Lecture Notes

On

COMPUTERS SECURITY

Assembled by:

Assoc. Prof. Noha A. Hikal

2018–2019

Lecture Notes

On

COMPUTERS SECURITY

Assembled by:

Assoc. Prof. Noha A. Hikal

2018–2019

CHAPTER 1

INTRODUCTION TO COMPUTER SECURITY

INTRODUCTION:

The revolution of technologies based on computers is endless track. Since using computers and internet has become one of the most important parts of our daily routine, starting from space communications and ending, not endless, by entertainment for individuals. These revolutions is faced by huge security attacks, the effect of this attacks start from denying services and grow up towards information wars among countries. Information has become wealth for countries such as petroleum and priceless metals as well as for the organizations and individuals. This information is used as a weapon in economic wars between the organizations. Hence, computer security became a necessity. Unfortunately, as technologies grow, the attacks grow too. No definite defense technology can stand with this growth. In this chapter, an introduction to basic principles of computer security is introduced in a form of questions and their answers.

1- What is meant by computer Security?

Computer security is the protection of the items you value, called the assets of a computer or computer system. There are many types of assets, involving hardware, software, data, people, processes, or combinations of these. To determine what to protect, we must first identify what has value and to whom. Figure 1 introduces the valuable computer objects. These objects differ in the function, the attacks, and the defense technique. However, they all seek to verify same goals of security. The NIST Computer Security Handbook [NIST95] defines the term computer security as: "The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/ data, and telecommunications)."

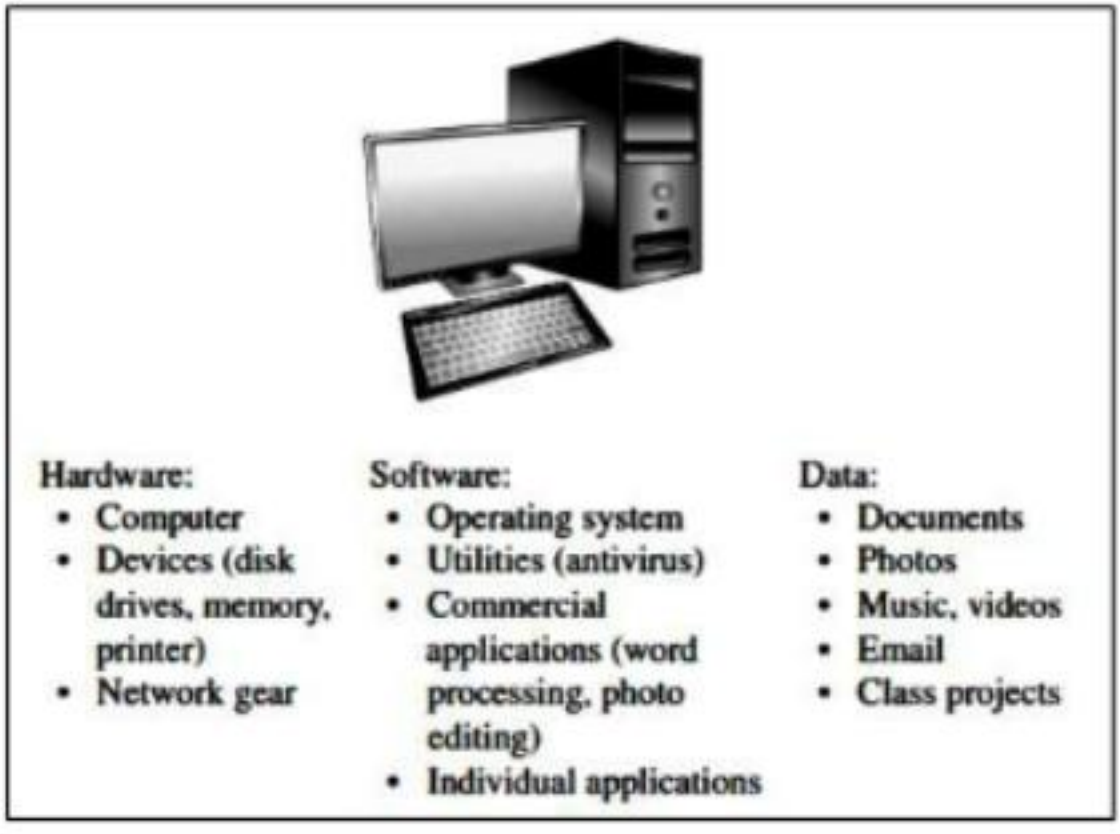

Figure 1, Valuable objects in Computer System

Protecting assets was difficult and not always effective. Today, however, asset protection is easier, with many factors working against the potential criminal. Very sophisticated alarm and camera systems silently protect secure places like banks whether people are around or not. The techniques of criminal investigation have become so effective that a person can be identified by genetic material (DNA), fingerprints, retinal patterns, voice, a composite sketch, ballistics evidence, or other hard-to-mask characteristics. The assets are stored in a safer form. For instance, many

bank branches now contain less cash than some large retail stores because much of a bank's business is conducted with checks, electronic transfers, credit cards, or debit cards. Sites that must store large amounts of cash or currency are protected with many levels of security: several layers of physical systems, complex locks, multiple-party systems requiring the agreement of several people to allow access, and other schemes. Significant improvements in transportation and communication mean that police can be at the scene of a crime in minutes; dispatchers can alert other officers in seconds about the suspects to watch for. From the criminal's point of view, the risk and required sophistication are so high that there are usually easier ways than bank robbery to make money.

The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications) is called: Computer Security.

2. *Security Goals*

We use the term "security" in many ways in our daily lives. A "security system" protects our house, warning the neighbors or the police if an unauthorized intruder tries to get in. And we speak of children's "physical security," hoping they are safe from potential harm. Just

as each of these terms has a very specific meaning in the context of its use, so too does the phrase "computer security." When we talk about computer security, we mean that we are addressing three important aspects of any computer related system. If someone steals your computer, scrambles your data, or looks at your private data files, the value of your computer has been diminished or your computer use has been harmed. These characteristics are considered the basic security goals. These three aspects, Confidentiality, Integrity, and Availability, make your computer valuable to you. These properties are called the C-I-A triad : Confidentiality, integrity, and Availability.

Confidentiality: the ability of a system to ensure that an asset is viewed only by authorized parties. It is the concealment of information or resources from unauthorized users. The need of keeping the information confident depends on the sensitivity of the information. (i.e; Military information versus personal information).

Here are some properties that could mean a failure of data confidentiality:

- An unauthorized person accesses a data item.
- An unauthorized process or program accesses a data item.
- A person authorized to access certain data accesses other data not authorized

• An unauthorized person accesses an approximate data value (for example, not knowing someone's exact salary but knowing that the salary falls in a particular range or exceeds a particular amount).
• An unauthorized person learns the existence of a piece of data (for example, knowing that a company is developing a certain new product or that talks are underway about the merger of two companies).

Achieving higher confidentiality is supported by:

i. Access Control: Ensures only authorized personnel have access to the information.
ii. Cryptography: When unauthorized access occurred, information is meaningless.

Integrity: the ability of a system to ensure that an asset is modified only by authorized parties. Integrity means trusting the worthiness of the data or resource, and to prevent improper or unauthorized change. Integrity includes: Data integrity; to ensure the correctness of the content of the data, and Origin integrity; to ensure the source of the data using authentication techniques.

if we say that we have preserved the integrity of an item, we may mean that the item is:

• Precise
• Accurate
• Unmodified

- Modified only in acceptable ways
- Modified only by authorized people
- Modified only by authorized processes
- Consistent
- Internally consistent
- Meaningful and usable

Integrity Mechanism falls into two classes:

i. Prevention: This uses technologies for blocking unauthorized attempt to modify data.

ii. Detection: This uses technologies to report when unauthorized modification occurs.

Availability: the ability of a system to ensure that an asset can be used by any authorized parties. The main objective of availability is to ensure that authorized access to data when desired is achievable. Attempt to block service (System down) or to make the service unavailable is called Denial of service attack (DoS) which is considered the most dangerous availability attack.

An object or service is thought to be available if the following are true:

- It is present in a usable form.
- It has enough capacity to meet the service's needs.
- It is making clear progress, and, if in wait mode, it has a bounded waiting time.
- The service is completed in an acceptable period of time. There is a timely response to our request.

• Resources are allocated fairly so that some requesters are not favored over others.

What can happen to harm the confidentiality, integrity, or availability of computer assets? If a thief steals your computer, you no longer have access, so you have lost Availability; furthermore, if the thief looks at the pictures or documents you have stored, your confidentiality is compromised. And if the thief changes the content of your music files but then gives them back with your computer, the integrity of your data has been harmed. You can envision many scenarios based around these three properties.

Security in computing addresses these three goals. One of the challenges in building a secure system is finding the right balance among the goals, which often conflict. For example, it is easy to preserve a particular object's confidentiality in a secure system simply by preventing everyone from reading that object. However, this system is not secure, because it does not meet the requirement of availability for proper access. That is, there must be a balance between confidentiality and availability.

But balance is not all. In fact, these three characteristics can be independent, can overlap (as shown in Figure 2, and can even be mutually exclusive. For example, we have seen that strong protection of confidentiality can

severely restrict availability. Figure 3 shows the relation between the degrees of security versus the degree of usability.

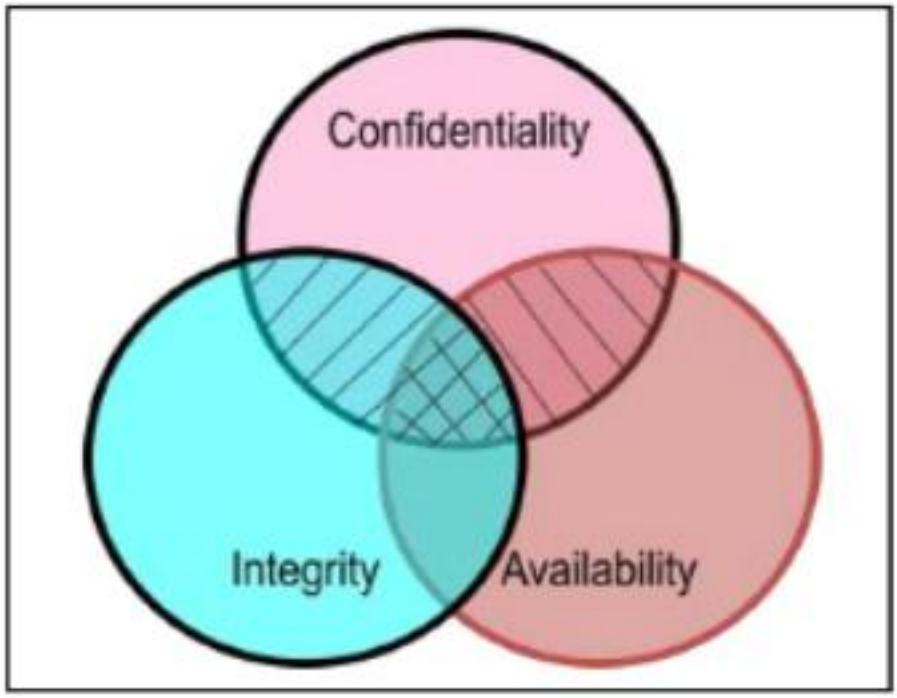

Figure 2. Relationship between security goals.

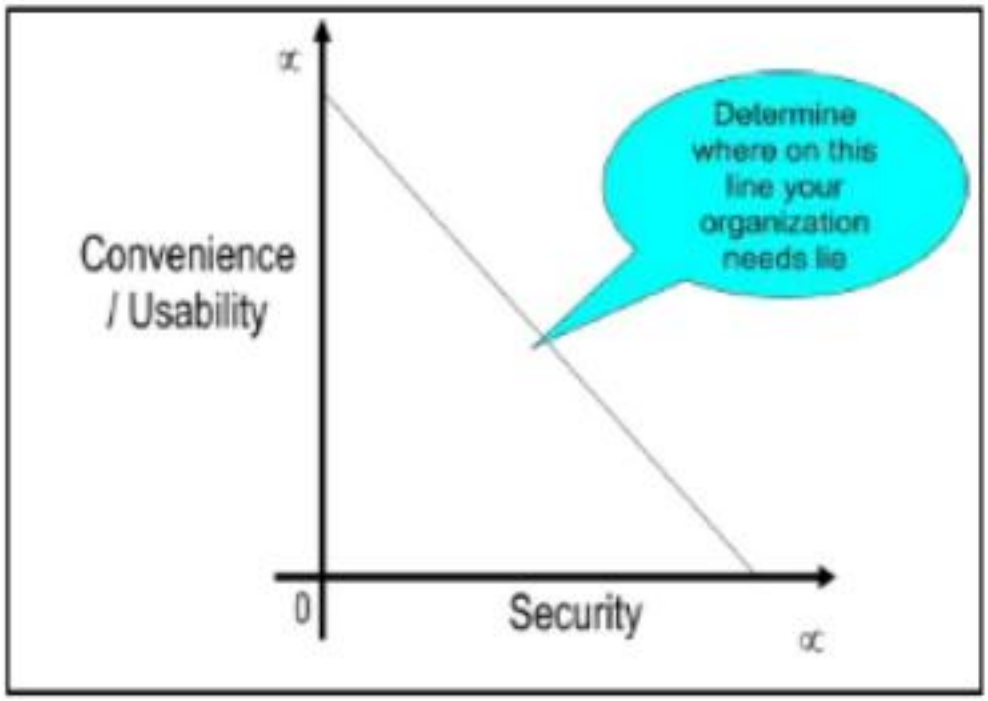

Figure 3. Usability versus Security

Much of computer security's past success has focused on confidentiality and integrity; full implementation of availability is security's next great challenge.

2- ***What is the difference between threats and vulnerability in computer system?***

Vulnerability: it refers to a weakness in the computer system (i,e; policy, design, implementation, or procedure) that can be exploited to cause harm or loss.

Threats: A threat to a computing system is a set of circumstances that has the potential to cause loss or harm. It refers to an action taken that uses one or more vulnerabilities to realize a threat. This could be someone following through on a threat or exploiting vulnerability.

Figure 4 shows the relationship between threats and vulnerabilities. While figure 5 shows the types of vulnerabilities we might find as they apply to the assets of hardware, software, and data. These three assets and the connections among them are all potential security weak points. Let us look in turn at the vulnerabilities of each asset. Concluding that

A threat is blocked by control of a Vulnerability

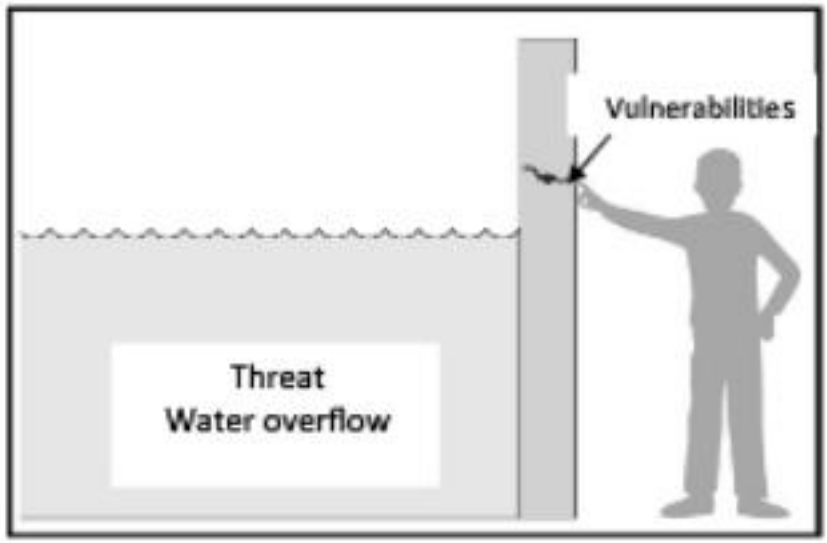

Figure 4. Threats and Vulnerability

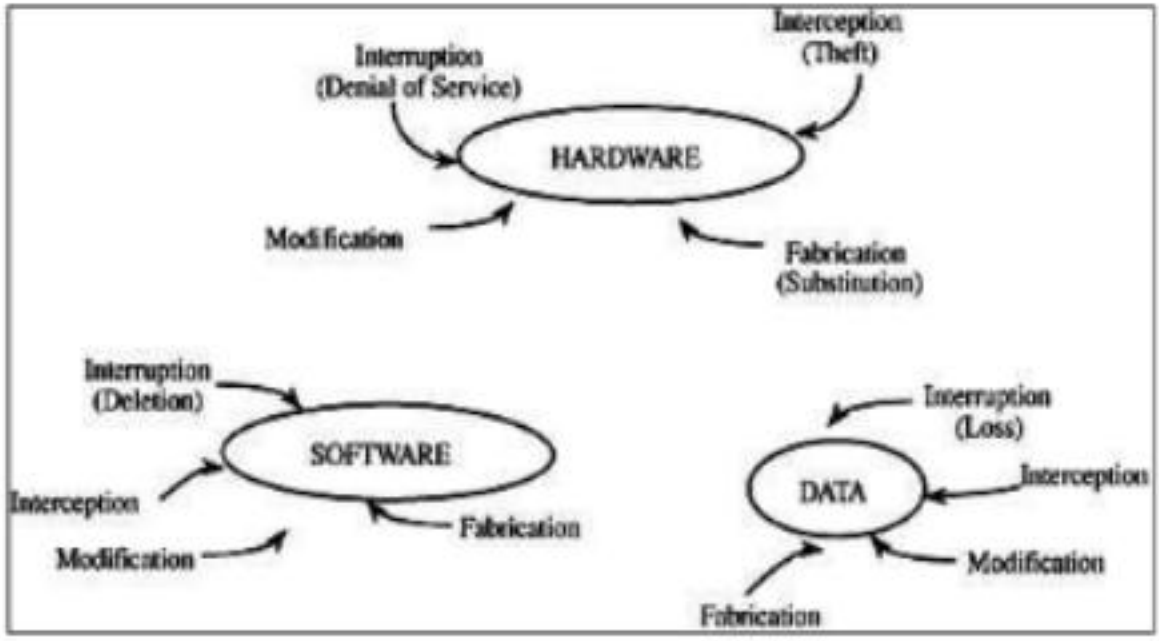

Figure 5. Vulnerabilities of Computing Systems.

-*Threats to hardware:*

Usually, it is the concern of a small staff of computer center professionals. Hardware threats is more visible than software, largely because it is composed of physical objects. Because we can see what devices are hooked to the system, it is rather simple to attack by adding devices, changing them, removing them,

Hardware threats can be: *Involuntary*: (like accidents; pouring water, food, etc). *Voluntary*: in which some actually wishes to do harm to the computer (bombs ,fires, ,theft, shorting out circuit boards)

-*Threats of software:*

It is the concern of all programmers, users, and analysis's. Software includes operating system, controllers, utility programs, and application programs that users expect. Software can be replaced, changed, or destroyed maliciously, or it can be modified, deleted, or misplaced accidentally. Sometimes, the attacks are obvious, as when the software no longer runs. More subtle are attacks in which the software has been altered but seems to run normally. Furthermore, it is possible to change a program so that it doesn't be able to do all it did before. Whether intentional or not, these attacks exploit the software's vulnerabilities as:

1. Software deletion.

Software is very easy to delete. Each of us has, at some point in our careers accidentally erased a file or saved a bad copy of a program, destroying a good previous copy. Because of software's high value to a commercial computing center, access to software is usually carefully controlled through a process called "configuration management". so that software cannot be deleted, destroyed, or replaced accidentally. Configuration management uses several techniques to ensure that each version or release retains its integrity. When configuration management is used, an old version or release can be replaced with a newer version only when it has been thoroughly tested to verify that the improvements work correctly without degrading the functionality and performance of other functions and services

2. Software theft.

This attack includes unauthorized copying of software. Software authors and distributors are entitled to fair compensation for use of their product. Unauthorized copying of software has not been stopped satisfactorily.

3. Software modification

A classic example of exploiting software vulnerability is the case in which a bank worker realized that software truncates the fractional interest on each account. In other words, if the monthly interest on an account is calculated

to be $14.5467, the software credits only $14.54 and ignores the $.0067. The worker amended the software so that the throw-away interest (the $.0067) was placed into his own account. Since the accounting practices ensured only that all accounts balanced, he built up a large amount of money from the thousands of account throw away without detection. It was only when he bragged to a colleague of his cleverness that the scheme was discovered.

Modification is done for either to cause the program fails during execution or fails in some special circumstances (logic bomb) or to cause it to do some unintended task.

The category of software modification includes:

- Trojan horse: a program that overtly does one thing while covertly doing another
- virus: a specific type of Trojan horse that can be used to spread its "infection" from one computer to another
- Trapdoor: a program that has a secret entry point information leaks in a program: code that makes information accessible to unauthorized people or programs

Countermeasures concentrate mainly in addressing a vulnerability to reduce the probability of an attack or the impact of a threat.

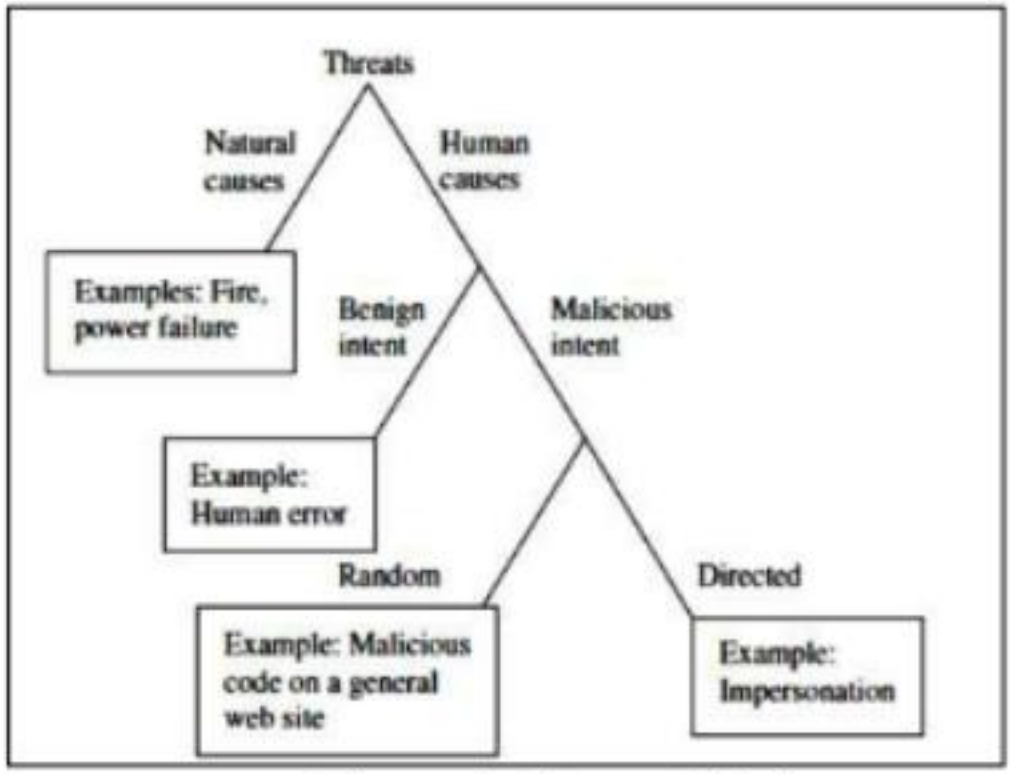

Figure 6. Threats Kinds

Threats to Data

Data items have greater public value than hardware and software because more people know how to use or interpret data. On the other hand, data incorrectly modified can cost human lives. Data items in context do relate to cost, perhaps measurable by the cost to reconstruct or redevelop damaged or lost data. Finally, inadequate security may lead to financial liability if certain personal data are made public. Thus, data have a definite value, even though that value is often difficult to measure.

Data security suggests the second principle of computer security. Principle of Adequate Protection: Computer items must be protected only until they lose their value. They must be protected to a degree consistent with their value. This principle says that "*Things with a short life can be protected by security measures that are effective only for that short time*". The notion of a small protection window applies primarily to data, but it can in some cases be relevant for software and hardware, too.

Figure 7 illustrates how the three goals of security apply to data. In particular, confidentiality prevents unauthorized disclosure of a data item, integrity prevents unauthorized modification, and availability prevents denial of authorized access.

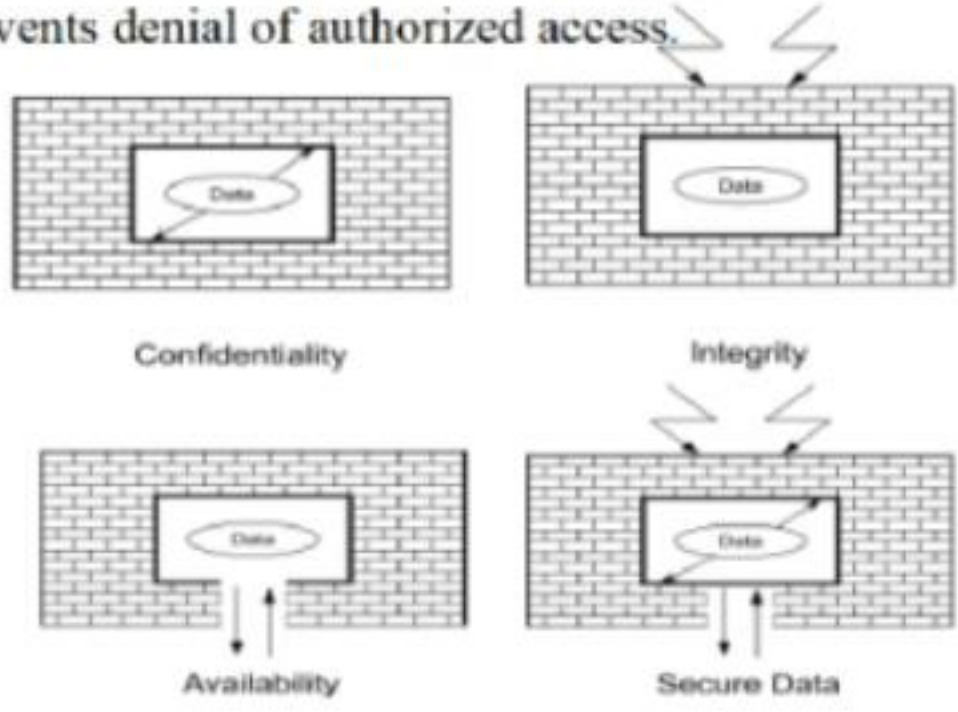

Figure 7. Security of Data.

