Security

The Security Problem

- Some of the most common types of ***violations*** include:
    - ***Breach of Confidentiality* -** Theft of private or confidential information, such as credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.
    - ***Breach of Integrity* -** Unauthorized ***modification*** of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.
    - ***Breach of Availability* -** Unauthorized ***destruction*** of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.
    - ***Theft of Service* -** Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.
    - ***Denial of Service, DOS* -** Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.
- One common attack is ***masquerading,*** in which the attacker pretends to be a trusted third party. A variation of this is the ***man-in-the-middle,*** in which the attacker masquerades as both ends of the conversation to two targets.
- A ***replay attack*** involves repeating a valid transmission. Sometimes this can be the entire attack, (such as repeating a request for a money transfer), or other times the content of the original message is replaced with malicious content.

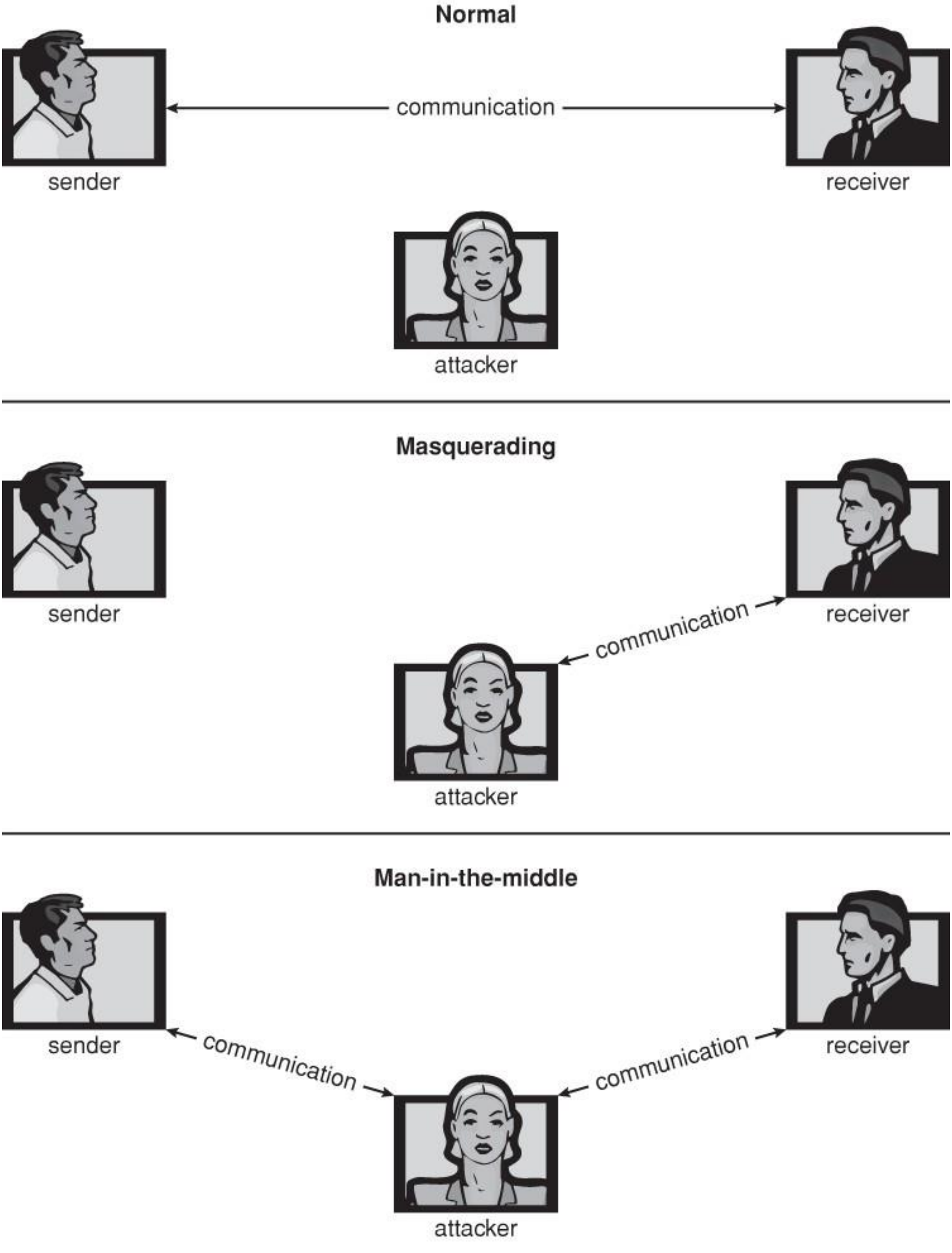

Figure 15.1 - Standard security attacks.

- There are four levels at which a system must be protected:
  1. **Physical -** The easiest way to steal data is to pocket the backup tapes. Also, access to the root console will often give the user special privileges, such as rebooting the system as root from removable media. Even general access to terminals in a computer room offers some opportunities for an attacker, although today's modern high-speed networking environment provides more and more opportunities for remote attacks.
  2. **Human -** There is some concern that the humans who are allowed access to a system be trustworthy, and that they cannot be coerced into breaching security. However more and more attacks today are made via ***social engineering,*** which basically means fooling trustworthy people into accidentally breaching security.

     - **Phishing** involves sending an innocent-looking e-mail or web site designed to fool people into revealing confidential information. E.g. spam e-mails pretending to be from e-Bay, PayPal, or any of a number of banks or credit-card companies.
     - **Dumpster Diving** involves searching the trash or other locations for passwords that are written down. (Note: Passwords that are too hard to remember, or which must be changed frequently are more likely to be written down somewhere close to the user's station.)
     - **Password Cracking** involves divining users passwords, either by watching them type in their passwords, knowing something about them like their pet's names, or simply trying all words in common dictionaries. (Note: "Good" passwords should involve a minimum number of characters, include non-alphabetical characters, and not appear in any dictionary (in any language), and should be changed frequently. Note also that it is proper etiquette to look away from the keyboard while someone else is entering their password.)
  3. **Operating System -** The OS must protect itself from security breaches, such as runaway processes (denial of service), memory-access violations, stack overflow violations, the launching of programs with excessive privileges, and many others.
  4. **Network -** As network communications become ever more important and pervasive in modern computing environments, it becomes ever more important to protect this area of the system. (Both protecting the network itself from attack, and protecting the local system from attacks coming in through the network.) This is a growing area of concern as wireless communications and portable devices become more and more prevalent.

Program Threats

- There are many common threats to modern systems. Only a few are discussed here.

Trojan Horse

- A ***Trojan Horse*** is a program that secretly performs some maliciousness in addition to its visible actions.
- Some Trojan horses are deliberately written as such, and others are the result of legitimate programs that have become infected with ***viruses,*** (see below.)
- One dangerous opening for Trojan horses is long search paths, and in particular paths which include the current directory (".") as part of the path. If a dangerous program having the same name as a legitimate program (or a common mis-spelling, such as "sl" instead of "ls") is placed anywhere on the path, then an unsuspecting user may be fooled into running the wrong program by mistake.
- Another classic Trojan Horse is a login emulator, which records a users account name and password, issues a "password incorrect" message, and then logs off the system. The user then tries again (with a proper login prompt), logs in successfully, and doesn't realize that their information has been stolen.
- Two solutions to Trojan Horses are to have the system print usage statistics on logouts, and to require the typing of non-trappable key sequences such as Control-Alt-Delete in order to log in. (This is why modern Windows systems require the Control-Alt-Delete sequence to commence logging in, which cannot be emulated or caught by ordinary programs. I.e. that key sequence always transfers control over to the operating system.)
- ***Spyware*** is a version of a Trojan Horse that is often included in "free" software downloaded off the Internet. Spyware programs generate pop-up browser windows, and may also accumulate information about the user and deliver it to some central site. (This is an example of ***covert channels,*** in which surreptitious communications occur.) Another common task of spyware is to send out spam e-mail messages, which then purportedly come from the infected user.

Trap Door

- A ***Trap Door*** is when a designer or a programmer (or hacker) deliberately inserts a security hole that they can use later to access the system.
- Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never be trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door.
- A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.

Logic Bomb

- **A *Logic Bomb* is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event.**
- A classic example is the ***Dead-Man Switch***, which is designed to check whether a certain person (e.g. the author) is logging in every day, and if they don't log in for a long time (presumably because they've been fired), then the logic bomb goes off and either opens up security holes or causes other problems.

Stack and Buffer Overflow

- This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow. Consider what happens in the following code, for example, if argv[1] exceeds 256 characters:
    - The strcpy command will overflow the buffer, overwriting adjacent areas of memory.
    - (The problem could be avoided using str***n***cpy, with a limit of 255 characters copied plus room for the null byte.)

```
#include
#define BUFFER_SIZE 256

int main( int argc, char * argv[ ] )
{
    char buffer[ BUFFER_SIZE ];

    if( argc < 2 )
        return -1;
    else {
        strcpy( buffer, argv[ 1 ] );
        return 0;
    }
}
```

Figure - C program with buffer-overflow condition.

