Graw Hill Osborne

Includes a Security Dictionary and contributions from topical experts!

The Complete Reference

Network Security

Examine and implement security strategies and discover proven techniques

Secure Windows®, Linux/UNIX, Novell and Wireless Security networks with logical, concise information

Understand legal issues and HIPPA legislation

Roberta Bragg
CISSP, MCSE: Security, Security+

Mark Rhodes-Ousley
CISSP

Keith Strassberg
CPA, CISSP

MORE THAN 20 CO-AUTHORS AND TECHNICAL REVIEWERS

CHAPTER

Intrusion-Detection Systems

by Roger A. Grimes

Intrusion-detection systems (IDSs) are yet another tool in a network administrator's computer security arsenal. Often thought of as a tertiary extra after antivirus software and firewalls, an IDS is often the best way to detect a security breach. As useful as they can be, however, IDSs remain largely immature and extremely resource intensive. Successfully deploying an IDS is one of the biggest challenges a security administrator can face.

This chapter is broken down into four sections. The first portion will introduce IDS concepts, and the second will discuss the IDS types available. Section three will discuss different IDS features in enough detail to help you evaluate different solutions. The fourth section will discuss real-life deployment considerations and will finish with a list of IDS vendors and products. By the end of this chapter, you should have a rich understanding of IDSs and be prepared to navigate the toughest operational issues.

IDS Concepts

The day after the widespread Bugbear worm broke out, a high-speed laser printer at one company began to spit out page after page of printer garbage characters. Was this a coincidence or the work of the worm? Because of a programming error, Bugbear does not correctly detect whether it is trying to infect a file or a printer share. When Bugbear tries to infect a printer share, the affected printer malfunctions and prints out lots of pages of control characters and machine language.

All the PCs on the network had up-to-date antivirus software, and no one had reported an infection. Still, there was the danger that Bugbear had slipped past the company's firewall and antivirus defenses and was now roaming their network. The administrator quickly plugged in a laptop, searched for and downloaded a Bugbear detection signature, fired up a copy of the open source Snort IDS, and waited. An hour passed by without any alert messages warning about Bugbear. The garbage printer pages were eventually tracked back to a newly installed, but buggy, Windows 2000 printer driver.

NOTE *Snort is a popular open source IDS for Unix and Windows and can be found at www.snort.org.*

It would have been easy to fire up Snort to double-check the other network defenses. An antivirus tool would not be alerted if Bugbear were simply trying to infect a share, and antivirus tools certainly wouldn't be running on the printer. Scanners would only be alerted if the worm was successful in writing a file to a vulnerable drive share, and then only if the antivirus signature database was up to date and configured for real-time monitoring. If Bugbear had slipped past perimeter security, the firewall would be unlikely to pick it up. Even if the firewall did pick up internal NetBIOS traffic originating on the network, it would not have been flagged as Bugbear-specific and would probably be lost among all the other NetBIOS false-positive messages that crop up on any firewall in a Windows environment. For this particular type of security event, an IDS is the best tool for the job.

Why Intrusion Detection

Intrusion detection (ID) is the process of monitoring for and identifying attempted unauthorized system access or manipulation. Most network administrators do ID all the time without realizing it. Security administrators are constantly checking system and security log files for something suspicious. An antivirus scanner is an ID system when it checks files and disks for known malware. Administrators use other security audit tools to look for inappropriate rights, elevated privileges, altered permissions, incorrect group memberships, unauthorized registry changes, malicious file manipulation, inactive user accounts, and unauthorized applications. An IDS is just another tool that can monitor host system changes or sniff network packets off the wire looking for signs of malicious intent.

Most IDSs are software programs installed on top of an operating system, but network-sniffing IDSs are increasingly being deployed as hardware appliances because of performance requirements. An IDS uses either a packet-level network interface driver to intercept packet traffic or it "hooks" the operating system to insert inspection subroutines. IDSs are a sort of virtual food-taster, deployed primarily for early detection, but increasingly used to prevent attacks.

When the IDS notices a possible malicious threat, called an *event*, it logs the transaction and takes appropriate action. The action may simply be to continue to log, send an alert, redirect the attack, or prevent the maliciousness. If the threat is high risk, the IDS will alert the appropriate people. Alerts can be sent by e-mail, Simple Network Management Protocol (SNMP), pager, or console broadcast. IDSs support the defense-in-depth security principle and can be used to detect a wide range of rogue events, including the following:

- Impersonation attempts
- Password cracking
- Protocol attacks
- Buffer overflows
- Installation of rootkits
- Rogue commands
- Software vulnerability exploits

- Malicious code, like viruses, worms, and Trojans
- Illegal data manipulation
- Unauthorized file access
- Denial of service (DoS) attacks

Threat Types

To really understand IDSs, you must understand the security threats and exploits they can detect and prevent. Threats can be classified as attacks or misuse, and they can exploit network protocols or work as malicious content at the application layer.

Attacks or Misuse

Attacks are unauthorized activity with malicious intent using specially crafted code or techniques. Attacks include denial of service attacks, virus or worm infections, buffer overflows, malcrafted requests, file corruption, malformed network packets, or unauthorized program execution. *Misuse* refers to unauthorized events without specially crafted code. In this case, the offending person used normally crafted traffic or requests and their implicit level of authorization to do something malicious. Misuse can also refer to unintended consequences, such as when a hapless new user overwrites a critical document with a blank page. Another misuse event could be a user mapping a drive to a file server share not intended by the network administrator.

NOTE *Some IDS experts define* misuse *as all internal security events, and attacks, as all external threats. I think that definition is inaccurate.*

Most IDSs are deployed to detect intentionally malicious attacks coming from external locations, but they are also proving of value within the corporate world for monitoring internal users. Security surveys often reveal internal misuse events as a leading cause of corporate data loss, and an IDS tool can track internal maliciousness almost as well as external attacks. In one case, a sharp security officer working in an IT department used an IDS to catch a fellow employee cracking passwords and reading confidential e-mail.

Network Protocol Attacks

Many of the security threats detected by IDSs exploit network protocols (layers three to six of the OSI model). Network protocols, such as TCP/IP, NetBIOS, AppleTalk, IPX, BGP, and hundreds of others, define standard ways of transmitting data to facilitate open communications. The data is sent in packets—packages of electronic bits (1's and 0's) framed in a particular format defined by a network protocol—but the protocols do not contemplate the consequences of malicious packet creation. TCP/IP (Transmission Control Protocol/Internet Protocol) running over the Ethernet is the most widely deployed network type, and most IDSs support it as their default protocol.

When information is sent between network hosts, commands and data sent by higher-layer application processes (such as FTP clients, web servers, and IM chat programs) are placed as payload content into discrete containers (called *datagrams* or *packets*), numbered, and sent from source to destination. When the packets arrive at the destination, they are reassembled (if needed), and the content is handed off to the destination application.

Network protocols define the packet's formatting and how the datagram is transmitte between source and destination. Malicious network protocol attacks interfere with the normal operation of this process. Understanding how this works requires some knowl of TCP/IP packet basics, so we'll quickly review that here.

TCP/IP Packet Analysis Basics This will be a quick overview. For more details, refer to authoritative TCP/IP book. Another good place to start is the TCP protocol specificati RFC 793 (www.rfc-editor.org/rfc/rfc793.txt), which explains basic TCP packet formati

The TCP and IP protocols work together, hence the name TCP/IP. IP handles routing t packets from source to destination, and TCP works to ensure reliable delivery. Among oth data, a layer three IP packet header contains the source and destination IP addresses, packe length, and protocol number. In TCP/IP, the most relevant protocol types are TCP (Transm Control Protocol), UDP (User Datagram Protocol), and ICMP (Internet Control Message Protocol). The *protocol number* for ICMP is 1, for TCP is 6, and for UDP is 17. You will see these numbers when doing packet analysis or rule-making. The IP packet header also ha a *fragmentation flag* that indicates whether the packet is a smaller part of a larger packet needing reassembly on the destination host.

TCP works at the transport layer of the OSI model and contains mechanisms to ma sure packets arrived at the destination successfully. A TCP header contains source and destination port numbers, sequencing and acknowledgement numbers, and six bit flag among the other data fields.

Each TCP session between two hosts is established ahead of time to make sure both parties are ready to communicate. The originating host sends a synchronization reques (SYN) to the intended destination. The receiving host acknowledges receipt of the requ and sends back its own synchronization response (ACK/SYN). The originating host th acknowledges (ACK) the destination host's own SYN request. This handshake makes up the infamous TCP three-way handshake: SYN, ACK/SYN, ACK; and makes TCP a *connection-oriented* protocol.

TCP is also *reliable* because it will direct a host to retransmit a packet if it is not acknowledged by the destination. Different protocol bits (called *flags*) are used to tell each communicating host whether a TCP packet is part of a starting, an established, or a disconnecting session (this is called *session state,* and because TCP keeps track of this information, it is called a *stateful* protocol). Each flag can be on (1) or off (0). The four most important state flags are:

- **SYN** Synchronization. This starts a TCP session.
- **ACK** Acknowledge. This acknowledges successful receipt of a prior packet.
- **FIN** Finish. This will gracefully end a TCP session.
- **RST** Reset. This will forcefully and immediately end a TCP session.

It's important that IDSs be stateful in order to detect abnormalities in packet flags in malicious network packets.

Other TCP/IP processes happen at the same time, such as agreeing on how much d can be sent at once and sending ARP (Address Resolution Protocol) packets on each sid local network to convert IP addresses to their Ethernet media addresses (MAC addresse you do network packet analysis, you will be accustomed to seeing TCP/IP traffic filled

dshakes and ARP packets. So much so that many IDSs and packet sniffers allow you to re (meaning *not capture*) handshake and ARP packets, while still capturing other types ata. Accurate IDSs need to be able to inspect all network traffic.

Port numbers identify the originating service or application the hosts are using to mmunicate. Port numbers below 1,024 are considered *well known* (www.iana.org/ ignments/port-numbers) and are reserved for particular applications. For example, t 25 is reserved for SMTP, 53 for DNS, and 21 for FTP. For the purposes of IDSs and ewalls, the destination port is usually of more interest. The source port number is often andom number above 1,023 and can be different each session. For example, a web client ntacting a web server will usually have a destination port number of 80, while its source rt number may be any number between 1,023 and 65,535. In practice, the source port is ually in a range just above 1,023, something like 1,060 or 3,728.

UDP is customarily used for small data transmissions where packet reassembly is t needed. UDP is *connectionless* and *stateless*, and as such, does not need flags. Examples common UDP protocols are DNS (Domain Name System), DHCP (Dynamic Host nfiguration Protocol), and SNMP (Simple Network Management Protocol). A UDP ader will contain source and destination port numbers, along with other information.

MP is used to troubleshoot and measure IP connections, and it also allows crackers to m information about host machines, such as the host operating system and version obtaining this information is called *fingerprinting*). ICMP is mostly known for its use in e ping troubleshooting utility.

NOTE Some services, like DNS, can use either TCP or UDP (or both at the same time) depending on the situation and vendor implementation.

g Exploits Abnormally crafted network packets are used for DoS attacks on host machines, to skirt past network perimeter defenses, to impersonate another user's session, r to crash a host's IP stack. Malicious network traffic works by playing tricks with the gitimate format settings of the IP protocol. For instance, using a specially crafted tool, a acker can set incompatible sequences of TCP flags, causing destination host machines to come confused and end up with a DoS condition. Other examples of maliciously formed CP traffic include a cracker setting an ACK flag in an originating session packet without ending an initial SYN packet to initiate traffic, or sending a SYN and FIN (start and stop) ombination at the same time. Port scanners often use the latter formation to make IP stacks nswer, even if a firewall-like blocking mechanisms are installed to stop normal port scanners.

Fragmentation and Reassembly Attacks Although not quite the security threat they once were, IP packets can be used in *fragmentation* attacks. TCP/IP fragmentation is allowed because all routers have a *maximum transmission unit* (MTU), which is the maximum number of bytes that they can send in a single packet. A large packet can be broken down into multiple smaller packets (known as *fragments*) and sent from source to destination. A *fragment offset* value located in each fragment tells the destination IP host how to reassemble the separate packets back into the larger packet.

Attacks can use fragment offset values to cause the packets to maliciously reassemble and intentionally cover up the header and payload of the first fragment. If an IDS or firewall allows fragmentation and does not reassemble the packets before inspection,

an exploit may slip by. For example, suppose a firewall does not allow FTP traffic, and an attacker sends fragmented packets posing as some other allowable traffic. If the packets act as SMTP e-mail packets headed to destination port 25, they could be passed through, but after they are past the firewall, they could reassemble to overwrite the original port number and become FTP packets to destination port 21.

