Module: 05
Network Security and Applications

Syllabus:

Lecture no	Content	Duration (Hr)	Self-Study (Hrs)
1	Network security basics	1	1
2	TCP/IP vulnerabilities (Layer wise)	1	1
3	Packet Sniffing, ARP spoofing, port scanning, IP spoofing	1	1
4	TCP syn flood, DNS Spoofing.	1	2
5	Denial of Service: Classic DOS attacks	1	2
6	ICMP flood	1	2
7	Source Address spoofing	1	2
8	SYN flood, UDP flood, Distributed Denial of Service	1	2
9	Internet Security Protocols: SSL, IPSEC, Secure Email: PGP	1	2
10	Firewalls, IDS and types, Honey pots	1	2

Theoretical Background:

7.2. Threats in Networks

Up to now, we have reviewed network concepts with very little discussion of their security implications. But our earlier discussion of threats and vulnerabilities, as well as outside articles and your own experiences, probably have you thinking about the many possible attacks against networks. This section describes some of the threats you have already hypothesized and perhaps presents you with some new ones. But the general thrust is the same: threats aimed to compromise confidentiality, integrity, or availability, applied against data, software, and hardware by nature, accidents, nonmalicious humans, and malicious attackers.

What Makes a Network Vulnerable?

An isolated home user or a stand-alone office with a few employees is an unlikely target for many attacks. But add a network to the mix and the risk rises sharply. Consider how a network differs from a stand-alone environment:

- *Anonymity.* An attacker can mount an attack from thousands of miles away and never come into direct contact with the system, its administrators, or users. The potential attacker is thus safe behind an electronic shield. The attack can be passed through many other hosts in an effort to disguise the attack's origin. And computer-to-computer authentication is not the same for computers as it is for humans; as illustrated by Sidebar 7-2, secure distributed authentication requires thought and attention to detail.

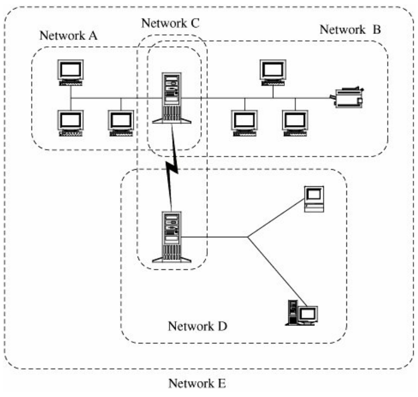
- *Many points of attackboth targets and origins.* A simple computing system is a self-contained unit. Access controls on one machine preserve the confidentiality of data on that processor. However, when a file is stored in a network host remote from the user, the data or the file itself may pass through many hosts to get to the user. One host's administrator may enforce rigorous security policies, but that administrator has no control over other hosts in the network. Thus, the user must depend on the access control mechanisms in each of these systems. An attack can come from any host to any host, so that a large network offers many points of vulnerability.

- *Sharing.* Because networks enable resource and workload sharing, more users have the potential to access networked systems than on single computers. Perhaps worse, access is afforded to *more systems,* so that access controls for single systems may be inadequate in networks.

- *Complexity of system.* In Chapter 4 we saw that an operating system is a complicated piece of software. Reliable security is difficult, if not impossible, on a large operating system, especially one not designed specifically for security. A network combines two or more possibly dissimilar operating systems. Therefore, a network operating/control system is likely to be more complex than an operating system for a single computing system. Furthermore, the ordinary desktop computer today has greater computing power than did many office computers in the last two decades. The attacker can use this power to advantage by causing the victim's computer to perform part of the attack's computation. And because an average computer is so powerful, most users do not know what their computers are really doing at any moment: What processes are active in the background while you are playing *Invaders from Mars*? This complexity diminishes confidence in the network's security.

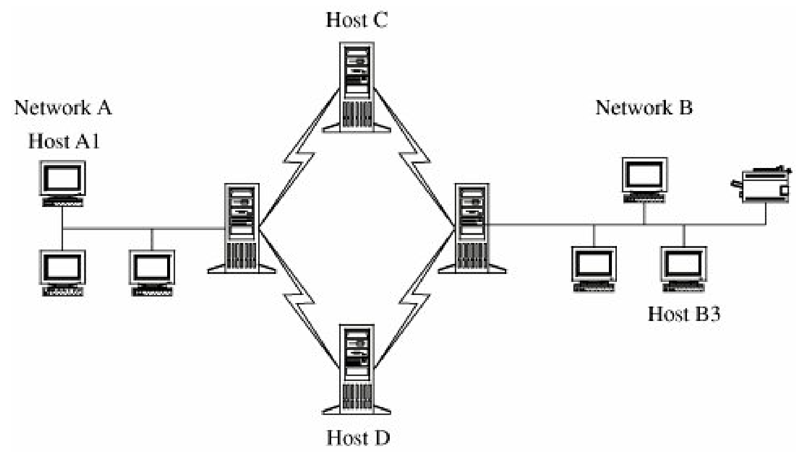
- *Unknown perimeter.* A network's expandability also implies uncertainty about the network boundary. One host may be a node on two different networks, so resources on one network are accessible to the users of the other network as well. Although wide accessibility is an advantage, this unknown or uncontrolled group of possibly malicious users is a security disadvantage. A similar problem occurs when new hosts can be added to the network. Every network node must be able to react to the possible presence of new, untrustable hosts. Figure 7-11 points out the problems in defining the boundaries of a network. Notice, for example, that a user on a host in network D may be unaware of the potential connections from users of networks A and B. And the host in the middle of networks A and B in fact belongs to A, B, C, and E. If there are different security rules for these networks, to what rules is that host subject?

Figure 7-11. Unclear Network Boundaries.



- *Unknown path.* Figure 7-12 illustrates that there may be many paths from one host to another. Suppose that a user on host A1 wants to send a message to a user on host B3. That message might be routed through hosts C or D before arriving at host B3. Host C may provide acceptable security, but not D. Network users seldom have control over the routing of their messages.

Figure 7-12. Uncertain Message Routing in a Network.



Reconnaissance

Now that we have listed many motives for attacking, we turn to how attackers perpetrate their attacks. Attackers do not ordinarily sit down at a terminal and launch an attack. A clever attacker investigates and plans before acting. Just as you might invest time in learning about a jewelry store before entering to steal from it, a network attacker learns a lot about a potential target before beginning the attack. We study the precursors to an attack so that if we can recognize characteristic behavior, we may be able to block the attack before it is launched.

Because most vulnerable networks are connected to the Internet, the attacker begins preparation by finding out as much as possible about the target. An example of information gathering is given in [HOB97]. (Not all information gathered is accurate, however; see Sidebar 7-4 for a look at reconnaissance combined with deception.)

Port Scan

An easy way to gather network information is to use a **port scan**, a program that, for a particular IP address, reports which ports respond to messages and which of several known vulnerabilities seem to be present. Farmer and Venema [FAR93] are among the first to describe the technique.

A port scan is much like a routine physical examination from a doctor, particularly the initial questions used to determine a medical history. The questions and answers by themselves may not seem significant, but they point to areas that suggest further investigation.

Port scanning tells an attacker three things: which standard ports or services are running and responding on the target system, what operating system is installed on the target system, and what applications and versions of applications are present. This information is readily available for the asking from a networked system; it can be obtained quietly, anonymously, without identification or authentication, drawing little or no attention to the scan.

Port scanning tools are readily available, and not just to the underground community. The nmap scanner by Fyodor at *www.insecure.org/nmap* is a useful tool that anyone can download. Given an address, nmap will report all open ports, the service they support, and the owner (user ID) of the daemon providing the service. (The owner is significant because it implies what privileges would descend upon someone who compromised that service.) Another readily available scanner is netcat, written by Hobbit, at *www.l0pht.com/users/l0pht*. (That URL is "letter ell," "digit zero," p-h-t.) Commercial products are a little more costly, but not prohibitive. Well-known commercial scanners are Nessus (Nessus Corp. [AND03]), CyberCop Scanner (Network Associates), Secure Scanner (Cisco), and Internet Scanner (Internet Security Systems).

Social Engineering

The port scan gives an external picture of a networkwhere are the doors and windows, of what are they constructed, to what kinds of rooms do they open? The attacker also wants to know what is inside the building. What better way to find out than to ask?

Suppose, while sitting at your workstation, you receive a phone call. "Hello, this is John Davis from IT support. We need to test some connections on the internal network. Could you please run the command ipconfig/all on your workstation and read to me the addresses it displays?" The request sounds innocuous. But unless you know John Davis and his job responsibilities well, the caller could be an attacker gathering information on the inside architecture.

Social engineering involves using social skills and personal interaction to get someone to reveal security-relevant information and perhaps even to do something that permits an attack. The point of social engineering is to persuade the victim to be helpful. The attacker often impersonates someone inside the organization who is in a bind: "My laptop has just been stolen and I need to change the password I had stored on it," or "I have to get out a very important report quickly and I can't get access to the following thing." This attack works especially well if the attacker impersonates someone in a high position, such as the division vice president or the head of IT security. (Their names can sometimes be found on a public web site, in a network registration with the Internet registry, or in publicity and articles.) The attack is often directed at someone low enough to be intimidated or impressed by the high-level person. A direct phone call and expressions of great urgency can override any natural instinct to check out the story.

Because the victim has helped the attacker (and the attacker has profusely thanked the victim), the victim will think nothing is wrong and not report the incident. Thus, the damage may not be known for some time.

An attacker has little to lose in trying a social engineering attack. At worst it will raise awareness of a possible target. But if the social engineering is directed against someone who is not skeptical, especially someone not involved in security management, it may well succeed. We as humans like to help others when asked politely.

Intelligence

From a port scan the attacker knows what is open. From social engineering, the attacker knows certain internal details. But a more detailed floor plan would be nice. **Intelligence** is the general term for collecting information. In security it often refers to gathering discrete bits of information from various sources and then putting them together like the pieces of a puzzle.