There are many threats to a computer system, as seen in Figure 8, including human-initiated and computer-initiated ones. A human who exploits vulnerability perpetrates an attack on the system. An attack can also be launched by another system, as when one system sends an overwhelming set of messages to another, virtually shutting down the second system's ability to function. Unfortunately, we have seen this type of attack frequently, as denial-of-service attacks flood servers with more messages than they can handle. In general, we can describe the relationship among threats, controls, and vulnerabilities in this way: A threat is blocked by control of vulnerability.

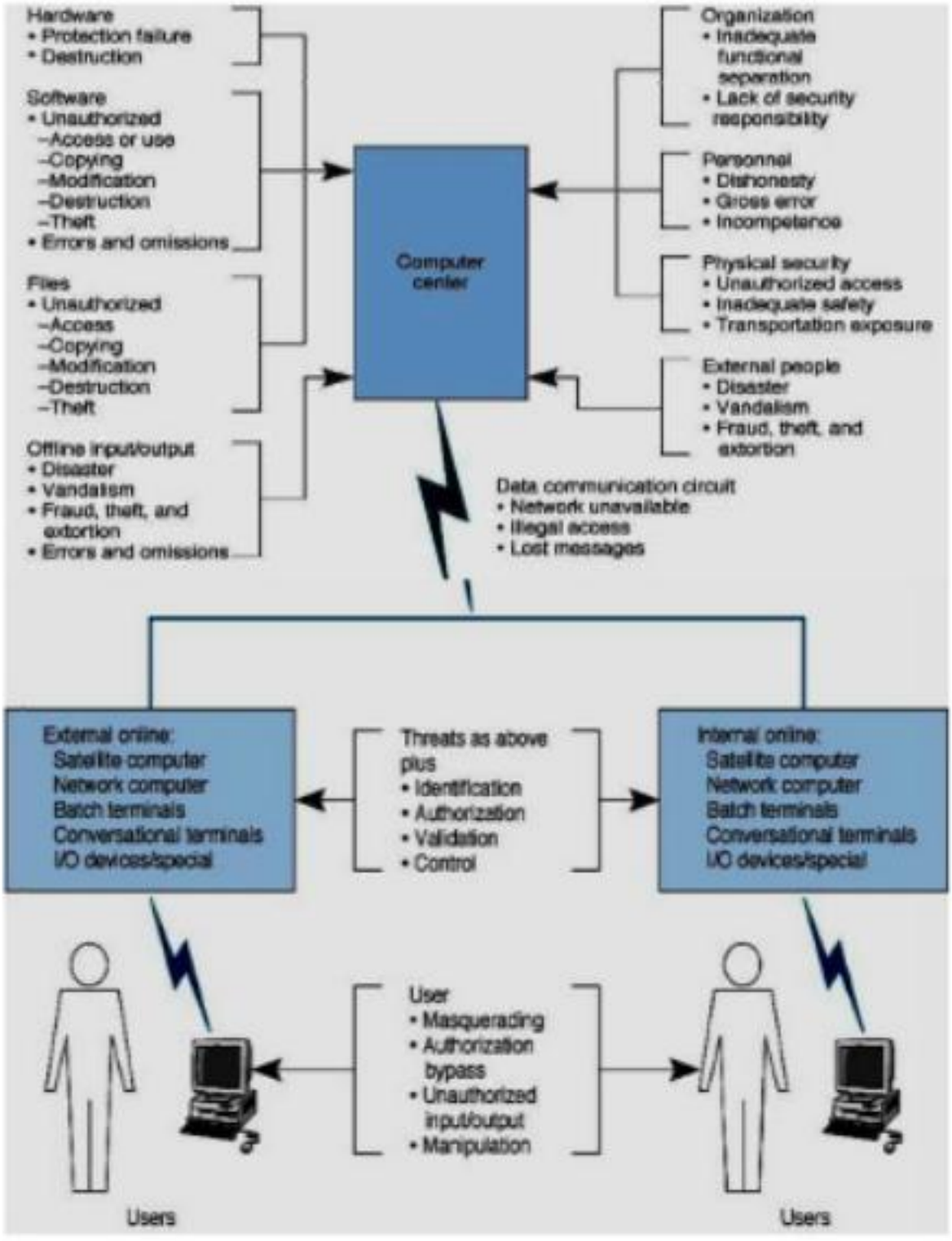

Figure 8, Example of Computer system threats

Each threat exploits vulnerabilities of the assets in computing systems. We can view any threat as being one of four kinds: interception, interruption, modification, and fabrication.;

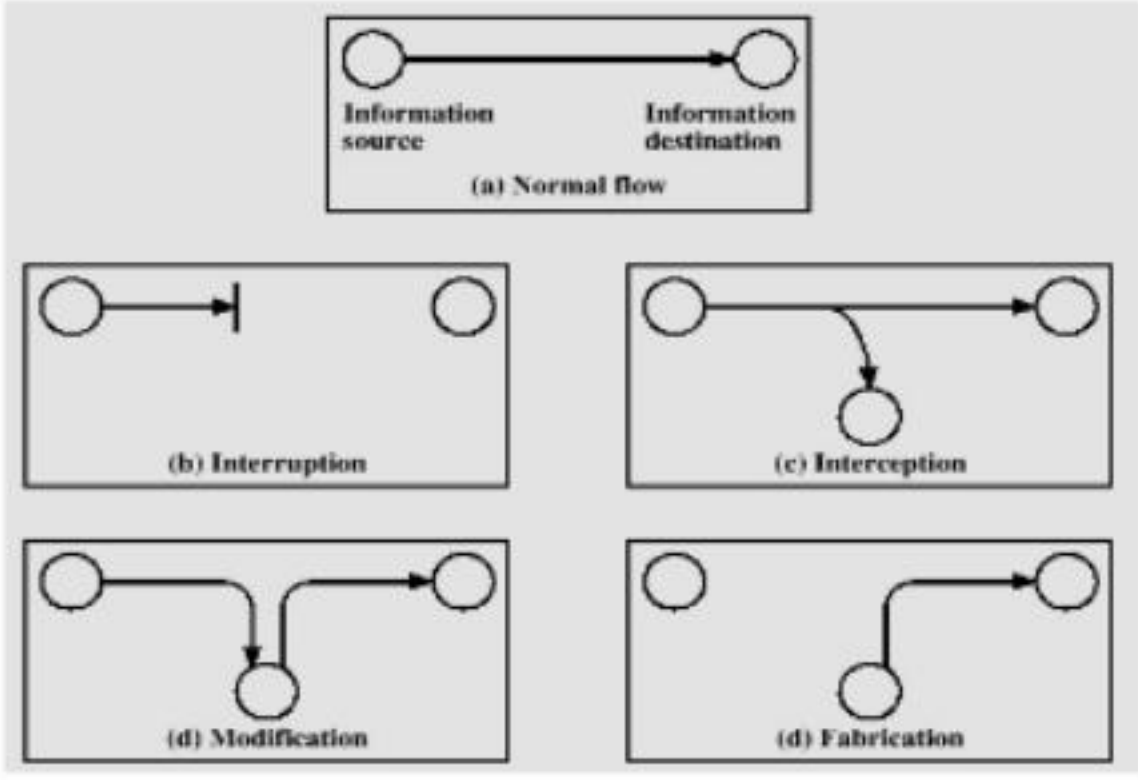

Figure 9. System Security Threats.

An interception: means that some unauthorized party has gained access to an asset. The outside party can be a person, a program, or a computing system. Examples of this type of failure are illicit copying of program or data files, or wiretapping to obtain data in a network..

An interruption, an asset of the system becomes lost, unavailable, or unusable. An example is malicious destruction of a hardware device, erasure of a program or data file, or malfunction of an operating system file manager so that it cannot find a particular disk file.

A modification is a threat occurs If an unauthorized party not only accesses but tampers with an asset. For example, someone might change the values in a database, alter a program so that it performs an additional computation, or modify data being transmitted electronically.

A fabrication: The intruder may insert spurious transactions to a network communication system or add records to an existing database. Sometimes these additions can be detected as forgeries, but if skillfully done, they are virtually indistinguishable from the real thing.

These four classes of threats interception, interruption, modification, and fabrication describe the kinds of problems we might encounter. In the next section, we look more closely at a system's vulnerabilities and how we can use them to set security goals.

NOTE THAT:

Any malicious attacker must have three things to implement his attack (MOM):

- Method: the skills, knowledge, tools, and other things with which to be able to pull off the attack

- Opportunity: the time and access to accomplish the attack

- Motive: a reason to want to perform this attack against this system

REMEMBER THAT:

THREATS ARE BLOCKED BY CONTROL OF VULNERABILITY

CHAPTER 2

CRYPTOGRAPHY

1- Meaning of Cryptography:

The word " Cryptography" is derived from a Greek words, *Kryptos*, means hidden and , *graphien*, means to write. Cryptography is the strongest tool for controlling against many kinds of security threats. Well-disguised data cannot be read, modified, or fabricated easily. Cryptography is rooted in higher mathematics: group and field theory, computational complexity, and even real analysis, not to mention probability and statistics. Fortunately, it is not necessary to understand the underlying mathematics to be able to use cryptography.

Most users of cryptography will never invent their own algorithms. There are some commonly used simple encryption methods: substitution and transposition. Also there are stronger, more sophisticated encryption algorithms. Because weak or flawed encryption provides only the illusion of protection, we also look at how encryption can fail. We analyze techniques used to break through the protective scheme and reveal the original text. Three very popular algorithms are in use today: DES, AES, and RSA.

Now, consider the steps involved in sending messages from a sender, S, to a recipient, R. If S entrusts the message to T, who then delivers it to R, T then becomes the transmission

medium. If an outsider, O, wants to access the message (to read, change, or even destroy it), we call O an interceptor or intruder. Any time after S transmits it via T, the message is vulnerable to exploitation, and O try to access the message in any of the following ways:

1- Block it, by preventing its reaching R, thereby affecting the availability of the message.

2- Intercept it, by reading or listening to the message, thereby affecting the confidentiality of the message.

3- Modify it, by seizing the message and changing it in some way, affecting the message's integrity.

4- Fabricate an authentic-looking message, arranging for it to be delivered as if it came from S, thereby also affecting the integrity of the message.

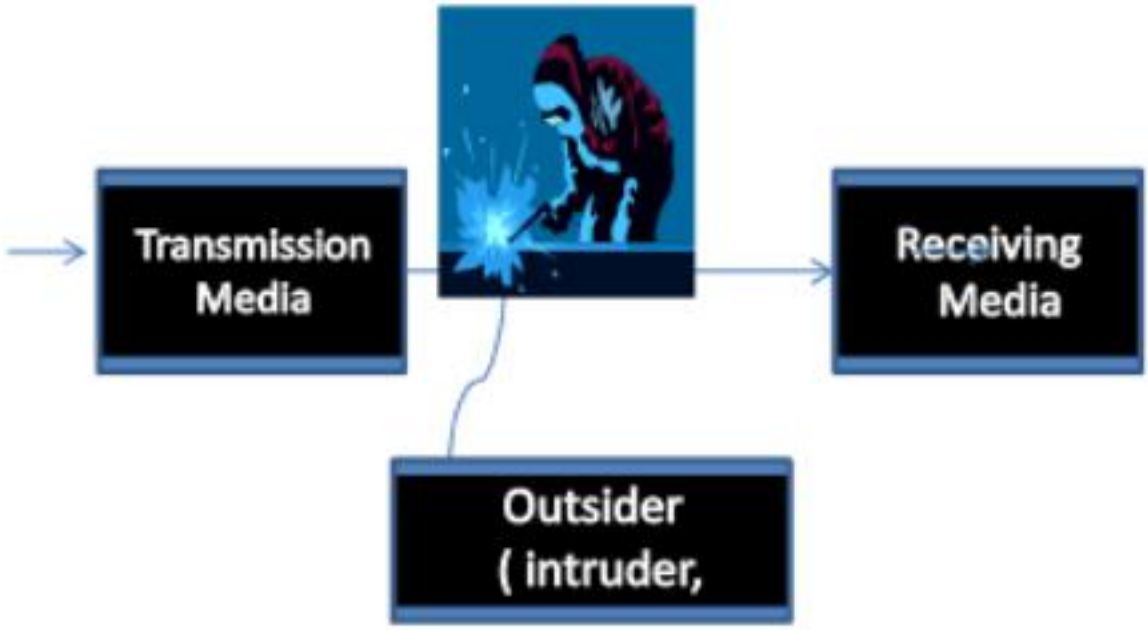

Figure 1: Cryptography System

Encryption is the process of encoding a message so that its meaning is not obvious; decryption is the reverse process, transforming an encrypted message back into its normal, original form. Alternatively, the terms encode and decode or encipher and decipher are used instead of encrypt and decrypt. That is, we say that we encode, encrypt, or encipher the original message to hide its meaning. Then, we decode, decrypt, or decipher it to reveal the original message. A system for encryption and decryption is called a cryptosystem.

The original form of a message is known as plaintext, and the

encrypted form is called ciphertext. This relationship is shown in Figure 1.

We use this formal notation to describe the transformations between plaintext and ciphertext. For example, we write C = E(P) and P = D(C), where C represents the ciphertext, E is the encryption rule, P is the plaintext, and D is the decryption rule. What we seek is a cryptosystem for which P = D(E(P)). In other words, we want to be able to convert the message to protect it from

an intruder, but we also want to be able to get the original message back so that the receiver can read it properly.

Encryption Algorithms

The cryptosystem involves a set of rules for how to encrypt the plaintext and how to decrypt the ciphertext. The encryption and decryption rules, called algorithms, often use a device called a key, denoted by K, so that the resulting ciphertext depends on the original plaintext message, the algorithm, and the key value. We write this dependence as C = E(K, P). Essentially, E is a set of encryption algorithms, and the key K selects one specific algorithm from the set. We see later in this chapter that a cryptosystem, such as the Caesar cipher, is keyless but that keyed encryptions are more difficult to break.

This process is similar to using mass-produced locks in houses. As

homeowner, it would be very expensive for you to contract with someone to invent and make a lock just for your house. In addition, you would not know whether a particular inventor's lock was really solid or how it compared with those of other inventors. A better solution is to have a few well-known, well-respected companies producing standard locks that differ according to the (physical) key. Then, you and your neighbor might have the same model of lock, but your key will open only your lock. In the same way, it is useful to have a few well-examined encryption algorithms that everyone could use, but the differing keys would prevent someone from breaking into what you are trying to protect. Sometimes the encryption and decryption keys are the same, so:

$$P = D(K, E(K,P)).$$

This form is called symmetric encryption because D and E are mirror-image processes. At other times, encryption and decryption keys come in pairs. Then, a decryption key, KD, inverts the encryption of key KE so that:

$$P = D(KD, E(KE,P))$$

Encryption algorithms of this form are called "asymmetric" because converting C back to P involves a series of steps and a key that are

different from the steps and key of E. The difference between symmetric and asymmetric encryption is shown in Figure2.

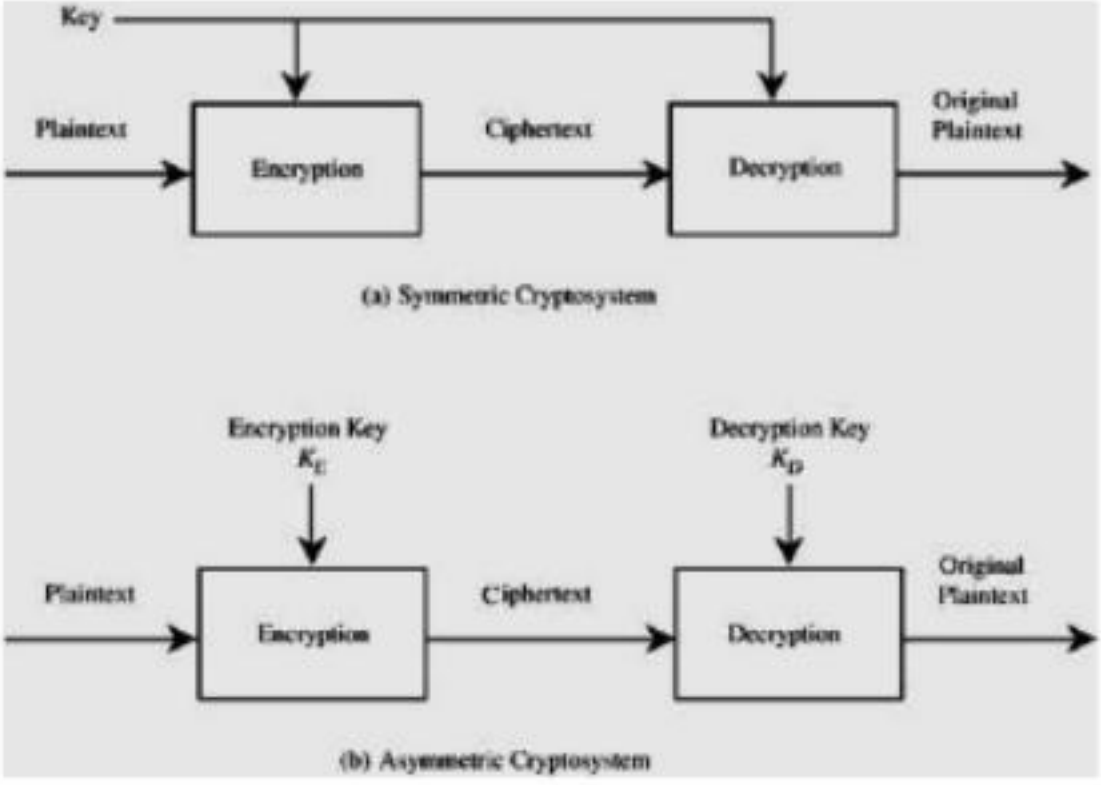

Figure 2. Encryption with Keys.

A key gives us flexibility in using an encryption scheme. We can create different encryptions of one plaintext message just by changing the key. Moreover, using a key provides additional security. If the encryption algorithm should fall into the interceptor's hands, future messages can still be kept secret because the interceptor will not know the key value. Both a cryptographer and a cryptanalyst attempt to translate coded material back to its original form. Normally, a

cryptographer works on behalf of a legitimate sender or receiver, whereas a cryptanalyst works on behalf of an unauthorized interceptor. Finally, cryptology is the research into and study of encryption and decryption; it includes both cryptography and cryptanalysis.

Cryptanalysis

A cryptanalyst's chore is to break an encryption. That is, the cryptanalyst attempts to deduce the original meaning of a ciphertext message. Better yet, he or she hopes to determine which decrypting algorithm matches the encrypting algorithm so that other messages encoded in the same way can be broken. For instance, suppose two countries are at war and the first country has intercepted encrypted messages of the second.

Cryptanalysts of the first country want to decipher a particular message so that the first country can anticipate the movements and resources of the second. But it is even better to discover the actual decryption algorithm; then the first country can easily break the encryption of all messages sent by the second country.

Thus, a cryptanalyst can attempt to do any or all of six different things:

1- Break a single message Recognize patterns in encrypted messages, to be able to break subsequent ones by applying a straightforward

decryption algorithm

1- Infer some meaning without even breaking the encryption, such as noticing an unusual frequency of communication or determining something by whether the communication was short or long

2- Deduce the key, to break subsequent messages easily

3- Find weaknesses in the implementation or environment of use of encryption

4- Find general weaknesses in an encryption algorithm, without necessarily having intercepted any messages

2- Substitution Ciphers :

In substitutions, the alphabet is scrambled, and each plaintext letter maps to a unique ciphertext letter. This technique is called a monoalphabetic cipher or simple substitution. A substitution is an acceptable way of encrypting text. In this section, we study several kinds of substitution ciphers.

2-1 The Caesar Cipher

The Caesar cipher has an important place in history. Julius Caesar is said to have been the first to use this scheme, in which each letter is translated to the letter a fixed number of places after it in the alphabet. Caesar used a shift of 3, so plaintext letter pi was enciphered as

ciphertext letter c_i by the rule

$c_i = E(p_i) = p_i + 3$

Using this encryption, the message

MANSOURA UNIVERSITY

would be encoded as

M A N S O U R A U N I V E R S I T Y

jxkplrx rkfsbopfqv

Advantages and Disadvantages of the Caesar Cipher

Most ciphers, and especially the early ones, had to be easy to perform in the field. In particular, it was dangerous to have the cryptosystem algorithms written down for the soldiers or spies to follow. Any cipher that was so complicated that its algorithm had to be written out was at risk of being revealed if the interceptor caught a sender with the written instructions. Then, the interceptor could readily decode any ciphertext messages intercepted (until the encryption algorithm could be changed).

The Caesar cipher is quite simple. During Caesar's lifetime, the simplicity did not dramatically compromise the safety of the encryption because anything written, even in plaintext, was rather well protected; few people knew how to read. The

pattern pi + 3 was easy to memorize and implement. A sender in the field could write out a plaintext and a ciphertext alphabet,

encode a message to be sent, and then destroy the paper containing the alphabets. Sidebar 2-3 describes actual use of a cipher similar to the Caesar cipher.

Its obvious pattern is also the major weakness of the Caesar cipher. A secure encryption should not allow an interceptor to use a small piece of the ciphertext to predict the entire pattern.

Cryptanalysis of the Caesar Cipher

Let us take a closer look at the result of applying Caesar's encryption technique to "MANSOURA UNIVERSITY." If we did not know the plaintext and were trying to guess it, we would have many clues from the ciphertext. For example, the break between the two words is preserved in the ciphertext, and double letters are preserved: The SS is translated to vv. We might also notice that when a letter is repeated, it maps again to the same ciphertext as it did previously. So the letters T, I, and E always translate to w, l, and h. These clues make this cipher easy to break.

2-2 Playfair

This cipher uses a 5 by 5 grid, in which the alphabet is placed, permuted by the keyword, and omitting the letter J (Figure 3). The plaintext is first conditioned by replacing J with I wherever it occurs, then dividing it into letter pairs, preventing double letters occurring in a pair by separating them with an x, and finally adding a z if necessary to complete the last letter pair. The example Playfair wrote on his napkin was ~~"L~~ord Granville's letter," which becomes ~~"l~~o rd gr an vi lx le sl et te rz".

It is then enciphered two letters at a time using the following rules:

• If two letters are in the same row or column, they are replaced by the succeeding letters. For example, ~~"a~~m" enciphers to ~~"L~~E."

• Otherwise, the two letters stand at two of the corners of a rectangle in the table, and we replace them with the letters at the other two corners of this rectangle. For example, ~~"l~~o" enciphers to ~~"M~~T." We can now encipher our specimen text as shown in Figure 3

P	A	L	M	E
R	S	T	O	N
B	C	D	F	G
H	I	K	Q	U
V	W	X	Y	Z

Plain:	lo rd gr an vi lx le sl et te rz
Cipher:	MT TB BN ES WH TL MP TA LN NL NV

Figure 3. Playfair encryption example

Variants of this cipher were used by the British army as a field cipher in World War I, and by the Americans and Germans in World War II. It's a substantial improvement on Vigenère, as the statistics an analyst can collect are of digraphs (letter pairs) rather than single letters, so the distribution is much flatter, and more ciphertext is needed for an attack. Again, it's not enough for the output of a block cipher to just look intuitively ~~"r~~andom." Playfair ciphertexts do look random, but they have the property that if you change a single letter of a plaintext pair, then often only a single letter of the ciphertext will change.

Cryptanalysis of Substitution Ciphers

The techniques described for breaking the Caesar cipher can also be used on other substitution ciphers. Short words, words with repeated patterns, and common initial and final letters all give clues for guessing the permutation. For a long message, this process can be extremely tedious. Fortunately, there are other approaches to breaking an encryption. In fact, analysts apply every technique at their disposal, using a combination of guess, strategy, and mathematical skill.

One-Time Pads

A one-time pad is sometimes considered the perfect cipher. The name comes from an encryption method in which a large, nonrepeating set of keys is written on sheets of paper, glued together into a pad. For example, if the keys are 20 characters long and a sender must transmit a message 300 characters in length, the sender would tear off the next 15 pages of keys. The sender would write the keys one at a time above the letters of the plaintext and encipher the plaintext with a prearranged chart (called a Vigenère tableau) that has all 26 letters in each column, in some scrambled order. The sender would then destroy the used keys.

For the encryption to work, the receiver needs a pad identical to that of the sender. Upon receiving a message, the receiver takes the appropriate number of keys and deciphers the message as if it were a plain substitution with a long key. Essentially, this algorithm gives the effect of a key as long as the number of characters in the pad.

The one-time pad method has two problems: the need for absolute synchronization between sender and receiver, and the need for an unlimited number of keys. Although generating a large number of random keys is no problem, printing, distributing, storing, and accounting for such keys are problems.

