, such as a router). Agents are designed to provide detection capability, but agents are normally not involved in alerting because it is more efficient to do so from a central host. Agents instead usually send information to a central server that sends alerts whenever predefined criteria are met. This requires that the server not only contain the addresses of operators who need to be notified, but also have an alerting mechanism.

Event Correlation:

Another extremely important function of the manager component is correlating events that have occurred to determine whether they have a common source, whether they were part of a series of related attacks, and so forth.

High-Level Analysis:

Still another function that the manager component may perform is high-level analysis of the events that the intrusion-detection or intrusion-prevention tool discovers. The manager component may, for example, track the progression of each attack from stage to stage, starting with the preparatory (doorknob rattling) stage. Additionally, this component can analyze the threat that each event constitutes, sending notification to the alert- generation function whenever a threat reaches a certain specified value. Sometimes high-level analysis is performed by a neural network or expert system that looks for patterns in large amounts of data.

The Importance of Being Able to View Packet Data:

Management consoles should allow operators to readily access all data that an IDS or IPS gathers, sends, and processes. A type of data not available in some implementations of IDSs and IPSs is packet data. Saving packet data, after all, requires huge amounts of disk space. But packet data are often one of the most useful types of data for operators. Errors in applications, the universal tendency for IDSs to produce false alarms, and tricks played by attackers can, for example, cause erroneous output or may prompt further questions. Going directly to packet data often resolves these issues and questions.

Suppose, for example, that an IDS reports a distributed scan from a certain range of IP addresses. Going to the IP portion of each packet will yield information such as the indicated IP source address, the version of the IP protocol, and the TTL (time-to-live parameter). If the apparent IP source addresses for the distributed scan are 15 hops away (as revealed by the output of the "traceroute or "tracert command), but the TTL parameter of most packets is equal to four or five, something is wrong. Many operating systems and applications assign an initial TTL value of 64 or 128 to IP packets they create. In this case, packets from hosts that are 15 hops away on the network should have TTL values equal to 115 (or perhaps 51) or so. The illogically low TTL values indicate that these packets have almost certainly been fabricated (spoofed).

Manager Deployment Considerations:

One of the most important deployment considerations for the manager component is ensuring that it runs on extremely high-end hardware (with a large amount of physical memory and a fast processor) and on a proven and reliable operating system platform (such as Solaris or Red Hat Linux). Continuous availability of the manager component is essential—any downtime generally renders an IDS or IPS totally worthless. Using RAID and deploying redundant servers in case one fails are additional measures that can be used to help assure continuous availability.

Decisions concerning where within a network to deploy the management console should (like the deployment of agents) be based on efficiency—the management component should be in a location within the network that minimizes the distance from agents with which it communicates—and on security, as discussed next.

Manager Security Considerations:

Of the three major components of a multi-tiered architecture, sensors are attacked most often, and compromised or disabled agents can cause considerable trouble, but a single successful attack on a management console is generally the worst imaginable outcome. Because of the centrality of the manager component to an entire IDS or IPS, such an attack can quickly result in all components in a multi-tiered architecture becoming compromised or unusable, and it can also result in destruction of all data and alerts that are centrally collected. It is thus advisable to devote considerable effort to hardening the host on which the management console runs.

Hardening includes implementing measures that prevent denial-of-service attacks, such as installing a properly configured firewall or TCP wrapper on the box that houses the manager component, ensuring that all vendor patches are installed, and not running unnecessary services. The server should also be protected by one or more external and possibly also internal firewalls and it should not be located in a portion of a network that has particularly high levels of traffic. The hardware platform on which the manager component runs must also be dedicated solely to this function, and it should have a special sensor—a watchdog function that is independent of the built-in logging capabilities—to provide specialized monitoring of all activity on that host.

Finally, providing suitable levels of encryption is critical. All communications between the manager component and any other component need to be encrypted using strong encryption, such as 192- or 256-bit Advanced Encryption Standard (AES) encryption. Sessions from remote workstations to the management console, in particular, need to be encrypted. If not, they could be hijacked, or their contents could be divulged to unauthorized persons. Any kind of data about individuals that is stored on the manager component should also be encrypted to avoid privacy breaches and to conform to state and local laws.