Today, most IDSs, operating systems, and firewalls have antifragmentation defenses. By default, a Windows host will drop fragmented packets.

Application Attacks

While network protocol attacks abound, most security threats exploit the application layer of the host. In these cases, the TCP/IP packets are constructed legitimately, but their data payload contains malicious content. Application attacks can be text commands used to exploit operating system or application holes, or they can contain malicious content such as a buffer overflow exploit, a maliciously-crafted command, or a computer virus. Application attacks include misappropriated passwords, password-cracking attempts, rootkit software, illegal data manipulation, unauthorized file access, and every other attack that doesn't rely on malformed network packets to work.

Content Obfuscation Most IDSs look for known malicious commands or data in a network packet's data payload. A byte-by-byte comparison is done between the payload and each potential threat signature in the IDS's database. If something matches, it's flagged as an event.

Because byte-scanning is relatively easy to do, crackers use encoding schemes to hide their malicious commands and content. *Encoding* schemes are non-plaintext character representations that eventually get converted to plaintext for processing. The flexibility of the Internet and international languages allow ASCII characters to be represented by many different encoding schemes, including hexadecimal, decimal-dotted notation, double-word decimal notation, octal notation, Unicode, and any combination thereof. Web URLs and commands have particularly flexible syntax. Complicating the issue, most browsers encountering common syntax mistakes, like reversed slashes or incorrect case, convert them to their legitimate form. Here is an example of one URL presented in different forms with syntax mistakes and encoding. Type them into your browser and see for yourself.

- http://www.mcgraw-hill.com (normal representation)
- http:\\198.45.19.151 (IP address and wrong slashes)
- http://%77%77%77%2E%6D%63%67%72%61%77%2D%68%69%6C%6C%2E%63%6F%6D (hexadecimal encoded)

NOTE *It is not unusual to see a few characters of encoding in a legitimate URL. It's when you see mostly character encoding that you should get suspicious.*

Encoding can be used to obscure text and data used to make up malicious commands. Crackers use all sorts of tricks to fool IDSs, including using tabs instead of spaces, changing values from lowercase to uppercase, splitting data commands into several different packets

sent over a long period of time, hiding parameters, prematurely ending requests, using excessively long URLs, and using text delimiters. Read uber-hacker, Rain Forest Puppy's *A Look at Whisker's Anti-IDS Tactics* at http://jeff.wwti.com/papers/Rain_Forest_Puppy/whiskerids.html for more details. Encoding tricks that fool popular security systems, including IDSs, are announced frequently on security mail lists.

Data Normalization An IDS signature database has to consider all character encoding schemes and tricks that can end up creating the same malicious pattern. This is usually accomplished by normalizing the data before inspection. Normalization reassembles fragments into single whole packets, converts encoded characters into plain ASCII text, fixes syntax mistakes, removes extraneous characters, converts tabs to spaces, removes common hacker tricks, and does its best to convert the data into its final intended form.

TIP *Snort handles data normalization through a series of optional preprocessors that can be activated by the main executable. Preprocessors inspect and manipulate the data before it is passed to detection routines. Snort's open source documentation is an excellent source of normalization details.*

Different types of IDSs excel at detecting different types of attacks, and we'll look at a variety in this chapter. Some work well at detecting network protocol exploits and others are better for application layer maliciousness.

Threats an IDS Cannot Detect

IDSs excel at catching known, definitive malicious attacks. While some experts will say that a properly defined IDS can catch any security threat, events involving misuse prove the most difficult to detect and prevent. For example, if an outside hacker uses social engineering tricks to get the CEO's password, there aren't many IDSs that will notice. If the webmaster accidentally posts a confidential document to a public directory available to the world, the IDS won't notice. If a cracker uses the default password of an administrative account that should have been changed right after the system was installed, few IDSs will notice. If a hacker gets inside the network and copies confidential files, that would be tough to notice. There are ways that an IDS could be used to detect each of the preceding misuse events, but they are more difficult to detect than straight-out attacks.

First-Generation IDSs

IDS development as we know it today began in the early 1980s, but only started growing in the PC marketplace in the late 1990s. First-generation IDSs focused almost exclusively on the benefit of early warning resulting from accurate detection. This continues to be a base requirement of IDSs, and vendors frequently bragged about their product's accuracy. The practical reality is that while most IDSs are considered fairly accurate, no IDS has ever been close to being perfectly accurate. While a plethora of antivirus scanners enjoy year-after-year 95 to 99 percent accuracy rates, IDSs never get over 90 percent accuracy against a wide spectrum of real-world attack traffic. Most are in the 80 percent range. Some test results show 100 percent detection rates, but in every such instance, the IDS was tuned after several previous, less accurate, rounds of testing. When an IDS misses a legitimate threat, it is called a *false-negative*. Most IDSs are plagued with even higher false-positive rates.

IDSs Have High False-Positive Rates

A *false-positive* is when the IDS says there is a security threat, but the traffic is not malicious or was never intended to be malicious. A common example is when an IDS flags an e-mail as infected with a particular virus because it is looking for some key text known to be in the message body of the e-mail virus (for example, the phrase "see my wife's photos"). When an e-mail intended to warn readers about the virus includes the keywords that the reader should be on the lookout for, it can create a false positive. The IDSs should only be flagging the e-mail as infected if it actually contains a virus, not just if it has the same message text.

Simply searching for text within the message body to detect malware is an immature detection choice. Many security web services that send subscribers early warning e-mails complain that nearly 10 percent of their e-mails are kicked back by overly zealous IDSs. Many of those same services have taken to purposely misrepresenting the warning text (by slightly changing the text, such as "see_my_wife's_photos") in a desperate attempt to get past the subscribers' poorly configured defenses. If the measure of IDS accuracy is the number of logged security events against legitimate attacks, accuracy plummets on most IDS products. This is the biggest problem facing IDSs, and solving it is considered the holy grail for IDS vendors. If you plan to get involved with IDSs, proving out false-positives will be a big part of your life.

In an effort to decrease false-positives, some IDSs are tuned to be more sensitive. They will wait for a highly definitive attack within a narrow set of parameters before they alert the administrator. While they deliver fewer false-positives, they have a higher risk of missing a legitimate attack. Other IDSs go the other route and report on almost everything. While they catch more of the legitimate threats, those legitimate warnings are buried in the logs between tons of false-positives. If administrators are so overwhelmed with false-positives that they don't want to read the logs, this can result in a human denial of service attack. Some attacks attempt to do just this and generate massive numbers of false-positives, hoping their one legitimate attack goes unnoticed.

So, which is a better practice? Higher false-positives or higher false-negatives? Most IDS products err on the side of reporting more events and requiring the user to fine-tune the IDS to ignore frequent false-positives. Fine-tuning an IDS means configuring sensitivity up or down to where you, the administrator, are comfortable with the number of false-negatives and false-positives. When you are talking with vendors or reviewing IDS products, inquire about which detection philosophy the IDS follows. If you don't know ahead of time, it will become apparent after you turn it on.

Second-Generation IDSs

The net effect of most IDSs being fairly accurate and none being highly accurate has resulted in vendors and administrators using other IDS features for differentiation. Here are some of those other features that may be more or less useful in different circumstances:

- Return on investment
- IDS type and detection model
- End-user interface
- IDS management
- Prevention mechanisms
- Performance

- Logging and alerting
- Reporting and analysis

All of these will be discussed in this chapter.

First-generation IDSs focused on accurate attack detection. *Second-generation* IDSs do that and work to simplify the administrator's life by offering a bountiful array of back-end options. They offer intuitive end-user interfaces, intrusion prevention, centralized device management, event correlation, and data analysis. Second-generation IDSs do more than just detect attacks—they sort them, prevent them, and attempt to add as much value as they can beyond mere detection.

Experienced IDS administrators know that half of the success or failure of an IDS is determined by all the back-end, non-sexy stuff. It's always exciting to catch a cracker hacking in real-time, and it's fun snooping on the snooper, so first-time implementers always spend most of their time learning about and implementing detection patterns. In doing so, though, they often breeze through or skip the reading on setting up the management features, configuring the database, and printing reports. They turn on their IDSs and are quickly overwhelmed because they didn't plan ahead. To increase your odds of a successful IDS deployment, remember this: For every hour you spend looking at cool detection signatures, spend an hour planning and configuring your logging, reporting, and analysis tools.

But before we even get to configuring the IDS, we have to justify its cost.

Return on IDS Investment

As fun as it can be to experiment and learn with IDSs, CEOs don't pay $2 to protect $1. In order to justify the expense of an IDS and all the time you'll be spending installing and maintaining it, you'll need to create a *return on investment* (ROI) analysis. The return on investment is the initial and ongoing cost of an asset, offset by future increases in revenue or decreases in expenses. A successful ROI takes everything, even speculative risks and emotional gut feelings, and quantifies them. Unless you are selling IDSs or managed services for IDSs, they probably won't generate revenue for your company. Accordingly, an IDS ROI analysis should show that the cost of an IDS will be offset by decreased future expenses relating to the assets it is protecting.

IDS Costs

As with most ROI calculations, the total cost of ownership (TCO) is easier to define in dollars than are the future savings. An IDS TCO should include costs for the following:

- IDS purchase (including install, updates, and support)
- Operating system (if separate)
- Hardware (if separate)
- Ongoing labor hours or managed services
- Training

IDSs can range in purchase cost from free, like Snort, to hundreds of thousands of dollars. It is common for an IDS to cost from $5,000 to $30,000 for a midsize business with a few hundred assets to protect. Larger organizations with thousands of computers, or industries with very valuable assets (such as in the banking industry), can expect prices to start in the hundreds of thousands of dollars.

If your IDS runs on a regular computer, it should have a high-end CPU (or several) and lots of memory and hard drive space. A typical IDS computer with operating system will cost somewhere from $3,000 to $15,000. You may require a separate computer for each network segment you want to monitor, another for a central management console, and one for the database engine.

It can take hundreds of hours to test, install, and fine-tune an IDS. If you use an IDS the way it's meant to be deployed, ongoing labor can run from a few hours per week to having a full-time person for a midsize organization. IDS administrators must have a firm understanding of network basics, medium to advanced knowledge of the operating system platforms they are protecting, and a moderate amount of computer security training. Because of the high level of training required across both hardware and software, an IDS administrator is usually a top-paid IT employee. A properly maintained IDS is not cheap, no matter what its initial cost.

IDS Benefits

Security system benefits are historically difficult to quantify because the benefits are estimations of how much won't have to be spent if a particular risk situation happens and the danger is averted. In the case of an IDS, the purchase and maintenance costs must be less than the cost of intrusion without early detection or prevention. In order to have a successful ROI, you must quantify the costs of downtime, data corruption, leaked information, recovery, customer reaction, media embarrassment, and so on, that could result from an intrusion. You have to quantify the value of the data stored on your company's network and servers. What would happen to the company's value if private information was taken or published?

You need to consider all the possible security threat scenarios, guess their chance of happening in the future if the current protection level were maintained, and estimate their potential cost to the company. A virus attack might only require two days of IT clean-up, valued at a few thousand dollars, but a strategic document being stolen from the CEO's home directory may prove priceless. Take each risk scenario, you'll need to assign a loss dollar value and assign it a percentage chance of occurring. The following table gives a few examples.

Threat Description	Loss Value	Chance of Occurring per Year	Exposure Risk
Virus attack	$4,000	200%	$8,000
File server compromise	$10,000	50%	$5,000
Web server compromise	$30,000	75%	$22,500
Strategic document taken	$1,000,000	5%	$50,000
Total exposure risk			**$85,500**

You will need to calculate the total exposure risk cost with and without an IDS. If your savings resulting from the IDS exceed the cost of running it, you've got a pretty good argument to take to management.

NOTE *Although it differs from this simple explanation, SecurityFocus has an excellent IDS ROI discussion document at www.securityfocus.com/printable/infocus/1608.*

Although the CEO usually makes a decision based on dollars and cents, any good ROI analysis will explain basic IDS concepts and explain why you picked a particular solution. The next section will cover basic IDS types and models.

IDS Types and Detection Models

Depending on what assets you want to protect, IDSs can protect a host or a network. All IDSs follow one of two intrusion-detection models—anomaly detection or signature detection—although some systems use parts of both where it's advantageous. Both types work by monitoring a wide population of events and triggering off predefined behaviors.