- So how does overflowing the buffer cause a security breach? Well the first step is to understand the structure of the stack in memory:
    - The "bottom" of the stack is actually at a high memory address, and the stack grows towards lower addresses.
    - However the address of an array is the lowest address of the array, and higher array elements extend to higher addresses. (I.e. an array "grows" towards the bottom of the stack.
    - In particular, writing past the top of an array, as occurs when a buffer overflows with too much input data, can eventually overwrite the return address, effectively changing where the program jumps to when it returns.

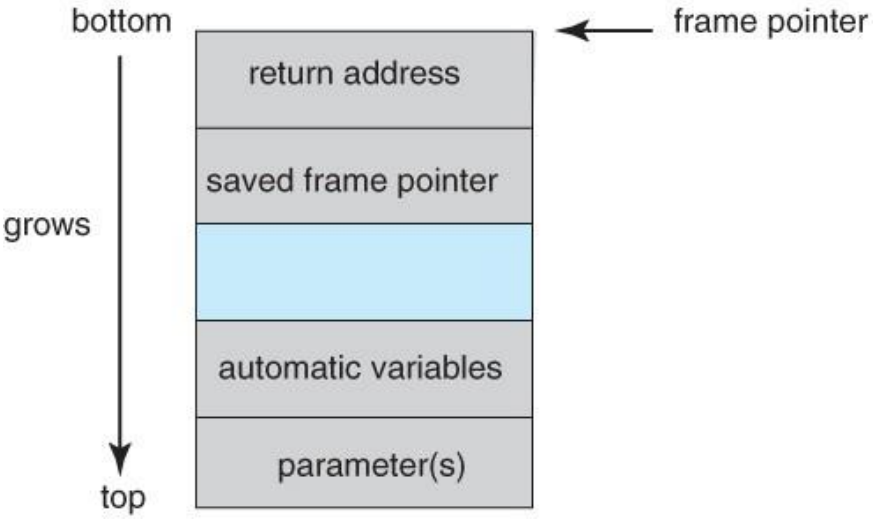

Figure 15.3 - The layout for a typical stack frame.

- Now that we know how to change where the program returns to by overflowing the buffer, the second step is to insert some nefarious code, and then get the program to jump to our inserted code.
- Our only opportunity to enter code is via the input into the buffer, which means there isn't room for very much. One of the simplest and most obvious approaches is to insert the code for "exec(/bin/sh)". To do this requires compiling a program that contains this instruction, and then using an assembler or debugging tool to extract the minimum extent that includes the necessary instructions.

- The bad code is then padded with as many extra bytes as are needed to overflow the buffer to the correct extent, and the address of the buffer inserted into the return address location. (Note, however, that neither the bad code or the padding can contain null bytes, which would terminate the strcpy.)
- The resulting block of information is provided as "input", copied into the buffer by the original program, and then the return statement causes control to jump to the location of the buffer and start executing the code to launch a shell.

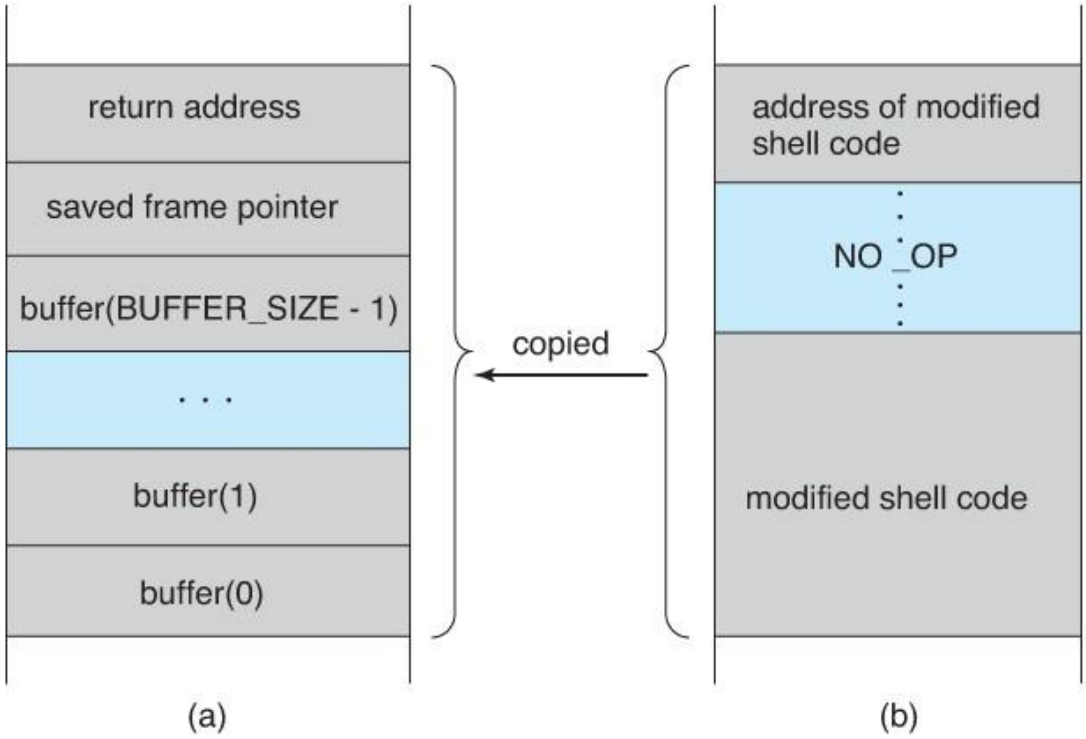

Figure 15.4 - Hypothetical stack frame for Figure 15.2, (a) before and (b) after.

- Unfortunately famous hacks such as the buffer overflow attack are well published and well known, and it doesn't take a lot of skill to follow the instructions and start attacking lots of systems until the law of averages eventually works out. (***Script Kiddies*** are those hackers with only rudimentary skills of their own but the ability to copy the efforts of others.)
- Fortunately modern hardware now includes a bit in the page tables to mark certain pages as non-executable. In this case the buffer-overflow attack would work up to a point, but as soon as it "returns" to an address in the data space and tries executing statements there, an exception would be thrown crashing the program.
- (More details about stack-overflow attacks are available on-line from http://www.insecure.org/stf/smashstack.txt)

Viruses

- A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc.
- Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures (such as the boot block.)
- Viruses are delivered to systems in a ***virus dropper,*** usually some form of a Trojan Horse, and usually via e-mail or unsafe downloads.
- Viruses take many forms (see below.) Figure 15.5 shows typical operation of a boot sector virus:

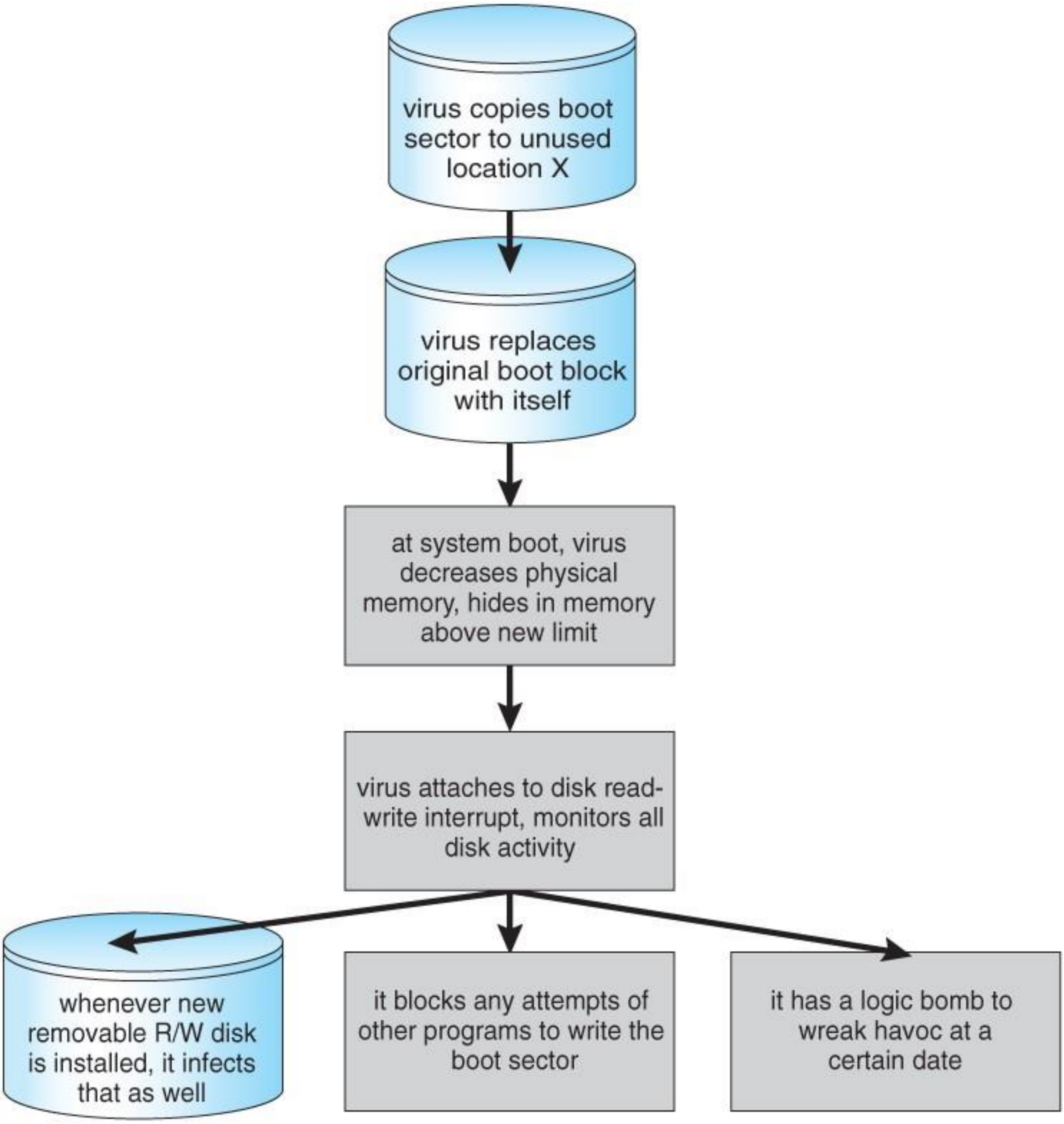
virus copies boot sector to unused location X
virus replaces original boot block with itself
at system boot, virus decreases physical memory, hides in memory above new limit
virus attaches to disk read-write interrupt, monitors all disk activity
whenever new removable R/W disk is installed, it infects that as well
it blocks any attempts of other programs to write the boot sector
it has a logic bomb to wreak havoc at a certain date