One commonly used intelligence technique is called "dumpster diving." It involves looking through items that have been discarded in rubbish bins or recycling boxes. It is amazing what we throw away without thinking about it. Mixed with the remains from lunch might be network diagrams, printouts of security device configurations, system designs and source code, telephone and employee lists, and more. Even outdated printouts may be useful. Seldom will the configuration of a security device change completely. More often only one rule is added or deleted or modified, so an attacker has a high probability of a successful attack based on the old information.

Gathering intelligence may also involve eavesdropping. Trained spies may follow employees to lunch and listen in from nearby tables as coworkers discuss security matters. Or spies may befriend key personnel in order to co-opt, coerce, or trick them into passing on useful information.

Most intelligence techniques require little training and minimal investment of time. If an attacker has targeted a particular organization, spending a little time to collect background information yields a big payoff.

Threats in Transit: Eavesdropping and Wiretapping

By now, you can see that an attacker can gather a significant amount of information about a victim before beginning the actual attack. Once the planning is done, the attacker is ready to proceed. In this section we turn to the kinds of attacks that can occur. Recall from Chapter 1 that an attacker has many ways by which to harm in a computing environment: loss of confidentiality, integrity, or availability to data, hardware or software, processes, or other assets. Because a network involves data in transit, we look first at the harm that can occur between a sender and a receiver. Sidebar 7-5 describes the ease of interception.

The easiest way to attack is simply to listen in. An attacker can pick off the content of a communication passing in the clear. The term **eavesdrop** implies overhearing without expending any extra effort. For example, we might say that an attacker (or a system administrator) is eavesdropping by monitoring all traffic passing through a node. The administrator might have a legitimate purpose, such as watching for inappropriate use of resources (for instance, visiting non-work-related web sites from a company network) or communication with inappropriate parties (for instance, passing files to an enemy from a military computer).

A more hostile term is **wiretap**, which means intercepting communications through some effort. **Passive wiretapping** is just "listening," much like eavesdropping. But **active wiretapping** means injecting something into the communication. For example, Marvin could replace Manny's communications with his own or create communications purported to be from Manny. Originally derived from listening in on telegraph and telephone communications, the term wiretapping usually conjures up a physical act by which a device extracts information as it flows over a wire. But in fact no actual contact is necessary. A wiretap can be done covertly so that neither the sender nor the receiver of a communication knows that the contents have been intercepted.

Wiretapping works differently depending on the communication medium used. Let us look more carefully at each possible choice.

Impersonation

In many instances, there is an easier way than wiretapping for obtaining information on a network: Impersonate another person or process. Why risk tapping a line, or why bother extracting one communication out of many, if you can obtain the same data directly?

Impersonation is a more significant threat in a wide area network than in a local one. Local individuals often have better ways to obtain access as another user; they can, for example, simply sit at an unattended workstation. Still, impersonation attacks should not be ignored even on local area networks, because local area networks are sometimes attached to wider area networks without anyone's first thinking through the security implications.

In an impersonation, an attacker has several choices:

- Guess the identity and authentication details of the target.
- Pick up the identity and authentication details of the target from a previous communication or from wiretapping.
- Circumvent or disable the authentication mechanism at the target computer.
- Use a target that will not be authenticated.
- Use a target whose authentication data are known.

Authentication Foiled by Guessing

Chapter 4 reported the results of several studies showing that many users choose easy-to-guess passwords. In Chapter 3, we saw that the Internet worm of 1988 capitalized on exactly that flaw. Morris's worm tried to impersonate each user on a target machine by trying, in order, a handful of variations of the user name, a list of about 250 common passwords and, finally, the words in a dictionary. Sadly, many users' accounts are still open to these easy attacks.

A second source of password guesses is default passwords. Many systems are initially configured with default accounts having GUEST or ADMIN as login IDs; accompanying these IDs are well-known passwords such as "guest" or "null" or "password" to enable the administrator to set up the system. Administrators often forget to delete or disable these accounts, or at least to change the passwords.

In a trustworthy environment, such as an office LAN, a password may simply be a signal that the user does not want others to use the workstation or account. Sometimes the password-protected workstation contains sensitive data, such as employee salaries or information about new products. Users may think that the password is enough to keep out a curious colleague; they see no reason to protect against concerted attacks. However, if that trustworthy environment is connected to an untrustworthy wider-area network, all users with simple passwords become easy targets. Indeed, some systems are not originally connected to a wider network, so their users begin in a less exposed situation that clearly changes when the connection occurs.

Dead accounts offer a final source of guessable passwords. To see how, suppose Professor Romine, a faculty member, takes leave for a year to teach at another university. The existing account may reasonably be kept on hold, awaiting the professor's return. But an attacker, reading a university newspaper online, finds out that the user is away. Now the attacker uses social engineering on the system administration ("Hello, this is Professor Romine calling from my temporary office at State University. I haven't used my account for quite a while, but now I need something from it urgently. I have forgotten the password. Can you please reset it to ICECREAM? No? Well, send me a new password by email to my account r1@stateuniv.edu.") Alternatively, the attacker can try several passwords until the password guessing limit is exceeded. The system then locks the account administratively, and the attacker uses a social engineering attack. In all these ways the attacker may succeed in resetting or discovering a password.

Authentication Thwarted by Eavesdropping or Wiretapping

Because of the rise in distributed and client-server computing, some users have access privileges on several connected machines. To protect against arbitrary outsiders using these accesses, authentication is required between hosts. This access can involve the user directly, or it can be done automatically on behalf of the user through a host-to-host authentication protocol. In either case, the account and authentication details of the subject are passed to the destination host. When these details are passed on the network, they are exposed to anyone observing the communication on the network. These same authentication details can be reused by an impersonator until they are changed.

Because transmitting a password in the clear is a significant vulnerability, protocols have been developed so that the password itself never leaves a user's workstation. But, as we have seen in several other places, the details are important.

Authentication Foiled by Avoidance

Obviously, authentication is effective only when it works. A weak or flawed authentication allows access to any system or person who can circumvent the authentication.

In a classic operating system flaw, the buffer for typed characters in a password was of fixed size, counting all characters typed, including backspaces for correction. If a user typed more characters than the buffer would hold, the overflow caused the operating system to bypass password comparison and act as if a correct authentication had been supplied. These flaws or weaknesses can be exploited by anyone seeking access.

Many network hosts, especially those that connect to wide area networks, run variants of Unix System V or BSD Unix. In a local environment, many users are not aware of which networked operating system is in use; still fewer would know of, be capable of, or be interested in exploiting flaws. However, some hackers regularly scan wide area networks for hosts running weak or flawed operating systems. Thus, connection to a wide area network, especially the Internet, exposes these flaws to a wide audience intent on exploiting them.

Nonexistent Authentication

If two computers are used by the same users to store data and run processes and if each has authenticated its users on first access, you might assume that computer-to-computer or local user-to-remote process authentication is unnecessary. These two computers and their users are a trustworthy environment in which the added complexity of repeated authentication seems excessive.

However, this assumption is not valid. To see why, consider the Unix operating system. In Unix, the file *.rhosts* lists trusted hosts and *.rlogin* lists trusted users who are allowed access without authentication. The files are intended to support computer-to-computer connection by users who have already been authenticated at their primary hosts. These "trusted hosts" can also be exploited by outsiders who obtain access to one system through an authentication weakness (such as a guessed password) and then transfer to another system that accepts the authenticity of a user who comes from a system on its trusted list.

Well-Known Authentication

Authentication data should be unique and difficult to guess. But unfortunately, the convenience of one well-known authentication scheme sometimes usurps the protection. For example, one computer manufacturer planned to use the same password to allow its remote maintenance personnel to access any of its computers belonging to any of its customers throughout the world. Fortunately, security experts pointed out the potential danger before that idea was put in place.

The system network management protocol (SNMP) is widely used for remote management of network devices, such as routers and switches, that support no ordinary users. SNMP uses a "community string," essentially a password for the community of devices that can interact with one another. But network devices are designed especially for quick installation with minimal configuration, and many network administrators do not change the default community string installed on a router or switch. This laxity makes these devices on the network perimeter open to many SNMP attacks.

Some vendors still ship computers with one system administration account installed, having a default password. Or the systems come with a demonstration or test account, with no required password. Some administrators fail to change the passwords or delete these accounts.

Trusted Authentication

Finally, authentication can become a problem when identification is delegated to other trusted sources. For instance, a file may indicate who can be trusted on a particular host. Or the authentication mechanism for one system can "vouch for" a user. We noted earlier how the Unix *.rhosts, .rlogin,* and */etc/hosts/equiv* files indicate hosts or users that are trusted on other hosts. While these features are useful to users who have accounts on multiple machines or for network management, maintenance, and operation, they must be used very carefully. Each of them represents a potential hole through which a remote useror a remote attackercan achieve access.

Spoofing

Guessing or otherwise obtaining the network authentication credentials of an entity (a user, an account, a process, a node, a device) permits an attacker to create a full communication under the entity's identity. Impersonation falsely represents a valid entity in a communication. Closely related is **spoofing**, when an attacker falsely carries on one end of a networked interchange. Examples of spoofing are masquerading, session hijacking, and man-in-the-middle attacks.

Masquerade

In a **masquerade** one host pretends to be another. A common example is URL confusion. Domain names can easily be confused, or someone can easily mistype certain names. Thus xyz.com, xyz.org, and xyz.net might be three different organizations, or one bona fide organization (for example, xyz.com) and two masquerade attempts from someone who registered the similar domain names. Names with or without hyphens (coca-cola.com versus cocacola.com) and easily mistyped names (l0pht.com versus lopht.com, or citibank.com versus citybank.com) are candidates for masquerading.

From the attacker's point of view, the fun in masquerading comes before the mask is removed. For example, suppose you want to attack a real bank, First Blue Bank of Chicago. The actual bank has the domain name BlueBank.com, so you register the domain name Blue-Bank.com. Next, you put up a web page at Blue-Bank.com, perhaps using the real Blue Bank logo that you downloaded to make your site look as much as possible like that of the Chicago bank. Finally, you ask people to log in with their name, account number, and password or PIN. (This redirection can occur in many ways. For example, you can pay for a banner ad that links to your site instead of the real bank's, or you can send e-mail to Chicago residents and invite them to visit your site.) After collecting personal data from several bank users, you can drop the connection, pass the connection on to the real Blue Bank, or continue to collect more information. You may even be able to transfer this connection smoothly to an authenticated access to the real Blue Bank so that the user never realizes the deviation. (First Blue Bank would probably win a suit to take ownership of the Blue-Bank.com domain.)