Host-Based IDS

Host-based IDSs (HIDSs) are installed on the host they are intended to monitor. The host can be a server, workstation, or any networked device (such as a printer, router, gateway). HIDSs install as a service or daemon, or they modify the underlying operating system's kernel or application to gain first inspection authority. While a HIDS may include the ability to sniff network traffic intended for the monitored host, they excel at monitoring and reporting direct interactions at the application layer. Application attacks can include memory modifications, maliciously crafted application requests, buffer overflows, or file-modification attempts. A HIDS can inspect each incoming command, looking for signs of maliciousness, or simply track unauthorized file changes.

File-integrity HIDSs (sometimes called *snapshot* or *checksum* HIDSs) take a cryptographic hash of important files in a known clean state, and then check them again later for comparison. If any changes are noted, the administrator is alerted. The following are two examples of file-integrity HIDSs:

- Tripwire (www.tripwire.com)
- Pedestal Software's INTACT (www.pedestalsoftware.com)

Behavior-monitoring HIDSs do real-time monitoring and will intercept potentially malicious behavior. For instance, a Windows HIDS will report on attempts to modify the registry, file manipulations, system access, password changes, privilege escalations, and other direct modifications to the host. On a Unix host, a behavior-monitoring HIDS may monitor attempts to access system binaries, attempts to download the /etc/passwd file, use of setuid and setgid, and additions to cron. A behavior-monitoring HIDS on a web server may monitor incoming requests and report maliciously crafted HTML responses, cross-site scripting attacks, or SQL injection code. Examples of behavior-monitoring HIDS include the following:

- Cisco's IDS Host Sensor (www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/prodlit/hid25_ds.htm)

- Okena's StormWatch (www.okena.com)
- Entercept Security Technologies' IDS solutions (www.entercept.com)

NOTE *At press time, StormWatch had been recently purchased by Cisco and was being renamed Cisco Security Agent.*

Real-Time or Snapshot?

Early warning and prevention are the greatest advantages of a *real-time* HIDS. Because a real-time HIDS is always monitoring system and application calls, it can stop potentially malicious events from happening in the first place. On the downside, real-time monitoring takes up significant CPU cycles, which may not be acceptable on a high-performance asset, like a popular web server or a large database server. Real-time behavior-monitoring only screens previously defined threats, and new attack vectors are devised several times a year, meaning that real-time monitors must be updated, much like databases for an antivirus scanner. Also, if an intrusion successfully gets by the real-time behavior blocker, the HIDS won't be able to provide as much detailed information about what happened thereafter as a snapshot HIDS would.

Snapshot HIDSs are reactive by nature. They can only report on maliciousness, not stop it. Snapshot HIDSs excel at forensic analysis. With one report you can capture all the changes between a known good state and the corrupted state. You will not have to piece together several different progressing states to see all the changes made since the baseline. Damage assessment is significantly easier than with a real-time HIDS because a snapshot HIDS can tell you exactly what has changed. You can use comparative reports to decide whether or not you have to completely rebuild the host or whether a piecemeal restoration can be done safely. You can also use the before and after snapshots as forensic evidence in an investigation.

Snapshot systems are useful outside the realm of computer security, too. You can use a snapshot system for configuration and change management. A snapshot can be valuable when you have to build many different systems with the same configuration settings as a master copy. You can configure the additional systems and use snapshot comparison to see if all configurations are identical. You can also run snapshot reports at a later date to see if anyone has made unauthorized changes to a host. The obvious disadvantage of a snapshot HIDS is that alerting and reporting is done after the fact. By then the changes have already occurred, and the damage is done.

RealSecure Desktop Protector—A Real-Time HIDS

Internet Security Systems (ISS, www.iss.net) offers a combination personal firewall and intrusion-detection system for workstations and servers called RealSecure Desktop Protector. Incorporating technology gained when ISS purchased BlackICE Defender, Desktop Protector is an inbound and outbound IDS. It contains hundreds of threat signatures and is smart enough to realize when a Trojan or virus is trying to piggyback on the operations of trusted applications. In Figure 14-1, Desktop Protector is shown logging BackOrifice and port scan attempts. Desktop Protector can report to and be managed by a central console and will make attempts to trace intruders back to their source.

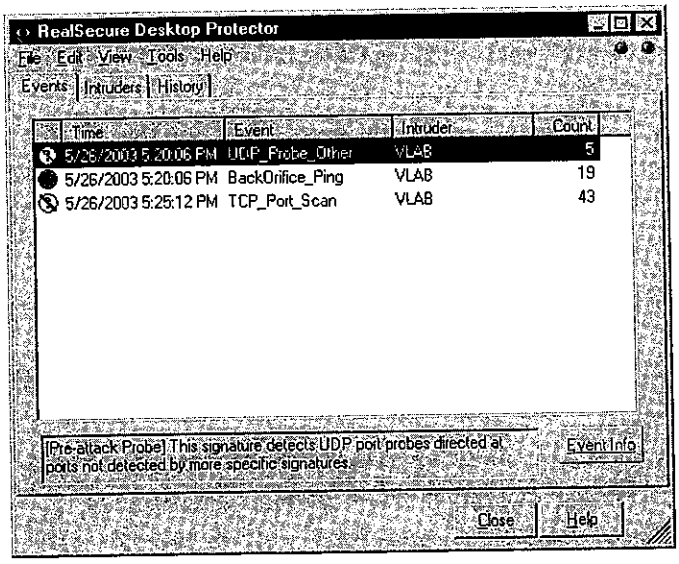

FIGURE 14-1 ISS's RealSecure Desktop Protector alerting about BackOrifice and port scan attempts

Tripwire: A Snapshot HIDS

Tripwire is a snapshot HIDS originally created as an open source project in 1992 by Gene Kim and Dr. Eugene Spafford. Tripwire Inc. (www.tripwire.com) was formed in 1997 by Kim to support a commercial version of Tripwire, although the open source version still coexists (http://sourceforge.net/projects/tripwire). Tripwire installs on Windows and Unix platforms and works by creating a baseline database of files and their digital fingerprints. It logs file size, creation date, security access controls, and alternate streams, and it documents 24 critical registry areas. It can track modifications, deletions, and last access dates. At any future date, a user can run another snapshot and get a report detailing any changes between the baseline and the new snapshot. Multiple instances can be managed and each can report to a central console.

Honeypots

So far, the HIDS we have talked about are for protection of production hosts. Some HIDSs, however, are created as sacrificial hosts to be exploited and compromised. A *honeypot* is a special type of HIDS where an entire expendable system is created solely to monitor, detect, and capture security threats against it. The honeypot can be a normal system mimicking a production system without all the patches applied, or it can be hosted using specialized *virtual honeypot* software. Virtual honeypots are created by dedicated emulation software that mimics a particular platform's IP stack, operating system responses, and sometimes even services, applications, and content.

For example, Honeyd, an open source virtual honeypot system can mimic over a hundred different systems all running at the same time, including most versions of Windows, Linux, and even a Cisco router. One single instance of Honeyd can behave like a Windows NT box running IIS 5.0, like a Windows 98 workstation, and like a Unix server with FTP, Telnet, and SSL services running. Honeypots often contain snapshot functionality and packet-capturing software so the security administrator can document all the crackers' malicious activities.

Honeypots can also accept malicious traffic that is *deflected* by a network perimeter device in order to slow down crackers or automated worms. The LaBrea program (http://labrea.sourceforge.net) is a *sticky* honeypot that answers malicious requests made to unallocated IP addresses and ports. By doing so, it can keep malicious hackers and worms busy for hours in a virtual environment where they can do no damage. There are also SMTP honeypots meant to track hiding spammers. See www.honeypots.net or www.honeynet.org for more details on honeypots.

Network-Based IDS (NIDS)

Network-based IDSs (NIDSs) are the more popularly talked about IDSs, and they work by capturing and analyzing network packets speeding by on the wire. Unlike HIDSs, NIDSs are designed to protect more than one host. They can protect a group of computer hosts, like a server farm, or monitor an entire network. Captured traffic is compared against protocol specifications and normal traffic trends or the packet's payload data is examined for malicious content. If a security threat is noted, the event is logged and an alert is generated.

Most IDSs are NIDSs, including these:

- Snort (www.snort.org)
- Symantec's Manhunt (www.symantec.com)
- Network Flight Recorder's NFR NID (www.nfr.com)
- Internet Security Systems' RealSecure Sensor (www.iss.net)

NIDS Physical Layer Considerations

With a HIDS, you install the software on the host that is to be monitored, and the software does all the work. Because NIDSs work by examining network packet traffic, including traffic not intended for the NIDS host on the network, they have a few extra deployment considerations. It is common for brand-new NIDS users to spend hours wondering why their IDS isn't generating any alerts. Sometimes it's because there is no threat traffic to alert on, and other times it's because the NIDS isn't set up to capture packets headed to other hosts. A sure sign that the network layer of your NIDS is misconfigured is that it only picks up broadcast traffic and traffic specifically headed for it. Traffic doesn't start showing up at the NIDS simply because it was turned on. You must configure your NIDS and the network so that the traffic you want to examine is physically passed to the NIDS. NIDSs must have promiscuous network cards with packet-level drivers, and they must be installed on each monitored network segment.

Packet-Level Drivers Network packets are captured using a packet-level software driver bound to a network interface card. Many Unix systems and Windows do not have native packet-level drivers built in, so it's common for IDS implementations to rely upon open source packet-level drivers like libpcap (http://sourceforge.net/projects/libpcap) or WinPcap 3.0 (http://winpcap.polito.it). An open source packet capturing and analyzing tool, such as tcpdump (www.tcpdump.org) or WinDump (http://windump.polito.it), is often used with open source IDSs. Most commercial IDSs have their own packet-level drivers and packet-sniffing software.

Promiscuous Mode In order for a NIDS to sniff packets, the packets have to be given to the packet-level driver by the network interface card. By default, most network cards are not *promiscuous*, meaning that they only read packets off the wire that are intended for them. This typically includes *unicast* packets, meant solely for one particular workstation, *broadcast* packets, meant for every computer that can listen to them, and *multicast* traffic, meant for two or more previously defined hosts. Most networks contain unicast and broadcast traffic. Multicast traffic isn't as common, but it is gaining popularity for web-streaming applications. By default, a network card in normal mode drops traffic destined for other computers and packets with transmission anomalies (resulting from collisions, bad cabling, and so on). If you are going to set up an IDS, make sure its network interface card has a *promiscuous mode* and is able to inspect all traffic passing by on the wire.

Sensors for Network Segments For the purposes of this chapter, a *network segment* can be defined as a single logical packet domain. For a NIDS, this definition means that all network traffic heading to and from all computers on the same network segment can be physically monitored.

In order for an IDS to effectively monitor a network, there should be at least one NIDS inspection device per network segment. This device can be a fully operational IDS computer or just a traffic repeater device known as a *sensor* or *tap*. Sensors and taps are small network devices specifically designed for NIDS with two or more network ports. They usually don't have keyboards or mice, and they must be configured by proprietary administrative software. One port plugs into the middle of a connection on the network segment to be monitored, and the other plugs into a cable leading to the central IDS console. They grab network traffic off the wire and pass it to the IDS console. Taps and sensors typically don't have an IP or MAC address, so they are more likely to remain invisible to intruders. Intrusion Inc. (www.intrusion.com) has a popular line of sensors and taps.

Any device on a shared hub shares the network segment because each port sees all network information headed to any port. You can place a sensor on any hub or bridge port and capture all traffic on the local segment. Routers are the edge points of segments, and you must place at least one sensor on each segment you wish to monitor. You may want to use more sensors than are physically required by the network topology for performance reasons, or to marry sensors to specific groups of computers within the same segment.

Most of today's Ethernet networks contain switch devices. With the notable exception of broadcast packets, switches only send packets to a single destination port. On a switched network, an IDS will not see its neighbor's non-broadcast traffic, and this presents a problem when you are trying to monitor an entire network segment.

Several solutions exist for this problem, depending on your hardware. First, many IDS administrators use taps or sensors to get around switch segmentation. However, that is expensive and impractical on a switched network if you want to monitor every port. Second, many switches support *port mirroring*, also called *port spanning* or *traffic redirection*. Port mirroring is accomplished by instructing the switch to copy all traffic to and from a specific port to another port where the IDS sits. This can be accomplished using remote monitoring (RMON), SNMP, or the vendor's proprietary method. If you are trying to monitor as much traffic as possible, you want the monitored port to be the highest utilized port on the switch

with the most widespread spectrum of traffic. This is usually a gateway port to another network or server farm, or a switch uplink to another segment.

Second, some switches will allow all ports on the switch to be mirrored to a specific management port, and others will only allow one port to be monitored at a time. The mirrored port may be any other normal switched port, or it may be a special management port with a serial interface. Be aware that some port mirroring implementations only copy traffic sent to the original port, but not sent from it. Check your switch's documentation for details. On the downside, port mirroring increases switch utilization as it copies traffic. Mirroring all ports can work to the detriment of overall network performance.