Figure 15.5 - A boot-sector computer virus.

- Some of the forms of viruses include:
    - **File -** A file virus attaches itself to an executable file, causing it to run the virus code first and then jump to the start of the original program. These viruses are termed ***parasitic,*** because they do not leave any new files on the system, and the original program is still fully functional.
    - **Boot -** A boot virus occupies the boot sector, and runs before the OS is loaded. These are also known as ***memory viruses***, because in operation they reside in memory, and do not appear in the file system.
    - **Macro -** These viruses exist as a macro (script) that are run automatically by certain macro-capable programs such as MS Word or Excel. These viruses can exist in word processing documents or spreadsheet files.
    - **Source code** viruses look for source code and infect it in order to spread.
    - **Polymorphic** viruses change every time they spread - Not their underlying functionality, but just their ***signature,*** by which virus checkers recognize them.
    - **Encrypted** viruses travel in encrypted form to escape detection. In practice they are self-decrypting, which then allows them to infect other files.
    - **Stealth** viruses try to avoid detection by modifying parts of the system that could be used to detect it. For example the read( ) system call could be modified so that if an infected file is read the infected part gets skipped and the reader would see the original unadulterated file.

    - **Tunneling** viruses attempt to avoid detection by inserting themselves into the interrupt handler chain, or into device drivers.
    - **Multipartite** viruses attack multiple parts of the system, such as files, boot sector, and memory.
    - **Armored** viruses are coded to make them hard for anti-virus researchers to decode and understand. In addition many files associated with viruses are hidden, protected, or given innocuous looking names such as "...".
- In 2004 a virus exploited three bugs in Microsoft products to infect hundreds of Windows servers (including many trusted sites) running Microsoft Internet Information Server, which in turn infected any Microsoft Internet Explorer web browser that visited any of the infected server sites. One of the back-door programs it installed was a ***keystroke logger,*** which records users keystrokes, including passwords and other sensitive information.
- There is some debate in the computing community as to whether a ***monoculture,*** in which nearly all systems run the same hardware, operating system, and applications, increases the threat of viruses and the potential for harm caused by them.

System and Network Threats

- Most of the threats described above are termed ***program threats***, because they attack specific programs or are carried and distributed in programs. The threats in this section attack the operating system or the network itself, or leverage those systems to launch their attacks.

Worms

- A ***worm*** is a process that uses the fork / spawn process to make copies of itself in order to wreak havoc on a system. Worms consume system resources, often blocking out other, legitimate processes. Worms that propagate over networks can be especially problematic, as they can tie up vast amounts of network resources and bring down large-scale systems.
- One of the most well-known worms was launched by Robert Morris, a graduate student at Cornell, in November 1988. Targeting Sun and VAX computers running BSD UNIX version 4, the worm spanned the Internet in a matter of a few hours, and consumed enough resources to bring down many systems.
- This worm consisted of two parts:
    1. A small program called a ***grappling hook,*** which was deposited on the target system through one of three vulnerabilities, and
    2. The main worm program, which was transferred onto the target system and launched by the grappling hook program.

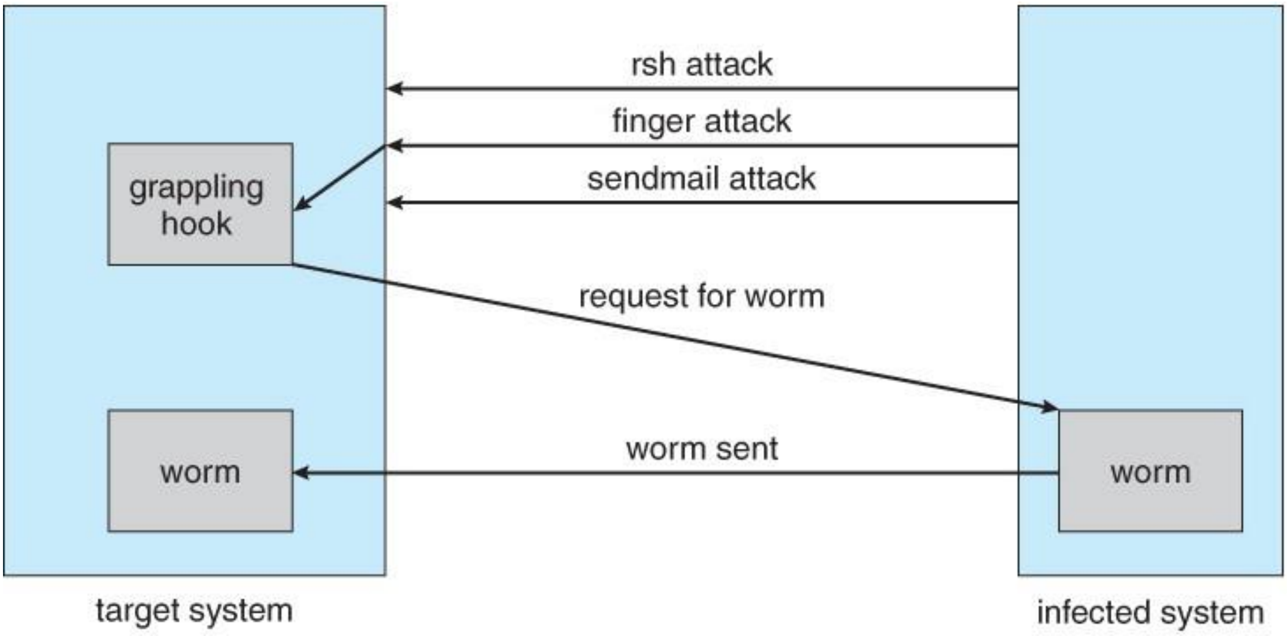

Figure 15.6 - The Morris Internet worm.

- The three vulnerabilities exploited by the Morris Internet worm were as follows:
    1. **rsh (remote shell)** is a utility that was in common use at that time for accessing remote systems without having to provide a password. If a user had an account on two different computers (with the same account name on both systems), then the system could be configured to allow that user to remotely connect from one system to the other without having to provide a password. Many

systems were configured so that ***any*** user (except root) on system A could access the same account on system B without providing a password.
2. **finger** is a utility that allows one to remotely query a user database, to find the true name and other information for a given account name on a given system. For example "finger joeUser@somemachine.edu" would access the finger daemon at somemachine.edu and return information regarding joeUser. Unfortunately the finger daemon (which ran with system privileges) had the buffer overflow problem, so by sending a special 536-character user name the worm was able to fork a shell on the remote system running with root privileges.
3. **sendmail** is a routine for sending and forwarding mail that also included a debugging option for verifying and testing the system. The debug feature was convenient for administrators, and was often left turned on. The Morris worm exploited the debugger to mail and execute a copy of the grappling hook program on the remote system.