Long Random Number Sequences

A close approximation of a one-time pad for use on computers is a random number generator. In fact, computer random numbers are not random; they really form a sequence with a very long period (that is, they go for a long time before repeating the sequence). In practice, a generator with a long period can be acceptable for a limited amount of time or plaintext.

To use a random number generator, the sender with a 300-character message would interrogate the computer for the next 300 random numbers, scale them to lie between 0 and 25, and use one number to encipher each character of the plaintext message.

The Vernam Cipher

The Vernam cipher is a type of one-time pad devised by Gilbert Vernam for AT&T. The Vernam cipher is immune to most cryptanalytic attacks. The basic encryption involves an arbitrarily long nonrepeating sequence of numbers that are combined with the plaintext. Vernam's invention used an arbitrarily long punched paper tape that fed into a teletype machine. The tape contained random numbers that were combined with characters typed into the teletype. The sequence of random numbers had no repeats, and each tape was used only

once. As long as the key tape does not repeat or is not reused, this type of cipher is immune to cryptanalytic attack because the available ciphertext does not display the pattern of the key. A model of this process is shown in Figure 4

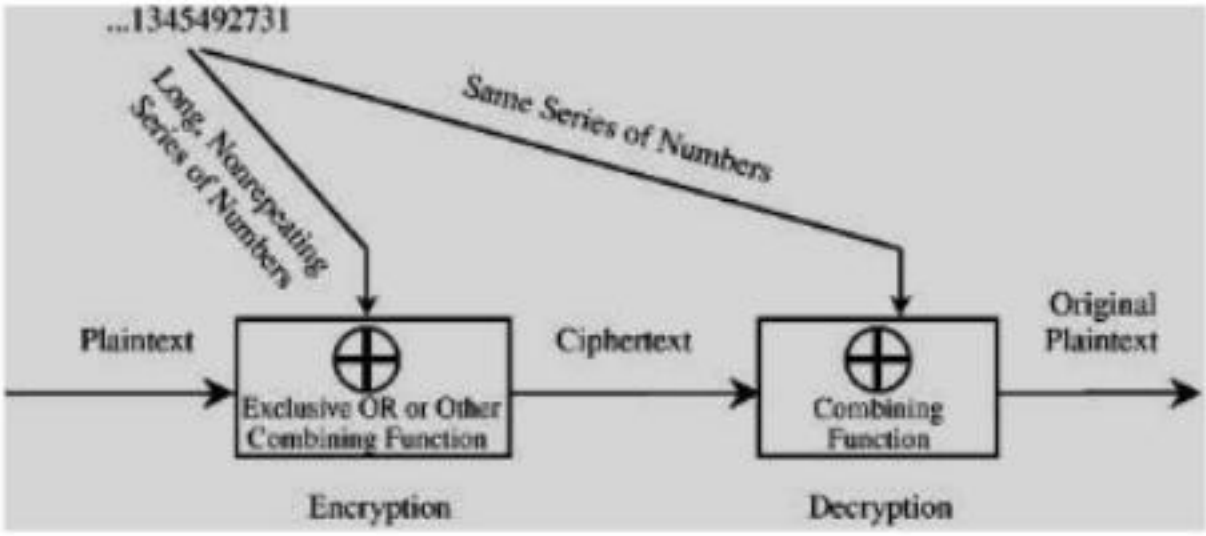

Figure 4. Vernam Cipher.

To see how this method works, we perform a simple Vernam encryption. Assume that the alphabetic letters correspond to their counterparts in arithmetic notation mod 26. That is, the letters are represented with numbers 0 through 25. To use the Vernam cipher, we sum this numerical representation with a stream of random two-digit numbers. For instance, if the message is

VERNAM CIPHER

the letters would first be converted to their numeric equivalents, as shown here:

Plaintext	V	E	R	N	A	M	C	I	P	H	E	R
Numeric Equivalent	21	4	17	13	0	12	2	8	15	7	4	17
+ Random Number	76	48	16	82	44	3	58	11	60	5	48	88
= Sum	97	52	33	95	44	15	60	19	75	12	52	105
= mod 26	19	0	7	17	18	15	8	19	23	12	0	1
Ciphertext	t	a	h	r	s	p	i	t	x	m	a	b

Next, we generate random numbers to combine with the letter codes. Suppose the following series of random two-digit numbers is generated.

76 48 16 82 44 03 58 11 60 05 48 88

The encoded form of the message is the sum mod 26 of each coded letter with the corresponding random number. The result is then encoded in the usual base-26 alphabet representation.

Thus, the message "VERNAM CIPHER " is encoded as "tahrsp itxmab"

Book Ciphers

Another source of supposedly "random" numbers is any book, piece of music, or other object of which the structure can be analyzed. Both the sender and receiver need access to

identical objects. For example, a possible one-time pad can be based on a telephone book. The sender and receiver might agree to start at page 35 and use two middle digits (ddd-DDdd) of each seven-digit phone number, mod 26, as a key letter for a substitution cipher. They use an already agreed-on table (a Vigenère tableau) that has all 26 letters in each column, in some scrambled order.

It would seem as though this cipher, too, would be impossible to break. Unfortunately, that is not true. The flaw lies in the fact that neither the message nor the key text is evenly distributed; in fact, the distributions of both cluster around high-frequency letters. For example, the four letters A, E, O, and T account for approximately 40 percent of all letters used in standard English text. Each ciphertext letter is really the intersection of a plaintext letter and a key letter. But if the probability of the plaintext or the key letter's being A, E, O, or T is 0.4, the probability of both being one of the four is 0.4 * 0.4 = 0.16, or nearly one in six. Using the top six letters (adding N and I) increases the sum of the frequencies to 50 percent and thus increases the probability for a pair to 0.25, or one in four.

Table 1: Vigenere Table

	0					5					10					15					20					25	
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	1
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	2
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	3
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	4
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	5
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	6
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	7
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	8
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	9
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	10
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	11
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	12
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	13
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	14
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	15
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	16
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	17
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	18
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	19
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	20
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	21
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	22
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	23
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	24
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	25

Substitutions are effective cryptographic devices. In fact, they were the basis of many cryptographic algorithms used for diplomatic communication through the first half of the twentieth century. Because they are interesting and intriguing, But substitution is not the only kind of encryption technique. In the next section, we introduce the other basic cryptographic

invention: the transposition (permutation). Substitutions and permutations together form a basis for some widely used commercial-grade encryption algorithms.

3- Transpositions (Permutations) :

A transposition is an encryption in which the letters of the message are rearranged. With transposition, the cryptography aims for diffusion, widely spreading the information from the message or the key across the ciphertext. Transpositions try to break established patterns. Because a transposition is a rearrangement of the symbols of a message, it is also known as

a permutation.

Columnar Transpositions

As with substitutions, we begin this study of transpositions by examining a simple example. The columnar transposition is a rearrangement of the characters of the plaintext into columns.

The following set of characters is a five-column transposition. The plaintext characters are written in rows of five and arranged one row after another, as shown here.

c_1 c_2 c_3 c_4 c_5

c_6 c_7 c_8 c_9 c_{10}

c_{11} c_{12}..... etc

You form the resulting ciphertext by reading down the columns, as shown in Figure 5.

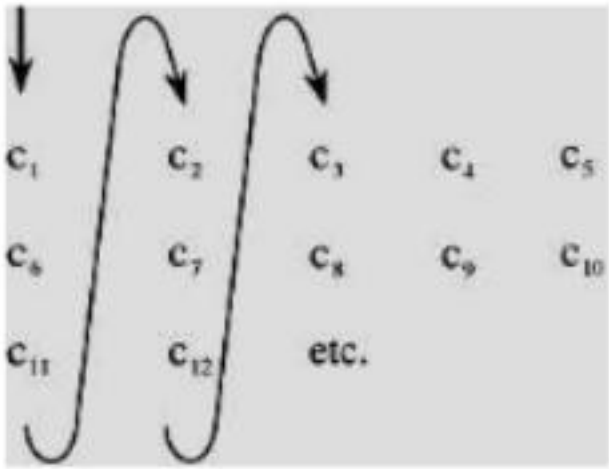

Figure 5. Columnar Transposition.

For instance, suppose you want to write the plaintext message THIS IS A MESSAGE TO SHOW HOW A COLUMNAR TRANSPOSITION WORKS. We arrange the letters in five columns as

T H I S I
S A M E S
S A G E T
O S H O W
H O W A C
O L U M N

A R T R N

S P O S I

T I O N W

O R K S

The resulting ciphertext would then be read down the columns as

tssoh oaniw haaso lrsto imghw

utpir seeoa mrook istwc nasns

In this example, the length of this message happens to be a multiple of five, so all columns are the same length. However, if the message length is not a multiple of the length of a row, the last columns will be one or more letters short. When this happens, we sometimes use an infrequent letter, such as X, to fill in any short columns.

This cipher involves no additional work beyond arranging the letters and reading them off again. Therefore, the algorithm requires a constant amount of work per character, and the time needed to apply the algorithm is proportional to the length of the message. However, we must also consider the amount of space needed to record or store the ciphertext. So far, the other ciphers we have seen require only a constant amount of space (admittedly up to 262

locations). But the columnar transposition algorithm requires storage for all characters of the message, so the space required is not constant; it depends directly on the length of the message.

5- Block Ciphers

Block ciphers operate on ~~–e~~hunks," or blocks, of a message and apply the encryption algorithm to an entire message block at the same time. The transposition ciphers are examples of block

ciphers. The simple algorithm used in the challenge-response algorithm takes an entire word and reverses its letters. The more complicated columnar transposition cipher works on an entire message (or a piece of a message) and encrypts it using the transposition algorithm and a secret keyword. Most modern encryption algorithms implement some type of block cipher.

6- Stream Ciphers

Stream ciphers are ciphers that operate on each character or bit of a message (or data stream) one character/bit at a time. The Caesar cipher is an example of a stream cipher. The one-time

pad is also a stream cipher because the algorithm operates on each letter of the plaintext message independently. Stream ciphers can also function as a type of block cipher. In such operations there is a buffer that fills up to real-time data that is then encrypted as a block and transmitted to the recipient.

7-Hash Functions

Later in this chapter, you'll learn how cryptosystems implement digital signatures to provide proof that a message originated from a particular user of the cryptosystem and to ensure that the message was not modified while in transit between the two parties. Before you can completely understand that concept, we must first explain the concept of *hash functions*. This section explores the basics of hash functions and looks at several common hash functions used in modern digital signature algorithms.

Hash functions have a very simple purpose—they take a potentially long message and generate a unique output value derived from the content of the message. This value is commonly referred to as the *message digest*.

Message digests can be generated by the sender of a message and transmitted to the recipient along with the full message for two reasons. First, the recipient can use the same hash function to recompute the message digest from the full message. They can then compare the computed message digest to the transmitted one to ensure that the message sent by the originator is the same one received by the recipient. If the message digests do not match, it indicates that the message was somehow modified while in transit. Second, the message digest can be used to implement a digital signature algorithm.

In most cases, a message digest is 128 bits or larger. However, a single-digit value can be used to perform the function of parity, a low-level or single-digit checksum value used to provide a single individual point of verification. In most cases, the longer the message digest, the more reliable

its verification of integrity.

According to RSA Security, there are five basic requirements for a cryptographic hash function:

- The input can be of any length.
- The output has a fixed length.

- The hash function is relatively easy to compute for any input.

-The hash function is one-way (meaning that it is extremely hard to determine the input when provided with the output

_ The hash function is collision free (meaning that it is extremely hard to find two messages that produce the same hash value).

The *Secure Hash Algorithm (SHA)* and its successor, SHA-1, are government standard hash functions developed by the National Institute of Standards and Technology (NIST) and are specified in an official government publication—the Secure Hash Standard (SHS), also known as Federal Information Processing Standard (FIPS) 180.

SHA-1 takes an input of virtually any length (in reality, there is an upper bound of approximately 2,097,152 terabytes on the algorithm) and produces a 160-bit message digest. Due to the mathematical structure of the hashing algorithm, this provides 80 bits of protection against collision attacks. The SHA-1 algorithm processes a message in 512-bit blocks. Therefore, if the message length is not a multiple of 512, the SHA algorithm pads the message with

additional data until the length reaches the next highest multiple of 512.

Although SHA-1 is the current official standard for federal government applications, it is not quite strong enough. It was designed to work with the old Data Encryption Standard (DES) and its follow-on, Triple DES (3DES). The new Advanced Encryption Standard (described in the preceding chapter) supports key lengths of up to 256 bits. Therefore, the government is currently evaluating three new hash functions to replace SHA-1 in the near future:

-SHA-256 produces a 256-bit message digest and provides 128 bits of protection against collision attacks.

-SHA-512 produces a 512-bit message digest and provides 256 bits of protection against collision attacks.

- SHA-384 uses a truncated version of the SHA-512 hash to produce a 384-bit digest that supports 192 bits of protection against collision attacks.

MD2

The *MD2 (Message Digest 2)* hash algorithm was developed by Ronald Rivest (the same Rivest of Rivest, Shamir, and Adleman fame) in 1989 to provide a secure hash function for 8-bit processors. MD2 pads the message

so that its length is a multiple of 16 bytes. It then computes a 16-byte checksum and appends it to the end of the message. A 128-bit message digest is then generated by using the entire original message along with the appended checksum. Cryptanalytic attacks exist against improper implementations of the MD2 algorithm. Specifically,
Nathalie Rogier and Pascal Chauvaud discovered that if the checksum is not appended to the message before digest computation, collisions may occur.

MD4

The next year, in 1990, Rivest enhanced his message digest algorithm to support 32-bit processors and increase the level of security. This enhanced algorithm is known as *MD4*. It first pads the message to ensure that the message length is 64 bits smaller than a multiple of 512 bits. For example, a 16-bit message would be padded with 432 additional bits of data to make it 448 bits, which is 64 bits smaller than a 512-bit message. The MD4 algorithm then processes 512-bit blocks of the message in three rounds of computation. The final output is a 128-bit message digest.

MD5

In 1991, Rivest released the next version of his message digest algorithm, which he called *MD5*.

It also processes 512-bit blocks of the message, but it uses four distinct rounds of computation to produce a digest of the same length as the MD2 and MD4 algorithms (128 bits). MD5 has the same padding requirements as MD4—the message length must be 64 bits less than a multiple of 512 bits.

MD5 implements additional security features that reduce the speed of message digest production significantly. Cryptanalysts have not yet proven that the full MD5 algorithm is vulnerable to collisions, but many experts suspect that such a proof may not be far away. However, MD5 is the strongest of Rivest's algorithms and remains in use today. MD5 is commonly seen in use in relation to file downloads, such as updates and patches, so the recipient can verify the integrity of a file after downloading and before installing or applying it to any system. Table 10.1 lists well-known hashing algorithms and their resultant hash value lengths in bits. Earmark this page for memorization.

Table 1: Hash algorithms

Name	Hash Value Length
Secure Hash Algorithm (SHA-1)	160
Message Digest 5 (MD5)	128
Message Digest 4 (MD4)	128
Message Digest 2 (MD2)	128
HMAC (Hash Message Authenticating Code)	variable
HAVAL (Hash of Variable Length) —an MD5 variant	128, 160, 192, 224, and 256 bits

8- The Data Encryption Standard

The United States government published the *Data Encryption Standard (DES)* in 1977 as a proposed standard cryptosystem for all government communications. Indeed, many government entities continue to use DES for cryptographic applications today, despite the fact that it was superseded by the *Advanced Encryption Standard (AES)* in December 2001. DES is a 64-bit block cipher that has four modes of operation: Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher

Feedback (CFB) mode, and Output Feedback (OFB) mode. These modes are explained in the following sections. All of the DES modes operate on 64 bits of plaintext at a time to generate 64-bit blocks of ciphertext. The key used by DES is 56 bits long.

DES utilizes a long series of exclusive OR (XOR) operations to generate the ciphertext. This process is repeated 16 times for each encryption/decryption operation. Each repetition is commonly referred to as a ~~-r~~ound" of encryption, explaining the statement that DES performs 16 rounds of encryption. In the following sections, we'll take a look at each of the four modes utilized by DES.

The DES algorithm is a careful and complex combination of two fundamental building blocks of encryption: substitution and transposition. The algorithm derives its strength from repeated application of these two techniques, one on top of the other, for a total of 16 cycles. The sheer complexity of tracing a single bit through 16 iterations of substitutions and transpositions has so far stopped researchers in the public from identifying more than a handful of general properties of the algorithm.

The algorithm begins by encrypting the plaintext as blocks of 64 bits. The key is 64 bits long, but in fact it can be any 56-bit number. (The extra 8 bits are often used as check digits and do not affect encryption in normal implementations.) The user can change the key at will any time there is uncertainty about the security of the old key. The algorithm accomplishes two things: ensuring that the output bits have no obvious relationship to the input bits and spreading the effect of one plaintext bit to other bits in the ciphertext. Substitution provides the confusion, and transposition provides the diffusion. In general, plaintext is affected by a series of cycles of a substitution then a permutation. The iterative substitutions and permutations are performed as outlined in Figure 5.

DES uses only standard arithmetic and logical operations on numbers up to 64 bits long, so it is suitable for implementation in software on most current computers. Although complex, the algorithm is repetitive, making it suitable for implementation on a single-purpose chip. In fact, several such chips are available on the market for use as basic components in devices that use DES encryption in an application.

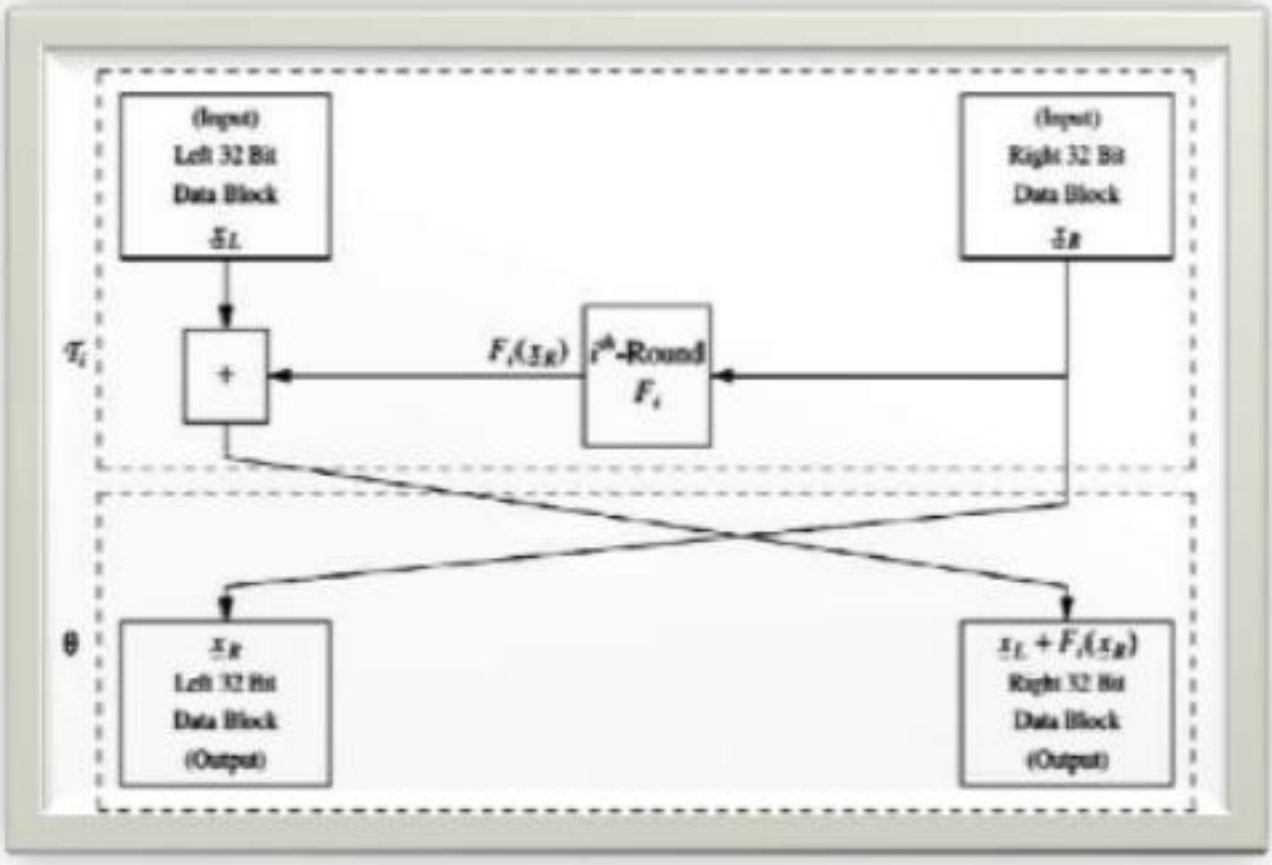

Figure 5. Cycles of Substitution and Permutation.

Double DES

As computing power has increased rapidly over the last few decades, and it promises to continue to do so. For this reason, the DES 56-bit key length is not long enough for some people to feel comfortable. Since the 1970s, researchers and practitioners have been interested in a longer-key version of DES. But we have a problem: The DES algorithm is fixed for a 56-bit key.

To address the discomfort, some researchers suggest using a double encryption for greater secrecy. The double encryption works in the following way. Take two keys, k1 and k2, and perform two encryptions, one on top of the other: E(k2, E(k1,m)). In theory, this approach should multiply the difficulty of breaking the encryption, just as two locks are harder to pick than one. Unfortunately, that assumption is false. Merkle and Hellman showed that two encryptions are no better than one. The basis of their argument is that the cryptanalyst works plaintext and ciphertext toward each other. The analyst needs two pairs of plaintext (call them P1 and P2) and corresponding ciphertext, C1 and C2, but not the keys used to encrypt them. The analyst computes and saves P1 encrypted under each possible key. The analyst then tries decrypting C1 with a single key and looking for a match in the saved Ps. A match is a possible pair of double keys, so the analyst checks the match with P2 and C2. Computing all the Ps takes 256 steps, but working backward from C1 takes only the same amount of time, for a total of 2 * 256 or 257, equivalent to a 57-bit key. Thus, the double encryption only doubles the work for the attacker.

Triple DES (3DES)

As mentioned in previous sections, the Data Encryption Standard's 56-bit key is no longer considered adequate in the face of modern cryptanalytic techniques and supercomputing power. However, an adapted version of DES, *Triple DES (3DES),* uses the same algorithm to produce a more secure encryption There are four versions of 3DES. The first simply encrypts the plaintext three times, using three different keys: K1, K2, and K3. It is known as DES-EEE3 mode (the *E*s indicate that there are three encryption operations, whereas the numeral 3 indicates that three different keys are used). DES-EEE3 can be expressed using the following notation, where E(K,P) represents the encryption of plaintext *P* with key *K*:

E(K1,E(K2,E(K3,P)))

DES-EEE3 has an effective key length of 168 bits.

The second variant (DES-EDE3) also uses three keys but replaces the second encryption operation with a decryption operation:

E(K1,D(K2,E(K3,P)))

The third version of 3DES (DES-EEE2) uses only two keys, K1 and K2, as follows:

E(K1,E(K2,E(K1,P)))

The fourth variant of 3DES (DES-EDE2) also uses two keys but uses a decryption operation in the middle:

E(K1,D(K2,E(K1,P)))

Both the third and fourth variants have an effective key length of 112 bits.

However, a simple trick does indeed enhance the security of DES. Using three keys adds significant strength. The so-called triple DES procedure is C = E(k3, E(k2, E(k1,m))). That is, you encrypt with one key, decrypt with the second, and encrypt with a third. This process gives a strength equivalent to a 112-bit key (because the double DES attack defeats the strength of one of the three keys).

A minor variation of triple DES, which some people also confusingly call triple DES, is C = E(k1, D(k2, E(k1,m))). That is, you encrypt with one key, decrypt with the second, and encrypt with the first again. This version requires only two keys. (The second decrypt step also makes this process work for single encryptions with one key: The decryption cancels the first encryption, so the net result is one encryption.) This approach is subject to another tricky attack, so its strength is rated at only about 80 bits.

In summary, ordinary DES has a key space of 56 bits, double DES is scarcely better, but two-key triple DES gives an effective length of 80 bits, and three-key triple DES gives a strength of 112 bits. Now, over three decades after the development of DES, a 56-bit key is inadequate for any serious confidentiality, but 80- and 112-bit effective key sizes

provide reasonable security.

Security of the DES Since its was first announced, DES has been controversial. Many researchers have questioned the security it provides. Much of this controversy has appeared in the open literature, but certain DES features have neither been revealed by the designers nor inferred by outside analysts.

9- The AES Encryption Algorithm

The AES is likely to be the commercial-grade symmetric algorithm of choice for years, if not decades. Let us look at it more closely.

In January 1997, NIST called for cryptographers to develop a new encryption system. As with the call for candidates from which DES as selected, NIST made several important restrictions. The algorithms had to be

- Unclassified
- Publicly disclosed
- Available royalty-free for use worldwide
- Symmetric block cipher algorithms, for blocks of 128 bits
- Usable with key sizes of 128, 192, and 256 bits

In August 1998, fifteen algorithms were chosen from among those submitted; in August 1999, the field of candidates was narrowed to five finalists. The five then underwent extensive public and private scrutiny. The final selection was made on the basis not only of security but also of cost or efficiency of operation and ease of implementation in software. The winning algorithm, submitted by two Dutch cryptographers, was Rijndael. Vincent Rijmen and described the four not chosen as also having adequate security for the AES no cryptographic flaws were identified in any of the five. Thus, the selection was based on efficiency and implementation characteristics.)