A variation of this attack is called **phishing**. You send an e-mail message, perhaps with the real logo of Blue Bank, and an enticement to click on a link, supposedly to take the victim to the Blue Bank web site. The enticement might be that your victim's account has been suspended or that you offer your victim some money for answering a survey (and need the account number and PIN to be able to credit the money), or some other legitimate-sounding explanation. The link might be to your domain Blue-Bank.com, the link might say *click here* to access your account (where the *click here* link connects to your fraudulent site), or you might use some other trick with the URL to fool your victim, like www.redirect.com/bluebank.com.

In another version of a masquerade, the attacker exploits a flaw in the victim's web server and is able to overwrite the victim's web pages. Although there is some public humiliation at having one's site replaced, perhaps with obscenities or strong messages opposing the nature of the site (for example, a plea for vegetarianism on a slaughterhouse web site), most people would not be fooled by a site displaying a message absolutely contrary to its aims. However, a clever attacker can be more subtle. Instead of differentiating from the real site, the attacker can try to build a false site that resembles the real one, perhaps to obtain sensitive information (names, authentication numbers, credit card numbers) or to induce the user to enter into a real transaction. For example, if one bookseller's site, call it Books-R-Us, were overtaken subtly by another, called Books Depot, the orders may actually be processed, filled, and billed to the naïve users by Books Depot. Test your ability to distinguish phishing sites from real ones at http://survey.mailfrontier.com/survey/quiztest.html.

Phishing is becoming a serious problem, according to a trends report from the Anti-Phishing Working Group [APW05]. This group received over 12,000 complaints each month from March 2005 to March 2006, with the number peaking above 18,000 for March 2006.

Session Hijacking

Session hijacking is intercepting and carrying on a session begun by another entity. Suppose two entities have entered into a session but then a third entity intercepts the traffic and carries on the session in the name of the other. Our example of Books-R-Us could be an instance of this technique. If Books Depot used a wiretap to intercept packets between you and Books-R-Us, Books Depot could simply monitor the information flow, letting Books-R-Us do the hard part of displaying titles for sale and convincing the user to buy. Then, when the user has completed the order, Books Depot intercepts the "I'm ready to check out" packet, and finishes the order with the user, obtaining shipping address, credit card details, and so forth. To Books-R-Us, the transaction would look like any other incomplete transaction: The user was browsing but for some reason decided to go elsewhere before purchasing. We would say that Books Depot had hijacked the session.

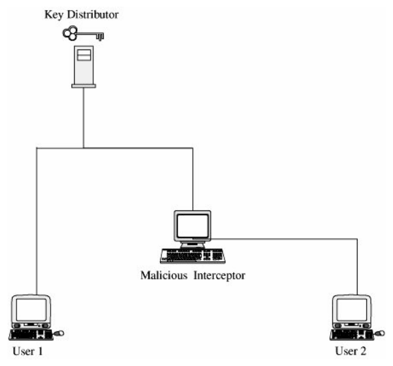
A different type of example involves an interactive session, for example, using Telnet. If a system administrator logs in remotely to a privileged account, a session hijack utility could intrude in the communication and pass commands as if they came from the administrator.

Man-in-the-Middle Attack

Our hijacking example requires a third party involved in a session between two entities. A **man-in-the-middle** attack is a similar form of attack, in which one entity intrudes between two others. We studied one form of this attack in Chapter 3. The difference between man-in-the-middle and hijacking is that a man-in-the-middle usually participates from the start of the session, whereas a session hijacking occurs after a session has been established. The difference is largely semantic and not too significant.

Man-in-the-middle attacks are frequently described in protocols. To see how an attack works, suppose you want to exchange encrypted information with your friend. You contact the key server and ask for a secret key with which to communicate with your friend. The key server responds by sending a key to you and your friend. One man-in-the-middle attack assumes someone can see and enter into all parts of this protocol. A malicious middleman intercepts the response key and can then eavesdrop on, or even decrypt, modify, and reencrypt any subsequent communications between you and your friend. This attack is depicted in Figure 7-15.

Figure 7-15. Key Interception by a Man-in-the-Middle Attack.



This attack would be changed with public keys, because the man-in-the-middle would not have the private key to be able to decrypt messages encrypted under your friend's public key. The man-in-the-middle attack now becomes more of the three-way interchange its name implies. The man-in-the-middle intercepts your request to the key server and instead asks for your friend's public key. The man-in-the-middle passes to you his own public key, not your friend's. You encrypt using the public key you received (from the man-in-the-middle); the man-in-the-middle intercepts and decrypts, reads, and reencrypts, using your friend's public key; and your friend receives. In this way, the man-in-the-middle reads the messages and neither you nor your friend is aware of the interception. A slight variation of this attack works for secret key distribution under a public key.

Denial of Service

So far, we have discussed attacks that lead to failures of confidentiality or integrityproblems we have also seen in the contexts of operating systems, databases, and applications. Availability attacks, sometimes called denial-of-service or DOS attacks, are much more significant in networks than in other contexts. There are many accidental and malicious threats to availability or continued service.

Transmission Failure

Communications fail for many reasons. For instance, a line is cut. Or network noise makes a packet unrecognizable or undeliverable. A machine along the transmission path fails for hardware or software reasons. A device is removed from service for repair or testing. A device is saturated and rejects incoming data until it can clear its overload. Many of these problems are temporary or automatically fixed (circumvented) in major networks, including the Internet.

However, some failures cannot be easily repaired. A break in the single communications line to your computer (for example, from the network to your network interface card or the telephone line to your modem) can be fixed only by establishment of an alternative link or repair of the damaged one. The network administrator will say "service to the rest of the network was unaffected," but that is of little consolation to you.

From a malicious standpoint, you can see that anyone who can sever, interrupt, or overload capacity to you can deny you service. The physical threats are pretty obvious. We consider instead several electronic attacks that can cause a denial of service.

Connection Flooding

The most primitive denial-of-service attack is flooding a connection. If an attacker sends you as much data as your communications system can handle, you are prevented from receiving any other data. Even if an occasional packet reaches you from someone else, communication to you will be seriously degraded.

More sophisticated attacks use elements of Internet protocols. In addition to TCP and UDP, there is a third class of protocols, called **ICMP** or **Internet Control Message Protocols**. Normally used for system diagnostics, these protocols do not have associated user applications. ICMP protocols include

- *ping*, which requests a destination to return a reply, intended to show that the destination system is reachable and functioning
- *echo*, which requests a destination to return the data sent to it, intended to show that the connection link is reliable (ping is actually a version of echo)
- *destination unreachable*, which indicates that a destination address cannot be accessed
- *source quench*, which means that the destination is becoming saturated and the source should suspend sending packets for a while

These protocols have important uses for network management. But they can also be used to attack a system. The protocols are handled within the network stack, so the attacks may be difficult to detect or block on the receiving host. We examine how these protocols can be used to attack a victim.

Echo-Chargen

This attack works between two hosts. Chargen is a protocol that generates a stream of packets; it is used to test the network's capacity. The attacker sets up a chargen process on host A that generates its packets as echo packets with a destination of host B. Then, host A produces a stream of packets to which host B replies by echoing them back to host A. This series puts the network infrastructures of A and B into an endless loop. If the attacker makes B both the source and destination address of the first packet, B hangs in a loop, constantly creating and replying to its own messages.

Ping of Death

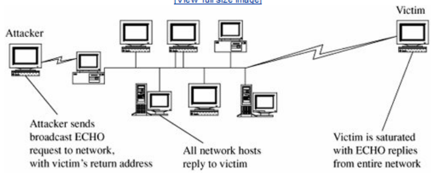
A **ping of death** is a simple attack. Since ping requires the recipient to respond to the ping request, all the attacker needs to do is send a flood of pings to the intended victim. The attack is limited by the smallest bandwidth on the attack route. If the attacker is on a 10-megabyte (MB) connection and the path to the victim is 100 MB or more, the attacker cannot mathematically flood the victim alone. But the attack succeeds if the numbers are reversed: The attacker on a 100-MB connection can easily flood a 10-MB victim. The ping packets will saturate the victim's bandwidth.

Smurf

The **smurf** attack is a variation of a ping attack. It uses the same vehicle, a ping packet, with two extra twists. First, the attacker chooses a network of unwitting victims. The attacker spoofs the source address in the ping packet so that it appears to come from the victim. Then, the attacker sends this request to the network in **broadcast mode** by setting the last byte of the address to all 1s; broadcast mode packets are distributed to all hosts on the network. The attack is shown in Figure 7-16.

Figure 7-16. Smurf Attack.

[View full size image]

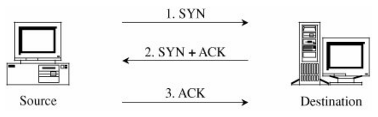

Attacker
Victim
Attacker sends broadcast ECHO request to network, with victim's return address
All network hosts reply to victim
Victim is saturated with ECHO replies from entire network

Syn Flood

Another popular denial-of-service attack is the **syn flood**. This attack uses the TCP protocol suite, making the session-oriented nature of these protocols work against the victim.

For a protocol such as Telnet, the protocol peers establish a virtual connection, called a **session**, to synchronize the back-and-forth, command-response nature of the Telnet terminal emulation. A session is established with a three-way TCP handshake. Each TCP packet has flag bits, two of which are denoted SYN and ACK. To initiate a TCP connection, the originator sends a packet with the SYN bit on. If the recipient is ready to establish a connection, it replies with a packet with both the SYN and ACK bits on. The first party then completes the exchange to demonstrate a clear and complete communication channel by sending a packet with the ACK bit on, as shown in Figure 7-17.