NOTE *Like a tap, port mirroring does not readily reveal itself to crackers who might otherwise note the IDS's presence.*

Third, if your switch does not support port mirroring, you might see if you can change your switch into bridge mode, in which all traffic is replicated among all ports. This effectively nullifies the benefits of having a switch, so the cost/benefit justification should be analyzed. If your switch doesn't have a bridge mode, some IDS administrators have resorted to intentional ARP or MAC address poisoning to fake their switch or to force it to bridge mode. This method should be implemented only in an emergency monitoring situation.

Address Resolution Protocol (ARP) is a protocol used to translate IP addresses to MAC addresses, which is the address every Ethernet packet eventually uses to communicate. There are several tools that will falsely answer to ARP queries as if they were the intended host, so that it will become a man-in-the-middle passer. Others will simply try to overwhelm the switch's routing matrix table with false ARP and MAC address floods. Some switches, when their address matrix table is overwhelmed, will automatically fail over to bridge mode.

Another common solution is to place your IDS and all the computers you want to protect on an Ethernet repeater or hub so that all traffic is shared among all ports. This solution could be used when searching for particular exploit traffic targeting specific servers, or to insert a portable IDS laptop into a known exploited segment. Be aware that if you are doing this to get around a switch, you are effectively nullifying the switch's faster performance by making all the hosts on the segment into a single packet domain again.

Snort: An Example NIDS

Snort (www.snort.org) is an open source NIDS written by Marty Roesch in 1998, and it is perhaps the most popular NIDS in use today. Snort was originally a Unix-only program, but it has been successfully ported to Windows and is used in a variety of commercial offerings. It has three modes: network sniffer, network packet logger, and NIDS. Snort enjoys large open source community support with dozens of add-ons. You can load nearly 2,000 signature-detection rules covering exploits for Windows, Unix, IIS, Apache, IM, Oracle, SQL, viruses, back doors, pornography, DoS, RPC, ICMP, POP, SMTP, port scans, and more. Figure 14-2 shows Snort console messages resulting from a rapid injection of exploit attempts on a test network. Snort has multiple tools to ease data analysis, and several front ends. It can be joined to external databases for packet and event logging, linked to reporting tools, managed by central consoles, and it can participate in a variety of alert systems.

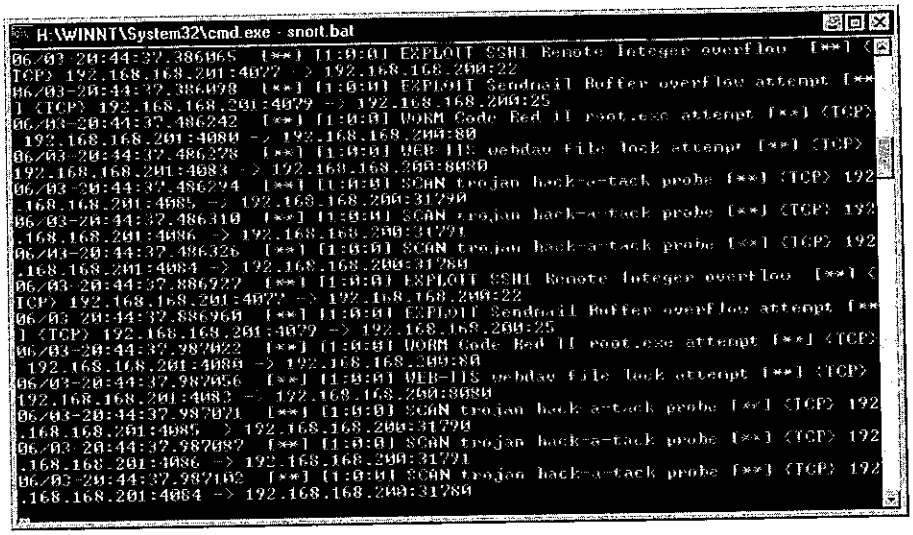

FIGURE 14-2 Snort in console mode logging exploit attempts

On the downside, Snort is not the easiest IDS to learn, configure, or deploy. Even with GUI front ends, learning to use Snort means editing text-based configuration files, learning case-sensitive command-line syntax, and deciphering cryptic error messages. Also, its widespread use also makes it a popular target for attack. Several hacking tools have been explicitly developed to exploit weaknesses in Snort. Despite its shortcomings, using Snort and reading its accompanying documentation is an excellent way to learn the basics of NIDSs.

Anomaly-Detection (AD) Model

Anomaly detection (AD) was proposed in 1985 by noted security laureate Dr. Dorothy E. Denning, and it works by establishing accepted baselines and noting exceptional differences. Baselines can be established for a particular computer host or for a particular network segment. Some IDS vendors refer to AD systems as *behavior-based* since they look for deviating behaviors. If an IDS looks only at network packet headers for differences, it is called *protocol anomaly detection*.

Several IDSs have anomaly-based detection engines, including Okena's StormWatch (www.okena.com) and Symantec's DeepSight Threat Management System (http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=158). Snort has a basic protocol-anomaly-detection component to quickly identify packets that don't follow normal network rules. Several massively distributed AD systems monitor the overall health of the Internet, and a handful of high-risk Internet threats have been minimized over the last few years because unusual activity was noticed by a large number of correlated AD systems.

The goal of AD is to be able to detect a wide range of malicious intrusions, including those for which no previous detection signature exists. By defining known good behaviors, an AD system can alert to everything else. Anomaly detection is statistical in nature and works on the concept of measuring the number of events happening in a given time interval for

a monitored metric. A simple example is someone logging in with the incorrect password too many times, causing an account to be locked out and generating a message to the security log. Anomaly-detection IDSs expand the same concept to cover network traffic patterns, application events, and system utilization. Here are some other events AD systems can monitor and trigger off:

- Unusual user account activity
- Excessive file and object accesses
- High CPU utilization
- Inappropriate protocol use
- Unusual workstation login location
- Unusual login frequency
- High number of concurrent logins
- High number of sessions
- Any code manipulation
- Unexpected privileged use or escalation attempts
- Unusual content

An accepted baseline may be that network utilization on a particular segment never rises above 20 percent and routinely only includes HTTP, FTP, and SMTP traffic. An AD baseline might be that there are no unicast packets between workstations and only unicasts between servers and workstations. If a DoS attack pegs the network utilization above 20 percent for an extended period of time, or someone tries to telnet to a server on a monitored segment, the IDS would create a security event. Excessive repetition of identical characters in an HTTP response might be indicative of a buffer overflow attempt.

When an AD system is installed, it monitors the host or network and creates a monitoring policy based upon the learned baseline. The IDS or installer chooses which events to measure and how long the AD system should measure to determine a baseline. The installer must make sure that nothing unusual is happening during the sampling period that might skew the baseline.

Anomalies are empirically measured as a statistically significant change from the baseline norm. The difference can be measured as a number, a percentage, or as a number of standard deviations. In some cases, like the access of an unused system file or the use of an inactive account, one instance is enough to trigger the AD system. For normal events with ongoing activity, two or more statistical deviations from the baseline measurement creates an alert.

AD Advantages

AD systems are great at detecting a sudden high value for some metric. For example, when the Slammer SQL worm ate up all available CPU cycles and bandwidth on affected servers and networks within seconds of infection, you can bet AD systems went off. They did not need to wait until an antivirus vendor released an updated signature. As another example, if your AD system defines a buffer overflow as any traffic with over a thousand repeating characters, it will catch any buffer overflow, known or unknown, that exceeds that definition.

It doesn't need to know the character used or how the buffer overflow works. If your network usually experiences 10 FTP sessions in a day, and all of a sudden it experiences 1,000, an AD system is a likely candidate to catch the suspicious activity.

Unfortunately, in the same vein, AD systems aren't great at telling how something was compromised, only that it was compromised.

AD Disadvantages

Where AD IDSs fail horribly is in establishing an initial baseline and in detecting malicious activity that does not violate an accepted behavioral norm, especially in the realm of malicious content. For instance, an SMTP e-mail may contain a link to a malicious web site or contain an infected file attachment. An AD system doesn't have the expert knowledge to determine whether the content is or isn't malicious.

Because AD systems are subject to time-event relationships, it is possible for a hacker to use the existence of an AD system to their advantage. For example, the hacker can port-scan a network with a long enough time interval between each port scanned so that the AD IDS doesn't flag the event as statistically significant. It is also possible for malicious hackers to fool the AD system by gradually training the tool to learn a new baseline.

Another major disadvantage is that defining the baseline norm in a chaotic changing world can be difficult. An AD system's baseline may be smart enough to know that network utilization slows from 11 A.M. to 1 P.M. every weekday because of lunch schedules, but it will probably mark as anomalous the "normal" increase in web and e-mail traffic after a national news story breaks. There is no easy way to tie complex, seemingly chance human reactions to a statistically based system.

It's interesting to note that although Dr. Denning is considered the mother of AD, in her landmark 1985 AD paper entitled *An Intrusion Detection Model* (www.cs.georgetown.edu/~denning/infosec/ids-model.rtf), she felt that using AD as the only model for an IDS system was a flawed idea. She understood that if a cracker knew that an IDS was only using AD, they could easily exploit a known system hole with a strong chance of quickly succeeding the first time. Dr. Denning felt that AD should only be used as an adjunct technology to attempt to catch vulnerabilities without a known signature.

Signature-Detection Model

Signature detection or *misuse* IDSs are the most popular type of IDS, and they work by using databases of known bad behaviors and patterns. This is nearly the exact opposite of AD systems. When you think of a signature-detection IDS, think of it as an antivirus scanner for network traffic. Signature-inspection engines can query any portion of a network packet or look for a specific series of data bytes. The defined patterns of code are called *signatures*, and often they are included as part of a governing *rule* when used within an IDS.

Signatures are byte sequences that are unique to a particular malady. A byte signature may contain a sample of virus code, a malicious combination of keystrokes used in a buffer overflow, or text that indicates the cracker is looking for the presence of a particular file in a particular directory. For performance reasons, the signature must be crafted so that it is the shortest possible sequence of bytes needed to reliably detect its related threat. It must be highly accurate in detecting the threat and not cause false-positives. Signatures and rules can be collected together into larger sets called *signature databases* or *rule sets*.

Signature-Detection Rules

Rules are the heart of any signature-detection engine. A rule usually contains the following information as a bare minimum:

- Unique signature byte sequence
- Protocol to examine (such as TCP, UDP, ICMP)
- IP port requested
- IP addresses to inspect (destination and source)
- Action to take if a threat is detected (such as allow, deny, alert, log, disconnect)

Most IDSs come with hundreds of predefined signatures and rules. They are either all turned on automatically, or you can pick and choose. Each activated rule or signature adds processing time for analyzing each event. If you were to turn on every rule and inspection option of a signature-detection IDS, there is a good chance it would quickly be unable to keep up with traffic inspection. Administrators should activate the rules and options with an acceptable cost/benefit tradeoff.

Most IDSs also allow you to make custom rules and signatures, which is essential for responding immediately to new threats or for fine-tuning an IDS. Here are some hints when creating rules and signatures:

- Byte signatures should be as short as possible, but reliable, and they should not cause false-positives.
- Similar rules should be near each other. Organizing your rules speeds up future maintenance tasks.
- Some IDSs and firewalls require rules that block traffic to appear before rules that allow traffic. Check with your vendor to see if rule placement matters.
- Create wide-sweeping rules that do the quickest filtering first. For example, if a network packet has a protocol anomaly, it should cause an alert event without the packet ever getting to the more processor-intensive content scanning.
- To minimize false-positives, rules should be as specific as possible, including information that specifically narrows down the population of acceptable packets to be inspected.

A Rule Example Here's an example rule from Snort:

```
alert tcp $XNET any -> any 80 (msg:"WEB-IIS CodeRed "; flags:A+;
uricontent:"scripts/root.exe?";)
```

With this Snort rule, an alert event is created if Snort notices a TCP packet destined for port 80 with only the ACK flag set (meaning that a session has been previously negotiated) and having payload text containing "scripts/root.exe?". The "scripts/root.exe?" query is used by crackers to take control of IIS servers infected with the Code Red worm.

Some threats, like polymorphic viruses or multiple-vector worms, require multiple signatures to identify the same threat. For instance, many computer worms arrive as infected executables, spread over internal drive shares, send themselves out with their

own SMTP engines, drop other Trojans and viruses, and use Internet chat channels to spread. Each vector of attack would require a different signature.