The Rijndael cipher allows the use of three key strengths: 128 bits, 192 bits, and 256 bits. The original specification for AES called for the processing of 128-bit blocks, but Rijndael exceeded this specification, allowing cryptographers to use a block size equal to the key length. The number of encryption rounds depends upon the key length chosen:

- 128-bit keys require 9 rounds of encryption.
- 192-bit keys require 11 rounds of encryption.
- 256-bit keys require 13 rounds of encryption.

Rijndael is a fast algorithm that can be implemented easily on simple

processors. Although it has a strong mathematical foundation, it primarily uses substitution; transposition; and the shift, exclusive OR, and addition operations. Like DES, AES uses repeat cycles. There are 10, 12, or 14 cycles for keys of 128, 192, and 256 bits, respectively. In Rijndael, the cycles are called "rounds." Each cycle consists of four steps.

- Byte substitution: This step uses a substitution box structure similar to the DES, substituting each byte of a 128-bit block according to a substitution table. This is a straight diffusion operation.
- Shift row: A transposition step. For 128- and 192-bit block sizes, row n is shifted left circular (n - 1) bytes; for 256-bit blocks, row 2 is shifted 1 byte and rows 3 and 4 are shifted 3 and 4 bytes, respectively. This is a straight confusion operation.
- Mix column: This step involves shifting left and exclusive-ORing bits with themselves. These operations provide both confusion and diffusion.
- Add subkey: Here, a portion of the key unique to this cycle is exclusive-ORed with the cycle result. This operation provides confusion and incorporates the key.

Note that the steps perform both diffusion and confusion on

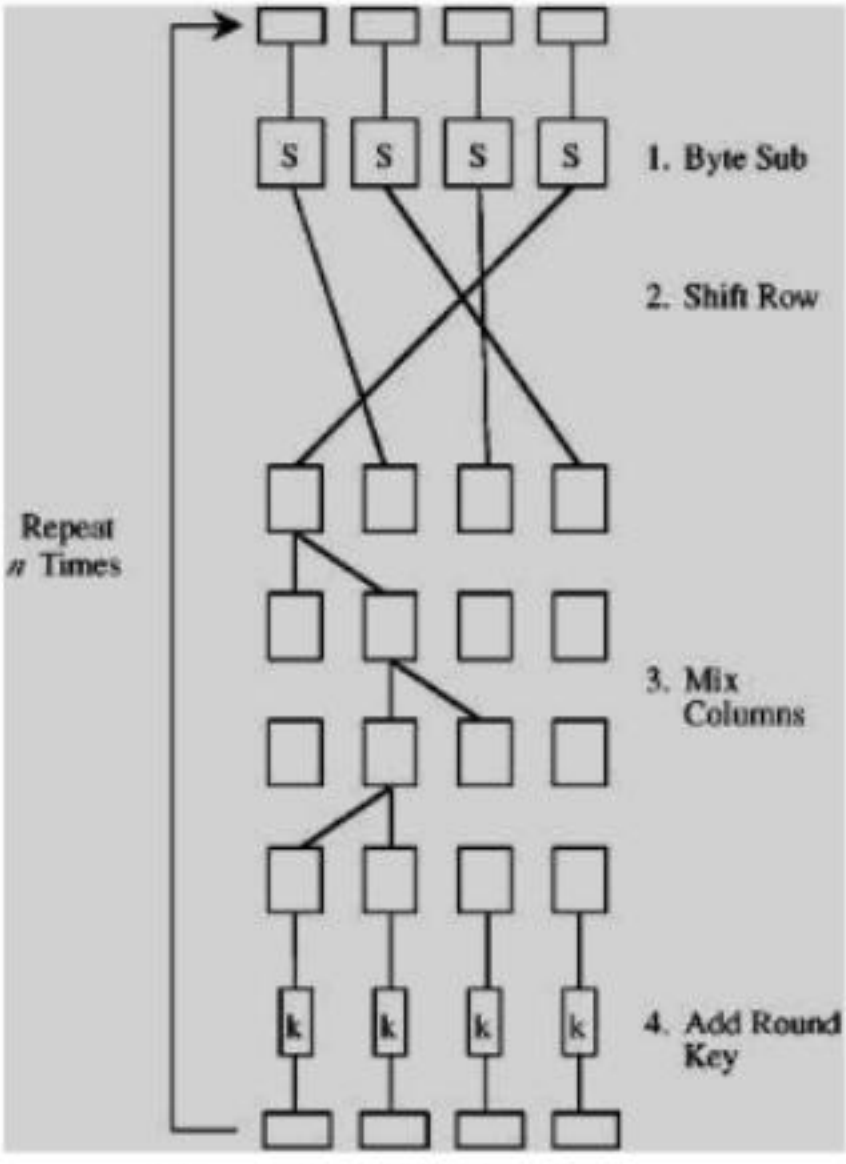

Figure 5 AES Algorithm

the input data. Bits from the key are combined with intermediate result bits frequently, so key bits are also well diffused throughout the result. Furthermore, these four steps are extremely fast. The AES algorithm is depicted in Figure 5

10-RSA

The most famous public key cryptosystem is named after its creators. In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman proposed the *RSA* public key algorithm that remains a worldwide standard today. They patented their algorithm and formed a commercial venture known as RSA Security to develop mainstream implementations of their security technology.

Today, the RSA algorithm forms the security backbone of a large number of well-known security infrastructures produced by companies like Microsoft, Nokia, and Cisco. The RSA algorithm depends upon the computational difficulty inherent in factoring large prime numbers. Each user of the cryptosystem generates a pair of public and private keys using the algorithm described in the following steps:

1.Choose two large prime numbers (approximately 200 digits each), labeled p and q.

2.Compute the product of those two numbers,

$n=p^*q$.

3.Select a number, e,that satisfies the following two requirements:

a.e is less than n.

b. e and $(n-1)(q-1)$ are relatively prime—that is, the two numbers have no common factors other than 1.

4. Find a number, d, such that $(ed-1) \bmod (p-1)(q-1) = 0$.

5. Distribute e and n as the public key to all cryptosystem users. Keep D secret as the private key.

If Alice wants to send an encrypted message to Bob, she generates the ciphertext (C) from the plaintext (P) using the following formula (where e is Bob's public key and n is the product of p and q created during the key generation process):

C = Pe mod n When Bob receives the message, he performs the following calculation to retrieve the plaintext message:

$$P = Cd \bmod n$$

COMPARISONS

1- Comparison between Symmetric and Asymmetric encryption

TERM	MEANING POTENTIAL	CONFUSION
Symmetric Methodology	-Uses one key which both encrypts and decrypts using the same symmetric encryption algorithm - The key is distributed to the two communicating parties in a secure manner prior to transfer of encrypted data	Often called private or private key methodology
Asymmetric methodology	-Uses symmetric encryption algorithms and symmetric keys to encrypt data --Uses asymmetric encryption algorithms and asymmetric keys to encrypt the symmetric key. The two keys are created and are linked together. -The symmetric key encrypted with one must be	Often called public or public key methodology

	decrypted by the other (in either direction) using the same asymmetric encryption algorithm. -The two linked asymmetric keys are created together. One must be distributed to the owner, and the other to the party which is keeping these keys (often called the CA) in a secure manner prior to transfer of data	

2- Comparison between Stream Cipher and Block Cipher:

	Stream Cipher	Block Cipher:
Advantages	- Speed of transformation - Low error propagation	-High diffusion - Immunity to insertion of symbols
Disadvantages	- Low Diffusion	-Slowness

3- Comparison between DES and Triple DES.

TYPE	DESCRIPTION
DES (Data Encryption Standard)	Popular, product cipher used by the Data Encryption Standard of the US Government 64-bit block cipher, 64-bit key (only 56 are needed), 16 rounds Operates in four modes: [redacted]CB—Electronic Code Book (native DES), using two distinct algorithms [redacted]BC—Cipher Block Chaining in which the

	encryption of each block depends upon the encryption of the previous block [redacted]FB—Output Feedback, used as a random number generator [redacted]FB—Cipher Feedback, used for message authentication codes
3-DES or Triple DES	64-bit block cipher, using the DES cipher 3 times, three distinct 56-bit keys Strong under all attacks

CHAPTER 3
PROGRAMS SECURITY

Introduction:

A program is considered secure if it can withstand and enforces the expected security goals as mentioned in chapter one. Also, takes too long time to break, it runs for a period of time without failures, or any drop in security requirements

As discussed in Chapter one that security implies some degree of trust that the program enforces expected confidentiality, integrity, and availability. From the point of view of a program or a programmer, one way to assess security or quality is to ask people to name the characteristics of software that contribute to its overall security. However, we are likely to get different answers from different people. This difference occurs because the importance of the characteristics depends on who is analyzing the software. For example, one person may decide that code is secure if it has run for a period of time with no apparent failures. But another person may decide that any potential fault in meeting security requirements makes code insecure. Another view of security is fitness for purpose. In this chapter we will discuss the standards for program security.

2-Program Faults vs Program Error:

A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most bugs arise from mistakes and errors made in either a program's source code or its design, or in components and operating systems used by such programs. A few are caused by compilers producing incorrect code.

However, in computer science technical writing, especially in software engineering, there are differences in meaning of error, fault, failure, and mistake, such that:

- *Error:* "A difference…between a computed result and the correct result"
- *Fault:* "An incorrect step, process, or data definition in a computer program"
- Failure: "The incorrect result of a fault"
- Mistake: "A human action that produces an incorrect result"

One approach to judging quality in security has been fixing faults. You might argue that a module in which 100 faults were discovered and fixed is better than another in which only 20 faults were discovered and fixed, suggesting that more rigorous analysis and testing had led to the finding of the larger number of faults.

To understand program security, we can examine programs to see whether they behave as their designers intended or users expected. Unexpected behavior is called a program security flaw; it is inappropriate program behavior caused by program vulnerability.

The terms "vulnerability" and "flaw" do not map directly to the characterization of faults and failures. A flaw can be either a fault or failure, and vulnerability usually describes a class of flaws, such as a buffer overflow. In spite of the inconsistency, it is important for us to remember that we must view vulnerabilities and flaws from two perspectives, cause and effect. Thus, we need to know what fault caused the problem and what failure (if any) is visible to the user.

Program security flaws can derive from any kind of software fault. That is, they cover everything from a misunderstanding of program requirements to a one-character error in coding or even typing. The flaws can result from problems in a single code component or from the failure of several programs or program pieces to interact compatibly through a shared interface. The security flaws can reflect code that was intentionally designed or coded to be malicious or code that was simply developed in a sloppy or misguided way. Thus,

it makes sense to divide program flaws into two separate logical categories: inadvertent human errors versus malicious, intentionally induced flaws.
These categories help us understand some ways to prevent the inadvertent and intentional insertion of flaws into future code, but we still have to address their effects, regardless of the intention.

An inadvertent error can cause just as much harm to users and their organizations as can an intentionally induced flaw. Furthermore, a system attack often exploits an unintentional security flaw to perform intentional damage. From reading the popular press you might conclude that intentional security incidents (called cyber-attacks) are the biggest security threat today. In fact, plain, unintentional human errors are more numerous and cause much more damage.

There is no ""silver bullet" to achieve security effortlessly, and false security solutions impede real progress toward more secure programming. There are two reasons for this distressing situation:

i. Program controls apply at the level of the individual program and programmer. When we test a system, we try to make sure that the functionality prescribed in the requirements are implemented in the code. That is, we take a

"should do" checklist and verify that the code does what it is supposed to do. However, security is also about preventing certain actions: a "shouldn't do" list. A system shouldn't do anything not on its "should do" list. It is almost impossible to ensure that a program does precisely what its designer or user intended, and nothing more.

ii. Programming and software engineering techniques change and evolve far more rapidly than do computer security techniques. So we often find ourselves trying to secure last year's technology while software developers are rapidly adopting today's and next year's technology.

.

3- Nonmalicious Program Errors

Being human, programmers and other developers make many mistakes, most of which are unintentional and nonmalicious. Many such errors cause program malfunctions but do not lead to more serious security vulnerabilities. However, a few classes of errors have plagued programmers and security professionals for decades, and there is no reason to believe they will disappear. In this section we consider three classic error types that have enabled many recent security breaches. We explain each type, why it is relevant to security, and how it can be prevented or mitigated.

1- Buffer Overflows

A buffer (or array or string) is a space in which data can be held. A buffer resides in memory. Because memory is finite, a buffer's capacity is finite. For this reason, in many programming languages the programmer must declare the buffer's maximum size so that the compiler can set aside that amount of space.

A buffer overflow is the computing equivalent of trying to pour two liters of water into a one-liter pitcher: Some water is going to spill out and make a mess. And in computing,

Let us look at an example to see how buffer overflows can happen. Suppose a C language program contains the declaration:

```
char sample[10];
```

The compiler sets aside 10 bytes to store this buffer, one byte for each of the 10 elements of the array, sample[0] through sample[9]. Now we execute the statement:

```
sample[10] = 'B';
```

The subscript is out of bounds (that is, it does not fall between 0 and 9), so we have a problem. The nicest outcome (from a security perspective) is for the compiler to detect the problem and mark the error during compilation. However, if the statement were

```
sample[i] = 'B';
```

we could not identify the problem until i was set during execution to a too-big subscript. It would be useful if, during execution, the system produced an error message warning of a subscript out of bounds. Unfortunately, in some languages, buffer sizes do not have to be predefined, so there is no way to detect an out-of-bounds error. More importantly, the code needed to check each subscript against its potential maximum value takes time and space during execution, and the resources are applied to catch a problem that occurs relatively infrequently. Even if the compiler were careful in analyzing the buffer declaration and use, this same problem can be caused with pointers, for which there is no reasonable way to define a proper limit. Thus, some compilers do not generate the code to check for exceeding bounds.

Let us examine this problem more closely. It is important to recognize that the potential overflow causes a serious problem only in some instances. The problem's occurrence depends on what is adjacent to the array sample. For example, suppose each of the ten elements of the array sample is filled with the letter A and the erroneous reference uses the letter B, as follows:

```
for (i=0; i<=9; i++)
        sample[i] = 'A'; sample[10] = 'B'
```

All program and data elements are in memory during execution, sharing space with the operating system, other code, and resident routines. So there are four cases to consider in deciding where the 'B' goes, as shown in **Figure 3.1**. If the extra character overflows into the user's data space, it simply overwrites an existing variable value (or it may be written into an as-yet unused location), perhaps affecting the program's result, but affecting no other program or data.

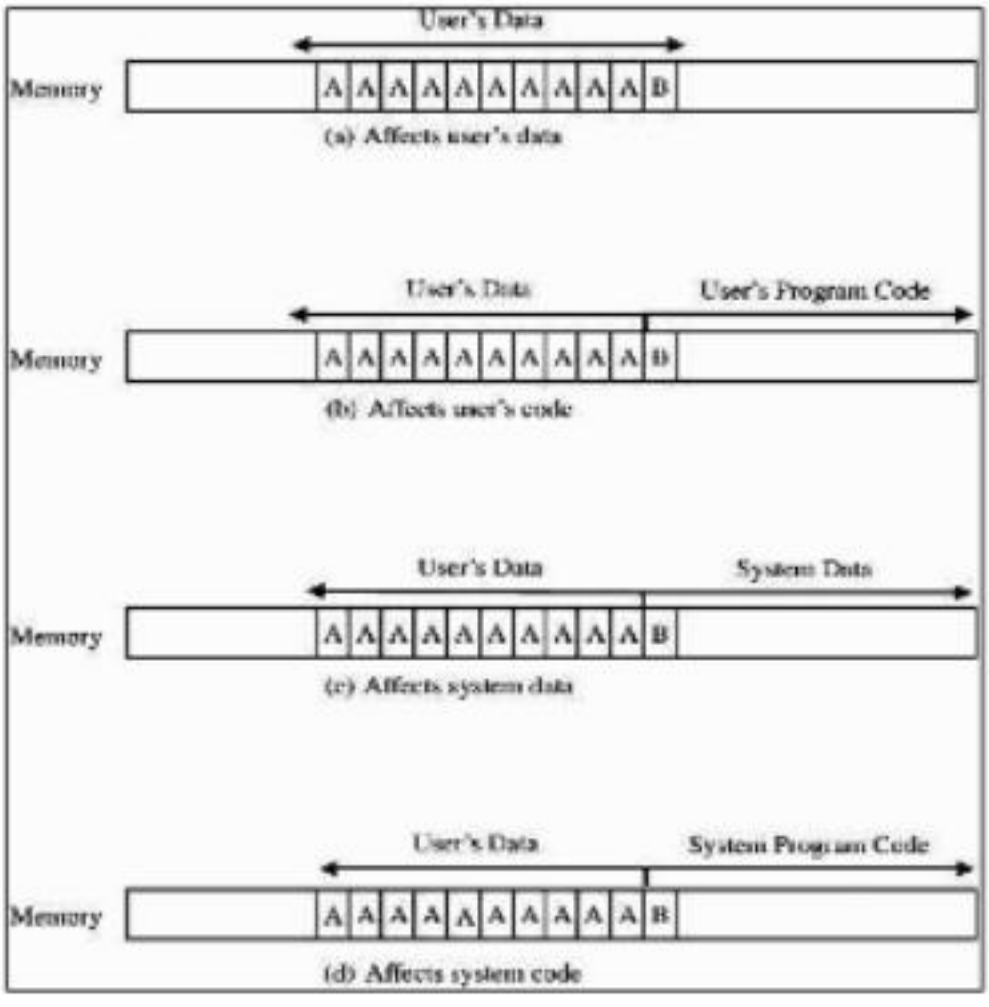

Figure 3.1. Places Where a Buffer Can Overflow.

In the second case, the 'B' goes into the user's program area. If it overlays an already executed instruction (which will not be executed again), the user should perceive no effect. If it overlays an instruction that is not yet executed, the machine will try to execute an instruction with operation code 0x42, the internal code for the character 'B'. If there is no instruction with operation code 0x42, the system will halt on an illegal instruction exception. Otherwise, the machine will use subsequent bytes as if they were the rest of the instruction, with success or failure depending on the meaning of the contents. Again, only the user is likely to experience an effect.

The most interesting cases occur when the system owns the space immediately after the array that overflows. Spilling over into system data or code areas produces similar results to those for the user's space: computing with a faulty value or trying to execute an improper operation.

In this section we consider Security Implication. Remember, however, that even if a flaw came from an honest mistake, the flaw can still cause serious harm. A malicious attacker can exploit these flaws.

Let us suppose that a malicious person understands the damage that can be done by a buffer overflow; that is,

we are dealing with more than simply a normal, errant programmer. The malicious programmer looks at the four cases illustrated in Figure 3.1 and thinks deviously about the last two: What data values could the attacker insert just after the buffer to cause mischief or damage, and what planned instruction codes could the system be forced to execute? There are many possible answers, some of which are more malevolent than others. Here, we present two buffer overflow attacks that are used frequently.

First, the attacker may replace code in the system space. Remember that every program is invoked by the operating system and that the operating system may run with higher privileges than those of a regular program. Thus, if the attacker can gain control by masquerading as the operating system, the attacker can execute many commands in a powerful role. Therefore, by replacing a few instructions right after returning from his or her own procedure, the attacker regains control from the operating system, possibly with raised privileges. If the buffer overflows into system code space, the attacker merely inserts overflow data that correspond to the machine code for instructions.

On the other hand, the attacker may make use of the stack pointer or the return register. Sub- procedure calls are handled with a stack, a data structure in which the

most recent item inserted is the next one removed (last arrived, first served). This structure works well because procedure calls can be nested, with each return causing control to transfer back to the immediately preceding routine at its point of execution. Each time a procedure is called, its parameters, the return address (the address immediately after its call), and other local values are pushed onto a stack. An old stack pointer is also pushed onto the stack, and a stack pointer register is reloaded with the address of these new values. Control is then transferred to the sub-procedure.

As the sub-procedure executes, it fetches parameters that it finds by using the address pointed to by the stack pointer. Typically, the stack pointer is a register in the processor. Therefore, by causing an overflow into the stack, the attacker can change either the old stack pointer (changing the context for the calling procedure) or the return address (causing control to transfer where the attacker wants when the sub-procedure returns). Changing the context or return address allows the attacker to redirect execution to a block of code the attacker wants.

In both these cases, a little experimentation is needed to determine where the overflow is and how to control it. But the work to be done is relatively small probably a day or two for a competent analyst.

3-2 Incomplete Mediation

Incomplete mediation is another security problem that has been with us for decades. Attackers are exploiting it to cause security problems.

Definition

Consider the example of the URL:

http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212&parm2=2009Jan17

The two parameters look like a telephone number and a date. Probably the user's web browser enters those two values in their specified format for easy processing on the server's side. What would happen if parm2 were submitted as 1800Jan01? Or 1800Feb30? Or 2048Min32? Or 1Aardvark2Many?

Something would likely fail. As with buffer overflows, one possibility is that the system would fail catastrophically, with a routine's failing on a data type error as it tried to handle a month named "Min" or even a year (like 1800) that was out of range. Another possibility is that the receiving program would continue to execute but would generate a very wrong result. Then again, the processing server might have a default condition The possibilities are endless. Nevertheless, unchecked data values represent a serious potential vulnerability.

One way to address the potential problems is to try to

anticipate them. For instance, the programmer in the examples above may have written code to check for correctness on the client's side (that is, the user's browser). The client program can search for and screen out errors. Or, to prevent the use of nonsense data, the program can restrict choices only to valid ones. For example, the program supplying the parameters might have solicited them by using a drop-down box or choice list from which only the twelve conventional months would have been possible choices. Similarly, the year could have been tested to ensure that the value was between 1995 and 2015, and date numbers would have to have been appropriate for the months in which they occur (no 30th of February, for example). Using these verification techniques, the programmer may have felt well insulated from the possible problems a careless or malicious user could cause.

However, the program is still vulnerable. By packing the result into the return URL, the programmer left these data fields in a place the user can access (and modify). In particular, the user could edit the URL line, change any parameter values, and resend the line. On the server side, there is no way for the server to tell if the response line came from the client's browser or as a result of the user's editing the URL directly. We say in this case that the data values are not completely mediated: The sensitive data (namely, the parameter

values) are in an exposed, uncontrolled condition.
Incomplete mediation is easy to exploit, but it has been exercised less often than buffer overflows..

To demonstrate this flaw's security implications, we use a real example; only the name of the vendor has been changed to protect the guilty. Things, Inc., was a very large, international vendor of consumer products, called Objects. The company was ready to sell its Objects through a web site, using what appeared to be a standard e-commerce application. The management at Things decided to let some of its in-house developers produce the web site so that its customers could order Objects directly from the web.

To accompany the web site, Things developed a complete price list of its Objects, including pictures, descriptions, and drop-down menus for size, shape, color, scent, and any other properties. For example, a customer on the web could choose to buy 20 of part number 555A Objects. If the price of one such part were $10, the web server would correctly compute the price of the 20 parts to be $200. Then the customer could decide whether to have the Objects shipped by boat, by ground transportation, or sent electronically. If the customer were to choose boat delivery, the customer's web browser would complete a form with parameters like these:

http://www.things.com/order.asp?custID=101&part=55

5A&qy=20&price
=10&ship=boat&shipcost=5&total=205

So far, so good; everything in the parameter passage looks correct. But this procedure leaves the parameter statement open for malicious tampering. Things should not need to pass the price of the items back to itself as an input parameter; presumably Things knows how much its Objects cost, and they are unlikely to change dramatically since the time the price was quoted a few screens earlier.

A malicious attacker may decide to exploit this peculiarity by supplying instead the following URL, where the price has been reduced from $205 to $25:

http://www.things.com/order.asp?custID=101&part=55
5A&qy=20&price 1&ship=boat&shipcost=5&total=25

It worked. The attacker could have ordered Objects from Things in any quantity at any price. And yes, this code was running on the web site for a while before the problem was detected. From a security perspective, the most serious concern about this flaw was the length of time that it could have run undetected.

3-3 Time-of-Check to Time-of-Use Errors

The *time-of-check-to-time-of-use (TOCTTOU or TOC/TOU)* issue is a timing vulnerability that occurs when a program checks access permissions too far in

advance of a resource request. For example, if an operating system builds a comprehensive list of access permissions for a user upon logon and then consults that list throughout the logon session, a TOCTTOU vulnerability exists. If the system administrator revokes a particular permission, that restriction would not be applied to the user until the next time they log on. If the user is logged on when the access revocation takes place, they will have access to the resource indefinitely. The user simply needs to leave the session open for days and the new restrictions will never be applied.

The third programming flaw we investigate involves synchronization. To improve efficiency, modern processors and operating systems usually change the order in which instructions and procedures are executed. In particular, instructions that appear to be adjacent may not actually be executed immediately after each other, either because of intentionally changed order or because of the effects of other processes in concurrent execution.

Definition

Access control is a fundamental part of computer security; we want to make sure that only those who should access an object are allowed that access. Every requested access must be governed by an access policy stating who is allowed access to what; then the request

must be mediated by an access-policy-enforcement agent. But an incomplete mediation problem occurs when access is not checked universally. The time-of-check to time-of-use (TOCTTOU) flaw concerns mediation that is performed with a "bait and switch" in the middle. It is also known as a serialization or synchronization flaw.

4- Viruses and Other Malicious Code:

Programs are considered security threats. The programs operate on data, taking action only when data and state changes trigger it. Much of the work done by a program is invisible to users who are not likely to be aware of any malicious activity. For instance, when was the last time you saw a bit? Do you know in what form a document file is stored? If you know a document resides somewhere on a disk, can you find it? Can you tell if a game program does anything in addition to its expected interaction with you? Which files are modified by a word processor when you create a document? Which programs execute when you start your computer or open a web page? Most users cannot answer these questions. However, since users usually do not see computer data directly, malicious people can make programs serve as vehicles to access and change data and other programs. The effects of malicious code and in detail and several kinds of programs that can be used

for interception or modification of data are explained.