- Once in place, the worm undertook systematic attacks to discover user passwords:
  1. First it would check for accounts for which the account name and the password were the same, such as "guest", "guest".
  2. Then it would try an internal dictionary of 432 favorite password choices. (I'm sure "password", "pass", and blank passwords were all on the list.)
  3. Finally it would try every word in the standard UNIX on-line dictionary to try and break into user accounts.
- Once it had gotten access to one or more user accounts, then it would attempt to use those accounts to rsh to other systems, and continue the process.
- With each new access the worm would check for already running copies of itself, and 6 out of 7 times if it found one it would stop. (The seventh was to prevent the worm from being stopped by fake copies.)
- Fortunately the same rapid network connectivity that allowed the worm to propagate so quickly also quickly led to its demise - Within 24 hours remedies for stopping the worm propagated through the Internet from administrator to administrator, and the worm was quickly shut down.
- There is some debate about whether Mr. Morris's actions were a harmless prank or research project that got out of hand or a deliberate and malicious attack on the Internet. However the court system convicted him, and penalized him heavy fines and court costs.
- There have since been many other worm attacks, including the W32.Sobig.F@mm attack which infected hundreds of thousands of computers and an estimated 1 in 17 e-mails in August 2003. This worm made detection difficult by varying the subject line of the infection-carrying mail message, including "Thank You!", "Your details", and "Re: Approved".

Port Scanning

- ***Port Scanning*** is technically not an attack, but rather a search for vulnerabilities to attack. The basic idea is to systematically attempt to connect to every known (or common or possible) network port on some remote machine, and to attempt to make contact. Once it is determined that a particular computer is listening to a particular port, then the next step is to determine what daemon is listening, and whether or not it is a version containing a known security flaw that can be exploited.
- Because port scanning is easily detected and traced, it is usually launched from ***zombie systems,*** i.e. previously hacked systems that are being used without the knowledge or permission of their rightful owner. For this reason it is important to protect "innocuous" systems and accounts as well as those that contain sensitive information or special privileges.
- There are also port scanners available that administrators can use to check their own systems, which report any weaknesses found but which do not exploit the weaknesses or cause any problems. Two such systems are ***nmap*** (http://www.insecure.org/nmap) and ***nessus*** (http://www.nessus.org). The former identifies what OS is found, what firewalls are in place, and what services are listening to what ports. The latter also contains a database of known security holes, and identifies any that it finds.

Denial of Service

- ***Denial of Service (DOS)*** attacks do not attempt to actually access or damage systems, but merely to clog them up so badly that they cannot be used for any useful work. Tight loops that repeatedly request system services are an obvious form of this attack.
- DOS attacks can also involve social engineering, such as the Internet chain letters that say "send this immediately to 10 of your friends, and then go to a certain URL", which clogs up not only the Internet mail system but also the web server to which everyone is directed. (Note: Sending a "reply all" to such a message notifying everyone that it was just a hoax also clogs up the Internet mail service, just as effectively as if you had forwarded the thing.)
- Security systems that lock accounts after a certain number of failed login attempts are subject to DOS attacks which repeatedly attempt logins to all accounts with invalid passwords strictly in order to lock up all accounts.
- Sometimes DOS is not the result of deliberate maliciousness. Consider for example:

    - A web site that sees a huge volume of hits as a result of a successful advertising campaign.
    - CNN.com occasionally gets overwhelmed on big news days, such as Sept 11, 2001.
    - CS students given their first programming assignment involving fork( ) often quickly fill up process tables or otherwise completely consume system resources. :-)
    - (Please use ipcs and ipcrm when working on the inter-process communications assignment !)

Cryptography as a Security Tool

- Within a given computer the transmittal of messages is safe, reliable and secure, because the OS knows exactly where each one is coming from and where it is going.
- On a network, however, things aren't so straightforward - A rogue computer (or e-mail sender) may spoof their identity, and outgoing packets are delivered to a lot of other computers besides their (intended) final destination, which brings up two big questions of security:
    - **Trust -** How can the system be sure that the messages received are really from the source that they say they are, and can that source be trusted?
    - **Confidentiality -** How can one ensure that the messages one is sending are received only by the intended recipient?
- Cryptography can help with both of these problems, through a system of **secrets** and **keys.** In the former case, the key is held by the sender, so that the recipient knows that only the authentic author could have sent the message; In the latter, the key is held by the recipient, so that only the intended recipient can receive the message accurately.
- Keys are designed so that they cannot be divined from any public information, and must be guarded carefully. (***Asymmetric encryption*** involve both a public and a private key.)

Encryption

- The basic idea of encryption is to encode a message so that only the desired recipient can decode and read it. Encryption has been around since before the days of Caesar, and is an entire field of study in itself. Only some of the more significant computer encryption schemes will be covered here.
- The basic process of encryption is shown in Figure 15.7, and will form the basis of most of our discussion on encryption. The steps in the procedure and some of the key terminology are as follows:
    1. The **sender** first creates a **message, m** in plaintext.
    2. The message is then entered into an **encryption algorithm, E,** along with the **encryption key, Ke.**
    3. The encryption algorithm generates the **ciphertext, c, = E(Ke)(m).** For any key k, E(k) is an algorithm for generating ciphertext from a message, and both E and E(k) should be efficiently computable functions.
    4. The ciphertext can then be sent over an unsecure network, where it may be received by **attackers.**
    5. The **recipient** enters the ciphertext into a **decryption algorithm, D,** along with the **decryption key, Kd.**
    6. The decryption algorithm re-generates the plaintext message, m, = D(Kd)(c). For any key k, D(k) is an algorithm for generating a clear text message from a ciphertext, and both D and D(k) should be efficiently computable functions.
    7. The algorithms described here must have this important property: Given a ciphertext c, a computer can only compute a message m such that c = E(k)(m) if it possesses D(k). (In other words, the messages can't be decoded unless you have the decryption algorithm and the decryption key.)

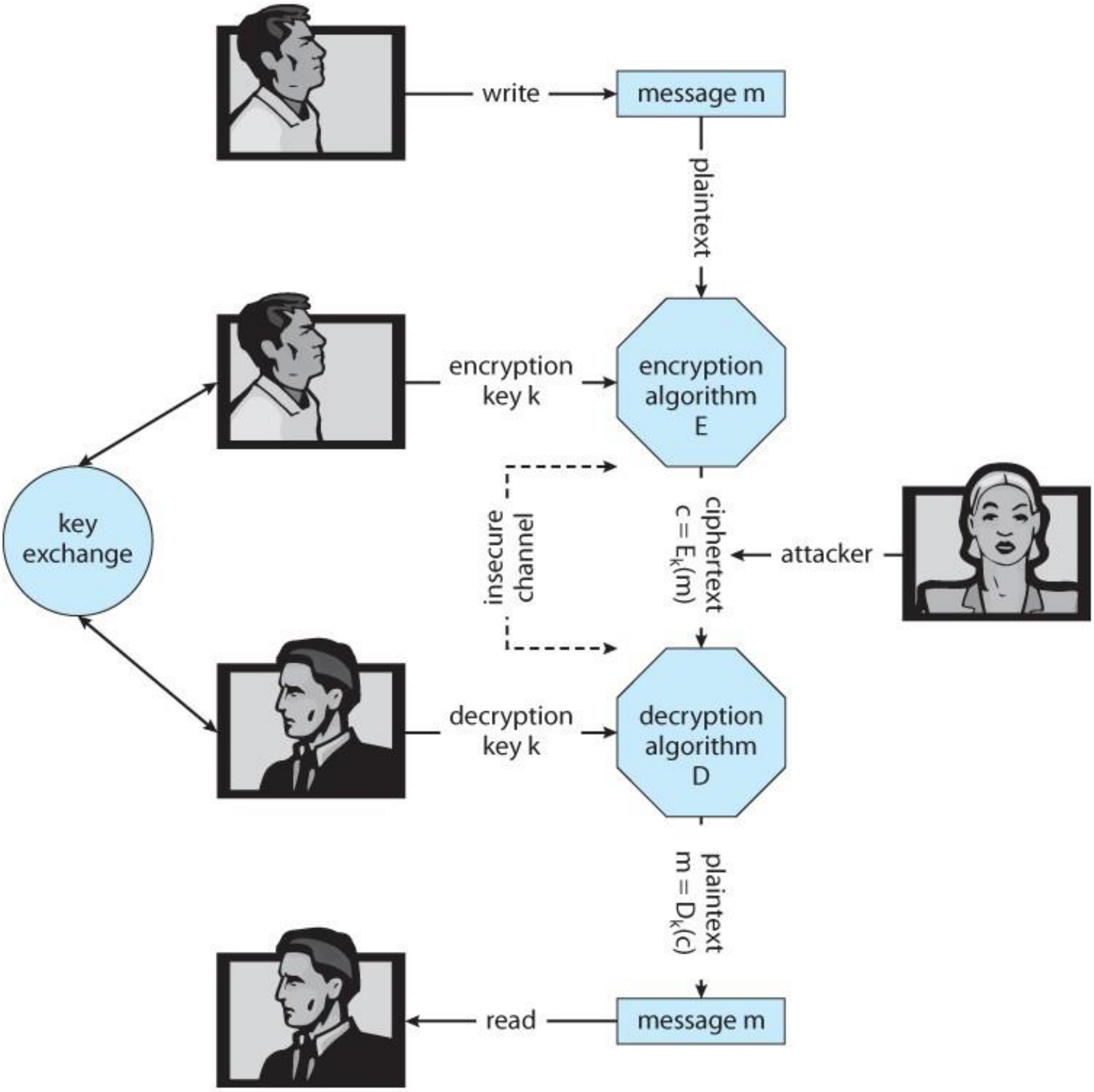

Figure 15.7 - A secure communication over an insecure medium.