Advantages of Signature Detection

Signature-detection IDSs are proficient at recognizing known threats. Once a good signature is created, signature detectors are great at finding patterns, and because signature-detection IDSs are popular, a signature to catch a new popular attack usually exists within hours of it first being reported. This applies to most open source and commercial vendors.

Another advantage of a signature-detection IDS is that it will specifically identify the threat, whereas an AD engine can only point out a generality. An AD IDS might alert you that a new TCP port opened on your file server, but a signature-detection IDS will tell you what exploit was used. Because a signature-detection engine can better identify specific threats, it has a better chance at providing the correct countermeasure for intrusion prevention.

Disadvantages of Signature Detection

Although signature-detection IDSs are the most popular type of IDS, there are several disadvantages as compared to AD IDSs.

Cannot Recognize Unknown Attacks Just like antivirus scanners, signature-detection IDSs are not able to recognize previously unknown attacks. Crackers can change one byte in the malware program (creating a variant) to invalidate an entire signature. Hundreds of new malware threats are created every year, and signature-based IDSs are always playing catch-up. To be fair, there hasn't been a significant threat in the last few years that didn't have a signature identified by the next day, but there is increased exposure in the so-called "zero-hour."

Performance Suffers as Signatures or Rules Grow Because each network packet or event is compared against the signature database, or at least a subset of the signature database, performance suffers as rules increase. Most IDS administrators using signature detection usually end up only using the most common signatures, and not the less common rules. The more helpful vendors will rank the different rules with threat risks so the administrator can make an informed risk tradeoff decision. While this is an efficient use of processing cycles, it does decrease detection reliability.

Some vendors are responding by including *generic* signatures that detect more than one event. To do so, their detection engines support wildcards to represent a series of bytes, like this:

Virus A has a signature of	`14 90 90 90 56 76 56 64 64`
Virus B has a signature of	`14 80 90 90 56 76 56 13 10`
A wildcard signature for viruses A and B is	`14 ? 90 90 56 76 56 *`

Of course, the use of wildcard signatures increases the chance of false-positives. Antivirus vendors faced a similar dilemma last decade, and viruses were called generic boot sector or generic file infectors. Some vendors went so far that they rarely identified any threat by its specific name. Security administrators were not happy with the results, and vendors had to return to using more specific signatures.

Because signatures are small, unique series of bytes, all a threat coder has to do is change one byte that is identified in the signature to make the threat undetectable. Threats with small changes like these are called *variants*. Luckily, most variants share some common portion of code that is still unique to the whole class of threats, so that one appropriate signature, or the use of wildcards, can identify the whole family.

There are a few other disadvantages of signature-detection systems, but because they are also disadvantages for AD-based systems, they will be discussed in the "IDS Weaknesses" section later in the chapter.

Wireless IDSs

Wireless networks, particularly those using the 802.11 standards, are becoming very popular. The press is full of stories of *war-driving*, where a laptop user with a wireless card intercepts unauthorized wireless transmissions and captures data. If the wireless traffic is not encrypted, whether by using Wired Equivalent Privacy (WEP) or some other encryption method, gaining unauthorized access and reading content can be trivial.

There are many threats that attack wireless protocols and cause problems, even for encrypted links. In response, several IDS vendors have created *wireless IDSs* (WIDSs). A WIDS can, of course, detect the regular assortment of IDS threats, but if you want to do that, you're better off inserting a regular IDS on your wired network intercepting traffic coming off the wireless portion. WIDSs are specifically designed to look for attacks against your wireless infrastructure, including MAC spoofing, man-in-the-middle attacks, unauthorized wireless access points, DoS attempts, and other specialized wireless attacks. AirDefense (www.airdefense.net) and Network Chemistry (www.networkchemistry.com) are two vendors concentrating on 802.11 IDSs. Expect to see the wireless IDS market grow significantly, as portable computing devices (such as PDAs, cell phones, and others) exceed desktop devices over the next decade.

CAUTION *WEP is weak encryption, and cracking it is considered trivial by encryption specialists.*

What Type of IDS Should You Use?

There are dozens of IDSs to choose from. The first thing you need to do is survey the computer assets you want to protect, and identify the most valuable computer assets that should get a higher level of security assurance. These devices are usually the easiest ones to use when making an ROI case to management. New IDS administrators should start small, learn, fine-tune, and then grow. A HIDS should be used when you want to protect a specific valuable host asset. A NIDS should be used for general network awareness and as an early warning detector across multiple hosts.

You need to pick an IDS that supports your network topology, operating system platforms, budget, and experience. If you work in a pure Windows shop, you will want to avoid all the IDSs that work with and protect Unix environments. There are numerous Unix-only choices from vendors that are usually known for their Windows applications, and their IDSs are Unix-based because they bought another company's product and are working to port the technology to Windows. If you're not a Unix person, don't let a vendor talk you into a Unix solution, unless they are the ones installing, configuring, and managing it. Even then, be cautious, because you will be responsible for supporting the product if the vendor

relationship does not work out. If you have a significant amount of wireless traffic exposed in public areas, consider investing in a WIDS. If you have high-speed links that you need to monitor, make sure your IDS has been rated and tested at the same traffic levels.

Should your IDS be based on anomaly or signature detection? When possible, use a product that does both. The best IDSs utilize all techniques, combining the strengths of each type to provide a greater defense strategy. IDS vendors, including Internet Security Systems and Cisco, have developed top products by using both technologies. They also have both NIDS and HIDS products and use central consoles to manage all the devices and coordinate attack activity monitoring. The reviews of these hybrid devices have been very favorable, and the rest of the IDS industry is noticing. With that said, most IDSs are signature-based with demonstrated IDS excellence.

IDS Features

As discussed earlier in the chapter, IDSs are more than detection engines. Detection is their main purpose, but if you can't configure the system or get the appropriate information out of the IDS, it won't be much help. This section will discuss the end-user interface, IDS management, intrusion prevention, performance, logging and alerting, and reporting and data analysis.

IDS End-User Interfaces

IDS end-user interfaces let you configure the product and see ongoing detection activities. You should be able to configure operational parameters, rules, alert events, actions, log files, and update mechanisms. IDS interfaces come in two flavors: syntactically difficult command prompts, or less-functional GUIs.

Historically, IDSs are command-line beasts with user-configurable text files. Command-line consoles are available on the host computer or can be obtained by a Telnet session or proprietary administrative software. The configuration files control the operation of the IDS detection engine, define and hold the detection rules, and contain the log files and alerts. You configure the files, save them, and then run the IDS. If any runtime errors appear, you have to reconfigure and rerun. A few of the command-line IDS programs have spawned GUI consoles that hide the command-line complexities. A good GUI for Windows-based Snort is IDScenter (www.engagesecurity.com/). After selecting and unselecting various options, the IDScenter GUI edits the necessary Snort configuration files and runs Snort.

Although text-based user interfaces may be fast and configurable, they aren't loved by the masses. Hence, more and more IDSs are coming with user-friendly GUIs that make installation a breeze and configuration a matter of point-and-click. With few exceptions, the GUIs tend to be less customizable than their text-based cousins and, if connected to the detection engine in real time, can cause slowness. Many of the GUI consoles, like IDScenter, present a pretty picture to the end-user but end up writing settings to text files, getting the benefits of both worlds.

NOTE *A frequent complaint of new GUI IDS users is that once the IDS is turned on, "nothing happens!" This is because the IDS is (1) not detecting any defined threats, (2) not placed appropriately in the network topology to be able to sniff traffic, or (3) not configured to display events to the screen (because doing so wastes valuable CPU cycles).*

IDS Management

Central to the IDS field are the definitions of *management console* and *agent*. An IDS agent (which can be a *probe, sensor,* or *tap*) is the software process or device that does the actual data collection and inspection. If you plan to monitor more than two network segments, you can separately manage multiple sensors by connecting them to a central management console. This allows you to concentrate your IDS expertise at one location.

IDS management consoles usually fulfill two central roles: configuration and reporting. If you have multiple agents, a central console can configure and update multiple distributed agents at once. For example, if you discover a new type of attack, you can use the central console to update the attack definitions for all sensors at the same time. A central console also aids in determining agent status—active and online or otherwise.

NOTE *If the management console and sensors run on different machines, traffic between the two should be protected. This is often accomplished using SSL or a proprietary vendor method.*

In environments with more than one IDS agent, it is crucial to report captured events to a central console. This is known as event *aggregation*. If the central console attempts to organize seemingly distinct multiple events into a smaller subset of related attacks, it is known as event *correlation*. For example, if a remote intruder port-scans five different hosts, each running its own sensor, a central console can combine the events into one larger event. To aid in this type of correlation analysis, most consoles allow you to sort events by

- Destination IP address
- Source IP address
- Type of attack
- Type of protocol
- Time of attack

You can also customize the policy that determines whether or not two separate events are related. For example, you can tell the console to link all IP fragmentation attacks in the last five minutes into one event, no matter how many source IP addresses were involved. Agents are configured to report events to the central console, and then the console handles the job of alerting system administrators. This centralization of duties helps with setting useful alert thresholds and specifying who should be alerted. Changes to the alert notification list can be made on one computer instead of on numerous distributed agents.

A management console can also play the role of expert analyzer. Lightweight IDSs perform the role of agent and analyzer on one machine. In larger environments with many distributed probes, agents collect data and send it to the central console without determining whether the monitored event was malicious or not. The central console manages the database, warehousing all the collected event data. As shown in Figure 14-3, the database may be maintained on a separate computer connected with a fast link.

Of course, having a central management console means having a single point of failure. If the management console goes down, alerts will not be passed on, and malicious traffic may not be recorded. Despite this risk, however, if you have more than one sensor, a management console is a necessity. And if a central console is helpful for managing multiple IDS sensors, it also holds true that it can be helpful for managing information from even more computer security devices.

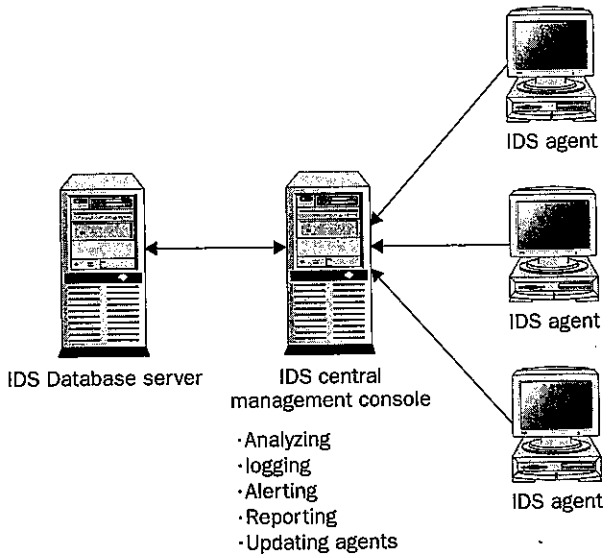

FIGURE 14-3 Example of a distributed IDS topology

Multiple security systems can report to a centralized *enterprise-management system* (EMS), bringing together logs and alerts from several disparate sources. For example, all logs and alerts from all IDSs, perimeter firewalls, personal firewalls, antivirus scanners, and operating systems can be tied together. Events from all logs can be gathered, analyzed, and reported on from one location. This is the ultimate in event correlation, offering one place to get a quick snapshot of security or to get trend information.

For example, suppose the Slammer SQL worm hit some perimeter firewalls and they reported stopping the worm's attempt on UDP port 1434, but the EMS console also shows very high, sustained utilization on a few Windows 2000 workstations running Microsoft SQL Desktop Engine starting around the same time. An intuitive security administrator would want to investigate those workstations and look for a misconfigured firewall, because they are signs that the worm was successful in getting into the organization. Without an EMS console tracking events from multiple locations and sources, the intrusion might go unnoticed for longer.

EMSs can also coordinate signature and product updates. There is great growth in the field of EMSs, and some log-analysis vendors are promoting their EMS tools as cross-platform and as supporting multiple IDS products.

A common problem when multiple vendors are involved is that each security system defines events and reports problems differently. Identical events may have different names, log files might have different data formats, and agent event thresholds may have different settings. For example, one agent may define a port scan as three incremental UDP probes (that is, 1, 2, 3) in less than five seconds, originating from the same host, whereas another may define a port scan as five UDP probes on any port in under ten seconds. Also, although a management console may centrally collect and report all events, it might not be possible to drill down into the details without going to each originating system. For example, the EMS may report unauthorized outgoing port attempts across multiple workstations, but you might have to go to each computer to find the process trying to open up the ports. There may be one place to collect all the events, but the real challenge is in communicating events and actions in a standard way between different devices and vendors.