Figure 7-17. Three-Way Connection Handshake



Occasionally packets get lost or damaged in transmission. The destination maintains a queue called the **SYN_RECV** connections, tracking those items for which a SYNACK has been sent but no corresponding ACK has yet been received. Normally, these connections are completed in a short time. If the SYNACK (2) or the ACK (3) packet is lost, eventually the destination host will time out the incomplete connection and discard it from its waiting queue.

The attacker can deny service to the target by sending many SYN requests and never responding with ACKs, thereby filling the victim's SYN_RECV queue. Typically, the SYN_RECV queue is quite small, such as 10 or 20 entries. Because of potential routing delays in the Internet, typical holding times for the SYN_RECV queue can be minutes. So the attacker need only send a new SYN request every few seconds and it will fill the queue.

Attackers using this approach usually do one more thing: They spoof the nonexistent return address in the initial SYN packet. Why? For two reasons. First, the attacker does not want to disclose the real source address in case someone should inspect the packets in the SYN_RECV queue to try to identify the attacker. Second, the attacker wants to make the SYN packets indistinguishable from legitimate SYN packets to establish real connections. Choosing a different (spoofed) source address for each one makes them unique. A SYNACK packet to a nonexistent address results in an ICMP Destination Unreachable response, but this is not the ACK for which the TCP connection is waiting. (Remember that TCP and ICMP are different protocol suites, so an ICMP reply does not necessarily get back to the sender's TCP handler.)

Teardrop

The **teardrop** attack misuses a feature designed to improve network communication. A network IP datagram is a variable-length object. To support different applications and conditions, the datagram protocol permits a single data unit to be fragmented, that is, broken into pieces and transmitted separately. Each fragment indicates its length and relative position within the data unit. The receiving end is responsible for reassembling the fragments into a single data unit.

In the teardrop attack, the attacker sends a series of datagrams that cannot fit together properly. One datagram might say it is position 0 for length 60 bytes, another position 30 for 90 bytes, and another position 41 for 173 bytes. These three pieces overlap, so they cannot be reassembled properly. In an extreme case, the operating system locks up with these partial data units it cannot reassemble, thus leading to denial of service.

For more on these and other denial of service threats, see [CER99 and MAR05].

Traffic Redirection

As we saw earlier, at the network layer, a router is a device that forwards traffic on its way through intermediate networks between a source host's network and a destination's network. So if an attacker can corrupt the routing, traffic can disappear.

Routers use complex algorithms to decide how to route traffic. No matter the algorithm, they essentially seek the best path (where "best" is measured in some combination of distance, time, cost, quality, and the like). Routers are aware only of the routers with which they share a direct network connection, and they use gateway protocols to share information about their capabilities. Each router advises its neighbors about how well it can reach other network addresses. This characteristic allows an attacker to disrupt the network.

To see how, keep in mind that, in spite of its sophistication, a router is simply a computer with two or more network interfaces. Suppose a router advertises to its neighbors that it has the best path to every other address in the whole network. Soon all routers will direct all traffic to that one router. The one router may become flooded, or it may simply drop much of its traffic. In either case, a lot of traffic never makes it to the intended destination.

DNS Attacks

Our final denial-of-service attack is actually a class of attacks based on the concept of domain name server. A **domain name server** (**DNS**) is a table that converts domain names like ATT.COM into network addresses like 211.217.74.130; this process is called resolving the domain name. A domain name server queries other name servers to resolve domain names it does not know. For efficiency, it caches the answers it receives so it can resolve that name more rapidly in the future. A pointer to a DNS server can be retained for weeks or months.

In the most common implementations of Unix, name servers run software called **Berkeley Internet Name Domain** or **BIND** or **named** (a shorthand for "name daemon"). There have been numerous flaws in BIND, including the now-familiar buffer overflow.

By overtaking a name server or causing it to cache spurious entries (called **DNS cache poisoning**), an attacker can redirect the routing of any traffic, with an obvious implication for denial of service.

In October 2002, a massive flood of traffic inundated the top-level domain DNS servers, the servers that form the foundation of the Internet addressing structure. Roughly half the traffic came from just 200 addresses. Although some people think the problem was a set of misconfigured firewalls, nobody knows for sure what caused the attack.

An attack in March 2005 used a flaw in a Symantec firewall to allow a change in the DNS records used on Windows machines. The objective of this attack was not denial of service, however. In this attack, the poisoned DNS cache redirected users to advertising sites that received money from clients each time a user visited the site. Nevertheless, the attack also prevented users from accessing the legitimate sites.

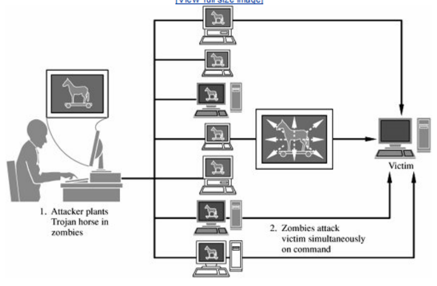
Distributed Denial of Service

The denial-of-service attacks we have listed are powerful by themselves, and Sidebar 7-7 shows us that many are launched. But an attacker can construct a two-stage attack that multiplies the effect many times. This multiplicative effect gives power to distributed denial of service.

To perpetrate a **distributed denial-of-service** (or **DDoS)** attack, an attacker does two things, as illustrated in Figure 7-18. In the first stage, the attacker uses any convenient attack (such as exploiting a buffer overflow or tricking the victim to open and install unknown code from an e-mail attachment) to plant a Trojan horse on a target machine. That Trojan horse does not necessarily cause any harm to the target machine, so it may not be noticed. The Trojan horse file may be named for a popular editor or utility, bound to a standard operating system service, or entered into the list of processes (daemons) activated at startup. No matter how it is situated within the system, it will probably not attract any attention.

Figure 7-18. Distributed Denial-of-Service Attack.

[View full size image]


Victim
1. Attacker plants Trojan horse in zombies
2. Zombies attack victim simultaneously on command

The attacker repeats this process with many targets. Each of these target systems then becomes what is known as a **zombie**. The target systems carry out their normal work, unaware of the resident zombie.

At some point the attacker chooses a victim and sends a signal to all the zombies to launch the attack. Then, instead of the victim's trying to defend against one denial-of-service attack from one malicious host, the victim must try to counter *n* attacks from the *n* zombies all acting at once. Not all of the zombies need to use the same attack; for instance, some could use smurf attacks and others, could use syn floods to address different potential weaknesses.

Network Security Controls:

SSL Encryption

The **SSL (Secure Sockets Layer**) protocol was originally designed by Netscape to protect communication between a web browser and server. It is also known now as **TLS,** for **transport layer security**. SSL interfaces between applications (such as browsers) and the TCP/IP protocols to provide server authentication, optional client authentication, and an encrypted communications channel between client and server. Client and server negotiate a mutually supported suite of encryption for session encryption and hashing; possibilities include triple DES and SHA1, or RC4 with a 128-bit key and MD5.

To use SSL, the client requests an SSL session. The server responds with its public key certificate so that the client can determine the authenticity of the server. The client returns part of a symmetric session key encrypted under the server's public key. Both the server and client compute the session key, and then they switch to encrypted communication, using the shared session key.

The protocol is simple but effective, and it is the most widely used secure communication protocol on the Internet. However, remember that SSL protects only from the client's browser to the server's decryption point (which is often only to the server's firewall or, slightly stronger, to the computer that runs the web application). Data are exposed from the user's keyboard to the browser and throughout the recipient's company. Blue Gem Security has developed a product called LocalSSL that encrypts data after it has been typed until the operating system delivers it to the client's browser, thus thwarting any keylogging Trojan horse that has become implanted in the user's computer to reveal everything the user types.

IPSec

As noted previously, the address space for the Internet is running out. As domain names and equipment proliferate, the original, 30-year-old, 32-bit address structure of the Internet is filling up. A new structure, called **IPv6** (version 6 of the IP protocol suite), solves the addressing problem. This restructuring also offered an excellent opportunity for the Internet Engineering Task Force (IETF) to address serious security requirements.

As a part of the IPv6 suite, the IETF adopted **IPSec**, or the **IP Security Protocol Suite**. Designed to address fundamental shortcomings such as being subject to spoofing, eavesdropping, and session hijacking, the IPSec protocol defines a standard means for handling encrypted data. IPSec is implemented at the IP layer, so it affects all layers above it, in particular TCP and UDP. Therefore, IPSec requires no change to the existing large number of TCP and UDP protocols.

IPSec is somewhat similar to SSL, in that it supports authentication and confidentiality in a way that does not necessitate significant change either above it (in applications) or below it (in the TCP protocols). Like SSL, it was designed to be independent of specific cryptographic protocols and to allow the two communicating parties to agree on a mutually supported set of protocols.

The basis of IPSec is what is called a **security association**, which is essentially the set of security parameters for a secured communication channel. It is roughly comparable to an SSL session. A security association includes

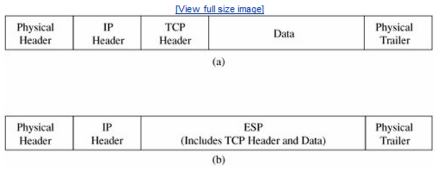
- encryption algorithm and mode (for example, DES in block-chaining mode)
- encryption key
- encryption parameters, such as the initialization vector
- authentication protocol and key
- lifespan of the association, to permit long-running sessions to select a new cryptographic key as often as needed
- address of the opposite end of association
- sensitivity level of protected data (usable for classified data)

A host, such as a network server or a firewall, might have several security associations in effect for concurrent communications with different remote hosts. A security association is selected by a **security parameter index (SPI)**, a data element that is essentially a pointer into a table of security associations.

The fundamental data structures of IPSec are the **AH** (**authentication header**) and the **ESP** (**encapsulated security payload**). The ESP replaces (includes) the conventional TCP header and data portion of a packet, as shown in Figure 7-27. The physical header and trailer depend on the data link and physical layer communications medium, such as Ethernet.

Figure 7-27. Packets: (a) Conventional Packet; (b) IPSec Packet.