Symmetric Encryption

- With ***symmetric encryption*** the same key is used for both encryption and decryption, and must be safely guarded. There are a number of well-known symmetric encryption algorithms that have been used for computer security:
  - The ***Data-Encryption Standard, DES,*** developed by the National Institute of Standards, NIST, has been a standard civilian encryption standard for over 20 years. Messages are broken down into 64-bit chunks, each of which are encrypted using a 56-bit key through a series of substitutions and transformations. Some of the transformations are hidden (black boxes), and are classified by the U.S. government.
  - DES is known as a ***block cipher,*** because it works on blocks of data at a time. Unfortunately this is a vulnerability if the same key is used for an extended amount of data. Therefore an enhancement is to not only encrypt each block, but also to XOR it with the previous block, in a technique known as ***cipher-block chaining.***
  - As modern computers become faster and faster, the security of DES has decreased, to where it is now considered insecure because its keys can be exhaustively searched within a reasonable amount of computer time. An enhancement called ***triple DES*** encrypts the data three times using three separate keys (actually two encryptions and one decryption) for an effective key length of 168 bits. Triple DES is in widespread use today.

- The ***Advanced Encryption Standard, AES,*** developed by NIST in 2001 to replace DES uses key lengths of 128, 192, or 256 bits, and encrypts in blocks of 128 bits using 10 to 14 rounds of transformations on a matrix formed from the block.
- The ***twofish algorithm,*** uses variable key lengths up to 256 bits and works on 128 bit blocks.
- ***RC5*** can vary in key length, block size, and the number of transformations, and runs on a wide variety of CPUs using only basic computations.
- ***RC4*** is a ***stream cipher,*** meaning it acts on a stream of data rather than blocks. The key is used to seed a pseudo-random number generator, which generates a ***keystream*** of keys. RC4 is used in ***WEP,*** but has been found to be breakable in a reasonable amount of computer time.

Asymmetric Encryption

- With ***asymmetric encryption,*** the decryption key, Kd, is not the same as the encryption key, Ke, and more importantly cannot be derived from it, which means the encryption key can be made publicly available, and only the decryption key needs to be kept secret. (or vice-versa, depending on the application.)
- One of the most widely used asymmetric encryption algorithms is ***RSA,*** named after its developers - Rivest, Shamir, and Adleman.
- RSA is based on two large prime numbers, ***p*** and ***q,*** (on the order of 512 bits each), and their product ***N.***
  - Ke and Kd must satisfy the relationship:
(Ke * Kd) % [(p - 1) * (q - 1)] = = 1
  - The encryption algorithm is:
c = E(Ke)(m) = m^Ke % N
  - The decryption algorithm is:
m = D(Kd)(c) = c^Kd % N
- An example using small numbers:
  - p = 7
  - q = 13
  - N = 7 * 13 = 91
  - (p - 1) * (q - 1) = 6 * 12 = 72
  - Select Ke < 72 and relatively prime to 72, say 5
  - Now select Kd, such that (Ke * Kd) % 72 = = 1, say 29
  - The public key is now (5, 91) and the private key is (29, 91)
  - Let the message, m = 42
  - Encrypt: c = 42^5 % 91 = 35
  - Decrypt: m = 35^29 % 91 = 42

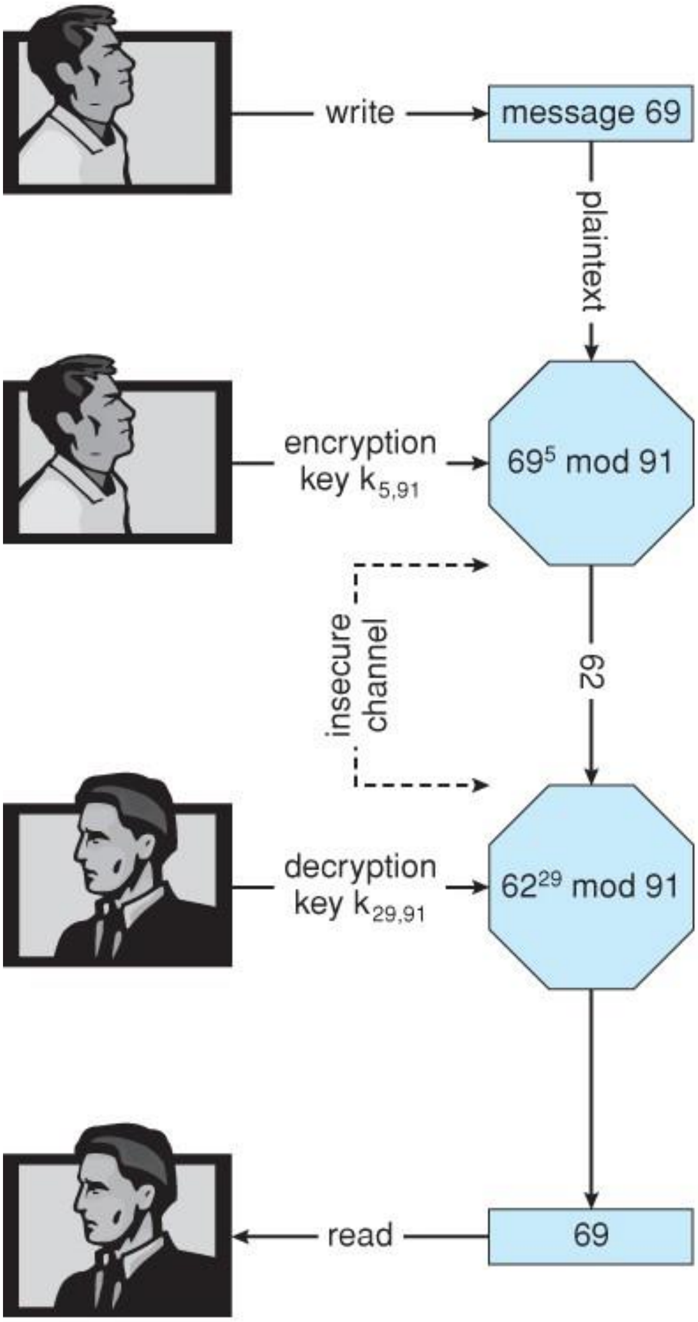

Figure 15.8 - Encryption and decryption using RSA asymmetric cryptography

- Note that asymmetric encryption is much more computationally expensive than symmetric encryption, and as such it is not normally used for large transmissions. Asymmetric encryption is suitable for small messages, authentication, and key distribution, as covered in the following sections.

Authentication

- Authentication involves verifying the identity of the entity who transmitted a message.
- For example, if D(Kd)(c) produces a valid message, then we know the sender was in possession of E(Ke).
- This form of authentication can also be used to verify that a message has not been modified
- Authentication revolves around two functions, used for ***signatures*** (or ***signing***), and ***verification:***
  - A signing function, ***S(Ks)*** that produces an ***authenticator, A,*** from any given message m.
  - A Verification function, ***V(Kv,m,A)*** that produces a value of "true" if A was created from m, and "false" otherwise.
  - Obviously S and V must both be computationally efficient.
  - More importantly, it must not be possible to generate a valid authenticator, A, without having possession of S(Ks).

  - Furthermore, it must not be possible to divine S(Ks) from the combination of (m and A), since both are sent visibly across networks.
- Understanding authenticators begins with an understanding of hash functions, which is the first step:
  - ***Hash functions, H(m)*** generate a small fixed-size block of data known as a ***message digest,*** or ***hash value*** from any given input data.
  - For authentication purposes, the hash function must be ***collision resistant on m.*** That is it should not be reasonably possible to find an alternate message m' such that H(m') = H(m).
  - Popular hash functions are **MD5,** which generates a 128-bit message digest, and **SHA-1,** which generates a 160-bit digest.
- Message digests are useful for detecting (accidentally) changed messages, but are not useful as authenticators, because if the hash function is known, then someone could easily change the message and then generate a new hash value for the modified message. Therefore authenticators take things one step further by encrypting the message digest.
- A ***message-authentication code, MAC,*** uses symmetric encryption and decryption of the message digest, which means that anyone capable of verifying an incoming message could also generate a new message.
- An asymmetric approach is the ***digital-signature algorithm,*** which produces authenticators called ***digital signatures.*** In this case Ks and Kv are separate, Kv is the public key, and it is not practical to determine S(Ks) from public information. In practice the sender of a message signs it (produces a digital signature using S(Ks)), and the receiver uses V(Kv) to verify that it did indeed come from a trusted source, and that it has not been modified.
- There are three good reasons for having separate algorithms for encryption of messages and authentication of messages:
  1. Authentication algorithms typically require fewer calculations, making verification a faster operation than encryption.
  2. Authenticators are almost always smaller than the messages, improving space efficiency. (?)
  3. Sometimes we want authentication only, and not confidentiality, such as when a vendor issues a new software patch.
- Another use of authentication is ***non-repudiation,*** in which a person filling out an electronic form cannot deny that they were the ones who did so.