Intrusion-Detection Messaging

To deal with this IDS Tower of Babel, there have been several attempts over the years to define and format events with a common descriptive language. The Internet Engineering Task Force's (IETF's) Intrusion Detection Exchange Format working group (IDWG) has submitted an Internet draft defining a language specification and protocol to be used in or between different vendor's IDS offerings. The IDWG's output uses Extensible Markup Language (XML), and its goal is to correlate data between multiple companies and sites in anticipation of detecting and fighting a global Internet intrusion event, such as a widespread worm. See www.ietf.org/html.charters/idwg-charter.html for more details.

The IDWG's efforts were inspired by the Common Intrusion Detection Framework (CIDF) project, which is currently dormant (www.isi.edu/gost/cidf). Mitre's Common Vulnerabilities and Exposures (CVE) dictionary (www.cve.mitre.org) is the most popular threat index in the security world. It gives distinctive vulnerabilities unique names and unifying descriptions. Several IDSs reference CVE numbers in their alerting and logging. Another security-data exchange language is called Application Vulnerability Description Language (AVDL) and is being defined by the AVDL Technical Committee (www.avdl.org). It was created to specifically address application-layer vulnerabilities.

In trying to establish a unified messaging standard, there are at least a handful of different standards—which means no standard at all. The need for a common intrusion-detection language is great, and it is likely that a common standard will develop in the near future. Make sure your IDS vendor appears to be getting on the right standards bandwagon as the winner emerges.

Intrusion-Prevention Systems (IPSs)

It has been a concern since the beginning of IDS development that IDSs be able to do more than just monitor and report maliciousness. What good is a device that only tells you you've been maligned when the real value is in preventing the intrusion? That's like a car alarm telling you that your car has been stolen, after the fact. Like intrusion detection, intrusion prevention has long been practiced by network administrators as a daily part of their routine. Setting access controls, requiring passwords, enabling real-time antivirus scanning, updating patches, and installing perimeter firewalls are all examples of common intrusion-prevention controls. Intrusion-prevention controls, as they apply to IDSs, involve real-time countermeasures taken against a specific, active threat. For example, the IDS might notice a ping flood and deny all future traffic originating from the same IP address. Or a host-based IDS might stop a malicious program from modifying system files.

Going far beyond mere monitoring and alerting, second-generation IDSs are being called *intrusion-prevention systems* (IPSs). They either stop the attack themselves or interact with an external system to put down the threat.

If the IDS, as shown in Figure 14-4, is a mandatory inspection point with the ability to filter real-time traffic, it is considered *inline*. Inline IDSs can drop packets, reset connections, and route suspicious traffic to quarantined areas for inspection. If the IDS isn't inline and is only inspecting the traffic, it still can instruct other network perimeter systems to stop an exploit. This may be done by sending scripted commands to a firewall, instructing it to deny all traffic from the remote cracker's IP address, calling a virus scanner to clean a malicious file, or simply telling the monitored host to deny the hacker's intended modification.

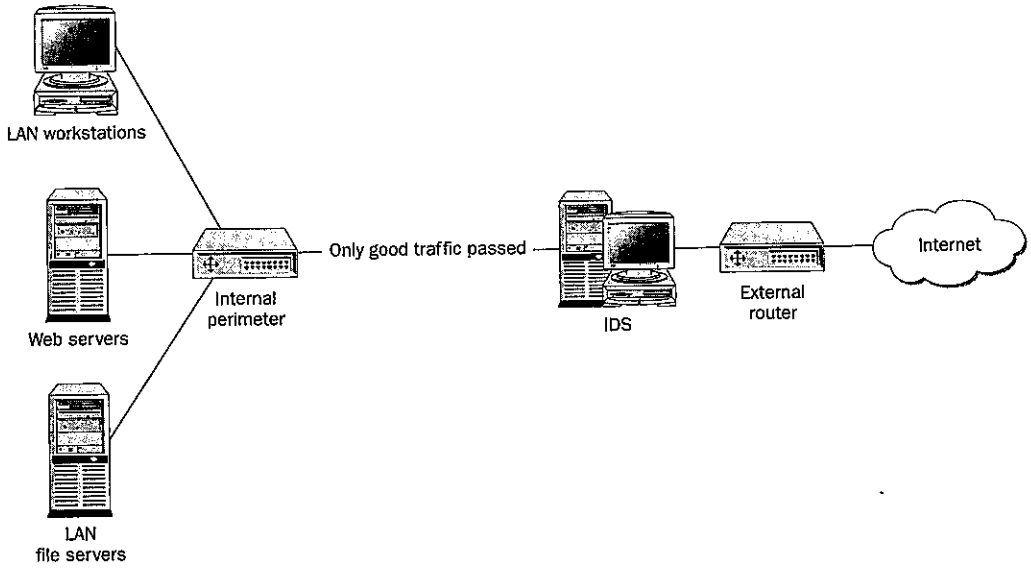

FIGURE 14-4 IDS placed to drop malicious packets before they can enter the network

In order for an IDS to cooperate with an external device, they must share a common scripting language, API, or some other communicating mechanism. Another common IPS method is for the IDS device to send reset (RST) packets to both sides of the connection, forcing both source and destination hosts to drop the communication. This isn't seen as very accurate, because often the successful exploit has happened by the time a forced reset has occurred, and the sensors themselves can get in the way and drop the RST packets.

Although IPS is the new buzz phrase in the IDS field, firewalls and most IDSs have been doing intrusion prevention for years. The name change is mostly marketing hype driven by practical realities. Attack detection is a commodity these days, but most IDSs do it fairly accurately and none do it perfectly. Vendors are trying to differentiate their products by what they do with what they detect. The ultimate goal of any IDS should be to notice an attack and stop it without human intervention. Expect many advances in IPS in the coming years, although it's concerning that vendors are moving full-steam ahead with intrusion prevention before making detection more accurate.

IPS Disadvantages

The biggest disadvantage of an inline IPS is the hit to network performance. As an inline choke point, every packet is potentially subjected to thousands of pattern comparisons. The increase in network latency may not be acceptable.

Another problem to consider is what happens if an inline device fails. Does it fail open or closed, and which method is preferred? If the device fails closed, then your network is down until you get the IPS issue resolved (or you can have redundant failover paths, of course). If it

fails open, then security exploits could get through. If the device fails, it probably won't send an alert to let you know, and if it fails open, you could be without security for a long period of time without realizing it. Companies where security is a top priority want IPSs to fail closed and are willing to accept related downtime.

A well-known consequence of IPSs is their ability to exacerbate the effects of a false-positive. With IDSs, a false-positive leads to wasted log space and time, as the administrator researches the threat's legitimacy. IPSs are proactive, and a false-positive means a legitimate service or host is being denied. How many of us have installed a personal firewall (a desktop IDS) only to have Network Neighborhood stop working or had problems retrieving e-mail? Malicious crackers have even used prevention countermeasures as a DoS attack (see the "IDS Weaknesses" section later in this chapter).

A Good Defense Is an Offense

Some IPSs go on the counterattack. They may try to locate and identify the cracker's host machine, send the hacker's ISP a complaint message, or even modify the machine hosting the threat to stop the attack. Although the legalities of doing so are questionable, some less-accepted IPSs will reverse-infect or shut down a compromised host to stop an attack.

For example, when the Fizzer worm began attacking IM chat channels in May 2003, a group of channel sysops went on the offensive. The Fizzer Task Force, as they called themselves, created channel bots that would recognize infected workstations, modify them, and clean the worm. This type of offense is nothing new and has been done since the early days of malicious mobile code. However, most security professionals eschew unauthorized system modification, no matter what the intent.

Is It a Firewall or an IDS?

With the growing importance of intrusion prevention, most firewalls are beginning to look a lot like IDSs, and IDSs can look a lot like firewalls. Although there is no hard and fast rule, one way of distinguishing is that if the device inspects payload content to make its decision or identifies the exploit by name, it's an IDS. Historically, firewalls make decisions by IP address and port number at layers three and four. IDSs can do that, but they can also identify the particular exploit if there is a previously defined pattern.

For example, a basic firewall may tell you that it blocked an unauthorized inbound connection attempt on port 31,337, where an IDS would call it a BackOrifice remote access Trojan scan. A firewall would tell you that it blocked a port 80 attempt to one of your workstations, where an IDS could tell you it was the Code Red worm. A firewall could tell you it had dropped a series of fragment packets, where an IDS could tell you it was a teardrop attack.

An IDS can compile several different connection attempts and recognize they were part of one port-scan event, and maybe even identify the port scanning tool (such as Superscan or nmap). A firewall would report each separate connection attempt as a separate event. IDSs have more expert knowledge and can identify exploits by popular name. Because it makes practical sense, the two markets are beginning to merge into one as both mature (see "The Future of IDS" section later in the chapter).

IDS Performance

An inline signature-based NIDS has a tough job to do. It has to capture packets traveling along a network segment at wire speeds, normalize the data stream, keep track of session state, and compare each packet against an array of stored signatures, all in fractions of a second. If an IDS is placed as an Internet perimeter device, it's probably inspecting data on a T-1 line, a cable or DSL modem, or something faster. Inside the network, most corporations have Fast Ethernet workstation segments (100 Mbps) and Gigabit Ethernet (a billion bits per second) backbones. A small corporate link will have anywhere from hundreds to tens of thousands of packets to interrogate each second. A large entity can have millions of packets per second that need to be inspected. It's natural to question whether an IDS can keep up with inspecting traffic at today's fastest data rates. The short answer is no, not at gigabit speeds.

There were a handful of independent benchmarking performance tests done against gigabit IDSs in 2002 and 2003. In most cases, as speed increased, reliability decreased. *Network World* magazine did an article in June 2002 (www.nwfusion.com/techinsider/2002/0624security1.html) that tested eight different NIDSs under real-world, high-traffic conditions. Under loads as low as 9 to 12 Mbps, seven of the eight IDSs locked up frequently, both consoles and sensors. The remaining IDS that did not lock up wasn't the most accurate device in the review. The good news? High-performance IDSs are getting better each year, and most IDSs can capture, analyze, and store more data than you can manage. Neohapsis's Open Security Evaluation Criteria (OSEC, http://osec.neohapsis.com) has evaluated a handful of NIDSs successfully handling traffic rates of 750 to 1,500 Mbps, and their testing methodology is as sound as any.

Hardware Appliances

Many vendors are answering the need for faster NIDSs with custom stand-alone hardware appliances. Nearly half the commercial NIDSs on the market are appliances. IDS appliances can be closed-system Intel PCs with a specialized operating system or can be installed as solid-state circuitry chips. IDS appliances often have application-specific integrated circuit (ASIC) or Field Programmable Gate Array (FPGA) chips. ASIC chips have burned-in instructions and cannot easily be upgraded in the field. FPGAs are slightly slower than ASIC chips, but they interact with software, allowing signature and engine upgrades to be done without replacing the chip. Both chip types are rated to perform IDS instructions over a thousand times faster than IDSs running on a regular CPU. As a result, IDS appliances are becoming more popular. Many network vendors offer IDSs as add-on modules to their network switches and routers.

NOTE *Most IDS benchmarking tests cannot be trusted to represent real-world applications because most tests are done using traffic generators that do not mimic normal network traffic patterns and deviations. Especially do not trust vendor benchmarks, as they often put specific conditions on the traffic, dropping a large amount of the inbound packets before inspection, inflating throughput figures. As with any security tool, try before you buy.*

Performance Tuning

If you use an inline IDS, you may run into performance problems even at speeds much lower than Gigabit Ethernet. This means dropped packets, slower network throughput, or lockups. To optimize the speed of your inline IDS, consider the following:

- Use one or more fast CPUs
- Use a fast disk subsystem
- Oversupply RAM
- Use a fast network card with top-rated throughput performance (sometimes a new driver can improve speed dramatically)
- Consider using an IDS appliance with ASIC or FPGA chip technology
- Turn off unneeded rules
- Turn off full packet capturing
- Decrease the amount of data your IDS shows on the screen
- Use efficient databases with your IDS
- Talk to your vendor about how to increase performance and keep accuracy

Turning off unneeded rules is a good way to increase IDS performance. For example, if you don't have a Unix server at your location, consider not using any rule that detects Unix exploits. Sure, you may miss a Unix cracker trying to use a particular exploit, but if it is not ever going to be successful against your environment, it has a questionable cost/benefit tradeoff when measured against performance pressures. Another example is if you have a good enterprise antivirus solution, you could consider turning off antivirus signature rules.

IDS Logging and Alerting

When security events are detected by IDSs, they generate alerts and log files.