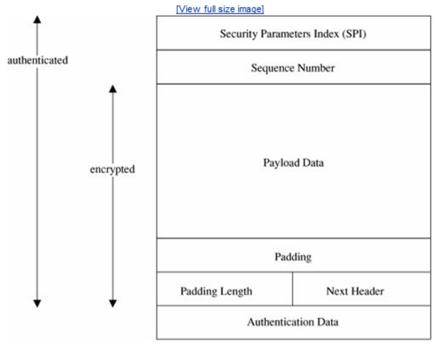
[View full size image]



The ESP contains both an authenticated portion and an encrypted portion, as shown in Figure 7-28. The sequence number is incremented by one for each packet transmitted to the same address using the same SPI, to preclude packet replay attacks. The payload data is the actual data of the packet. Because some encryption or other security mechanisms require blocks of certain sizes, the padding factor and padding length fields contain padding and the amount of padding to bring the payload data to an appropriate length. The next header indicates the type of payload data. The authentication field is used for authentication of the entire object.

Figure 7-28. Encapsulated Security Packet.

[View full size image]



As with most cryptographic applications, the critical element is key management. IPSec addresses this need with **ISAKMP** or **Internet Security Association Key Management Protocol**. Like SSL, ISAKMP requires that a distinct key be generated for each security association. The ISAKMP protocol is simple, flexible, and scalable. In IPSec, ISAKMP is implemented through **IKE** or ISAKMP **key exchange**. IKE provides a way to agree on and manage protocols, algorithms, and keys. For key exchange between unrelated parties IKE uses the DiffieHellman scheme (also described in Chapter 2). In DiffieHellman, each of the two parties, X and Y, chooses a large prime and sends a number g raised to the power of the prime to the other. That is, X sends g^x and Y sends g^y. They both raise what they receive to the power they kept: Y raises g^x to $(g^x)^y$ and X raises g^y to $(g^y)^x$, which are both the same; voilà, they share a secret $(g^x)^y = (g^y)^x$. (The computation is slightly more complicated, being done in a finite field $mod(n)$, so an attacker cannot factor the secret easily.) With their shared secret, the two parties now exchange identities and certificates to authenticate those identities. Finally, they derive a shared cryptographic key and enter a security association.

The key exchange is very efficient: The exchange can be accomplished in two messages, with an optional two more messages for authentication. Because this is a public key method, only two keys are needed for each pair of communicating parties. IKE has submodes for authentication (initiation) and for establishing new keys in an existing security association.

IPSec can establish cryptographic sessions with many purposes, including VPNs, applications, and lower-level network management (such as routing). The protocols of IPSec have been published and extensively scrutinized. Work on the protocols began in 1992. They were first published in 1995, and they were finalized in 1998 (RFCs 24012409) [KEN98].

Honeypots

How do you catch a mouse? You set a trap with bait (food the mouse finds attractive) and catch the mouse after it is lured into the trap. You can catch a computer attacker the same way.

In a very interesting book, Cliff Stoll [STO89] details the story of attracting and monitoring the actions of an attacker. Cheswick [CHE90, CHE02] and Bellovin [BEL92c] tell a similar story. These two cases describe the use of a **honeypot**: a computer system open to attackers.

You put up a honeypot for several reasons:

- to watch what attackers do, in order to learn about new attacks (so that you can strengthen your defenses against these new attacks)
- to lure an attacker to a place in which you may be able to learn enough to identify and stop the attacker
- to provide an attractive but diversionary playground, hoping that the attacker will leave your real system alone

A honeypot has no special features. It is just a computer system or a network segment, loaded with servers and devices and data. It may be protected with a firewall, although you want the attackers to have some access. There may be some monitoring capability, done carefully so that the monitoring is not evident to the attacker.

What Is a Firewall?

A firewall is a device that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network. Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important, which means nonfirewall functions should not be done on the same machine. Because a firewall is executable code, an attacker could compromise that code and execute from the firewall's device. Thus, the fewer pieces of code on the device, the fewer tools the attacker would have by compromising the firewall. Firewall code usually runs on a proprietary or carefully minimized operating system.

The purpose of a firewall is to keep "bad" things outside a protected environment. To accomplish that, firewalls implement a security policy that is specifically designed to address what bad things might happen. For example, the policy might be to prevent any access from outside (while still allowing traffic to pass *from* the inside *to* the outside). Alternatively, the policy might permit accesses only from certain places, from certain users, or for certain activities. Part of the challenge of protecting a network with a firewall is determining which security policy meets the needs of the installation.

People in the firewall community (users, developers, and security experts) disagree about how a firewall should work. In particular, the community is divided about a firewall's default behavior. We can describe the two schools of thought as "that which is not expressly forbidden is permitted" (default permit) and "that which is not expressly permitted is forbidden" (default deny). Users, always interested in new features, prefer the former. Security experts, relying on several decades of experience, strongly counsel the latter. An administrator implementing or configuring a firewall must choose one of the two approaches, although the administrator can often broaden the policy by setting the firewall's parameters.

Design of Firewalls

Remember from Chapter 5 that a reference monitor must be

- always invoked
- tamperproof
- small and simple enough for rigorous analysis

A firewall is a special form of reference monitor. By carefully positioning a firewall within a network, we can ensure that all network accesses that we want to control must pass through it. This restriction meets the "always invoked" condition. A firewall is typically well isolated, making it highly immune to modification. Usually a firewall is implemented on a separate computer, with direct connections only to the outside and inside networks. This isolation is expected to meet the "tamperproof" requirement. And firewall designers strongly recommend keeping the functionality of the firewall simple.

Types of Firewalls

Firewalls have a wide range of capabilities. Types of firewalls include

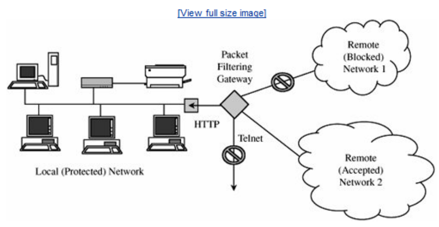
- packet filtering gateways or screening routers
- stateful inspection firewalls
- application proxies
- guards
- personal firewalls

Because a firewall is a type of host, it often is as programmable as a good-quality workstation. While a screening router *can* be fairly primitive, the tendency is to host even routers on complete computers with operating systems because editors and other programming tools assist in configuring and maintaining the router. However, firewall developers are minimalists: They try to eliminate from the firewall all that is not strictly necessary for the firewall's functionality. There is a good reason for this minimal constraint: to give as little assistance as possible to a successful attacker. Thus, firewalls tend not to have user accounts so that, for example, they have no password file to conceal. Indeed, the most desirable firewall is one that runs contentedly in a back room; except for periodic scanning of its audit logs, there is seldom reason to touch it.

Packet Filtering Gateway

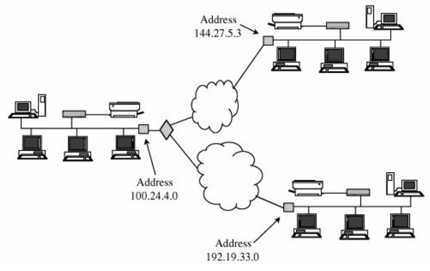
A packet filtering gateway or screening router is the simplest, and in some situations, the most effective type of firewall. A packet filtering gateway controls access to packets on the basis of packet address (source or destination) or specific transport protocol type (such as HTTP web traffic). As described earlier in this chapter, putting ACLs on routers may severely impede their performance. But a separate firewall behind (on the local side) of the router can screen traffic before it gets to the protected network. Figure 7-34 shows a packet filter that blocks access from (or to) addresses in one network; the filter allows HTTP traffic but blocks traffic using the Telnet protocol.

Figure 7-34. Packet Filter Blocking Addresses and Protocols.


[View full size image]
Packet Filtering Gateway
Remote (Blocked) Network 1
HTTP
Telnet
Remote (Accepted) Network 2
Local (Protected) Network

For example, suppose an international company has three LANs at three locations throughout the world, as shown in Figure 7-35. In this example, the router has two sides: inside and outside. We say that the local LAN is on the inside of the router, and the two connections to distant LANs through wide area networks are on the outside. The company might want communication *only* among the three LANs of the corporate network. It could use a screening router on the LAN at 100.24.4.0 to allow *in* only communications destined to the host at 100.24.4.0 and to allow *out* only communications addressed either to address 144.27.5.3 or 192.19.33.0.

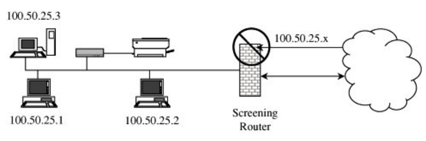
Figure 7-35. Three Connected LANs.



Packet filters do not "see inside" a packet; they block or accept packets solely on the basis of the IP addresses and ports. Thus, any details in the packet's data field (for example, allowing certain Telnet commands while blocking other services) is beyond the capability of a packet filter.

Packet filters can perform the very important service of ensuring the validity of inside addresses. Inside hosts typically trust other inside hosts for all the reasons described as characteristics of LANs. But the only way an inside host can distinguish another inside host is by the address shown in the source field of a message. Source addresses in packets can be forged, so an inside application might think it was communicating with another host on the inside instead of an outside forger. A packet filter sits between the inside network and the outside net, so it can know if a packet from the outside is forging an inside address, as shown in Figure 7-36. A screening packet filter might be configured to block all packets from the *outside* that claimed their source address was an *inside* address. In this example, the packet filter blocks all packets claiming to come from any address of the form 100.50.25.*x* (but, of course, it permits in any packets with *destination* 100.50.25.*x*).

Figure 7-36. Filter Screening Outside Addresses.



The primary disadvantage of packet filtering routers is a combination of simplicity and complexity. The router's inspection is simplistic; to perform sophisticated filtering, the filtering rules set needs to be very detailed. A detailed rules set will be complex and therefore prone to error. For example, blocking all port 23 traffic (Telnet) is simple and straightforward. But if some Telnet traffic is to be allowed, each IP address from which it is allowed must be specified in the rules; in this way, the rule set can become very long.