Key Distribution

- Key distribution with symmetric cryptography is a major problem, because all keys must be kept secret, and they obviously can't be transmitted over unsecure channels. One option is to send them ***out-of-band,*** say via paper or a confidential conversation.
- Another problem with symmetric keys, is that a separate key must be maintained and used for each correspondent with whom one wishes to exchange confidential information.
- Asymmetric encryption solves some of these problems, because the public key can be freely transmitted through any channel, and the private key doesn't need to be transmitted anywhere. Recipients only need to maintain one private key for all incoming messages, though senders must maintain a separate public key for each recipient to which they might wish to send a message. Fortunately the public keys are not confidential, so this ***key-ring*** can be easily stored and managed.
- Unfortunately there are still some security concerns regarding the public keys used in asymmetric encryption. Consider for example the following man-in-the-middle attack involving phony public keys:

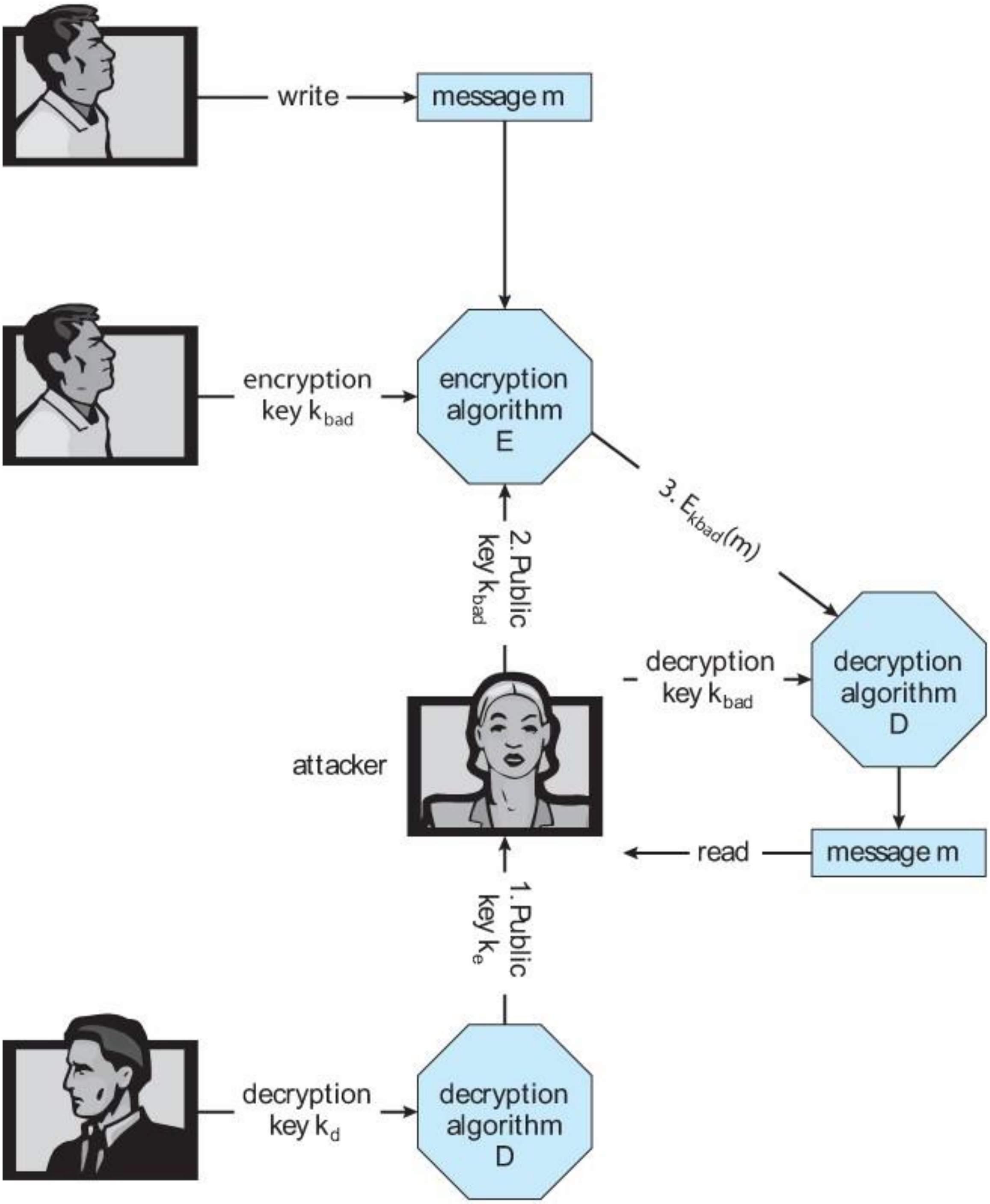

Figure 15.9 - A man-in-the-middle attack on asymmetric cryptography.

- One solution to the above problem involves ***digital certificates,*** which are public keys that have been digitally signed by a trusted third party. But wait a minute - How do we trust that third party, and how do we know ***they*** are really who they say they are? Certain ***certificate authorities*** have their public keys included within web browsers and other certificate consumers before they are distributed. These certificate authorities can then vouch for other trusted entities and so on in a web of trust, as explained more fully in section 15.4.3.

Implementation of Cryptography

- Network communications are implemented in multiple layers - Physical, Data Link, Network, Transport, and Application being the most common breakdown.
- Encryption and security can be implemented at any layer in the stack, with pros and cons to each choice:
  - Because packets at lower levels contain the contents of higher layers, encryption at lower layers automatically encrypts higher layer information at the same time.
  - However security and authorization may be important to higher levels independent of the underlying transport mechanism or route taken.
- At the network layer the most common standard is **IPSec,** a secure form of the IP layer, which is used to set up **Virtual Private Networks, VPNs.**

- At the transport layer the most common implementation is SSL, described below.

An Example: SSL

- SSL (Secure Sockets Layer) 3.0 was first developed by Netscape, and has now evolved into the industry-standard TLS protocol. It is used by web browsers to communicate securely with web servers, making it perhaps the most widely used security protocol on the Internet today.
- SSL is quite complex with many variations, only a simple case of which is shown here.
- The heart of SSL is ***session keys***, which are used once for symmetric encryption and then discarded, requiring the generation of new keys for each new session. The big challenge is how to safely create such keys while avoiding man-in-the-middle and replay attacks.
- Prior to commencing the transaction, the server obtains a ***certificate*** from a ***certification authority, CA,*** containing:
    - Server attributes such as unique and common names.
    - Identity of the public encryption algorithm, E( ), for the server.
    - The public key, k_e for the server.
    - The validity interval within which the certificate is valid.
    - A digital signature on the above issued by the CA:
        - a = S(K_CA)((attrs, E(k_e), interval)
- In addition, the client will have obtained a public ***verification algorithm,*** V(K_CA), for the certifying authority. Today's modern browsers include these built-in by the browser vendor for a number of trusted certificate authorities.
- The procedure for establishing secure communications is as follows:
    1. The client, c, connects to the server, s, and sends a random 28-byte number, n_c.
    2. The server replies with its own random value, n_s, along with its certificate of authority.
    3. The client uses its verification algorithm to confirm the identity of the sender, and if all checks out, then the client generates a 46 byte random **premaster secret, pms,** and sends an encrypted version of it as cpms = E(k_s)(pms)
    4. The server recovers pms as D(k_s)(cpms).
    5. Now both the client and the server can compute a shared 48-byte ***master secret, ms,*** = f(pms, n_s, n_c)
    6. Next, both client and server generate the following from ms:
        - Symmetric encryption keys k_sc_crypt and k_cs_crypt for encrypting messages from the server to the client and vice-versa respectively.
        - MAC generation keys k_sc_mac and k_cs_mac for generating authenticators on messages from server to client and client to server respectively.
    7. To send a message to the server, the client sends:
        - c = E(k_cs_crypt)(m, S(k_cs_mac))(m)))
    8. Upon receiving c, the server recovers:
        - (m,a) = D(k_cs_crypt)(c)
        - and accepts it if V(k_sc_mac)(m,a) is true.
- This approach enables both the server and client to verify the authenticity of every incoming message, and to ensure that outgoing messages are only readable by the process that originally participated in the key generation.
- SSL is the basis of many secure protocols,including ***Virtual Private Networks, VPNs,*** in which private data is distributed over the insecure public internet structure in an encrypted fashion that emulates a privately owned network.

User Authentication

- A lot of chapter 14, Protection, dealt with making sure that only certain users were allowed to perform certain tasks, i.e. that a users privileges were dependent on his or her identity. But how does one verify that identity to begin with?

Passwords

- Passwords are the most common form of user authentication. If the user is in possession of the correct password, then they are considered to have identified themselves.
- In theory separate passwords could be implemented for separate activities, such as reading this file, writing that file, etc. In practice most systems use one password to confirm user identity, and then authorization is based upon that identification. This is a result of the classic trade-off between security and convenience.