4-1 Kinds of Malicious Code

Malicious code or rogue program is the general name for unanticipated or undesired effects in programs or program parts, caused by an agent intent on damage. By this definition, most faults found in software inspections, reviews, and testing do not qualify as malicious code, because we think of them as unintentional. However, keep in mind as you read this chapter those unintentional faults can in fact invoke the same responses as intentional malevolence; a benign cause can still lead to a disastrous effect.

The terminology of malicious code is sometimes used imprecisely. In fact each malicious code has its own definition and effect on the system as:.

- ***Virus***: is a program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them. It infects other healthy subjects by attaching itself to the program and either destroying it or coexisting with it. Because viruses are insidious, we cannot assume that a clean program yesterday is still clean today. Moreover, a good program can be modified to include a copy of the virus program, so the infected good program itself begins to act as a virus, infecting other programs. The infection

usually spreads at a geometric rate, eventually overtaking an entire computing system and spreading to all other connected systems.

A virus can be either *transient* or *resident.* A transient virus has a life that depends on the life of its host; the virus runs when its attached program executes and terminates when its attached program ends. (During its execution, the transient virus may spread its infection to other programs.) A resident virus locates itself in memory; then it can remain active or be activated as a stand-alone program, even after its attached program ends.

- **A Trojan horse** is malicious code that, in addition to its primary effect, has a second, nonobvious malicious effect. As an example of a computer Trojan horse, consider a login script that solicits a user's identification and password, passes the identification information on to the rest of the system for login processing, but also retains a copy of the information for later, malicious use. In this example, the user sees only the login occurring as expected, so there is no evident reason to suspect that any other action took place.

- **Hoaxes**

  No discussion of viruses is complete without mentioning the nuisance and wasted resources caused by virus hoaxes. Almost every e-mail user has, at one time or another, received a message forwarded by a friend or relative that warns of the latest virus threat to roam the Internet. Invariably, this purported "virus" is the most destructive virus ever unleashed and no antivirus package is able to detect and/or eradicate it. One famous example of such a hoax is the Good Times virus warning that first surfaced on the Internet in 1994 and still circulates today. For more information on this topic, the renowned virus hoax expert Rob Rosenberger edits a website that contains a comprehensive repository of virus hoaxes. You can find it at www.vmyths.com

- **A logic bomb** is a class of malicious code that "detonates" or goes off when a specified condition occurs. A time bomb is a logic bomb whose trigger is a time or date. *logic bombs* are malicious code objects that infect a system and lie dormant until they are triggered by the occurrence of one or more conditions such as time, program launch, website logon, and so on.

The vast majority of logic bombs are programmed into custom-built applications by software developers seeking to ensure that their work is destroyed if they unexpectedly leave the company. The previous chapter provided several examples of this type of logic bomb.

However, it's important to remember that, like any malicious code object, logic bombs come in many shapes and sizes. Indeed, many viruses and Trojan horses contain a logic bomb component. The famous Michelangelo virus caused a media frenzy when it was discovered in 1991 due to the logic bomb trigger it contained. The virus infects a system's Master Boot Record through the sharing of infected floppy disks and then hides itself until March 6—the birthday of the famous Italian artist Michelangelo Buonarroti. On that date, it springs into action, reformatting the hard drives of infected systems and destroying all of the data they contain.

- **A trapdoor** or backdoor is a feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges. For instance, an automated bank teller program might allow anyone entering the number 990099 on the keypad to process the log of everyone's transactions at that machine. In this example, the trapdoor could be intentional,

for maintenance purposes, or it could be an illicit way for the implementer to wipe out any record of a crime.

- **A worm** is a program that spreads copies of itself through a network., was for nonmalicious purposes. The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses copied program or data files). Additionally, the worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.
- **A rabbit** is a virus or worm that self-replicates without

  bound, with the intention of exhausting some computing resource. A rabbit might create copies of itself and store them on disk in an effort to completely fill the disk, for example. These definitions match current careful usage. The distinctions among these terms are small, and often the terms are confused, especially in the popular press. The term "virus" is often used to refer to any piece of malicious code. Furthermore, two or more forms of malicious code can be combined to produce a third kind of problem. For instance, a virus can be a time bomb

if the viral code that is spreading will trigger an event after a period of time has passed. The kinds of malicious code and a comparisons are summarized in Table 1 and Table 2.

Table 1. Types of Malicious Code.

Code Type	Characteristics
Virus	Attaches itself to program and propagates copies of itself to other programs
Trojan horse	Contains unexpected, additional functionality
Logic bomb	Triggers action when condition occurs Time bomb Triggers action when specified time occurs
Trapdoor	Allows unauthorized access to functionality
Worm	Propagates copies of itself through a network
Rabbit	Replicates itself without limit to exhaust resources

Table 2. Comparison between malicious codes

Code Type	Can replicate itself	Need a Host program	Side effect
Viruses	Yes	Yes	Gain control of the computer
Bacteria	Yes	No	Consuming processor capacity, system memory, disk space
Worms	Yes	No	Consume network resources and may lead to a near shutdown
Trapdoors	No	Yes	Complete logon into software
Logic bombs	No	Yes	Once triggered , bomb may modify or delete data, delete entire file, machine halt, inflict some other damage
Trojan Horses	No	yes	A piece of code hides inside a program to perform masked function (i.e, modification, deletion,….)

A program virus attaches itself to a program in three ways:

Appended virus, the virus is activated whenever the program is run. This kind of attachment is usually easy to program. In the simplest case, a virus inserts a copy of itself into the executable program file before the first executable instruction. Then, all the virus instructions execute first; after the last virus instruction, control flows naturally to what used to be the first program instruction. Such a situation is shown in **Figure** 3.2

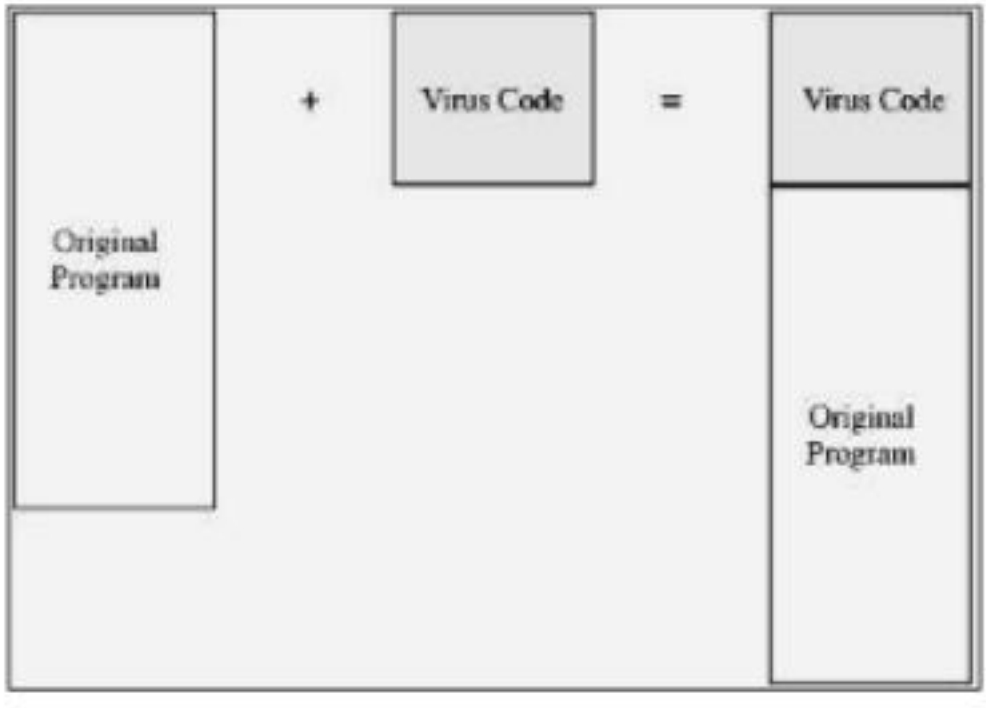

Figure 3.2. Virus Appended to a Program.

This kind of attachment is simple and usually effective. The virus writer does not need to know anything about

the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus.

***Surround virus*:** The virus performs its task and then transfers to the original program. Typically, the user is unaware of the effect of the virus if the original program still does all that it used to. Most viruses attach in this manner. . For example, a virus writer might want to prevent the virus from being detected. If the virus is stored on disk, its presence will be given away by its file name, or its size will affect the amount of space used on the disk. The virus writer might arrange for the virus to attach itself to the program that constructs the listing of files on the disk. If the virus regains control after the listing program has generated the listing but before the listing is displayed or printed, the virus could eliminate its entry from the listing and falsify space counts so that it appears not to exist. A surrounding virus is shown in Figure 3.3.

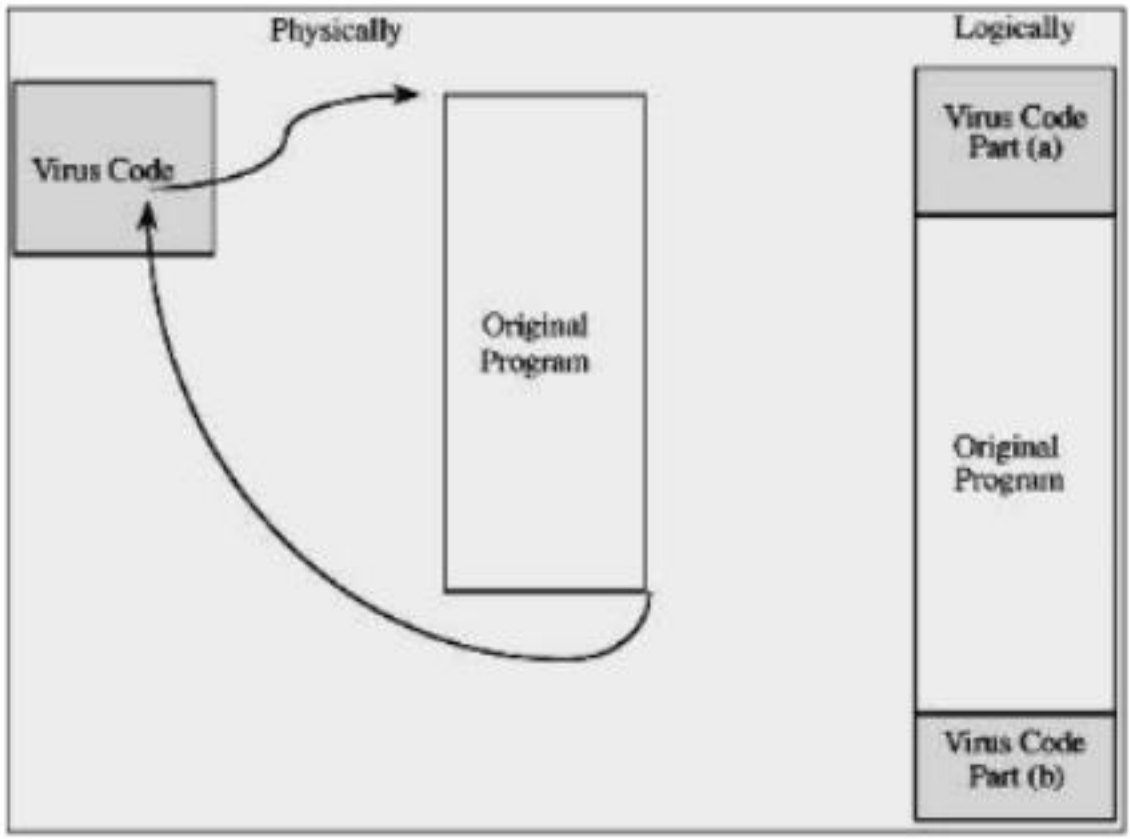

Figure 3.3. Virus Surrounding a Program.

Integrated Viruses and Replacements is a third situation occurs when the virus replaces some of its target, integrating itself into the original code of the target. Such a situation is shown in **Figure** 3.4. Clearly, the virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.

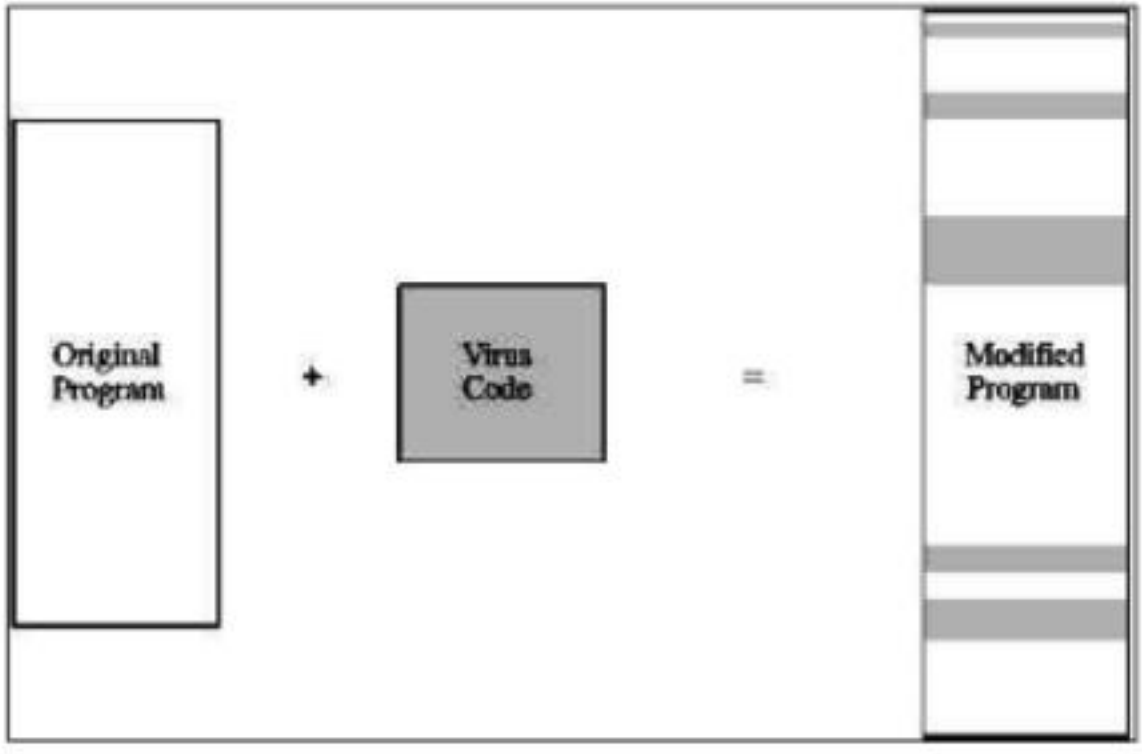

Figure 3.4. Virus Integrated into a Program.

Finally, the virus can replace the entire target, either mimicking the effect of the target or ignoring the expected effect of the target and performing only the virus effect. In this case, the user is most likely to perceive the loss of the original program.

4-2 How Viruses Gain Control ?

The virus (V) has to be invoked instead of the target (T). Essentially, the virus either has to seem to be T, saying effectively "I am T" or the virus has to push T out of the way and become a substitute for T, saying effectively "Call me instead of T." The virus can

assume T's name by replacing (or joining to) T's code in a file structure; this invocation technique is most appropriate for ordinary programs. The virus can overwrite T in storage (simply replacing the copy of T in storage, for example). Alternatively, the virus can change the pointers in the file table so that the virus is located instead of T whenever T is accessed through the file system. These two cases are shown in Figure 3.5.

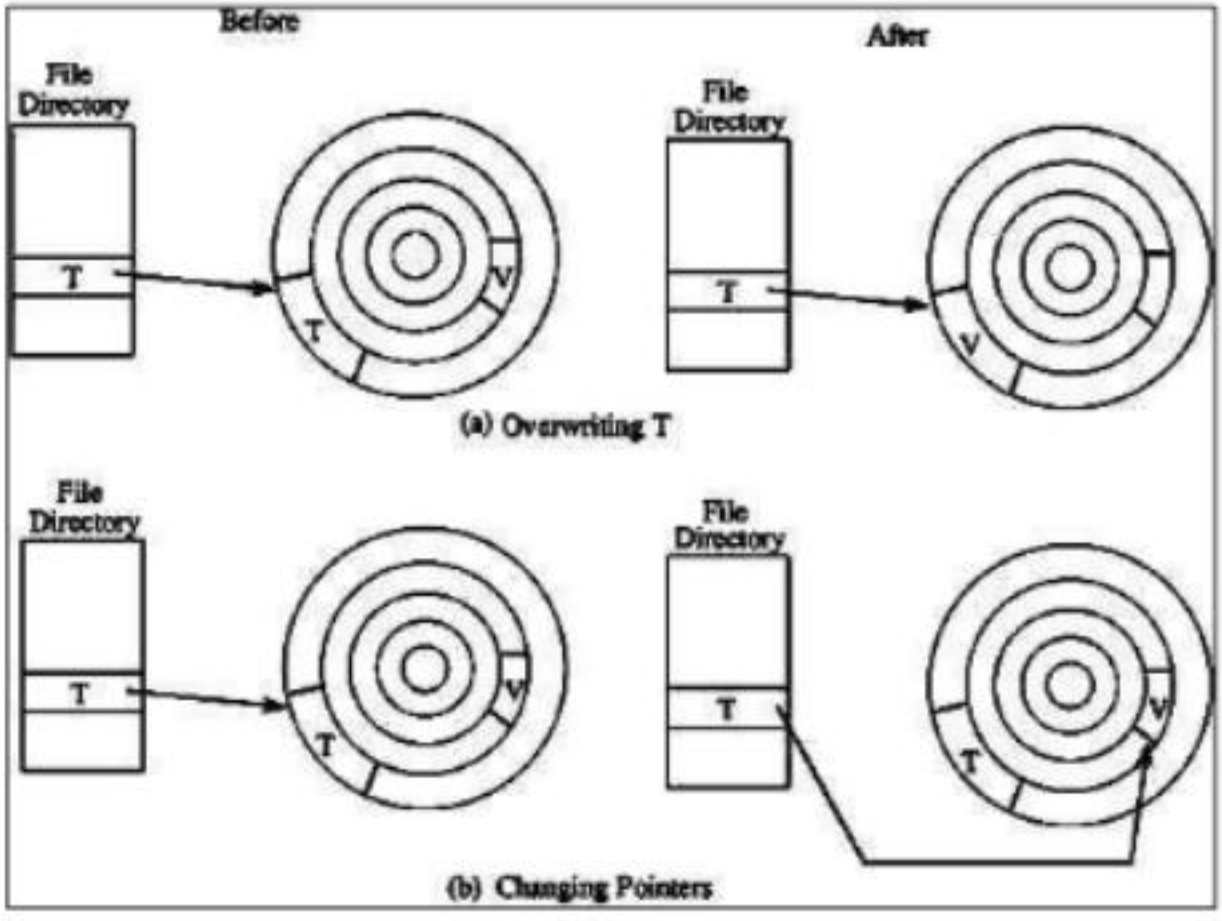

Figure 3.5. Virus Completely Replacing a Program.

The virus can supplant T by altering the sequence that would have invoked T to now invoke the virus V; this invocation can be used to replace parts of the resident operating system by modifying pointers to those resident parts, such as the table of handlers for different kinds of interrupts.

4-3 Homes for Viruses

The virus writer may find these qualities appealing in a virus:

- It is hard to detect.
- It is not easily destroyed or deactivated.
- It spreads infection widely.
- It can reinfect its home program or other programs.
- It is easy to create.
- It is machine independent and operating system independent.

Few viruses meet all these criteria. The virus writer chooses from these objectives when deciding what the virus will do and where it will reside.

4-4 Boot Sector Viruses

A special case of virus attachment, but formerly a fairly popular one, is the so-called boot sector virus. When a computer is started, control begins with firmware that determines which hardware components are present, tests them, and transfers control to an operating system. A given hardware platform can run many different operating systems, so the operating system is not coded in firmware but is instead invoked dynamically, perhaps even by a user's choice, after the hardware test.

The operating system is software stored on disk. Code copies the operating system from disk to memory and transfers control to it; this copying is called the bootstrap (often boot) load because the operating system figuratively pulls itself into memory by its bootstraps. The firmware does its control transfer by reading a fixed number of bytes from a fixed location on the disk (called the boot sector) to a fixed address in memory and then jumping to that address (which will turn out to contain the first instruction of the bootstrap loader). The bootstrap loader then reads into memory the rest of the operating system from disk. To run a different operating system, the user just inserts a disk with the new operating system and a bootstrap loader. When the user reboots from this new disk, the loader there brings in and runs another operating system. This

same scheme is used for personal computers, workstations, and large mainframes.

To allow for change, expansion, and uncertainty, hardware designers reserve a large amount of space for the bootstrap load. The boot sector on a PC is slightly less than 512 bytes, but since the loader will be larger than that, the hardware designers support "chaining," in which each block of the bootstrap is chained to (contains the disk location of) the next block. This chaining allows big bootstraps but also simplifies the installation of a virus. The virus writer simply breaks the chain at any point, inserts a pointer to the virus code to be executed, and reconnects the chain after the virus has been installed. This situation is shown in **Figure 3.6.**

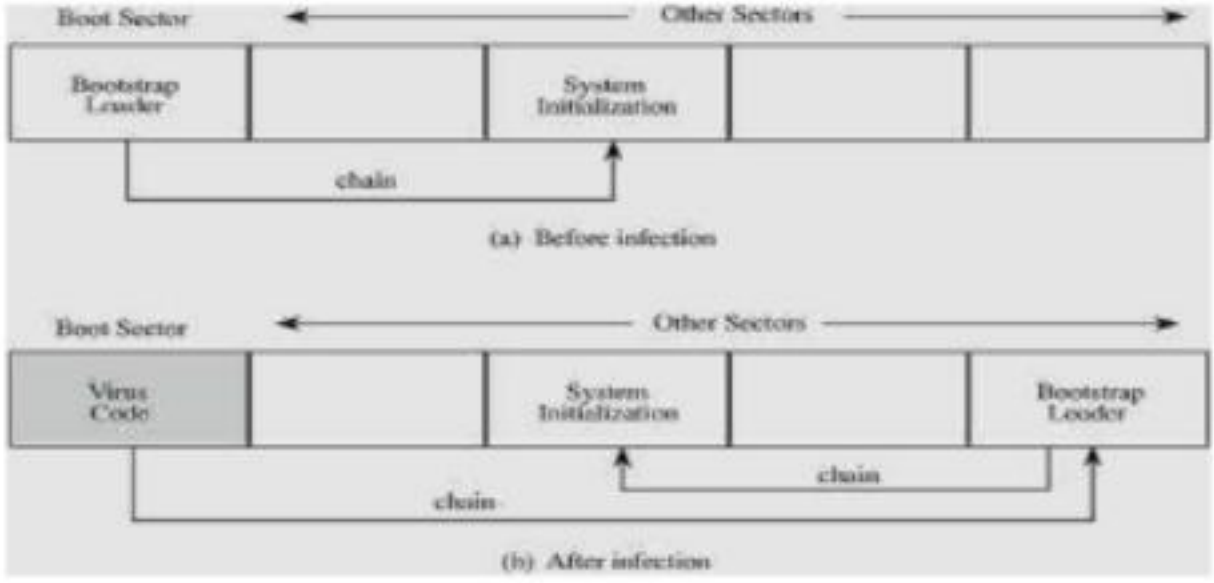

Figure 3.6. Boot Sector Virus Relocating Code.

The boot sector is an especially appealing place to house a virus. The virus gains control very early in the boot process, before most detection tools are active, so that it can avoid, or at least complicate, detection. The files in the boot area are crucial parts of the operating system. Consequently, to keep users from accidentally modifying or deleting them with disastrous results, the operating system makes them "invisible" by not showing them as part of a normal listing of stored files, preventing their deletion. Thus, the virus code is not readily noticed by users.

4-5 Memory-Resident Viruses

Some parts of the operating system and most user programs execute, terminate, and disappear, with their space in memory being available for anything executed later. For very frequently used parts of the operating system and for a few specialized user programs, it would take too long to reload the program each time it was needed. Such code remains in memory and is called "resident" code. Examples of resident code are the routine that interprets keys pressed on the keyboard, the code that handles error conditions that arise during a program's execution, or a program that acts like an alarm clock, sounding a signal at a time the user determines. Resident routines are sometimes called TSRs or "terminate and stay resident" routines.

Virus writers also like to attach viruses to resident code

because the resident code is activated many times while the machine is running. Each time the resident code runs, the virus does too. Once activated, the virus can look for and infect uninfected carriers. For example, after activation, a boot sector virus might attach itself to a piece of resident code. Then, each time the virus was activated it might check whether any removable disk in a disk drive was infected and, if not, infect it. In this way the virus could spread its infection to all removable disks used during the computing session.

A virus can also modify the operating system's table of programs to run. On a Windows machine the registry is the table of all critical system information, including programs to run at startup. If the virus gains control once, it can insert a registry entry so that it will be reinvoked each time the system restarts. In this way, even if the user notices and deletes the executing copy of the virus from memory, the virus will return on the next system restart.

4-6 Other Homes for Viruses

A virus that does not take up residence in one of these cozy establishments has to fend more for itself. But that is not to say that the virus will go homeless. One popular home for a virus is an application program. Many applications, such as word processors and spreadsheets, have a "macro" feature, by which a user can record a series of commands and repeat them with

one invocation. Such programs also provide a "startup macro" that is executed every time the application is executed. A virus writer can create a virus macro that adds itself to the startup directives for the application. It also then embeds a copy of itself in data files so that the infection spreads to anyone receiving one or more of those files.