Stateful Inspection Firewall

Filtering firewalls work on packets one at a time, accepting or rejecting each packet and moving on to the next. They have no concept of "state" or "context" from one packet to the next. A **stateful inspection firewall** maintains state information from one packet to another in the input stream.

One classic approach used by attackers is to break an attack into multiple packets by forcing some packets to have very short lengths so that a firewall cannot detect the signature of an attack split across two or more packets. (Remember that with the TCP protocols, packets can arrive in any order, and the protocol suite is responsible for reassembling the packet stream in proper order before passing it along to the application.) A stateful inspection firewall would track the sequence of packets and conditions from one packet to another to thwart such an attack.

Application Proxy

Packet filters look only at the headers of packets, not at the data *inside* the packets. Therefore, a packet filter would pass anything to port 25, assuming its screening rules allow inbound connections to that port. But applications are complex and sometimes contain errors. Worse, applications (such as the e-mail delivery agent) often act on behalf of all users, so they require privileges of all users (for example, to store incoming mail messages so that inside users can read them). A flawed application, running with all users' privileges, can cause much damage.

An **application proxy gateway**, also called a **bastion host**, is a firewall that simulates the (proper) effects of an application so that the application receives only requests to act properly. A proxy gateway is a two-headed device: It looks to the inside as if it is the outside (destination) connection, while to the outside it responds just as the insider would.

An application proxy runs pseudoapplications. For instance, when electronic mail is transferred to a location, a sending process at one site and a receiving process at the destination communicate by a protocol that establishes the legitimacy of a mail transfer and then actually transfers the mail message. The protocol between sender and destination is carefully defined. A proxy gateway essentially intrudes in the middle of this protocol exchange, seeming like a destination in communication with the sender that is outside the firewall, and seeming like the sender in communication with the real destination on the inside. The proxy in the middle has the opportunity to screen the mail transfer, ensuring that only acceptable e-mail protocol commands are sent to the destination.

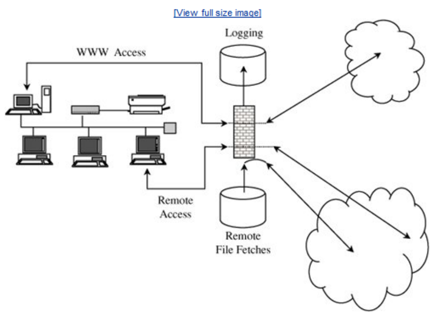
As an example of application proxying, consider the FTP (file transfer) protocol. Specific protocol commands fetch (*get*) files from a remote location, store (*put*) files onto a remote host, list files (*ls*) in a directory on a remote host, and position the process (*cd*) at a particular point in a directory tree on a remote host. Some administrators might want to permit gets but block puts, and to list only certain files or prohibit changing out of a particular directory (so that an outsider could retrieve only files from a prespecified directory). The proxy would simulate both sides of this protocol exchange. For example, the proxy might accept get commands, reject put commands, and filter the local response to a request to list files.

To understand the real purpose of a proxy gateway, let us consider several examples.

- A company wants to set up an online price list so that outsiders can see the products and prices offered. It wants to be sure that (a) no outsider can change the prices or product list and (b) outsiders can access only the price list, not any of the more sensitive files stored inside.
- A school wants to allow its students to retrieve any information from World Wide Web resources on the Internet. To help provide efficient service, the school wants to know what sites have been visited and what files from those sites have been fetched; particularly popular files will be cached locally.
- A government agency wants to respond to queries through a database management system. However, because of inference attacks against databases, the agency wants to restrict queries that return the mean of a set of fewer than five values.
- A company with multiple offices wants to encrypt the data portion of all e-mail to addresses at its other offices. (A corresponding proxy at the remote end will remove the encryption.)
- A company wants to allow dial-in access by its employees, without exposing its company resources to login attacks from remote nonemployees.

Each of these requirements can be met with a proxy. In the first case, the proxy would monitor the file transfer protocol *data* to ensure that only the price list file was accessed, and that file could only be read, not modified. The school's requirement could be met by a logging procedure as part of the web browser. The agency's need could be satisfied by a special-purpose proxy that interacted with the database management system, performing queries but also obtaining the number of values from which the response was computed and adding a random minor error term to results from small sample sizes. The requirement for limited login could be handled by a specially written proxy that required strong user authentication (such as a challengeresponse system), which many operating systems do not require. These functions are shown in Figure 7-37.

Figure 7-37. Actions of Firewall Proxies.



The proxies on the firewall can be tailored to specific requirements, such as logging details about accesses. They can even present a common user interface to what may be dissimilar internal functions. Suppose the internal network has a mixture of operating system types, none of which support strong authentication through a challengeresponse token. The proxy can demand strong authentication (name, password, and challengeresponse), validate the challengeresponse itself, and then pass on only simple name and password authentication details in the form required by a specific internal host's operating system.

The distinction between a proxy and a screening router is that the proxy interprets the protocol stream to an application, to control actions through the firewall on the basis of things visible *within* the protocol, not just on external header data.

Guard

A guard is a sophisticated firewall. Like a proxy firewall, it receives protocol data units, interprets them, and passes through the same or different protocol data units that achieve either the same result or a modified result. The guard decides what services to perform on the user's behalf in accordance with its available knowledge, such as whatever it can reliably know of the (outside) user's identity, previous interactions, and so forth. The degree of control a guard can provide is limited only by what is computable. But guards and proxy firewalls are similar enough that the distinction between them is sometimes fuzzy. That is, we can add functionality to a proxy firewall until it starts to look a lot like a guard.

Guard activities can be quite sophisticated, as illustrated in the following examples:

- A university wants to allow its students to use e-mail up to a limit of so many messages or so many characters of e-mail in the last so many days. Although this result could be achieved by modifying e-mail handlers, it is more easily done by monitoring the common point through which all e-mail flows, the mail transfer protocol.
- A school wants its students to be able to access the World Wide Web but, because of the slow speed of its connection to the web, it will allow only so many characters per downloaded image (that is, allowing text mode and simple graphics, but disallowing complex graphics, animation, music, or the like).
- A library wants to make available certain documents but, to support fair use of copyrighted matter, it will allow a user to retrieve only the first so many characters of a document. After that amount, the library will require the user to pay a fee that will be forwarded to the author.
- A company wants to allow its employees to fetch files via ftp. However, to prevent introduction of viruses, it will first pass all incoming files through a virus scanner. Even though many of these files will be nonexecutable text or graphics, the company administrator thinks that the expense of scanning them (which should pass) will be negligible.

Each of these scenarios can be implemented as a modified proxy. Because the proxy decision is based on some quality of the communication data, we call the proxy a guard. Since the security policy implemented by the guard is somewhat more complex than the action of a proxy, the guard's code is also more complex and therefore more exposed to error. Simpler firewalls have fewer possible ways to fail or be subverted.

Personal Firewalls

Firewalls typically protect a (sub)network of multiple hosts. University students and employees in offices are behind a real firewall. Increasingly, home users, individual workers, and small businesses use cable modems or DSL connections with unlimited, always-on access. These people need a firewall, but a separate firewall computer to protect a single workstation can seem too complex and expensive. These people need a firewall's capabilities at a lower price.

A **personal firewall** is an application program that runs on a workstation to block unwanted traffic, usually from the network. A personal firewall can complement the work of a conventional firewall by screening the kind of data a single host will accept, or it can compensate for the lack of a regular firewall, as in a private DSL or cable modem connection.

Just as a network firewall screens incoming and outgoing traffic for that network, a personal firewall screens traffic on a single workstation. A workstation could be vulnerable to malicious code or malicious active agents (ActiveX controls or Java applets), leakage of personal data stored on the workstation, and vulnerability scans to identify potential weaknesses. Commercial implementations of personal firewalls include Norton Personal Firewall from Symantec, McAfee Personal Firewall, and Zone Alarm from Zone Labs (now owned by CheckPoint).

The personal firewall is configured to enforce some policy. For example, the user may decide that certain sites, such as computers on the company network, are highly trustworthy, but most other sites are not. The user defines a policy permitting download of code, unrestricted data sharing, and management access from the corporate segment, but not from other sites. Personal firewalls can also generate logs of accesses, which can be useful to examine in case something harmful does slip through the firewall.

Combining a virus scanner with a personal firewall is both effective and efficient. Typically, users forget to run virus scanners daily, but they do remember to run them occasionally, such as sometime during the week. However, leaving the virus scanner execution to the user's memory means that the scanner detects a problem only after the factsuch as when a virus has been downloaded in an e-mail attachment. With the combination of a virus scanner and a personal firewall, the firewall directs all incoming e-mail to the virus scanner, which examines every attachment the moment it reaches the target host and before it is opened.

A personal firewall runs on the very computer it is trying to protect. Thus, a clever attacker is likely to attempt an undetected attack that would disable or reconfigure the firewall for the future. Still, especially for cable modem, DSL, and other "always on" connections, the static workstation is a visible and vulnerable target for an ever-present attack community. A personal firewall can provide reasonable protection to clients that are not behind a network firewall.

Comparison of Firewall Types

We can summarize the differences among the several types of firewalls we have studied in depth. The comparisons are shown in Table 7-8.

Table 7-8. Comparison of Firewall Types.

Packet Filtering	Stateful Inspection	Application Proxy	Guard	Personal Firewall
Simplest	More complex	Even more complex	Most complex	Similar to packet filtering firewall
Sees only addresses and service protocol type	Can see either addresses or data	Sees full data portion of packet	Sees full text of communication	Can see full data portion of packet
Auditing difficult	Auditing possible	Can audit activity	Can audit activity	Canand usually doesaudit activity
Screens based on connection rules	Screens based on information across packetsin either header or data field	Screens based on behavior of proxies	Screens based on interpretation of message content	Typically, screens based on information in a single packet, using header or data

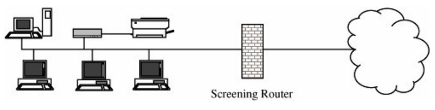
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex addressing rules	Complex guard functionality can limit assurance	Usually starts in "deny all inbound" mode, to which user adds trusted addresses as they appear

Example Firewall Configurations

Let us look at several examples to understand how to use firewalls. We present situations designed to show how a firewall complements a sensible security policy and architecture.