Password Vulnerabilities

- Passwords can be guessed.
    - Intelligent guessing requires knowing something about the intended target in specific, or about people and commonly used passwords in general.
    - Brute-force guessing involves trying every word in the dictionary, or every valid combination of characters. For this reason good passwords should not be in any dictionary (in any language), should be reasonably lengthy, and should use the full range of allowable characters by including upper and lower case characters, numbers, and special symbols.
- "Shoulder surfing" involves looking over people's shoulders while they are typing in their password.
    - Even if the lurker does not get the entire password, they may get enough clues to narrow it down, especially if they watch on repeated occasions.
    - Common courtesy dictates that you look away from the keyboard while someone is typing their password.
    - Passwords echoed as stars or dots still give clues, because an observer can determine how many characters are in the password. :-(
- "Packet sniffing" involves putting a monitor on a network connection and reading data contained in those packets.
    - SSH encrypts all packets, reducing the effectiveness of packet sniffing.
    - However you should still never e-mail a password, particularly not with the word "password" in the same message or worse yet the subject header.
    - Beware of any system that transmits passwords in clear text. ("Thank you for signing up for XYZ. Your new account and password information are shown below".) You probably want to have a spare throw-away password to give these entities, instead of using the same high-security password that you use for banking or other confidential uses.
- Long hard to remember passwords are often written down, particularly if they are used seldomly or must be changed frequently. Hence a security trade-off of passwords that are easily divined versus those that get written down. :-(
- Passwords can be given away to friends or co-workers, destroying the integrity of the entire user-identification system.
- Most systems have configurable parameters controlling password generation and what constitutes acceptable passwords.
    - They may be user chosen or machine generated.
    - They may have minimum and/or maximum length requirements.
    - They may need to be changed with a given frequency. (In extreme cases for every session.)
    - A variable length history can prevent repeating passwords.
    - More or less stringent checks can be made against password dictionaries.

Encrypted Passwords

- Modern systems do not store passwords in clear-text form, and hence there is no mechanism to look up an existing password.
- Rather they are encrypted and stored in that form. When a user enters their password, that too is encrypted, and if the encrypted version match, then user authentication passes.
- The encryption scheme was once considered safe enough that the encrypted versions were stored in the publicly readable file "/etc/passwd".
    - They always encrypted to a 13 character string, so an account could be disabled by putting a string of any other length into the password field.
    - Modern computers can try every possible password combination in a reasonably short time, so now the encrypted passwords are stored in files that are only readable by the super user. Any password-related programs run as setuid root to get access to these files. (/etc/shadow)
    - A random seed is included as part of the password generation process, and stored as part of the encrypted password. This ensures that if two accounts have the same plain-text password that they will not have the same encrypted password. However cutting and pasting encrypted passwords from one account to another will give them the same plain-text passwords.

One-Time Passwords

- One-time passwords resist shoulder surfing and other attacks where an observer is able to capture a password typed in by a user.
    - These are often based on a **challenge** and a **response.** Because the challenge is different each time, the old response will not be valid for future challenges.
        - For example, The user may be in possession of a secret function f(x). The system challenges with some given value for x, and the user responds with f(x), which the system can then verify. Since the challenger gives a different (random) x each time, the answer is constantly changing.

        - A variation uses a map (e.g. a road map) as the key. Today's question might be "On what corner is SEO located?", and tomorrow's question might be "How far is it from Navy Pier to Wrigley Field?" Obviously "Taylor and Morgan" would not be accepted as a valid answer for the second question!
    - Another option is to have some sort of electronic card with a series of constantly changing numbers, based on the current time. The user enters the current number on the card, which will only be valid for a few seconds. A ***two-factor authorization*** also requires a traditional password in addition to the number on the card, so others may not use it if it were ever lost or stolen.
    - A third variation is a ***code book,*** or ***one-time pad.*** In this scheme a long list of passwords is generated, and each one is crossed off and cancelled as it is used. Obviously it is important to keep the pad secure.

Biometrics

- Biometrics involve a physical characteristic of the user that is not easily forged or duplicated and not likely to be identical between multiple users.
    - Fingerprint scanners are getting faster, more accurate, and more economical.
    - Palm readers can check thermal properties, finger length, etc.
    - Retinal scanners examine the back of the users' eyes.
    - Voiceprint analyzers distinguish particular voices.
    - Difficulties may arise in the event of colds, injuries, or other physiological changes.

Implementing Security Defenses

Security Policy

- A security policy should be well thought-out, agreed upon, and contained in a living document that everyone adheres to and is updated as needed.
- Examples of contents include how often port scans are run, password requirements, virus detectors, etc.

Vulnerability Assessment

- Periodically examine the system to detect vulnerabilities.
    - Port scanning.
    - Check for bad passwords.
    - Look for suid programs.
    - Unauthorized programs in system directories.
    - Incorrect permission bits set.
    - Program checksums / digital signatures which have changed.
    - Unexpected or hidden network daemons.
    - New entries in startup scripts, shutdown scripts, cron tables, or other system scripts or configuration files.
    - New unauthorized accounts.
- The government considers a system to be only as secure as its most far-reaching component. Any system connected to the Internet is inherently less secure than one that is in a sealed room with no external communications.
- Some administrators advocate "security through obscurity", aiming to keep as much information about their systems hidden as possible, and not announcing any security concerns they come across. Others announce security concerns from the rooftops, under the theory that the hackers are going to find out anyway, and the only one kept in the dark by obscurity are honest administrators who need to get the word.

Intrusion Detection

- Intrusion detection attempts to detect attacks, both successful and unsuccessful attempts. Different techniques vary along several axes:
    - The time that detection occurs, either during the attack or after the fact.
    - The types of information examined to detect the attack(s). Some attacks can only be detected by analyzing multiple sources of information.
    - The response to the attack, which may range from alerting an administrator to automatically stopping the attack (e.g. killing an offending process), to tracing back the attack in order to identify the attacker.
        - Another approach is to divert the attacker to a ***honeypot***, on a ***honeynet.*** The idea behind a honeypot is a computer running normal services, but which no one uses to do any real work.

Such a system should not see any network traffic under normal conditions, so any traffic going to or from such a system is by definition suspicious. Honeypots are normally kept on a honeynet protected by a ***reverse firewall,*** which will let potential attackers in to the honeypot, but will not allow any outgoing traffic. (So that if the honeypot is compromised, the attacker cannot use it as a base of operations for attacking other systems.) Honeypots are closely watched, and any suspicious activity carefully logged and investigated.

- Intrusion Detection Systems, IDSs, raise the alarm when they detect an intrusion. Intrusion Detection and Prevention Systems, IDPs, act as filtering routers, shutting down suspicious traffic when it is detected.
- There are two major approaches to detecting problems:
    - ***Signature-Based Detection*** scans network packets, system files, etc. looking for recognizable characteristics of known attacks, such as text strings for messages or the binary code for "exec /bin/sh". The problem with this is that it can only detect previously encountered problems for which the signature is known, requiring the frequent update of signature lists.
    - ***Anomaly Detection*** looks for "unusual" patterns of traffic or operation, such as unusually heavy load or an unusual number of logins late at night.
        - The benefit of this approach is that it can detect previously unknown attacks, so called ***zero-day attacks.***
        - One problem with this method is characterizing what is "normal" for a given system. One approach is to benchmark the system, but if the attacker is already present when the benchmarks are made, then the "unusual" activity is recorded as "the norm."
        - Another problem is that not all changes in system performance are the result of security attacks. If the system is bogged down and really slow late on a Thursday night, does that mean that a hacker has gotten in and is using the system to send out SPAM, or does it simply mean that a CS 385 assignment is due on Friday? :-)
        - To be effective, anomaly detectors must have a very low ***false alarm (false positive)*** rate, lest the warnings get ignored, as well as a low ***false negative*** rate in which attacks are missed.

Virus Protection

- Modern anti-virus programs are basically signature-based detection systems, which also have the ability (in some cases) of ***disinfecting*** the affected files and returning them back to their original condition.
- Both viruses and anti-virus programs are rapidly evolving. For example viruses now commonly mutate every time they propagate, and so anti-virus programs look for families of related signatures rather than specific ones.
- Some antivirus programs look for anomalies, such as an executable program being opened for writing (other than by a compiler.)
- Avoiding bootleg, free, and shared software can help reduce the chance of catching a virus, but even shrink-wrapped official software has on occasion been infected by disgruntled factory workers.
- Some virus detectors will run suspicious programs in a ***sandbox,*** an isolated and secure area of the system which mimics the real system.
- ***Rich Text Format, RTF,*** files cannot carry macros, and hence cannot carry Word macro viruses.
- Known safe programs (e.g. right after a fresh install or after a thorough examination) can be digitally signed, and periodically the files can be re-verified against the stored digital signatures. (Which should be kept secure, such as on off-line write-only medium.)