5- Virus Signatures

A virus cannot be completely invisible. Code must be stored somewhere, and the code must be in memory to execute. Moreover, the virus executes in a particular way, using certain methods to spread. Each of these characteristics yields a telltale pattern, called a signature that can be found by a program that looks for it. The virus's signature is important for creating a program, called a virus scanner that can detect and, in some cases, remove viruses. The scanner searches memory and long-term storage, monitoring execution and watching for the telltale signatures of viruses.

.

CHAPTER 4

AUTHENTICATION

1-What is meant by Authentication?

For most systems, authentication is the first defense line. Authentication is a technical action that prevents unauthorized people (or unauthorized processes) from entering a computer system. Authentication is a critical building block of computer security since it is the basis for most types of access control and for establishing user accountability. Access control often requires that the system be able to identify and differentiate among users.

Computer systems recognize people based on the authentication data the systems receive. Authentication presents several challenges: collecting authentication data, transmitting the data securely, and knowing whether the person who was originally authenticated is still the person using the computer system. For example, a user may walk

away from a terminal while still logged on, and another person may start using it. There are three means of authenticating a user's identity which can be used alone or in combination:

1- *Something the individual knows* (a secret e.g., a password, Personal Identification Number (PIN), or cryptographic key)
2- *Something the individual has* (a token e.g., an ATM card or a smart card)
3- *Something the individual is* (a biometric e.g., such characteristics as a voice pattern, handwriting dynamics, or a fingerprint).

While it may appear that any of these means could provide strong authentication, there are problems associated with each. If people wanted to pretend to be someone else on a computer system, they can guess or learn that individual's password; they can also steal or fabricate tokens. Each method also has drawbacks for legitimate users and system administrators: users forget passwords and may lose tokens, and administrative overhead for keeping

track of authentication data and tokens can be substantial. Biometric systems have significant technical, user acceptance, and cost problems as well.

This chapter explains current authentication technologies and their benefits and drawbacks as they relate to the three means of authentication. Although some of the technologies make use of cryptography because it can significantly strengthen authentication.

2- Authentication Based on Something the User Knows:

The most common form of authentication is a user ID coupled with a password. This technique is based solely on something the user knows. There are other techniques besides conventional passwords that are based on knowledge, such as knowledge of a cryptographic key.

2-1 Passwords

In general, password systems work by requiring the user to enter a user ID and password (or passphrase or personal identification number). The system compares the password to a previously stored password for that user ID. If there is a match, the user is authenticated and granted access.

Benefits of Passwords:

Passwords have been successfully providing security for computer systems for a long time. They are integrated into many operating systems, and users and system administrators are familiar with them. When properly managed in a controlled environment, they can provide effective security.

Problems With Passwords.

The security of a password system is dependent upon keeping passwords secret. Unfortunately, there are many ways that the secret may be divulged. All of the problems discussed below can be significantly mitigated by

improving password security. However, there is no fix for the problem of electronic monitoring, except to use more advanced authentication (e.g., based on cryptographic techniques or tokens).

1- Guessing or finding passwords. If users select their own passwords, they tend to make them easy to remember. That often makes them easy to guess. The names of people's children, pets, or favorite sports teams are common examples. On the other hand, assigned passwords may be difficult to remember, so users are more likely to write them down. Many computer systems are shipped with administrative accounts that have preset passwords. Because these passwords are standard, they are easily "guessed." Although security practitioners have been warning about this problem for years, many system administrators still do not change default passwords. Another method of learning passwords is to observe someone entering a password or PIN. The observation can be done by someone in the same room.

2- Giving passwords away: Users may share their passwords.

3- Electronic monitoring: When passwords are transmitted to a computer system, they can be electronically monitored. This can happen on the network used to transmit the password or on the computer system itself. Simple encryption of a password that will be used again does not solve this problem because encrypting the same password will create the same ciphertext; the ciphertext becomes the password.

4- Accessing the password file: If the password file is not protected by strong access controls, the file can be downloaded. Password files are often protected with one-way encryption so that plain-text passwords are not available to system administrators or hackers (if they successfully bypass access controls). Even if the file is encrypted, brute force can be used to learn passwords if the file is downloaded.

5- Passwords Used as Access Control: Some mainframe operating systems and many PC applications use passwords as a means of restricting access to specific resources within a system. Instead of using mechanisms such as access control lists, access is granted by entering a password. The result is a proliferation of passwords that can reduce the overall security of a system. While the use of passwords as a means of access control is common, it is an approach that is often less than optimal and not cost-effective.

Some categories of passwords that researchers have found easy to guess are as follows:

- Passwords based on account names
- Account name followed by a number
- Passwords based on user names
- Initials repeated 0 or more times
- All letters lower- or uppercase
- Name reversed
- First initial followed by last name reversed
- Passwords based on computer names
- Dictionary words
- Dictionary words with some or all letters capitalized
- Reversed dictionary words with some or all letters capitalized
- Dictionary words with arbitrary letters turned into control characters
- Patterns from the keyboard
- Passwords shorter than six characters
- Passwords containing only digits
- Passwords containing only uppercase or lowercase letters, or letters and numbers, or letters and punctuation
- Passwords that look like license plate numbers

3-Based on Something the User has:

Although some techniques are based solely on something the user has, most of the techniques described in this section are combined with something the user knows. This combination can provide significantly stronger security than either something the user knows or possesses alone. Objects that a user possesses for the purpose of authentication are called tokens. This section divides tokens into two categories: memory tokens and smart tokens.

3-1 Memory Token:

Memory tokens store, but do not process, information. Special reader/writer devices control the writing and reading of data to and from the tokens. The most common type of memory token is a magnetic striped card, in which a thin strips of magnetic material is affixed to the surface of a card (e.g., as on the back of credit cards). A common application of memory tokens for authentication to computer systems is the automatic teller machine

(ATM) card. This uses a combination of something the user possesses (the card) with something the user knows (the PIN). Some computer systems authentication technologies are based solely on possession of a token, but they are less common. Token-only systems are more likely to be used in other applications.

Benefits of Memory Token Systems:

1- Memory tokens when used with PINs provide significantly more security than passwords.
2- Memory cards are inexpensive to produce.
3- The hacker must have both a valid token and the corresponding PIN. This is much more difficult than obtaining a valid password and user ID combination.

Problems With Memory Token Systems.

1- Most of the problems associated with them relate to their cost, administration, token loss, user dissatisfaction, and the compromise of PINs.

2- Most of the techniques for increasing the security of memory token systems relate to the protection of PINs.

3- Requires special reader. The need for a special reader increases the cost of using memory tokens. The readers used for memory tokens must include both the physical unit that reads the card and a processor that determines whether the card and/or the PIN entered with the card is valid. If the PIN or token is validated by a processor that is not physically located with the reader, then the authentication data is vulnerable to electronic monitoring

4- Token loss. A lost token may prevent the user from being able to log in until a replacement is provided. This can increase administrative overhead costs.

5- User Dissatisfaction. In general, users want computers to be easy to use. Many users find it inconvenient to carry and present a token. However, their dissatisfaction may be reduced if they see the need for increased security.

3-1-1 Smart Token

A smart token expands the functionality of a memory token by incorporating one or more integrated circuits into the token itself. When used for authentication, a smart token is another example of authentication based on something a user is (i.e., the token itself). A smart token typically requires a user also to provide something the user knows (i.e., a PIN or password) in order to "unlock" the smart token for use. There are many different types of smart tokens as shown in Figure 1.

Figure 1, Authentication techniques based on something user has

In general, smart tokens can be divided three different ways based on physical characteristics, interface, and protocols used. These three divisions are not mutually exclusive.

- *Physical Characteristics*. Smart tokens can be divided into two groups: smart cards and other types of tokens.

  A smart card looks like a credit card, but incorporates an embedded microprocessor. Smart cards are defined by an International Standards Organization (ISO) standard. Smart tokens that are not smart cards can look like calculators, keys, or other small portable objects.

- *Interface*. Smart tokens have either a manual or an electronic interface. Manual or human interface tokens have displays and/or keypads to allow humans to communicate with the card. Smart tokens with electronic interfaces must be read by special reader/writers. Smart cards, described above,
  have an electronic interface. Smart tokens that look like calculators usually have a manual interface.

- *Protocol.* There are many possible protocols a smart token can use for authentication. In general, they can be divided into three categories: static password exchange, dynamic password generators, and challenge-response.

Static tokens work similarly to memory tokens, except that the users authenticate themselves to the token and then the token authenticates the user to the computer. A token that uses a dynamic password generator protocol creates a unique value. for example, an eight-digit number, that changes periodically (e.g., every minute). If the token has a manual interface, the user simply reads the current value and then types it into the computer system for authentication. If the token has an electronic interface, the transfer is done automatically. If the correct value is provided, the log-in is permitted, and the user is granted access to the system.

- Tokens that use a challenge-response protocol work by having the computer generate a challenge, such as a random string of numbers. The smart

token then generates a response based on the challenge. This is sent back to the computer, which authenticates the user based on the response. The challenge-response protocol is based on cryptography. Challenge-response tokens can use either electronic or manual interfaces.

Benefits of Smart Tokens

Smart tokens offer great flexibility and can be used to solve many authentication problems. The benefits of smart tokens vary, depending on the type used. In general, they provide greater security than memory cards. Smart tokens can solve the problem of electronic monitoring even if the authentication is done across an open network by using one-time passwords.

Problems with Smart Tokens

Like memory tokens, most of the problems associated with smart tokens relate to their cost, the administration of the system, and user dissatisfaction. Smart tokens are generally less

vulnerable to the compromise of PINs because authentication usually takes place on the card. (It is possible, of course, for someone to watch a PIN being entered and steal that card.) Smart tokens cost more than memory cards because they are more complex, particularly challenge-response calculators.

1. Need reader/writers or human intervention. Smart tokens can use either an electronic or a human interface.
2. Substantial Administration. Smart tokens, like passwords and memory tokens, require trong administration. For tokens that use cryptography, this includes key management.

3- Authentication Based on Something the User Is :

Biometric authentication technologies use the unique characteristics (or attributes) of an individual to authenticate that person's identity. These include physiological attributes (such as fingerprints, hand geometry, or retina patterns) or behavioral attributes

(such as voice patterns and hand-written signatures). Biometric authentication technologies based upon these

attributes have been developed for computer log-in applications.

Biometric authentication is technically complex and expensive, and user acceptance can be difficult. However, advances continue to be made to make the technology more reliable, less costly, and more user-friendly. Biometric systems can provide an increased level of security for computer systems, but the technology is still less matures than that of memory tokens or smart tokens. Imperfections in biometric authentication devices arise from technical difficulties in measuring and profiling physical attributes as well as from the somewhat variable nature of physical attributes. These may change, depending on various conditions. For example, a person's speech pattern may change under stressful conditions or when suffering from a sore throat or cold.

Due to their relatively high cost, biometric systems are typically used with other authentication means in environments requiring high security.

5-AUTHENTICATION PROCEDURES:

This section address the procedures used to verify the identity of the user. These procedures assume that the user or the process is in possession of a password that is shared between the user and an application or a system. In a given distributed network, clients as well as servers need to be authenticated. Consider a network where a user A need to request services from server S. Then user A has to be authenticated to the server S, and optionally, the server S has to be authenticated to user A. This is called two-party authentication as show in figure 2.

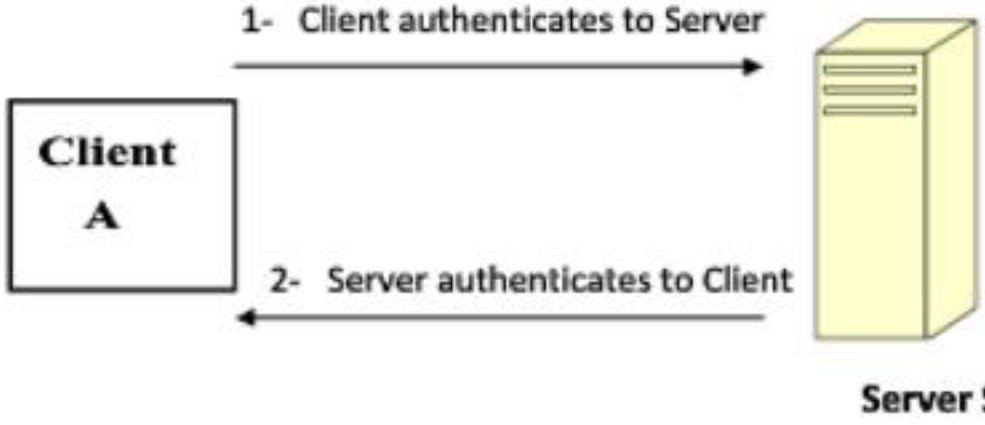

Figure 2: Two –Party Authentication

Many environments require a trusted third party for authentication. This third party ensures the identities of the client and the server. It is often called a security server that provides storage for the passwords and uses these stored passwords to verify the identities of the users and servers. This is called third party authentication. Procedures for achieving two-party authentication are presented below, followed by various schemes for third party authentication.

5-1 TWO- PARTY AUTHENTICATION:

Two-party authentication is used for achieving one-way authentication as well as two-way authentication.

One-Way Authentication

Consider the simple case in which the user wishes to log on to a host application, figure 3. In this case, let us assume that only the user needs to be authenticated to the application. Therefore, the host application is not to be authenticated to the user. Then the authenticated procedure simply involved the user sending his ID and the password to the application. The application verifies the password for the received ID, thereby authenticating the user. One-way authentication accomplishes the following:

1- It confirms that the application has authenticated the user's ID .

2- It assures the application that the ID and password was indeed sent by the legitimate user (no one alse has the user's password)

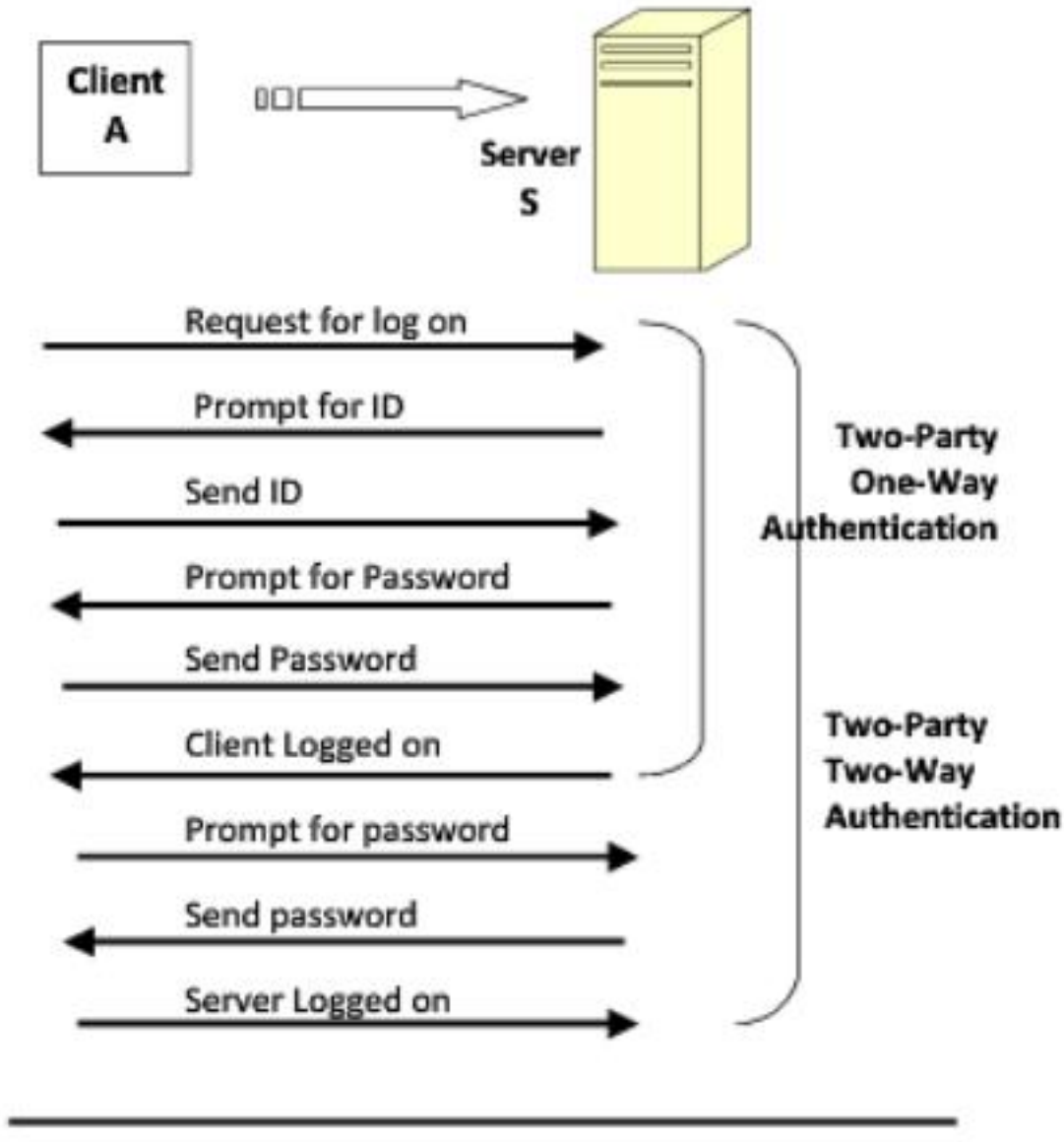

Figure 3: Two-Party One-Way Authentication and Two-Way Authentication.

Two-Way Authentication

For two-party two-way authentication, both parties are authenticated to each other, as shown in Figure 3. In addition to the two items listed under one-way authentication, two-way authentication also accomplishes the following:

1. The server has also been authenticated to the user.
2. The password for the server was sent by the legitimate server(since no one else has the server's password)

The two-party two-way authentication is not commonly used due to some practical problems. Consider a network where there are 50 users (or applications), and every user can communicate with any other user. So every user should be capable of authenticating every other user. Furthermore, for security reasons, every user has to store a password for each for the other users. This results in storing 49 passwords at every user

workstation. Now consider the case in which a user ia added or deleted or a password is changed. Such a change requires coordinating an update to the list of stored passwords for each of the 50 users. Therefore, it is considered inefficient technique.

5-2 THIRD-PARTY AUTHENTICATION SCHEMES:

A natural extension to the two-party two-way authentication is to use a trusted third party that stores all the passwords. This trusted third party is the only location where passwords are stored and maintained. Every user or application will send the ID and the password to the trusted third party for authentication. This approach improves the security and simplifies the storage and maintenance of passwords. Before proceeding to describe the third-party authentication scheme, we summarize the basic requirements for a third party authentication scheme:

1. The scheme should provide two-party two-way authentication; the third party should provide centralized storage and maintenance of the passwords.
2. The scheme should not transport passwords over the network. This is desirable since a plaintext or encrypted password can be copied from the network and replayed at the later time.
3. The passwords should not be stored at the client workstation. A stored password can be retrieved by an intruder when the user has stepped away from the workstation.
4. Once a user is logged on, the authentication scheme should provide a temporary secret to represent the user. In this way, repeated entries of passwords are not required. For example, a user may want to access the mail server to check the mail about six times in a day. It is desirable that the user not be required to enter and transmit the password six different times. Instead, the client workstation should use a temporary secret on behalf

of the user to access the mail server.

5. The security system should be capable of securely transmitting encryption keys among the clients and server.

KERBEROS SYSTEM:

Kerberos is a distributed network security system, which provides for authentication across unsecured networks. If requested by the application, integrity and encryption can also be provided. Kerberos was originally developed at the Massachusetts Institute of Technology (MIT) in the mid 1980s. There are two major releases of Kerberos, version 4 and 5, which are for practical purposes, incompatible.

Kerberos is a two-way third-party authentication relies on a symmetric key database using a key distribution center (KDC) which is known as the Kerberos server. A user or service (known as principals) is granted electronic "tickets" after properly communicating with the KDC. These

tickets are used for authentication between principals. All tickets include a time stamp, which limits the time period for which the ticket is valid. Therefore, Kerberos clients and server must have a secure time source, and be able to keep time accurately.

The practical side of Kerberos is its integration with the application level. Typical applications like FTP, telnet, POP, and NFS have been integrated with the Kerberos system. There are a variety of implementations which have varying levels of integration.

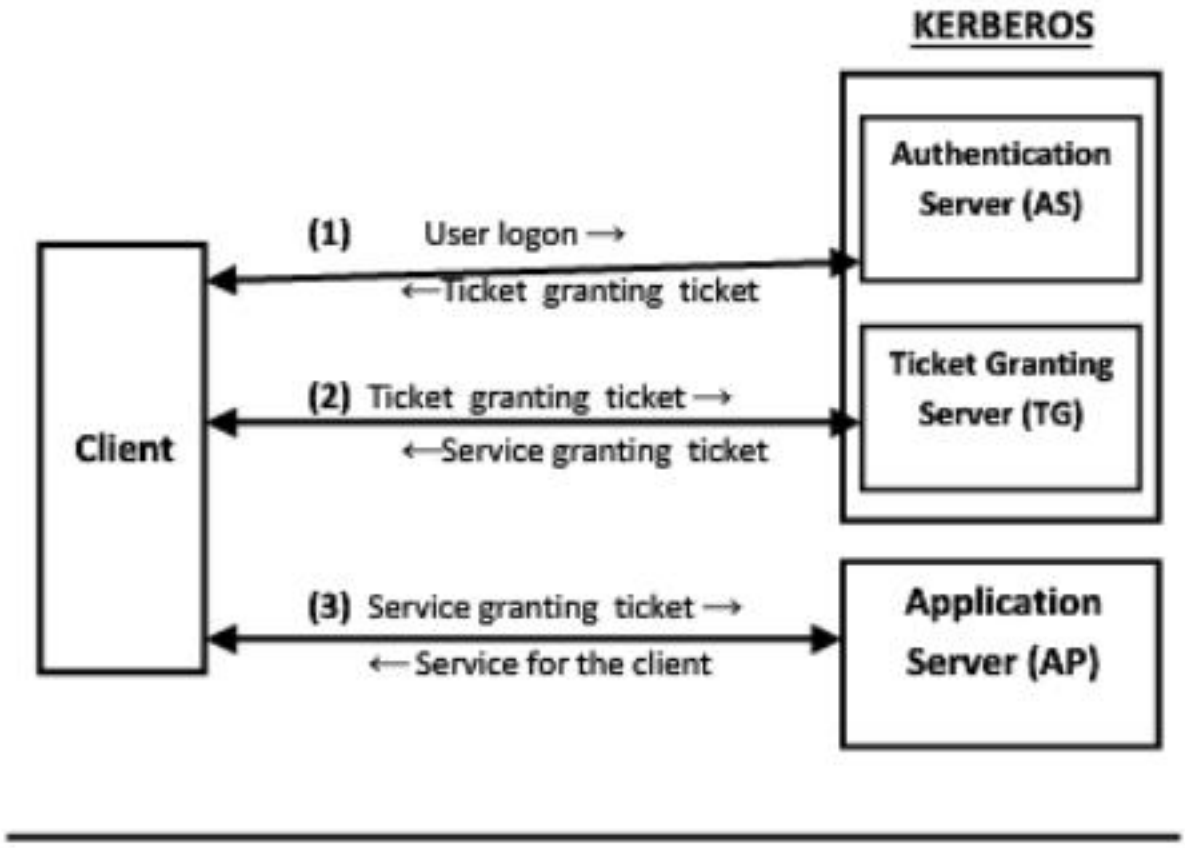

Figure 4: KERBEROS system

Table 1: KERBEROS Information Exchange.

Exchange	Frequency
1.Client-authentication server exchange	Once per user logon
2.Client ticket granting server exchange	Once per type of service required
3. Client application server exchange	Once per service request to any application server

The four entities are used in the Kerberos system as shown in figure 4 are:

1- **Client workstation:** client workstation is where the user interacts with the workstation and enters the ID and the password.

2- **Authentication Server (AS):** The authentication server provides the passwords storage and interacts with the client in authenticating the user. This interaction also includes providing a ticket-granting ticket to the client. The ticket-granting ticket from TG will be described in the next point. The service-granting ticket is a temporary secret to be used by the client to achieve

authentication with the application server.

3- **Ticket-Granting Server (TG):** The ticket-granting server provides a service granting ticket to the client for receiving service from an application server. The packet consists of TG's ID, a time stamp, a lifetime value, a session key, and a ticket-granting ticket. The time stamp informs the client of the time of ticket-granting ticket was issued. The lifetime parameter informs the client of the duration of the validity of this ticket-granting ticket.

4- **Application Server (AP):** The application server provides the desired services to the user through the client workstation.

5-3 PUBLIC KEY AUTHENTICATION:

The public key scheme doesn't require users to share or store their secret keys with any system. It uses one key for passwords encryption and different but related key for decryption.


THE
END

Chapter Five

FIREWALLS AND INTRUSION DETECTION SYSTEM

1- INTRODUCTION:

Network is more Vulnerable, because of:

1- Anonymity (no personal identification): An attacker can mount an attack from thousands of miles away and never come into direct contact with the system.

2- Many points of attack both targets and origins: A large network offers many points of vulnerability.

3- Sharing. Because networks enable resource and workload sharing, more users have the potential to access networked systems than on single computers.

4-Complexity of system: 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.

5-Unknown network boundary: A network's expandability 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.

6-Unknown path. there may be many paths from one host to another. The message might be routed through different hosts before arriving at destination host. Network users seldom have control over the routing of their messages.