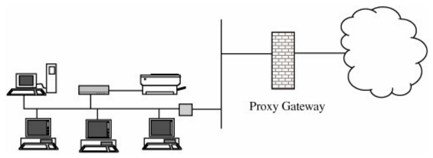
The simplest use of a firewall is shown in Figure 7-38. This environment has a screening router positioned between the internal LAN and the outside network connection. In many cases, this installation is adequate when we need only screen the address of a router.

Figure 7-38. Firewall with Screening Router.



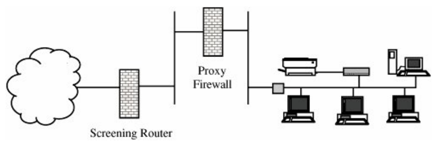
However, to use a proxy machine, this organization is not ideal. Similarly, configuring a router for a complex set of approved or rejected addresses is difficult. If the firewall router is successfully attacked, then all traffic on the LAN to which the firewall is connected is visible. To reduce this exposure, a proxy firewall is often installed on its own LAN, as shown in Figure 7-39. In this way the only traffic visible on that LAN is the traffic going into and out of the firewall.

Figure 7-39. Firewall on Separate LAN.



For even more protection, we can add a screening router to this configuration, as shown in Figure 7-40. Here, the screening router ensures address correctness to the proxy firewall (so that the proxy firewall cannot be fooled by an outside attacker forging an address from an inside host); the proxy firewall filters traffic according to its proxy rules. Also, if the screening router is subverted, only the traffic to the proxy firewall is visiblenot any of the sensitive information on the internal protected LAN.

Figure 7-40. Firewall with Proxy and Screening Router.



Although these examples are simplifications, they show the kinds of configurations firewalls protect. Next, we review the kinds of attacks against which firewalls can and cannot protect.

What Firewalls Canand CannotBlock

As we have seen, firewalls are not complete solutions to all computer security problems. A firewall protects only the perimeter of its environment against attacks from outsiders who want to execute code or access data on the machines in the protected environment. Keep in mind these points about firewalls.

- Firewalls can protect an environment only if the firewalls control the entire perimeter. That is, firewalls are effective only if no unmediated connections breach the perimeter. If even one inside host connects to an outside address, by a modem for example, the entire inside net is vulnerable through the modem and its host.
- Firewalls do not protect data outside the perimeter; data that have properly passed (outbound) through the firewall are just as exposed as if there were no firewall.
- Firewalls are the most visible part of an installation to the outside, so they are the most attractive target for attack. For this reason, several different layers of protection, called **defense in depth**, are better than relying on the strength of just a single firewall.
- Firewalls must be correctly configured, that configuration must be updated as the internal and external environment changes, and firewall activity reports must be reviewed periodically for evidence of attempted or successful intrusion.
- Firewalls are targets for penetrators. While a firewall is designed to withstand attack, it is not impenetrable. Designers intentionally keep a firewall small and simple so that even if a penetrator breaks it, the firewall does not have further tools, such as compilers, linkers, loaders, and the like, to continue an attack.
- Firewalls exercise only minor control over the content admitted to the inside, meaning that inaccurate data or malicious code must be controlled by other means inside the perimeter.

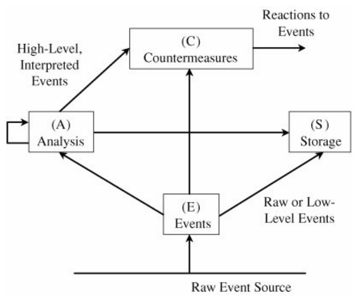
Firewalls are important tools in protecting an environment connected to a network. However, the environment must be viewed as a whole, all possible exposures must be considered, and the firewall must fit into a larger, comprehensive security strategy. Firewalls alone cannot secure an environment.

7.5. Intrusion Detection Systems

After the perimeter controls, firewall, and authentication and access controls block certain actions, some users are admitted to use a computing system. Most of these controls are preventive: They block known bad things from happening. Many studies (for example, see [DUR99]) have shown that most computer security incidents are caused by insiders, people who would not be blocked by a firewall. And insiders require access with significant privileges to do their daily jobs. The vast majority of harm from insiders is not malicious; it is honest people making honest mistakes. Then, too, there are the potential malicious outsiders who have somehow passed the screens of firewalls and access controls. Prevention, although necessary, is not a complete computer security control; detection during an incident copes with harm that cannot be prevented in advance. Halme and Bauer [HAL95] survey the range of controls to address intrusions.

Intrusion detection systems complement these preventive controls as the next line of defense. An **intrusion detection system (IDS)** is a device, typically anothe separate computer, that monitors activity to identify malicious or suspicious events. Kemmerer and Vigna [KEM02] survey the history of IDSs. An IDS is a sensor, like a smoke detector, that raises an alarm if specific things occur. A model of an IDS is shown in Figure 7-41. The components in the figure are the four basic elements of an intrusion detection system, based on the Common Intrusion Detection Framework of [STA96]. An IDS receives raw inputs from sensors. It saves those inputs, analyzes them, and takes some controlling action.

Figure 7-41. Common Components of an Intrusion Detection Framework.



IDSs perform a variety of functions:

- monitoring users and system activity
- auditing system configuration for vulnerabilities and misconfigurations
- assessing the integrity of critical system and data files
- recognizing known attack patterns in system activity
- identifying abnormal activity through statistical analysis
- managing audit trails and highlighting user violation of policy or normal activity
- correcting system configuration errors
- installing and operating traps to record information about intruders

No one IDS performs all of these functions. Let us look more closely at the kinds of IDSs and their use in providing security.

Types of IDSs

The two general types of intrusion detection systems are signature based and heuristic. **Signature-based** intrusion detection systems perform simple pattern-matching and report situations that match a pattern corresponding to a known attack type. **Heuristic** intrusion detection systems, also known as **anomaly based**, build a model of acceptable behavior and flag exceptions to that model; for the future, the administrator can mark a flagged behavior as acceptable so that the heuristic IDS will now treat that previously unclassified behavior as acceptable.

Intrusion detection devices can be network based or host based. A **network-based** IDS is a stand-alone device attached to the network to monitor traffic throughout that network; a **host-based** IDS runs on a single workstation or client or host, to protect that one host.

Early intrusion detection systems (for example, [DEN87b, LUN90a, FOX90, LIE89]) worked after the fact, by reviewing logs of system activity to spot potential misuses that had occurred. The administrator could review the results of the IDS to find and fix weaknesses in the system. Now, however, intrusion detection systems operate in real time (or near real time), watching activity and raising alarms in time for the administrator to take protective action.

Signature-Based Intrusion Detection

A simple signature for a known attack type might describe a series of TCP SYN packets sent to many different ports in succession and at times close to one another, as would be the case for a port scan. An intrusion detection system would probably find nothing unusual in the first SYN, say, to port 80, and then another (from the same source address) to port 25. But as more and more ports receive SYN packets, especially ports that are not open, this pattern reflects a possible port scan. Similarly, some implementations of the protocol stack fail if they receive an ICMP packet with a data length of 65535 bytes, so such a packet would be a pattern for which to watch.

The problem with signature-based detection is the signatures themselves. An attacker will try to modify a basic attack in such a way that it will not match the known signature of that attack. For example, the attacker may convert lowercase to uppercase letters or convert a symbol such as "blank space" to its character code equivalent %20. The IDS must necessarily work from a canonical form of the data stream in order to recognize that %20 matches a pattern with a blank space. The attacker may insert malformed packets that the IDS will see, to intentionally cause a pattern mismatch; the protocol handler stack will discard the packets because of the malformation. Each of these variations could be detected by an IDS, but more signatures require additional work for the IDS, which reduces performance.

Of course, signature-based IDSs cannot detect a new attack for which a signature is not yet installed in the database. Every attack type starts as a new pattern at some time, and the IDS is helpless to warn of its existence.

Signature-based intrusion detection systems tend to use **statistical analysis**. This approach uses statistical tools both to obtain sample measurements of key indicators (such as amount of external activity, number of active processes, number of transactions) and to determine whether the collected measurements fit the predetermined attack signatures.

Ideally, signatures should match every instance of an attack, match subtle variations of the attack, but not match traffic that is not part of an attack. However, this goal is grand but unreachable.

Heuristic Intrusion Detection

Because signatures are limited to specific, known attack patterns, another form of intrusion detection becomes useful. Instead of looking for matches, heuristic intrusion detection looks for behavior that is out of the ordinary. The original work in this area (for example, [TEN90]) focused on the individual, trying to find characteristics of that person that might be helpful in understanding normal and abnormal behavior. For example, one user might always start the day by reading e-mail, write many documents using a word processor, and occasionally back up files. These actions would be normal. This user does not seem to use many administrator utilities. If that person tried to access sensitive system management utilities, this new behavior might be a clue that someone else was acting under the user's identity.

If we think of a compromised system in use, it starts clean, with no intrusion, and it ends dirty, fully compromised. There may be no point in the trace of use in which the system changed from clean to dirty; it was more likely that little dirty events occurred, occasionally at first and then increasing as the system became more deeply compromised. Any one of those events might be acceptable by itself, but the accumulation of them and the order and speed at which they occurred could have been signals that something unacceptable was happening. The inference engine of an intrusion detection system performs continuous analysis of the system, raising an alert when the system's dirtiness exceeds a threshold.

Inference engines work in two ways. Some, called **state-based** intrusion detection systems, see the system going through changes of overall state or configuration. They try to detect when the system has veered into unsafe modes. Others try to map current activity onto a model of unacceptable activity and raise an alarm when the activity resembles the model. These are called **model-based** intrusion detection systems. This approach has been extended to networks in [MUK94]. Later work (for example, [FOR96, LIN99]) sought to build a dynamic model of behavior, to accommodate variation and evolution in a person's actions over time. The technique compares real activity with a known representation of normality.