Auditing, Accounting, and Logging

- Auditing, accounting, and logging records can also be used to detect anomalous behavior.
- Some of the kinds of things that can be logged include authentication failures and successes, logins, running of suid or sgid programs, network accesses, system calls, etc. In extreme cases almost every keystroke and electron that moves can be logged for future analysis. (Note that on the flip side, all this detailed logging can also be used to analyze system performance. The down side is that the logging also *affects* system performance (negatively!), and so a Heisenberg effect applies.)
- "The Cuckoo's Egg" tells the story of how Cliff Stoll detected one of the early UNIX break ins when he noticed anomalies in the accounting records on a computer system being used by physics researchers.

Tripwire Filesystem (New Sidebar)

- The tripwire filesystem monitors files and directories for changes, on the assumption that most intrusions eventually result in some sort of undesired or unexpected file changes.
- The tw.config file indicates what directories are to be monitored, as well as what properties of each file are to be recorded. (E.g. one may choose to monitor permission and content changes, but not worry about read access times.)
- When first run, the selected properties for all monitored files are recorded in a database. Hash codes are used to monitor file contents for changes.

- Subsequent runs report any changes to the recorded data, including hash code changes, and any newly created or missing files in the monitored directories.
- For full security it is necessary to also protect the tripwire system itself, most importantly the database of recorded file properties. This could be saved on some external or write-only location, but that makes it harder to change the database when legitimate changes are made.
- It is difficult to monitor files that are *supposed to* change, such as log files. The best tripwire can do in this case is to watch for anomalies, such as a log file that shrinks in size.
- Free and commercial versions are available at http://tripwire.org and http://tripwire.com.

Firewalling to Protect Systems and Networks

- Firewalls are devices (or sometimes software) that sit on the border between two security domains and monitor/log activity between them, sometimes restricting the traffic that can pass between them based on certain criteria.
- For example a firewall router may allow HTTP: requests to pass through to a web server inside a company domain while not allowing telnet, ssh, or other traffic to pass through.
- A common architecture is to establish a de-militarized zone, DMZ, which sort of sits "between" the company domain and the outside world, as shown below. Company computers can reach either the DMZ or the outside world, but outside computers can only reach the DMZ. Perhaps most importantly, the DMZ cannot reach any of the other company computers, so even if the DMZ is breached, the attacker cannot get to the rest of the company network. (In some cases the DMZ may have limited access to company computers, such as a web server on the DMZ that needs to query a database on one of the other company computers.)

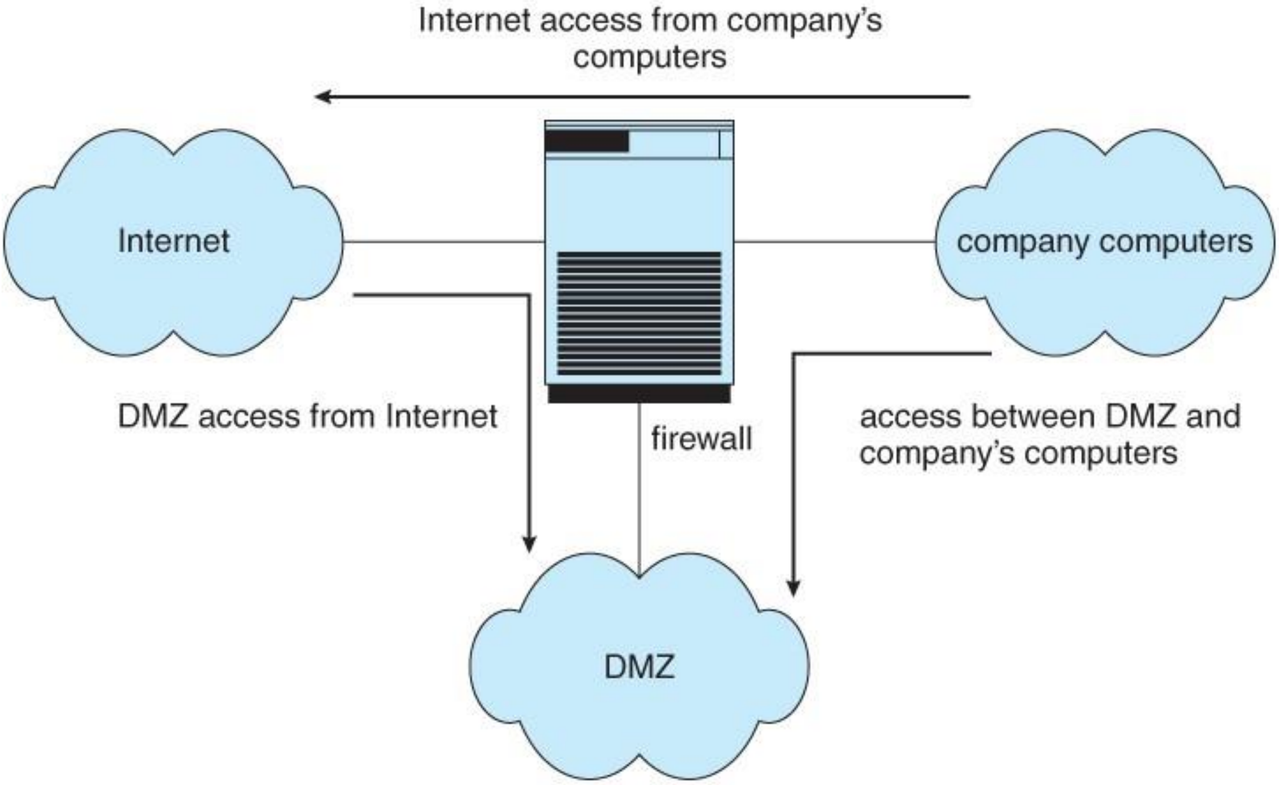
Internet access from company's computers
Internet
company computers
DMZ access from Internet
firewall
access between DMZ and company's computers
DMZ

Figure 15.10 - Domain separation via firewall.

- Firewalls themselves need to be resistant to attacks, and unfortunately have several vulnerabilities:
  - ***Tunneling,*** which involves encapsulating forbidden traffic inside of packets that are allowed.
  - Denial of service attacks addressed at the firewall itself.
  - Spoofing, in which an unauthorized host sends packets to the firewall with the return address of an authorized host.
- In addition to the common firewalls protecting a company internal network from the outside world, there are also some specialized forms of firewalls that have been recently developed:
  - A ***personal firewall*** is a software layer that protects an individual computer. It may be a part of the operating system or a separate software package.
  - An ***application proxy firewall*** understands the protocols of a particular service and acts as a stand-in (and relay) for the particular service. For example, and SMTP proxy firewall would accept SMTP requests from the outside world, examine them for security concerns, and forward only the "safe" ones on to the real SMTP server behind the firewall.
  - ***XML firewalls*** examine XML packets only, and reject ill-formed packets. Similar firewalls exist for other specific protocols.

  - ***System call firewalls*** guard the boundary between user mode and system mode, and reject any system calls that violate security policies.

Computer-Security Classifications

- No computer system can be 100% secure, and attempts to make it so can quickly make it unusable.
- However one can establish a level of trust to which one feels "safe" using a given computer system for particular security needs.
- The U.S. Department of Defense's "Trusted Computer System Evaluation Criteria" defines four broad levels of trust, and sub-levels in some cases:
  - **Level D** is the least trustworthy, and encompasses all systems that do not meet any of the more stringent criteria. DOS and Windows 3.1 fall into level D, which has no user identification or authorization, and anyone who sits down has full access and control over the machine.
  - **Level C1** includes user identification and authorization, and some means of controlling what users are allowed to access what files. It is designed for use by a group of mostly cooperating users, and describes most common UNIX systems.
  - **Level C2** adds individual-level control and monitoring. For example file access control can be allowed or denied on a per-individual basis, and the system administrator can monitor and log the activities of specific individuals. Another restriction is that when one user uses a system resource and then returns it back to the system, another user who uses the same resource later cannot read any of the information that the first user stored there. (I.e. buffers, etc. are wiped out between users, and are not left full of old contents.) Some special secure versions of UNIX have been certified for C2 security levels, such as SCO.
  - **Level B** adds sensitivity labels on each object in the system, such as "secret", "top secret", and "confidential". Individual users have different clearance levels, which controls which objects they are able to access. All human-readable documents are labeled at both the top and bottom with the sensitivity level of the file.
  - **Level B2** extends sensitivity labels to all system resources, including devices. B2 also supports covert channels and the auditing of events that could exploit covert channels.
  - B3 allows creation of access-control lists that denote users NOT given access to specific objects.
  - Class A is the highest level of security. Architecturally it is the same as B3, but it is developed using formal methods which can be used to ***prove*** that the system meets all requirements and cannot have any possible bugs or other vulnerabilities. Systems in class A and higher may be developed by trusted personnel in secure facilities.
  - These classifications determine what a system *can* implement, but it is up to security policy to determine *how* they are implemented in practice. These systems and policies can be reviewed and certified by trusted organizations, such as the National Computer Security Center. Other standards may dictate physical protections and other issues.