2- NETWORK SECURITY GOALS:

- Identification: user ID for each application.
- Authentication: verification of the identity of user (password one-way authentication, application to user is two-way authentication)
- Authorization: is the process of enforcing access rights for network access to each user ID (access rights include read, write, update)
- Access Control: the process of enforcing access rights for network access
- Confidentiality: the process used to protect secret information from unauthorized disclosure.
- Data integrity: allows detection of unauthorized modification of data (even through transmission)
- Non repudiation: is the capability to provide proof of the origin of data or proof of the delivery of data

3-ATTACKS TO COMPUTER NETWORKS:

- Breaching secret data – Confidential data is often stored and transmitted in encrypted form weak encryption or lack

of protection lead to data breaching.

- Unauthorized logons results from misuse of stolen and guessed passwords, lack of authentication
- Unauthorized denial of service the hacker is interested in shutting down the computer system, degrading its performance, consuming its resources i.e affect the availability and is usually done by injecting malicious software.
- Network Spoofing occurs when one host on the network is used to impersonate another host
- Denial of Services: when attacker consumes a resource so that no one else can use it.

4- DEFINITION OF A FIREWALL SYSTEM:

A firewall is a device that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network. 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. Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important. 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.

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:

(i)Default permits. (2) Default denies.

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.

The design of Firewalls must satisfies the following features: (i)Always invoked , (ii)Tamperproof, and (iii)Small and simple enough for rigorous analysis

"**Always invoked**"; We can ensure that all network accesses that we want to control must pass through the firewall by carefully positioning it within a network, it.

"**Tamperproof**"; 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.

Finally, firewall designers strongly recommend keeping the functionality of the firewall simple.

2-Types of Firewalls:

Types of firewalls include:

1- Packet filtering gateways (or screening routers)
2- Stateful inspection firewalls
3- Application proxies

4- Guards

5- Personal firewalls

Each type does different things; no one is necessarily "right" and the others "wrong." In this section, we examine each type to see what it is, how it works, and what its strengths and weaknesses are. In general, screening routers tend to implement rather simplistic security policies, whereas guards and proxy gateways have a richer set of choices for security policy. Simplicity in a security policy is not a bad thing; the important question to ask when choosing a type of firewall is what threats an installation needs to counter.

2-1 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). 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) are beyond the capability of a packet filter. Figure 1 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.

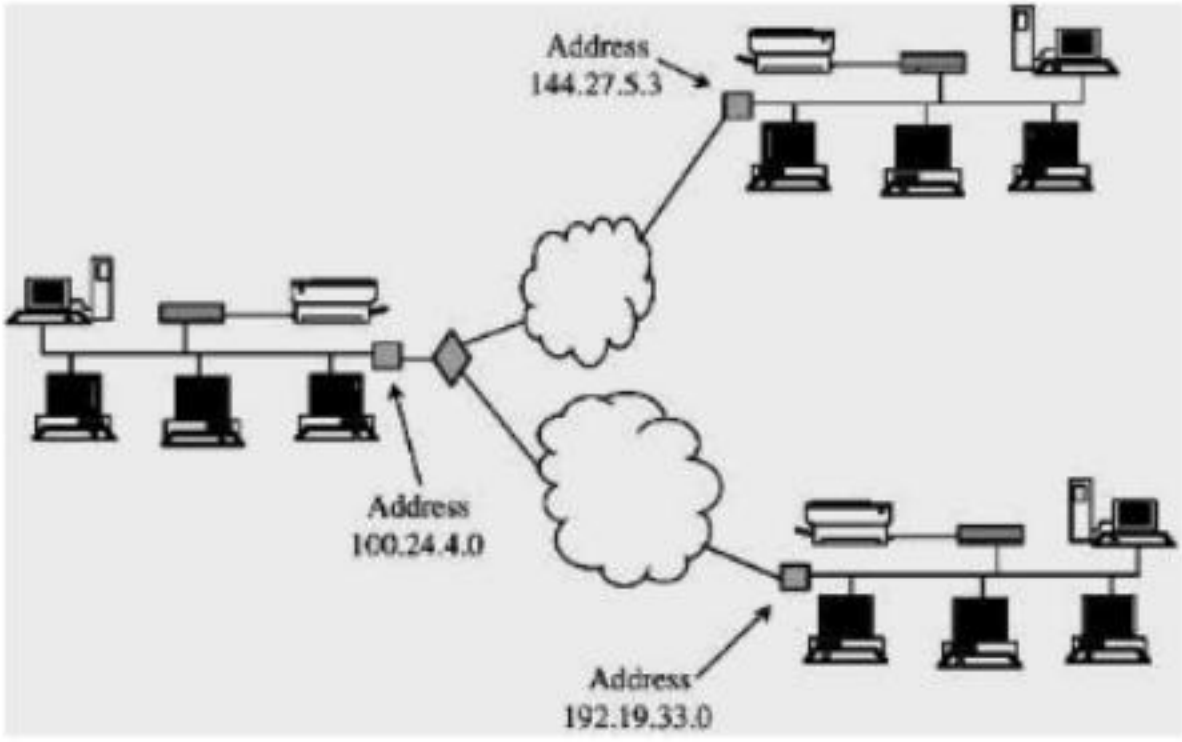

Figure 1. Packet Filter Blocking Addresses and Protocols.

For example, suppose an international company has three LANs at three locations throughout the world, as shown in figure 2. 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

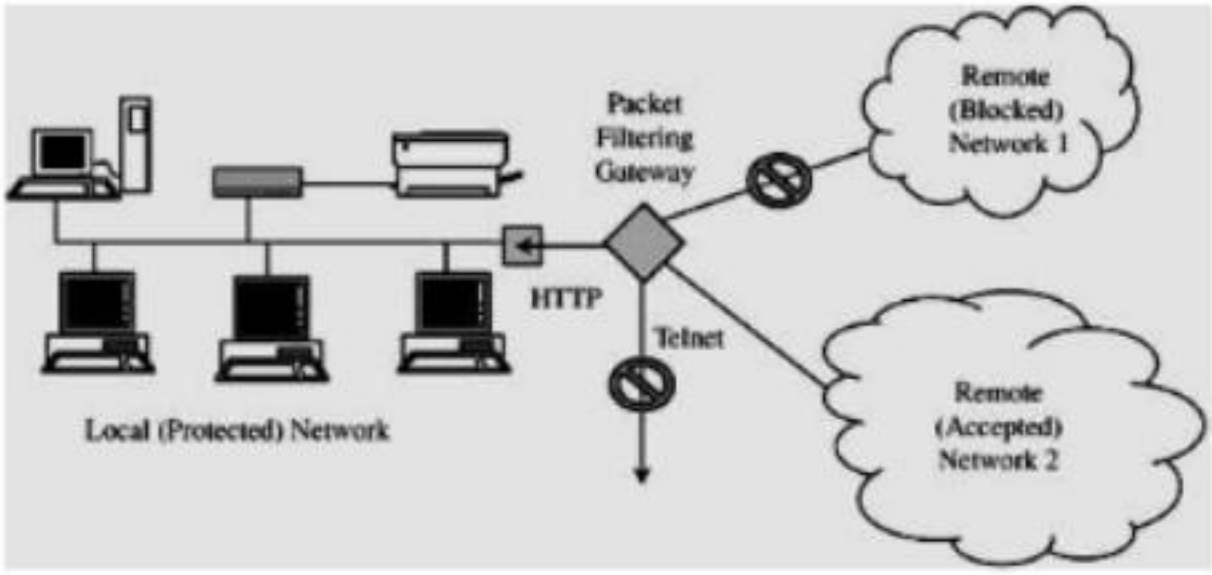

Figure 2. Three Connected LANs.

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 3. 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, it permits in any packets with destination 100.50.25.x).

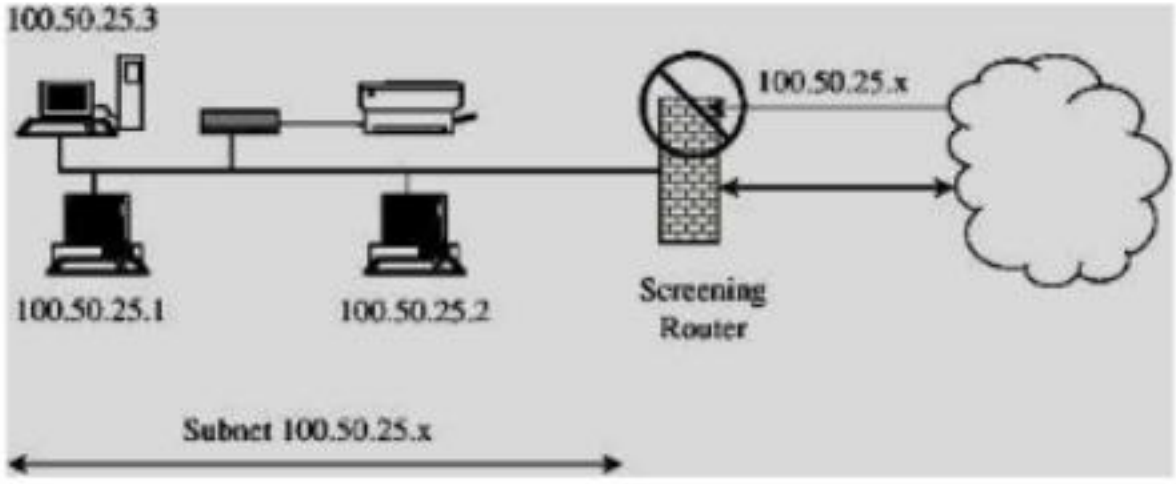

Figure 3. Filter Screening Outside Addresses.

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.

2-2 Stateful Inspection Firewall:

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. 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.

2-3 Application Proxy:

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.

Packet filters look only at the headers of packets, not at the data inside the packets. Therefore, a packet filter would pass anything to certain port, 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 runs pseudo applications. 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.

1- 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.

2- 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.

3- 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.

4- 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.)

5- 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 challenge response system), which many

operating systems do not require. These functions are shown in Figure 4.

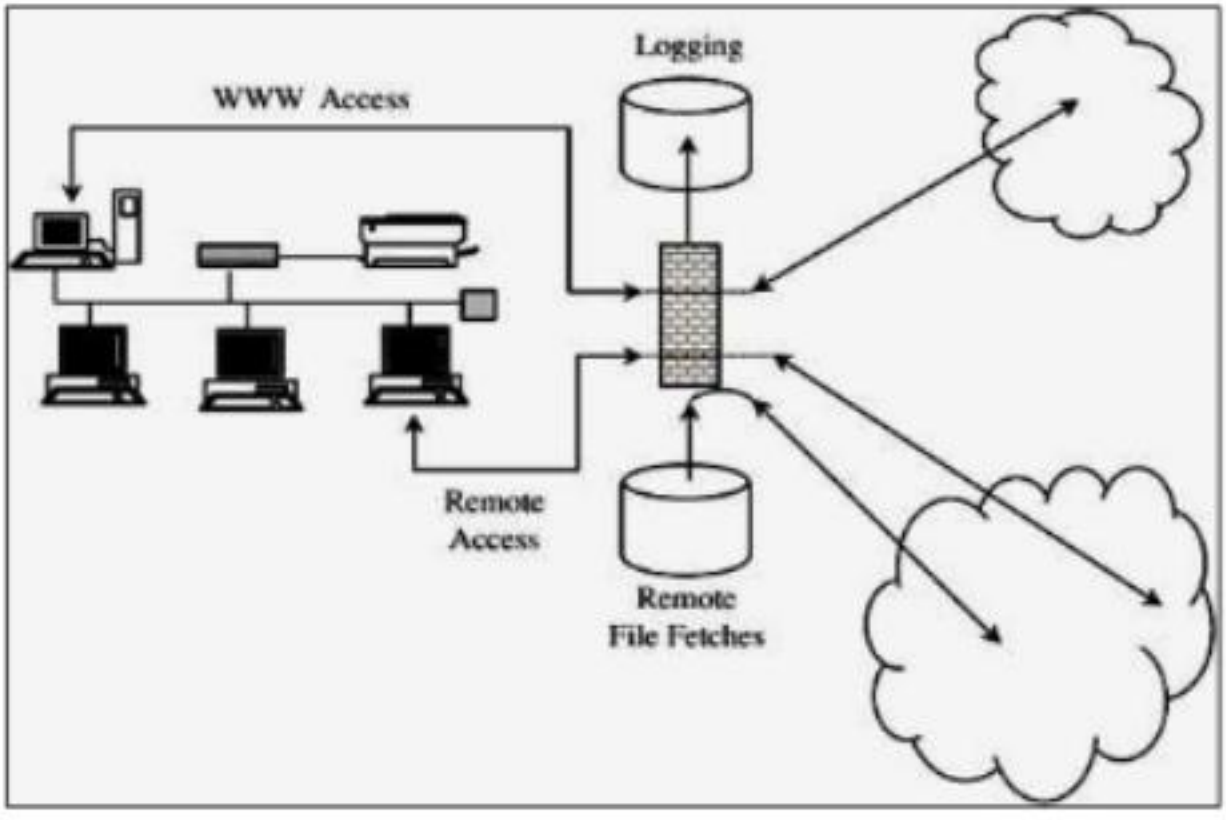

Figure 4. Actions of Firewall Proxies.

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.

2-4 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:

1- 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.

2- 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).

3- 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.

4- 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 non-executable text or graphics, the company administrator thinks that the expense of scanning them (which should pass) will be negligible.

2-5 Personal Firewalls

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.

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 screens traffic on a single workstation. 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. 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 fact such 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.

3- Example Firewall Configurations

The simplest use of a firewall is shown in Figure 5. 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

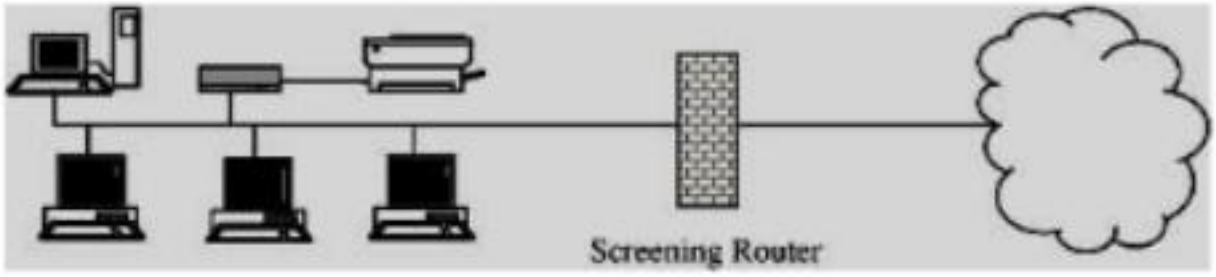

Figure 5. Firewall with Screening Router.

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 6. In this way the only traffic visible on that LAN is the traffic going into and out of the firewall.

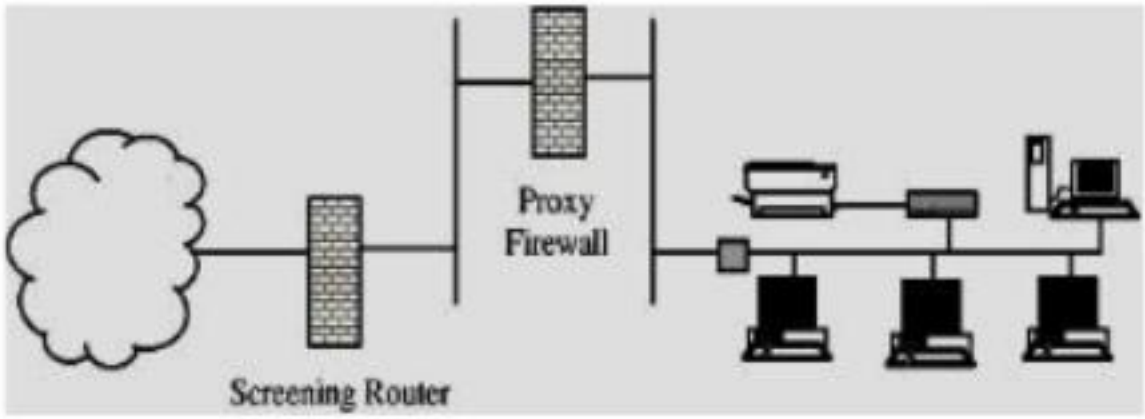

Figure 6. Firewall on Separate LAN.

4-What Firewalls Can and Cannot Block :

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.

1- 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.

1- 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.

3- 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.

4- 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.

5- 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.

6- Methods of Attacks

These are the common or well-known classes of attacks or attack methodologies:

1- Brute force and dictionary
2- Denial of service
3- Spoofing
4- Man-in-the-middle attacks
5- Spamming
6- Sniffers
7- Crackers

5-1 Brute Force and Dictionary Attacks

A brute force attack is an attempt to discover passwords for user accounts by systematically attempting every possible combination of letters, numbers, and symbols. With the speed of modern computers and the ability to employ distributed computing, brute force attacks are becoming successful even against strong passwords. With enough time, all passwords can be discovered using a brute force attack method. Brute force and dictionary attacks are often discussed together because they are waged against the same entity: passwords. Either type of attack can be waged against a password database file or against an active logon prompt.

A *dictionary attack* is an attempt to discover passwords by attempting to use every possible password from a predefined list of common or expected passwords. This type of attack is named such because the possible

password list is so long it is as if you are using the entire dictionary one word at a time to discover passwords.

Password attacks employ a specify cryptographic attack method known as the birthday attack. This attack can also be called reverse hash matching or the exploitation of collision. Basically, the attack exploits the fact that if two messages are hashed and the hash values are the same, then the two messages are probably the same. A way of expressing this in mathematical or cryptographic notation is H(M)=H(M'). Passwords are stored in an accounts database file on secured systems. However, instead of being stored as plain text, passwords are hashed and only their hash values are actually stored. This provides a reasonable level of protection. However, using reverse hash matching, a password cracker tool looks for possible passwords (through either brute force or dictionary methods) that have the same hash value as a value stored on the accounts database file. When a hash value match is discovered, then the tool is said to have cracked the password. Combinations of these two password attack methodologies can be used as well. For example, a brute force attack could use a dictionary list as the source of its guesswork.

Dictionary attacks are often successful due to the predictability of human nature to select passwords based on personal experiences. Unfortunately, those personal experiences are often broadcast to the world around you simply by the way you live and act on a daily basis. The

more data about a victim learned through intelligence gathering, dumpster diving, and social engineering, the more successful a custom dictionary list will be.

Protecting passwords from brute force and dictionary attacks requires numerous security precautions and rigid adherence to a strong security policy, which are:

First, physical access to systems must be controlled. If a malicious entity can gain physical access to an authentication server, they can often steal the password file within seconds. Once a password file is stolen, all passwords should be considered compromised.

Second, tightly control and monitor electronic access to password files. End users and non–account administrators have no need to access the password database file for normal daily work tasks. If you discover an unauthorized access to the database file, investigate immediately. If you

Can not determine that a valid access occurred, then consider all passwords compromised.

Third, craft a password policy that programmatically enforces strong passwords and prescribe means by which end users can create stronger passwords. The stronger and longer the password, the longer it will take for it to be discovered in a brute force attack. However, with enough time, *all* passwords can be discovered via brute force methods. Thus, changing passwords regularly is required to maintain security. Static passwords older than 30 days should be considered

compromised even if no other aspect of a security breach has been discovered.

Fourth, deploy two-factor authentication, such as using biometrics or token devices. If passwords are not the only means used to protect the security of a network, their compromise will not automatically result in a system breach.

Fifth, use account lockout controls to prevent brute force and dictionary attacks against logon prompts. For those systems and services that don't support account lockout controls,such as most FTP servers, employ extensive logging and an IDS to look for attempted fast and slow password attacks.

Sixth, encrypt password files with the strongest encryption available for your OS. Maintain rigid control over all media that have a copy of the password database file, such as backup tapes and some types of boot or repair disks.

Passwords are a poor security mechanism when used as the sole deterrent against unauthorized access. Brute force and dictionary attacks show that passwords alone offer little more than a temporary blockade.

5-2 Denial of Service

Denial of service (DoS) attacks are attacks that prevent the system from processing or responding to legitimate traffic or requests for resources and objects. The most common form of denial of service attacks is transmitting so many data packets to a server that it cannot processes them all. Other forms of denial of service attacks focus

on the exploitation of a known fault or vulnerability in an operating system, service, or application. Exploiting the fault often results in system crash or 100 percent CPU utilization. No matter what the actual attack consists of, any attack that renders the victim unable to perform normal activities can be considered a denial of service attack. Denial of service attacks can result in system crashes, system reboots, data corruption, blockage of services, and more.

Unfortunately, denial of service attacks based on *flooding* (i.e., sending sufficient traffic to a victim to cause a DoS) a server with data are a way of life on the Internet. In fact, there are no known means by which denial of service flood attacks in general can be prevented. Furthermore, due to the ability to spoof packets or exploit legitimate Internet services, it is often impossible to trace the actual origin of an attack and apprehend the culprit.

There are several types of DoS flood attacks. The first, or original, type of attack employed a single attacking system flooding a single victim with a steady stream of packets. Those packets could be valid requests that were never completed or malformed or fragmented packets that consume the attention of the victimized system. This simple form of DoS is easy to terminate just by blocking packets from the source IP address.

Another form of attack is called the *distributed denial of service (DDoS)*. A distributed denial of service occurs when the attacker compromises several systems and

uses them as launching platforms against one or more victims. The compromised systems used in the attack are often called slaves or zombies. A DDoS attack results in the victims being flooded with data from numerous sources. DDoS attacks can be stopped by blocking packets from the compromised systems. But this can also result in blocking legitimate traffic because the sources of the flood

packets are victims themselves and not the original perpetrator of the attack. These types of attacks are labeled as distributed because numerous systems are involved in the propagation of the attack against the victim.

A more recent form of DoS, called a *distributed reflective denial of service (DRDoS),* has been discovered. DRDoS attacks take advantage of the normal operation mechanisms of key Internet services, such as DNS and router update protocols. DRDoS attacks function by sending numerous update, session, or control packets to various Internet service servers or routers with a spoofed source address of the intended victim. Usually these servers or routers are part of the high-speed, high-volume Internet backbone trunks. What results is a flood of update packets, session acknowledgment responses, or error messages sent to the victim. A DRDoS attack can result in so much traffic that upstream systems are adversely affected by the sheer volume of data focused on the victim. This type of attack is called a reflective attack because the

high-speed backbone systems reflect the attack to the victim. Unfortunately, these types of attacks cannot be prevented because they exploit normal functions of the systems. Blocking packets from these key Internet systems will effectively cut the victim off from a significant section of the Internet.

Not all instances of DoS are the result of a malicious attack. Errors in coding operating systems, services, and applications have resulted in DoS conditions. For example, a process failing to release control of the CPU or a service consuming system resources out of proportion to the service requests it is handling can cause DoS conditions. Most vendors quickly release patches to correct these self-inflicted DoS conditions, so it is important to stay informed.

There have been many forms of DoS attacks committed over the Internet. Some of the more popular ones.

A *SYN flood attack* is waged by breaking the standard three-way handshake used by TCP/IP to initiate communication sessions. Normally, a client sends a SYN packet to a server, the server responds with a SYN/ACK packet to the client, and the client then responds with an ACK packet back to the server. This three-way handshake establishes a communication session that is used for data transfer until the session is terminated (using a three-way handshake with FIN and ACK packets). A SYN flood occurs when numerous SYN packets are sent to a server but the sender never

replies to the server's SYN/ACK packets with the final ACK.

A *Smurf attack* occurs when an amplifying server or network is used to flood a victim with useless data. An amplifying server or network is any system that generates multiple response packets, such as ICMP ECHO packets or special UDP packets, from a single submitted packet.

One common attack is to send a message to the broadcast of a subnet or network so that every node on the network produces one or more response packets. The attacker sends information request packets with the victim's spoofed source address to the amplification system. Thus, all of the response packets are sent to the victim. If the amplification network is capable of producing sufficient response packet traffic, the victim's system will experience a DoS. Figure 9.

shows the basic elements of a Smurf attack. The attacker sends multiple IMCP PING packets with a source address spoofed as the victim (V) and a destination address that is the same as the broadcast address of the amplification network (AN:B). The amplification network responds with multiplied volumes of echo packets to the victim, thus fully consuming the victim's connection bandwidth. Another DoS attack similar to Smurf is called Fraggle. Fraggle attacks employ spoofed UDP packets rather than ICMP packets.

Countermeasures for Smurf attacks include disabling directed broadcasts on all network border routers and configuring all systems to drop ICMP ECHO packets. An IDS may be able to detect this type of attack, but there are no means to prevent the attack other than blocking
the addresses of the amplification network. This tactic is problematic because the amplification network is usually also a victim.