Alternatively, intrusion detection can work from a model of known bad activity. For example, except for a few utilities (login, change password, create user), any other attempt to access a password file is suspect. This form of intrusion detection is known as **misuse intrusion detection**. In this work, the real activity is compared against a known suspicious area.

All heuristic intrusion detection activity is classified in one of three categories: good/benign, suspicious, or unknown. Over time, specific kinds of actions can move from one of these categories to another, corresponding to the IDS's learning whether certain actions are acceptable or not.

As with pattern-matching, heuristic intrusion detection is limited by the amount of information the system has seen (to classify actions into the right category) and how well the current actions fit into one of these categories.

Goals for Intrusion Detection Systems

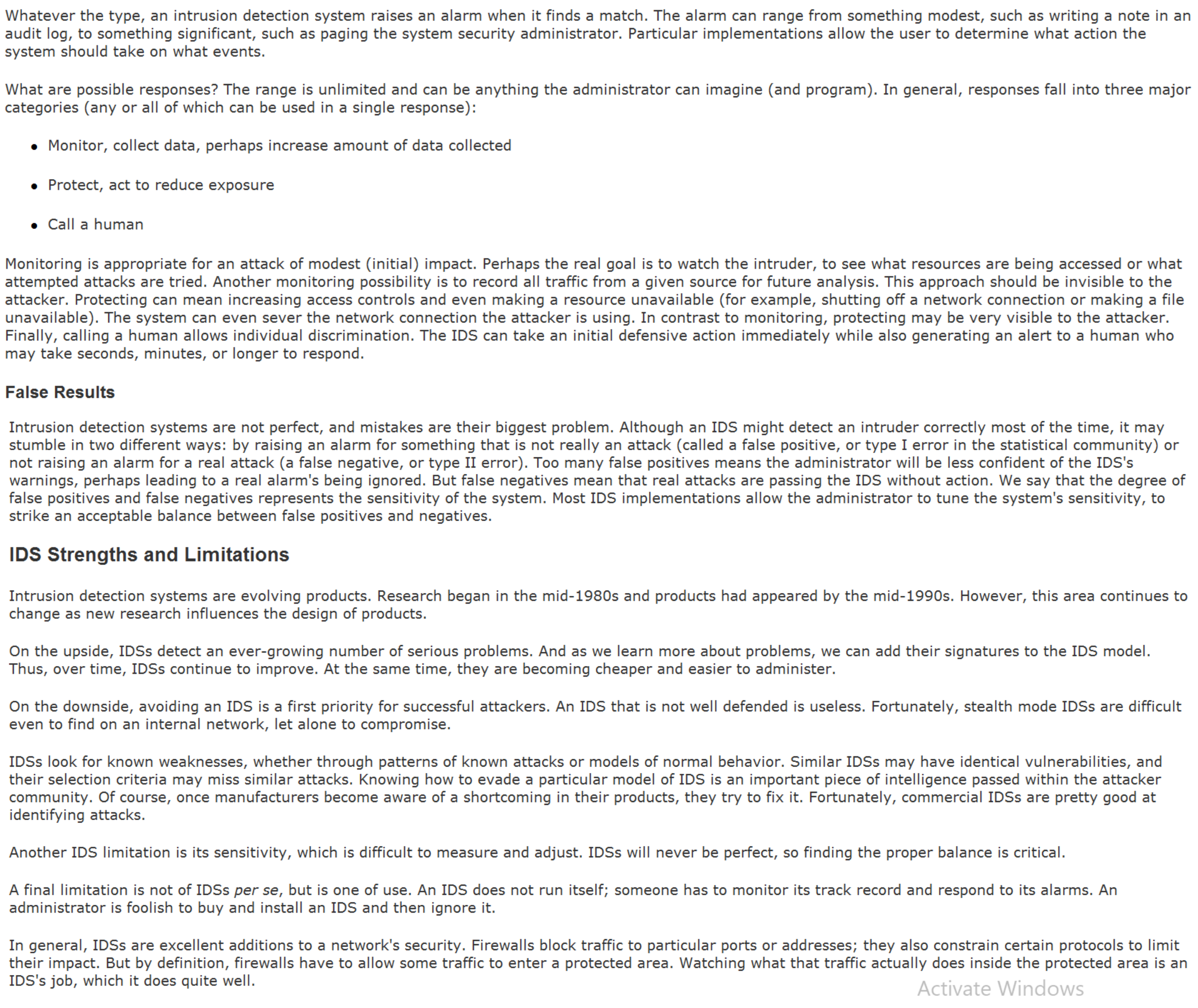
The two styles of intrusion detectionpattern matching and heuristicrepresent different approaches, each of which has advantages and disadvantages. Actual IDS products often blend the two approaches.

Ideally, an IDS should be fast, simple, and accurate, while at the same time being complete. It should detect all attacks with little performance penalty. An IDS could use someor allof the following design approaches:

- Filter on packet headers
- Filter on packet content
- Maintain connection state
- Use complex, multipacket signatures
- Use minimal number of signatures with maximum effect
- Filter in real time, online
- Hide its presence
- Use optimal sliding time window size to match signatures

Responding to Alarms

Whatever the type, an intrusion detection system raises an alarm when it finds a match. The alarm can range from something modest, such as writing a note in an audit log, to something significant, such as paging the system security administrator. Particular implementations allow the user to determine what action the system should take on what events.



Whatever the type, an intrusion detection system raises an alarm when it finds a match. The alarm can range from something modest, such as writing a note in an audit log, to something significant, such as paging the system security administrator. Particular implementations allow the user to determine what action the system should take on what events.

What are possible responses? The range is unlimited and can be anything the administrator can imagine (and program). In general, responses fall into three major categories (any or all of which can be used in a single response):

- Monitor, collect data, perhaps increase amount of data collected
- Protect, act to reduce exposure
- Call a human

Monitoring is appropriate for an attack of modest (initial) impact. Perhaps the real goal is to watch the intruder, to see what resources are being accessed or what attempted attacks are tried. Another monitoring possibility is to record all traffic from a given source for future analysis. This approach should be invisible to the attacker. Protecting can mean increasing access controls and even making a resource unavailable (for example, shutting off a network connection or making a file unavailable). The system can even sever the network connection the attacker is using. In contrast to monitoring, protecting may be very visible to the attacker. Finally, calling a human allows individual discrimination. The IDS can take an initial defensive action immediately while also generating an alert to a human who may take seconds, minutes, or longer to respond.

False Results

Intrusion detection systems are not perfect, and mistakes are their biggest problem. Although an IDS might detect an intruder correctly most of the time, it may stumble in two different ways: by raising an alarm for something that is not really an attack (called a false positive, or type I error in the statistical community) or not raising an alarm for a real attack (a false negative, or type II error). Too many false positives means the administrator will be less confident of the IDS's warnings, perhaps leading to a real alarm's being ignored. But false negatives mean that real attacks are passing the IDS without action. We say that the degree of false positives and false negatives represents the sensitivity of the system. Most IDS implementations allow the administrator to tune the system's sensitivity, to strike an acceptable balance between false positives and negatives.

IDS Strengths and Limitations

Intrusion detection systems are evolving products. Research began in the mid-1980s and products had appeared by the mid-1990s. However, this area continues to change as new research influences the design of products.

On the upside, IDSs detect an ever-growing number of serious problems. And as we learn more about problems, we can add their signatures to the IDS model. Thus, over time, IDSs continue to improve. At the same time, they are becoming cheaper and easier to administer.

On the downside, avoiding an IDS is a first priority for successful attackers. An IDS that is not well defended is useless. Fortunately, stealth mode IDSs are difficult even to find on an internal network, let alone to compromise.

IDSs look for known weaknesses, whether through patterns of known attacks or models of normal behavior. Similar IDSs may have identical vulnerabilities, and their selection criteria may miss similar attacks. Knowing how to evade a particular model of IDS is an important piece of intelligence passed within the attacker community. Of course, once manufacturers become aware of a shortcoming in their products, they try to fix it. Fortunately, commercial IDSs are pretty good at identifying attacks.

Another IDS limitation is its sensitivity, which is difficult to measure and adjust. IDSs will never be perfect, so finding the proper balance is critical.

A final limitation is not of IDSs *per se*, but is one of use. An IDS does not run itself; someone has to monitor its track record and respond to its alarms. An administrator is foolish to buy and install an IDS and then ignore it.

In general, IDSs are excellent additions to a network's security. Firewalls block traffic to particular ports or addresses; they also constrain certain protocols to limit their impact. But by definition, firewalls have to allow some traffic to enter a protected area. Watching what that traffic actually does inside the protected area is an IDS's job, which it does quite well.

References:

1. Behrouz Forouzan, “Cryptography & Network Security”

2. Charles P. Pfleeger," Security in Computing ", Pearson Education

Short Answer Type Questions:

1. What makes a network vulnerable?
2. Explain the various threats in transit in a network.
3. What makes the network vulnerable?
4. List the various TCP/IP vulnerabilities.
5. Define Packet Sniffing & Port Scanning.
6. Explain IP Spoofing.
7. What is ARP Spoofing?
8. Explain TCP SYN Flood.
9. What is DNS Spoofing?
10. List the Classic DOS Attacks.
11. What is DDOS?
12. How SSL works?
13. What is a Firewall? What are it's functions?
14. How is IDS different from Firewall?
15. Define Honeypots.

Long Answer Type Questions:

1. Explain the various threats & vulnerabilities in a network.
2. Explain in detail PGP protocol in detail for securing e-mail communications.
3. Explain DOS attacks in detail. What is DDOS? How can DOS attacks be mitigated?
4. Explain TCP/IP vulnerabilities in detail.
5. Explain IP Spoofing & it's types in detail.
6. What is the difference between spoofing & sniffing?
7. Explain SSL protocol in detail.
8. Explain Firewall & it's types in detail.
9. Explain IDS & it's types in detail.
10. Compare Firewall & IDS.
11. How honeypots help in mitigating attacks?
12. Explain in detail what Firewalls can & cannot block?
13. What are the strengths & limitations of IDS?

---*---**