Alerts

Alerts are high-priority events communicated to administrators in real time. The IDS's policy determines what security threats are considered high risk, and the priority level is set accordingly. Typically, you would not want an IDS administrator to respond as quickly to a NetBIOS scan against your appropriately firewalled network as you would to a successful DoS attack against your company's primary web server. When an event is considered high risk against a valuable asset, it should be communicated immediately.

Alerts can be sent using all sorts of methods, including these:

- Dialog box to a console screen
- SMTP or Short Messaging Service (SMS) message
- Alphanumeric text to a pager
- SNMP trap
- Console sound

Carefully contemplate what method should be used for communication. For example, most IDS alerts are sent via e-mail. In the case of a fast-spreading e-mail worm, the e-mail

system will be severely taxed, and finding an alert message among thousands of other messages might be daunting. For the same reason, storing a cell phone or pager e-mail address in a mail server's global address book would allow an e-mail worm to quickly fill up the pager or cell phone with infected messages and would obscure the informational alerts.

Alerts should be quick and to the point. They should describe location, event, and priority, and they should fit on a small display, like these two examples:

```
LAN3-1: Smurf attack-Medium
Corpweb3: DoS-High
```

More advanced IDS systems allow you to combine identical alerts occurring in a given time period as the same event. Although this might not seem important as you read this book, it becomes important to the administrator at 3 A.M. when one port scan turns into over a thousand different alerts in under a minute. Correlation thresholds allow a security administrator to be appropriately alerted for an event without feeling like the whole network is under siege.

Logs

IDS log files record all detected events regardless of priority and, after its detection engine, have the greatest influence on the speed and use of an IDS. IDS logs are used for data analysis and reporting. They can include just a barebones summary of events or a complete network packet decode. While complete network traces are preferable for forensics, they can quickly take up a lot of hard drive space. A small network can generate hundreds of events a minute, and a mid-sized network can generate tens of thousands. If you plan to store multiple days worth of logs with full packet decoding, the IDS's hard drive should be at least 80GB. Consider using SCSI drives with RAID-5 striping to increase write speed.

Log files can be saved to a scaleable database, like MS SQL or MySQL, or as plaintext files. While SQL may appear to be the easy database of choice because of its widescale use in most enterprises, high-speed IDSs can quickly outperform an SQL database. An IDS may be recording 100,000 events per second, and most SQL databases, within any reasonable price range, are only capable of handling thousands per second. Most high-speed IDSs save their logs to flat, but efficient, ASCII text files. The text files are imported to other database engines for analysis and reporting.

Recently, one administrator complained to an IDS vendor that their product was sluggish and nowhere near the benchmarks they reported. It turned out that he had unilaterally decided to hook the IDS to an SQL database instead of using the vendor's recommended proprietary solution. Once the switch was made to the right database, the IDS exceeded expectations. It also forced the administrator to decide between using an optimized database with fewer features or a slower database that supported all the other operational standards (such as reporting, data extraction, and the like).

Regardless of the log format an IDS uses, all log files must be rotated out frequently in order to maintain performance and to prevent lockups. Unfortunately, when you rotate a log file out, it complicates threat analysis, because you will have to merge multiple files to cover a greater time period. Common log file formats include fixed-width ASCII text, comma-delimited ASCII text, HTML, XML, CVS, syslog, SQL, and other proprietary standards. Some IDSs digitally sign their log files to authenticate them.

At a bare minimum, a log file should record the event location, timestamp (date and time to the hundredth of a second), description of the action attempted, criticality, and IDS response, if any. If the event was recorded using network packets, then the following additional information should be noted: source and destination IP addresses, protocol, and port number. The log should provide a short description of the attack and give links to the vendor or other vulnerability web sites for a more detailed explanation.

NOTE *Reporting event timestamps in UTC (Coordinated Universal Time) time will simplify your task when reporting events to external authorities in different time zones.*

Most vulnerability databases describe the security event as if it can only be a malicious attack, when in fact this is often not true. IDS vendor databases should also list reasons why the reported event may be a false-positive. For example, if the IDS reports an IP spoof event, it's helpful to read that IP spoofs can be created by poorly configured, but legitimate, VPN links. If you keep receiving port-scanning alerts that you trace to your ISP's DNS servers, you can learn that it is a normal behavior for them as they attempt to respond to misconfigured client workstations.

IDS Reporting and Analysis

Closely related to logging is reporting and analysis. Reporting tools (also called *log analyzers*) take the log files, find commonalities (such as attack types and threat origination), and summarize the results for a particular time period. Most commercial IDS products come with a handful of canned reports and allow customized reports to be defined. IDS administrators are free to use any reporting tools included, or they can extract data to be analyzed with other reporting tools, like Crystal Reports.

IDSs are good for displaying logs on the console, one screen at a time, or for reporting data on a daily basis. But depending on the IDS you are using, the reporting tool, and the amount of data being massaged, it can take from minutes to hours to generate older reports. It is not usual for an IDS report covering hundreds of thousands of events to take hours to run. Many IDS administrators managing high-speed IDSs run from one to three reports each week, looking for significant events or trending. They usually allow the reports to run at off hours to get the most CPU cycles, and they have the reports waiting for them in the morning. To improve speed, you have to either examine less data or throw more processing power at the problem. High numbers of false-positives and slow reporting ensures that many IDS administrators don't print reports. They find the most value in real-time alerts and daily log files. Many IDSs have HTML reports viewable through a browser, but they are slower than the other reporting methods.

On the analysis end, IDSs are far from being expert intelligence systems. They can recognize predefined patterns and track statistics, but it usually takes a trained IDS administrator to see the bigger picture. For example, an IDS may notice an unusual port connection coming for an end-user's workstation, but it takes a human to determine that the port attempt is originating from a new remote access Trojan. Next-generation IDSs are expected to have more intelligent analysis to provide administrators with conclusions instead of raw facts.

IDS Deployment Considerations

IDSs are beneficial tools, but they have weaknesses. They also need to be fine-tuned to be very useful, and if you intend to deploy one, you'll need to come up with a deployment plan to do so successfully. IDSs are a lot of hard work. This section of the chapter will discuss these issues and will also survey current IDS products and online resources.

IDS Weaknesses

An IDS can improve the overall security of any environment, but it is not without a tradeoff cost and technical weaknesses.

False-Positives

IDSs are frequently called alarm systems. If so, then surely IDSs are car alarms, going off with such frequency that the value of their alerts can quickly be overwhelmed with false-positives. It's the nature of the beast, and IDS vendors apparently can't tame the problem because it is still the number one IDS problem after more than a decade of trying. Non-malicious, innocent, network packet anomalies happen all the time. Misbehaving and nonconforming applications generate suspicious-looking traffic. Many Internet programs install adware (software that downloads ads from remote servers to a PC) that often uses unauthorized outgoing ports. You may want to know about adware so that you can get rid of it, but will your IDS be able to rank the criticality of the event as being lower than a more malicious attack?

Even a real attack can be considered a false positive when it's not relevant to your environment. If your network is 100 percent Windows, do you care that Unix exploits are being attempted against it? IDSs have to be trained to consider event *relevancy*. Even if you want to note all legitimate threats, those not relevant to your environment should have a lower priority ranking than threats that could be successful. Irrelevant events should be noted in summary reports, but should not be interpreted as alert events to bother the administrator.

Windows machines are famous for producing protocol-violating traffic. Windows will frequently ignore the negotiated TCP window size or will attempt to send voluminous amounts of data via a single UDP packet. If you are new to IDSs, you will surely discover that you have a lot more protocol traffic traveling around your network than you previously knew. An e-mail containing a new threat signature may set off your IDS even though the content is safely stored in the confines of message text.

Some hacker tools, like stick, are built exclusively to overwhelm IDSs with false-positives. Stick is a packet generator that uses Snort's own rules database to generate false-positive alerts. It can be used to overwhelm the Snort administrator with tons of false-positives and allow real threats to sneak by unnoticed. Snort's creator, Marty Roesch, created a plug-in called stream4 to defeat stick's stateless packets.

Many companies that install an IDS are excited to see the amount of information one can provide. The excitement quickly turns into frustration, and often the high number of false-positives exhausts the administrator to the point where they don't read the logs or pay attention to the alerts anymore. At that point, what's the use of having an IDS at all? To avoid that situation, keep on top of event logs and fine-tune your IDS to minimize the number of false-positives.

Expense

An IDS purchase can be free or expensive, but they are always expensive to maintain. They aren't install-and-forget systems. IDSs generate a huge amount of information—it is not unusual for one IDS to generate thousands of messages a day in a medium-sized environment of only a few hundred computers. Large environments with dozens of distributed sensors can get millions of messages a day. No matter how smart the expert system of an IDS is, someone trained is going to have to eventually review alerts and log files and respond to critical events. Before you buy an IDS, ask yourself the following questions:

- Do you have the expertise to run an IDS? If not, can you afford the expertise?
- Who is going to install your IDS?
- Who is going to maintain and fine-tune the IDS?
- Who is going to follow up on alerts?
- Who is going to configure rules?
- Who is going to write and update signatures?
- Who is going to sift through the log files and filter out all the false-positives?

Many companies have come to the conclusion that they do not have and can't afford the expertise in-house, so they hire an external firm (called a *managed security service provider* or MSSP) to manage their IDS for them. If the company can afford it, a managed solution is often the best choice. The MSSP has the expertise to quickly cull out false-positives and it knows when to respond to alerts. If a new threat breaks out infecting the Internet, managed clients are among the first to get notified and updated.

If your company decides to use an MSSP, it's important to get an acceptable uptime guarantee, acceptable response times, and other assurances that will minimize your staff's involvement.

Volume Limitations

Every IDS says it can handle high volumes of traffic, but in testing most have some percentage of dropped packets, missed events, and lockup problems when faced with high utilization. If you place an IDS inline as a perimeter funnel to approve all passing traffic, it can significantly slow down network throughput.

Lockups

IDSs seem to have more than their fair share of lockups, or they go so slowly that they appear to have locked up. Some have sensor lockups, others have problems at the management console. Usually cycling the logs or rebooting the IDS is enough to get rid of the problem, but should it really happen in the first place? And if your IDS is inline, does it fail opened or closed, and what are the security repercussions?

Spoofed IP Addresses

Crackers often spoof their source IP addresses, and if your IDS includes a reactive countermeasure, you could be blocking the wrong source. There have been cases where a cracker intentionally sent large numbers of attacks with wide ranges of spoofed IP addresses in a successful attempt to cause a DoS attack.

This happened to a non-profit company that ran a large Christian-based web site. After their well-known spokesman was quoted in the national media condemning alternate lifestyles, their main web server was defaced, and hundreds of thousands of port scans began arriving each hour. Their satellite and T-3 links were completely saturated with malicious traffic. Initially, everyone on the security team marveled over how well the intrusion-prevention device was doing and was grateful that the firewall was automatically blocking traffic from the originating hosts. Then they were able to prove that the packet addresses were spoofed, and they wondered why port scans would be sent that would never arrive back to a legitimate source?

Hours later, when the attack was finished, the company's intrusion-prevention system had blocked millions of legitimate IP addresses from interacting with their commercial web site. The cracker's knowledge of the intrusion system's countermeasures allowed them to turn the client's automated defenses into a DoS attack. Most of today's firewalls and IPSs have time limits on automatic lockouts to prevent these types of attacks.

Evasion Techniques

The field of IDS is full of evasion techniques, including

- Malware variants
- Fragmentation attacks
- Obfuscation and encoding
- Encrypted traffic
- Prolonged attacks designed not to set off IDSs.
- False-positive attacks

Most of these have already been covered in this chapter. The only topic we haven't touched on is the problem created by encrypted traffic.

Encryption Blocks Inspection

The network world is increasingly using encryption to prevent third-party eavesdropping. Common schemes used to encrypt network traffic include IPSec, PGP, SSH, DES, WEP, and AES. IPSec is a now a Windows encryption default, PGP is the world's most popular e-mail encryption program, and SSH is used nearly universally on Unix computers. For instance, if an SMTP user uses PGP to send and receive encrypted e-mail, the IDS will not be able to inspect the message for malicious content and attachments.

Crackers are increasingly using encryption in an effort to prevent inspection and detection. If payload content is encrypted, at best an IDS might be able to read packet headers to detect traffic and protocol anomalies. However, many VPN encryption schemes, like IPSec, even encode the TCP and IP headers, cutting the IDS almost entirely out of the inspection loop.

Data encryption is expected to rise over the next decade, and this trend will frustrate IDS administrators. If the encrypted streams are authorized, it may be possible to include the IDS as a trusted reader, sharing the encryption scheme and shared secret. If the encryption is unauthorized, as in the case of a cracker, the traffic must be captured before or after being encrypted. This is easier for HIDS than NIDS, of course. Expect to see more IDS products with decryption capabilities in the future.

The IDS as a Target

The hunter can become the hunted. Because most IDSs are software programs running on top of a hardened operating system, they can be attacked themselves in order to compromise a network. Sometimes the IDS, itself, opens up new TCP or UDP ports on a host in order to facilitate communication. Hackers can scan for and target these new ports or attack the existing operating system ports that the IDS monitors. Snort has been the target of at least two different attacks, one a buffer overflow and the other a DoS attack. Microsoft's ISA Server and Checkpoint's Firewall-1, both firewalls with IDS capabilities, have been the subject of DoS attacks. There exists the potential that a cracker may specifically target an organization knowing that the IDS the company is running contains an exploitable hole.

You should understand the value that IDSs bring to an organization's security, but you must also understand the weaknesses. The benefits of increased security must be weighed against the expertise and time needed to maintain an IDS and follow up on alerts.

IDS Fine-Tuning

Fine-tuning an IDS means doing three things: increasing inspection speed, decreasing false-positives, and using efficient logging and alerting.

Increasing Inspection Speed

Although we spent time covering the best ways to make an IDS faster in the "IDS Performance" section earlier in the chapter, how you tweak an IDS's runtime settings can significantly affect performance. What you should tweak depends on the IDS product and the network environment, although narrowing down what the IDS inspects and optimizing the rules are usually good places to start.

Most IDS administrators start off monitoring all packets and capturing full packet decodes. You can narrow down what packets an IDS inspects by telling it to include or ignore packets based upon source and destination addresses. For example, if you are most concerned with protecting your servers, modify the IDS's packet inspection engine so it only captures packets with server destination addresses. Another common packet filter is a rule that excludes broadcast packets between routers. Routers are always busy chatting and broadcasting to learn routes and reconstruct routing tables, but if you aren't worried about internal ARP poisoning, don't capture ARP packets. The more packets the IDS can safely ignore, the faster it will be.

Another strategy is to let other faster perimeter devices do the filtering. Routers and firewalls are usually faster than IDSs, so, when possible, configure the packet filters of your routers and firewalls to deny traffic that should not be on your network in the first place. For example, tell your router to deny IP address spoofs, and tell your firewall to drop all NetBIOS traffic originating from the Internet. The more traffic that you can block with the faster device, the higher performing your IDS will be. That's the way it should be—each security device should be configured to excel at what it does best, at the layer it does it best.

Configure your rules to optimize inspection. This means that if your IDS allows it, only have it inspect network packets in areas where the malicious content can be located. For example, suppose your IDS is looking for the phase "$$$ victim" as the signature to a FTP attack, and that phrase will always appear in byte positions 51–60. Configure the signature rule to only look in byte positions 51–60 for the malicious string. If data packets can routinely be 1,500 bytes long, you've decreased the IDS scanning effort by 99 percent. If the attack can

only be in established TCP sessions on port 21, tell the IDS only to look if the packet belongs to an established FTP session. No need to look in initial SYN packets or in packets with other port numbers for content that will never be there. And make sure that the destination address the rule looks for matches your FTP server or servers.

NOTE *Anyone interested in rule optimization should read Snort's documentation on creating and optimizing rules (www.snort.org). It is an excellent primer.*

Decreasing False-Positives

Because IDSs have so many false-positives, it should be the number one job of any IDS administrator to track down and troubleshoot false-positives. In most instances, false-positives will outweigh all other events. Track them all down, rule out maliciousness, and then appropriately modify the source or IDS to prevent them. Often the source of the false-positive is a misbehaving program or a chatty router. If you can't stop the source of the false-positive, modify the IDS so it will not track the event. The key is that you want your logs to be as accurate as they can be, and they should only alert you to events that need human intervention. Don't get into the habit of ignoring the frequently occurring false-positives in your logs as a way of doing business. This will quickly lead to the administrator missing the real events buried inside all the false-positives, or lead to the logs not being read at all.

Efficient Logging and Alerting

Hopefully, your IDS has several levels of prioritization so that you can be alerted to the most threatening events first. If a port scan and IIS buffer overflow exploit happens at the same time, and you have an unpatched IIS server, you want the IIS alert to get top billing. If you are sure your network is free of remote access Trojans, don't react to random external probes for NetBus and BackOrifice programs. While it is nice that the IDS logs unsuccessful probes, an administrator should only be involved with persistent or high-risk efforts.

Most vendor products come with their own preset levels of event criticalities, but when setting up the IDS, take the time to customize the criticalities for your environment. For instance, if you don't have any Apache web servers, set Apache exploit notices with a low level of prioritization. Or better yet, don't track or log them at all.

For performance reasons, if you need to capture and log full packet decodes, consider using an external packet-capture tool instead of using the IDS's own abilities. This allows you to more efficiently capture all traffic while not slowing the IDS down with packet decode logging duties. Some IDSs, like Snort, will allow you to capture packet decodes in a binary stream and later re-create the original decoded packets in the event that you need to see or replay them.

NIDS Deployment Plan

So you want to deploy your first NIDS. You've mapped your network, surveyed your needs, decided what to protect, and picked an IDS solution. Here are the steps to a successful NIDS deployment:

1. Document your environment's security policy.
2. Define human roles.

3. Decide the physical location of the IDS and sensors.
4. Configure the IDS sensors and management console to support your security policy.
5. Plan and configure device management (including the update policy).
6. Review and customize your detection mechanisms.
7. Plan and configure any prevention mechanisms.
8. Plan and configure your logging, alerting, and reporting.
9. Deploy the sensors and console (do not encrypt communication between sensors and links to lessen troubleshooting).
10. Test the deployment using IDS testing tools (initially use very broad rules to make sure the sensors are working).
11. Encrypt communications between the sensors and console.
12. Test the IDS setup with actual rules.
13. Analyze the results and troubleshoot any deficiencies.
14. Fine-tune the sensors, console, logging, alerting, and reporting.
15. Implement the IDS system in the live environment.
16. Define continuing education plans for the IDS administrator.
17. Repeat these steps as necessary over the life of the IDS.

As you can see, installing and testing an IDS is a lot of work. The key is to take small steps in your deployment, and plan and configure all the parts of your IDS before just turning it on. The more time you spend on defining reporting and database mechanisms at the beginning, the better the deployment will go.

During the initial tests, in step 10, it can help to use a test rule that is sure to trigger the IDS sensor or console on every packet. This will ensure that the physical part of the sensor is working and will let you test the logging and alerting mechanisms. Once you know the physical layer is working, you can remove that test rule (or comment it out or unselect it, in case you need it later). It is advisable not to turn on encryption, digital signing, or any other self-securing components until after you've tested the initial physical connections. This reduces troubleshooting time caused by mistyped passphrases or incorrectly configured security settings.

You can find lots of hacking tools on the Internet to test your IDS. Port scanners, penetration-testing tools, and vulnerability-assessment tools abound and work well. Just make sure to download your testing tools from a reliable site.

Lastly, keep on top of your logs, and research all critical events. Quickly rule out false-positives, and fine-tune your IDS on a regular basis to minimize false-positives and false-negatives. Once you get behind in your log duty, it's tough to catch up again. Successful IDS administrators track and troubleshoot everything as quickly as they can. The extra effort will pay dividends with smaller and more accurate logs.

The Future of IDS

The future of IDS is bright, if only because there is significant room for improvement. Here are a few things to look for in the future:

- **Decreasing false-positive detection** This is on the top of every vendor's fix-it list. You can expect products that will automatically recognize common false-positives (DNS port scans, IP spoofing caused by remote VPNs, authentication pings, and so on) and keep those alerts turned off by default. Vendors accepting a high number of false-positives as a cost of doing business will be gone from the marketplace.
- **IDSs becoming IPSs** If the IDS can detect the attack, it can try to prevent it.
- **IDSs using both anomaly detection and signature detection** IDSs will move to use both of these engines, and they should reside on host and network devices, all reporting to a central console. This console will aggregate and correlate events from all network security devices. Several vendors already do this, but more vendors are starting to follow this model.
- **IDSs becoming better expert systems** IDSs are trying to decrease false-positives, consider relevancy, and allow events to have varying levels of priorities. Ultimately, an IDS administrator would like the IDS to do its job perfectly and would only have to print out a monthly activity report to hand to management. While that dream may never happen, IDSs are already handling more exploits automatically with less human intervention. Events with high degrees of accuracy are handled automatically, and others that need further analysis are shuttled to administrators or to a safe quarantine area. For example, one company has a firewall that redirects all suspicious traffic to a honeypot network where it can do no harm, but where it can also be studied.
- **Convergence of functionality and vendors** Most of today's firewalls are becoming IDSs. It's not enough that they do simple port and address blocking. There are several vendors making the case for one inline device to do all the security inspection at once—firewalling, antivirus scanning, intrusion detection, content filtering, and vulnerability analysis. And it makes sense that if the device has to pull the packet aside to inspect it, maybe it should inspect it for all things. Many vendors are partnering with each other to provide such holistic solutions. Check Point's (www.checkpoint.com) OPSEC alliance and Content Vector Protocol (CVP) interoperates with over 350 vendors. On the downside, although Check Point says OPSEC is an open standard, it heavily favors their products. Expect to see other unified languages allow different security products to communicate and interoperate.

Successful small IDS vendors are being bought at a rapid pace by larger security vendors as a way to add IDS products and functionality to their offerings. Symantec, Cisco, and Network Associates have all recently acquired smaller IDS companies.

IDS Products

There are dozens of IDSs to choose from. Table 14-1 contains a list of IDSs, along with some online links to more IDS lists. The field of IDS vendors is constantly changing, with new products and players emerging. Most changes are the result of larger security or network vendors buying up smaller IDS vendors to complete their product lineup.*

Online IDS Resources

Here are some good online links for more information on IDSs:

- Honeypots.net (www.honeypots.net). Online resource for honeypots and IDSs. This is one of the biggest IDS document collections on the Net.
- SecurityFocus IDS section (www.securityfocus.com/cgi-bin/sfonline/ids_topics.pl)
- SANS InfoSec Reading Room-Intrusion Detection (www.sans.org/rr/intrusion) (www.sans.org/rr/catindex.php?cat_id=30)
- LogAnalysis.org (www.loganalysis.org/). A not-for-profit organization dedicated to furthering education surrounding log file analysis.

Summary

An intrusion-detection system should be a part of every network security administrator's protection plan. Along with other ID tools and methods, an IDS can monitor a host for system changes or sniff network packets off the wire, looking for malicious intent. Security administrators should consider using a combination of HIDS and NIDS, with both signature-detection and anomaly-based engines. Put a HIDS on your strategically valuable hosts, and place NIDS appropriately across the network for general early-warning detection. Central management consoles are helpful when multiple, distributed agents are involved.

An IDS can be installed purely as a monitoring and detection device, or it can participate as an inline device and prevent threats. An IDS's biggest weaknesses are the high number of false-positives and the significant maintenance effort needed to keep it up to date and finely tuned. IDS accuracy, performance, and functionality are all improving.

CHAPTER

Integrity and Availability Architecture

by Roberta Bragg, CISSP, MCSE: Security, Security+

Techniques that ensure network integrity and availability are often overlooked in a discussion of network security, yet both are important parts of a system-wide plan. They are often given less promotion because they involve careful, time-consuming, meticulous work—it means continuously following procedures, line by line, one after another, day in and day out. It's typically thankless work and is often delegated to new and junior IT personnel. I'm speaking here of backups, change control, and the patching and provisioning of redundant systems.

NOTE *Implementing sound antivirus policies and practices also contributes to the integrity and availability of systems. Be sure to include an effective program that blocks known and suspicious code at the gateway, scans e-mail and desktops, and blocks the execution of mobile code. (Chapter 29 discusses this in more detail).*

Despite the tedium of many of the tasks, some of the finest architectural work being done today is in the area of integrity and availability. Automated, intelligent processes are hot on the heels of solving the bone-wearying process of keeping systems patched. Online data vaulting and disk-to-disk backup are beginning to make it possible to keep pace with data—to provide resources for recovery when catastrophic disaster strikes. Shadow copy backups are making it possible for users to recover accidentally deleted files without the help of the IT department. It's possible to provide redundancy of systems such that it is common to speak not of 99 percent uptime but of 99.999 percent uptime.

No, its not exactly sexy to be in this side of the information security biz, but it's close. This chapter will look at what gives an information system integrity and availability. The solutions discussed here will provide

- Version control and change control
- Patching