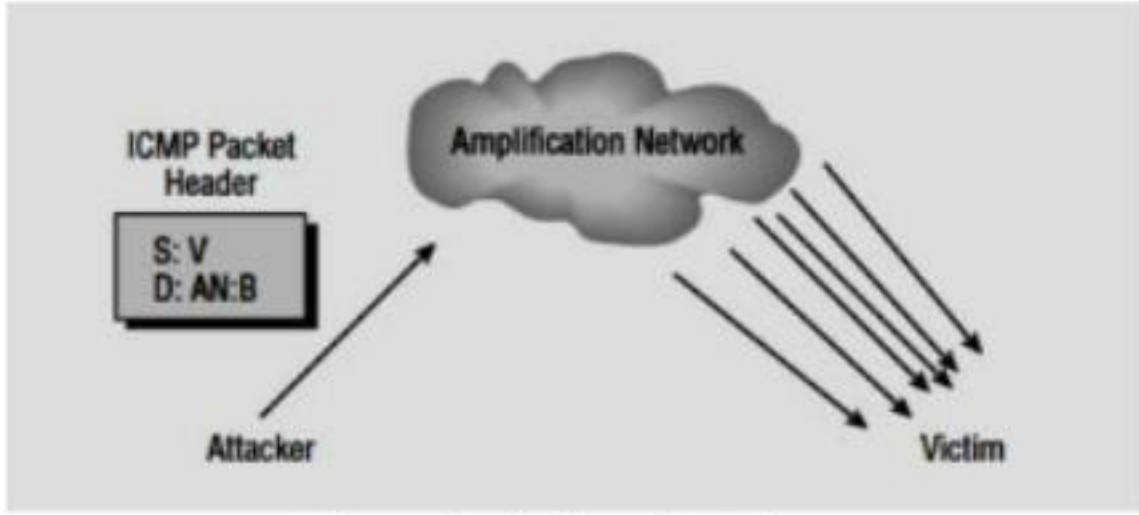

Figure 9 : A Smurf attack

A *ping of death attack*

Employs an oversized ping packet. Using special tools, an attacker can send numerous oversized ping packets to a victim. In many cases, when the victimized system attempts to process the packets, an error occurs, causing the system to freeze, crash, or reboot. The ping of death is more of a buffer overflow attack, but because it often results in a downed server, it is considered a DoS attack. Countermeasures to the ping of death attack include keeping up-to-date with OS and software patches,

properly coding in-house applications to prevent buffer overflows, avoiding running code with system- or root-level privileges, and blocking ping packets at border routers/firewalls.

A *WinNuke* attack is a specialized assault against Windows 95 systems. Out-of-band TCP data is sent to a victim's system, which causes the OS to freeze. Countermeasures for this attack consist of updating Windows 95 with the appropriate patch or changing to a different OS.

A *stream attack*

occurs when a large number of packets are sent to numerous ports on the victim system using random source and sequence numbers. The processing performed by the victim system attempting to make sense of the data will result in a DoS. Countermeasures include patching the system and using an IDS for dynamic blocking.

A *teardrop attack*

occurs when an attacker exploits a bug in operating systems. The bug exists in the routines used to reassemble (i.e., resequence) fragmented packets. An attacker sends numerous specially formatted fragmented packets to the victim, which causes the system to freeze or crash. Countermeasures for this attack include patching the OS and deploying an IDS for detection and dynamic blocking.

A *land attack*

occurs when the attacker sends numerous SYN packets to a victim and the SYN packets have been spoofed to use the same source and destination IP address and port number as the victim. This causes the system to think it sent a TCP/IP session opening packet to itself, which causes a system failure and usually results in a system freeze, crash, or reboot.

Countermeasures for this attack include patching the OS and deploying an IDS for detection and dynamic blocking.

5-3 Spoofing Attacks

Spoofing is the art of pretending to be something other than what you are. *Spoofing attacks* consist of replacing the valid source and/or destination IP address and node numbers with false ones.

Spoofing is involved in most attacks because it grants attackers the ability to hide their identity through misdirection. Spoofing is employed when an intruder uses a stolen username and password to gain entry, when an attacker changes the source address of a malicious packet, or when an attacker assumes the identity of a client to fool a server into transmitting controlled data.

Two specific types of spoofing attacks are impersonation and masquerading. Ultimately, these attacks are the same: someone is able to gain access to a secured system by pretending to be someone else. These attacks often result in an unauthorized person gaining

access to a system through a valid user account that has been compromised. Impersonation is considered a more active attack because it requires the capture of authentication traffic and the replay of that traffic in such a way as to gain access to the system. Masquerading is considered a more passive attack because the attacker uses previously stolen account credentials to log on to a secured system.

Countermeasures to spoofing attacks include patching the OS and software, enabling source/destination verification on routers, and employing an IDS to detect and block attacks. As a general rule of thumb, whenever your system detects spoofed information, it should record relevant data elements into a log file; then the system should drop or delete the spoof itself.

5-4 Man-in-the-Middle Attacks

A *man-in-the-middle attack* occurs when a malicious user is able to gain a position between the two endpoints of a communications link. There are two types of man-in-the-middle attacks.

One involves copying or sniffing the traffic between two parties; this is basically a sniffer attack (see the next section). The other involves attackers positioning themselves in the line of communication where they act as a store-and-forward or proxy mechanism (see Figure 10). The attacker functions as the receiver for data transmitted by the client and the transmitter for data sent to the server. The attacker is invisible to both ends of the communication link and is able to alter the content

or flow of traffic. Through this type of attack, the attacker can collect logon credentials or sensitive data as well as change the content of the messages exchanged between the two endpoints.

To perform this type of attack, the attacker must often alter routing information and DNS values, steal IP addresses, or defraud ARP lookups to impersonate the server from the perspective of the client and to impersonate the client from the perspective of the server. An offshoot of a man-in-the-middle attack is known as a *hijack attack.;*In this type of attack, a malicious user is positioned between a client and server and then interrupts the session and takes it over. Often, the malicious user impersonates the client to extract data from the server.

The server is unaware that any change in the communication partner has occurred. The client is aware that communications with the server have ceased, but no indication as to why the communications were terminated is available.

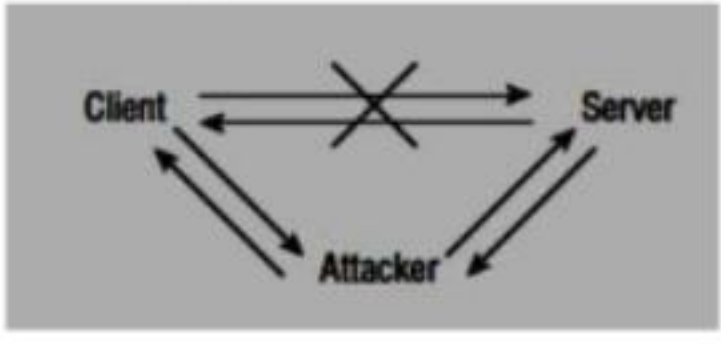

Figure 10 : A man-in-the-middle attack

Another type of attack, a *replay attack* (also known as a *playback attack*), is similar to hijacking. A malicious user records the traffic between a client and server; then the packets sent from the client to the server are played back or retransmitted to the server with slight variations of the time stamp and source IP address (i.e., spoofing). In some cases, this allows the malicious user to restart an old communication link with a server. Once the ommunication session is reopened, the malicious user can attempt to obtain data or additional access. The captured traffic is often authentication traffic (i.e., that which includes logon credentials, such as username and password), but it could also be service access traffic or message control traffic. Replay attacks can be prevented by employing complex sequencing rules and time stamps to prevent retransmitted packets from being accepted as valid.

Countermeasures to these types of attacks require improvement in the session establishment, identification, and authentication processes. Some man-in-the-middle attacks are thwarted through patching the OS and software. An IDS cannot usually detect a man-in-the-middle or hijack attack, but it can often detect the abnormal activities occurring via ~~-~~secured" communication links. Operating systems and many IDSs can often detect and block replay attacks.

5-5 Sniffer Attacks

A *sniffer attack* (also known as a *snooping attack*) is any activity that results in a malicious user obtaining information about a network or the traffic over that network. A sniffer is often a packet capturing program that duplicates the contents of packets traveling over the network medium into a file. Sniffer attacks often focus on the initial connections between clients and servers to obtain logon credentials (e.g., usernames and passwords), secret keys, and so on. When performed properly, sniffing attacks are invisible to all other entities on the network and often precede spoofing or hijack attacks. A replay attack (discussed in the preceding section) is a type of sniffer attack.

Countermeasures to prevent or stop sniffing attacks require improvement in physical access control, active monitoring for sniffing signatures (such as looking for packet delay, additional routing hops, or lost packets, which can be performed by some IDSs), and using encrypted traffic over internal and external network connections.

5-6 Spamming Attacks

Spam is the term describing unwanted e-mail, newsgroup, or discussion forum messages. Spam can be as innocuous as an advertisement from a well-meaning vendor or as malignant as floods of unrequested messages with viruses or Trojan horses attached. Spam is usually not a security threat but rather a type of denial of service attack. As the level of spam increases,

locating or accessing legitimate messages can be difficult. In addition to the nuisance value, spam consumes a significant portion of Internet resources (in the form of bandwidth and CPU processing), resulting in overall slower Internet performance and lower bandwidth availability for everyone.

Spamming attacks are directed floods of unwanted messages to a victim's e-mail inbox or other messaging system. Such attacks cause DoS issues by filling up storage space and preventing legitimate messages from being delivered. In extreme cases, spamming attacks can cause system freezes or crashes and interrupt the activity of other users on the same subnet or ISP.

Spam attack countermeasures include using e-mail filters, e-mail proxies, and IDSs to detect, track, and terminate spam flood attempts.

5-7 Crackers

Crackers are malicious users intent on waging an attack against a person or system. Crackers may be motivated by greed, power, or recognition. Their actions can result in stolen property (data, ideas, etc.), disabled systems, compromised security, negative public opinion, loss of market share, reduced profitability, and lost productivity.

A term commonly confused with *crackers* is *hackers,* who are technology enthusiasts with no malicious intent. Many authors and the media often use the term *hacker* when they are actually discussing issues relating to crackers.

Thwarting a cracker's attempts to breach your security or perpetrate DoS attacks requires vigilant effort to keep systems patched and properly configured. IDSs and honey pot systems often offer means to detect and gather crackers once they have breachedyour controlled perimeter

6-INTRUSION DETECTION SYSTEM:

An intrusion detection system (IDS) is a device, typically another separate computer that:

1- Monitors activity to identify malicious or suspicious events (used to detect intrusion attempts)

2- IDS can also used in detecting and identifying unauthorized or unusual activity on the system, Such a scheme requires specification of what constitute an undesirable activity and a means of automatically detecting such activity as it occurs.

3- IDS's receives raw inputs from sensors. It saves those inputs, analyzes them, and takes some controlling action.

4- IDSs watch for violations of confidentiality, integrity, and availability.

Attacks recognized by an IDS can come from external connections (such as the Internet or partner networks), viruses, malicious code, trusted internal subjects attempting to perform unauthorized activities, and unauthorized access attempts from trusted locations. An IDS is considered a form of a technical detective security

control.

An IDS can actively watch for suspicious activity, peruse audit logs, send alerts to administrators when specific events are discovered, lock down important system files or capabilities, track slow and fast intrusion attempts, highlight vulnerabilities, identify the intrusion's origination point, track down the logical or physical location of the perpetrator, terminate or interrupt attacks or intrusion attempts, and reconfigure routers and firewalls to prevent repeats of discovered attacks. IDS alerts can be sent or communicated with an on-screen notification (the most common), by playing a sound, via e-mail, via pager, or by recording information in a log file.

A response by an IDS can be active, passive, or hybrid. An active response is one that directly affects the malicious activity of network traffic or the host application. A passive response is one that does not affect the malicious activity but records information about the issue and notifies the administrator. A hybrid response is one that stops unwanted activity, records information about the event, and possibly even notifies the administrator.

Typically, the responses an IDS can take against an attack include port blocking, source address blocking, and disabling all communications over a specific cable segment. Whenever an IDS discovers abnormal traffic (e.g., spoofed) or violations of its security

policy, filters, and rules, it records a log detail of the issue and then drops, discards, or deletes the relevant packets.

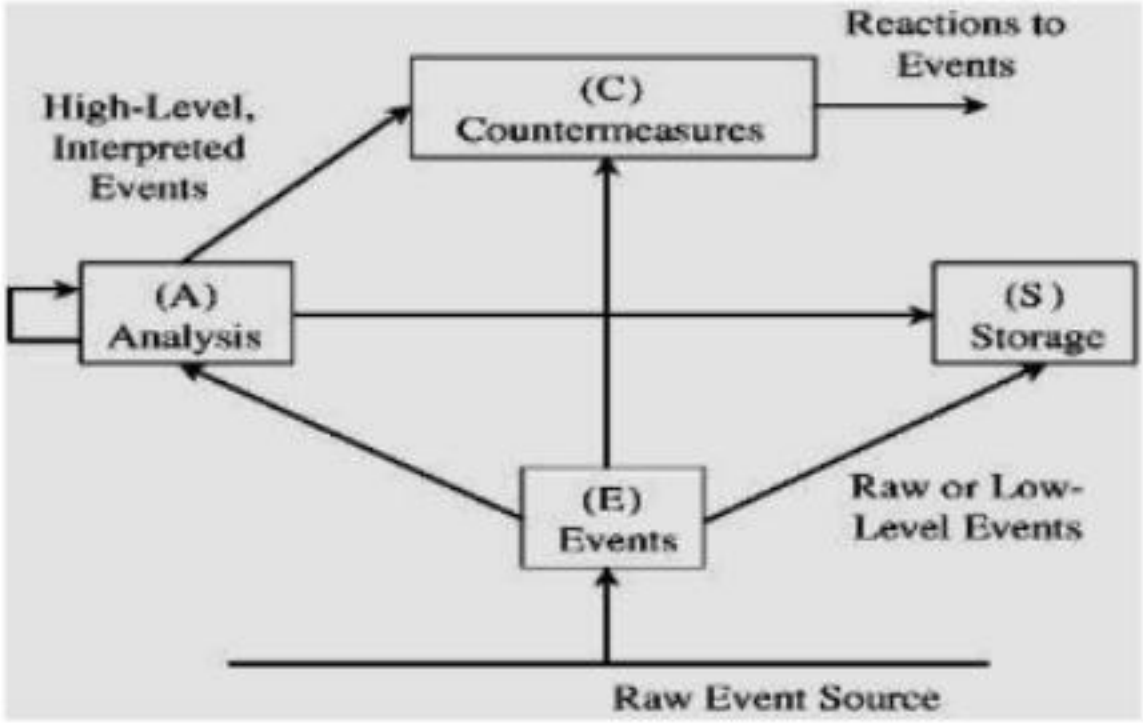

Figure 7: Common components of an intrusion detection framework

When an intrusion is detected, your first response should be to contain the intrusion. Intrusion containment prevents additional damage to other systems but may allow the continued infestation of already compromised systems. Later, once compromised systems are rebuilt from scratch, be sure to double-check compliance with your security policy—including checking ACLs, service configurations, and user account settings—before

connecting the reestablished system to your network. You should realize that if you wipe and re-create a system, none of the previous system, nor any intrusion footprints, will remain.

6-1 Types of IDS's:

IDS types are most commonly classified **by their performance** into two primary types of IDSs:

1- Signature-based intrusion detection systems: perform simple pattern-matching and report situations that match a pattern corresponding to a known attack type.

2- Heuristic intrusion detection systems, " 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 or not so that the heuristic IDS will now treat that previously unclassified behavior as how it was classified.

Profiles characterize the normal behavior of the subjects on an objects so the IDS can detect the abnormal.

Three candidates profiles:

1-logon and session activity

2-command or program usage

3-file access activity

Anomaly record is created when abnormal behavior appears, it consists of *–event, time, profile*"

Table 2: Denning's IDS model

Format	Example
Subject	Jhon (action initiator)
Action	File write
Object	Employee record file
Exception condition	No
Resources usage	10
Time stamp	0800 02112010

IDS types are most commonly classified **by their information source**. into two primary types of IDSs:

(i) Host based and (ii) Network based.

1- Host-Based IDS

A host-based IDS watches for questionable activity on a single computer system. Because the attention of a host-based IDS is focused on a single computer (whereas a network-based IDS must monitor the activity on an entire network), it can examine events in much greater detail than a network-based IDS can. A host-based IDS is able to pinpoint the files and processes compromised or

employed by a malicious user to perform unauthorized activity.

Host-based IDSs can detect anomalies undetected by network-based IDSs; however, a host-based IDS cannot detect network-only attacks or attacks on other systems. Because a host-based IDS is installed on the computer being monitored, crackers can discover the IDS software and disable it or manipulate it to hide their tracks. A host-based IDS has some difficulty with detect-ing and tracking down denial of service (DoS) attacks, especially those of a bandwidth con-sumption nature. A host-based IDS also consumes resources from the computer being monitored, thereby reducing the performance of that system. A host-based IDS is limited by the auditing capabilities of the host operating system and applications.

Host-based IDSs are considered more costly to manage than network-based IDSs. Host-based IDSs require that an installation on each server be monitored and require administrative attention at each point of installation, while network-based IDSs usually only require a single installation point. Host-based IDSs have other disadvantages as well; for example, they cause a significant host system performance degradation and they are easier for an intruder to discover and disable.

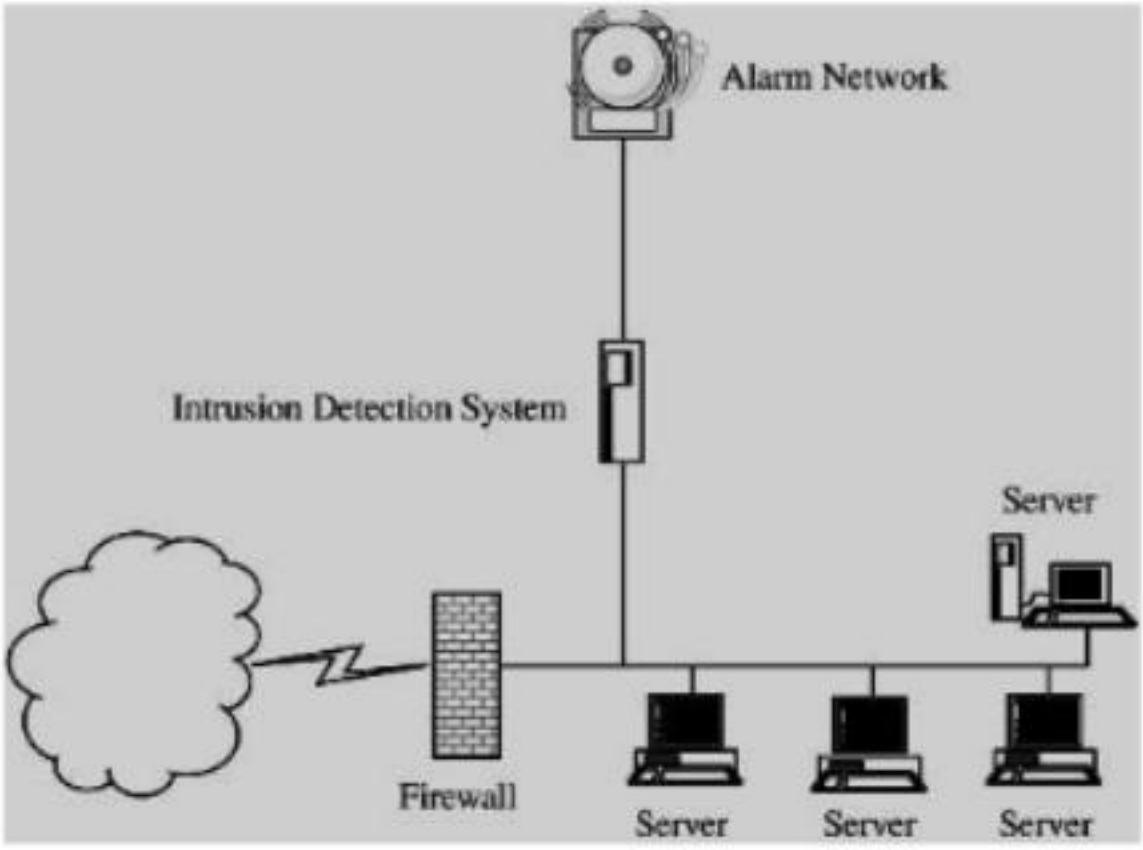

Figure 8: Stealth Mode IDS Connected to Two Networks.

2- Network-Based IDS:

A network-based IDS watches for questionable activity being performed over the network medium. Network-based IDSs detect attacks or event anomalies through the capture and evaluation of network packets. A single network-based IDS is capable of monitoring a large network if installed on a backbone of that network, where a majority of the network traffic occurs. Some versions of network-based IDSs use remote agents to collect data from various subnets and report to a central management console. Network-based IDSs are installed

onto single-purpose computers. This allows them to be hardened against attack, reduces the number of vulnerabilities to the IDS, and allows the IDS to operate in stealth mode. In stealth mode, the IDS is invisible to the network and intruders would have to know of its exact location and system identification to discover it. A network-based IDS has little negative affect on overall network performance, and because it is deployed on a single-purpose system, it doesn't adversely affect the performance of any other computer.

On networks with extremely large volumes of traffic, a network-based IDS may be unable to keep up with the flow of data. This could cause the IDS to miss an attack that occurred during high traffic levels. Network-based IDSs do not usually work well on switched networks, especially if the routers do not have a monitoring port. Network-based IDSs are used to monitor the content of traffic if it is encrypted during transmission over the network medium. They are usually able to detect the initiation of an attack or the ongoing attempts to perpetrate an attack (including DoS), but they are unable to provide information about whether an attack was successful or which specific systems, user accounts, files, or applications were affected.

Often, a network-based IDS can provide some limited functionality for discovering the source of an attack by performing Reverse Address Resolution Protocol (RARP) or Domain Name System (DNS) lookups. However, because most attacks are launched by

malicious individuals whose identity is masked through spoofing, this is not usually a fully reliable system capability.

An IDS should not be viewed as a single universal security solution. It is only part of a multi-faceted security solution for an environment. Although an IDS can offer numerous benefits, there are several drawbacks to consider. A host-based IDS may not be able to examine every detail if the host system is overworked and insufficient execution time is granted to the IDS processes. A network-based IDS can suffer the same problem if the network traffic load is high and it is unable to process packets efficiently and swiftly. A network-based IDS is also unable to examine the contents of encrypted traffic. A network-based IDS is not an effective network-wide solution on switched networks because it is unable to view all network traffic if it is not placed on a mirror port (i.e., a port specifically configured to send all data to the IDS). An IDS may initially produce numerous false alarms and requires significant management on an ongoing basis detection. Basically, the IDS uses a signature database and attempts to match all monitored events to it. If events match, then the IDS assumes that an attack is taking place (or has taken place). The IDS vendor develops the suspect chart by examining and inspecting numerous intrusions on various systems. What results is a description, or signature, of common attack methods. An IDS using knowledge-based detection functions in much the same

way as many antivirus applications.

The primary drawback to a knowledge-based IDS is that it is effective only against known attack methods. New attacks or slightly modified versions of known attacks often go unrecognized by the IDS. This means that the knowledge-based IDS lacks a learning model; that is, it is unable to recognize new attack patterns as they occur. Thus, this type of IDS is only as useful as its signature file is correct and up-to-date. Keeping the signature file current is an important aspect in maintaining the best performance from a knowledge-based IDS.

The second detection type is behavior-based detection. A behavior-based IDS is also called statistical intrusion detection, anomaly detection, and heuristics-based detection. Basically, behavior-based detection finds out about the normal activities and events on your system through watching and learning. Once it has accumulated enough data about normal activity, it can detect abnormal and possible malicious activities and events.

A behavior-based IDS can be labeled an expert system or a pseudo artificial intelligence system because it can learn and make assumptions about events. In other words, the IDS can act like a human expert by evaluating current events against known events. The more information provided to a behavior-based IDS about normal activities and events, the more accurate its anomaly detection becomes.

The primary drawback of a behavior-based IDS is that it produces many false alarms. The normal pattern of user and system activity can vary widely, and thus establishing a definition of normal or acceptable activity can be difficult. The more a security detection system creates false alarms, the less likely security administrators will heed its warnings, just as in the fable of the boy who cried wolf. Over time, the IDS can become more efficient and accurate, but the learning process takes considerable time. Using known behaviors, activity statistics, and heuristic evaluation of current versus previous events, a behavior-based IDS can detect unforeseen, new, and unknown vulnerabilities, attacks, and intrusion methods.

Although knowledge-based and behavior-based detection methods do have their differences, both employ an alarm-signal system. When an intrusion is recognized or detected, an alarm is triggered. The alarm system can notify administrators via e-mail or pop-up messages or by executing scripts to send pager messages. In addition to administrator notification, the alarm system can record alert messages in log and audit files as well as generate violation reports detailing the detected intrusions and discoveries of vulnerabilities.


THE
END

CHAPTER 6

TRUSTED OPERATING SYSTEMS

1- The meaning of Trusted Operating Systems:

Trusted software is often used as a safe way for general users to access sensitive data. Trusted programs are used to perform limited (safe) operations for users without allowing the users to have direct access to sensitive data.

An Operating System is trusted if we have confidence that it provides the following four services consistently and effectively: (i)Memory protection, (ii)File protection, (iii)General object access control, (iv)User authentication.

However, software is trusted software if we know that the code has been rigorously developed and analyzed, giving us reason to trust that the code does what it is expected to do and nothing more. Typically, trusted code can be a foundation on which other, untrusted, code runs. That is, the untrusted system's quality depends, in part, on the trusted code; the trusted code establishes the baseline for security of the overall system. In particular, an operating system can be trusted software when there is a basis for trusting that it correctly controls the accesses of components or systems run from it. For example, the operating system might be expected to limit users' accesses to certain files.

To trust any program, we base our trust on rigorous analysis and testing, looking for certain key characteristics:

- *Functional correctness.* The program does what it is supposed to, and it works correctly.
- *Enforcement of integrity.* Even if presented erroneous commands or commands from unauthorized users, the program maintains the correctness of the data with which it has contact.
- *Limited privilege:* The program is allowed to access secure data, but the access is minimized and neither the access rights nor the data are passed along to other untrusted programs or back to an untrusted caller.
- *Appropriate confidence level.* The program has been examined and rated at a degree of trust appropriate for the kind of data and environment in which it is to be used.

Security professionals prefer to speak of *trusted* instead of *secure* operating systems. A trusted system connotes one that meets the intended security requirements, is of high enough quality, and justifies the user's confidence in that

quality. That is, trust is perceived by the system's receiver or user, not by its developer, designer, or manufacturer.

Trusted process: a process that can affect system security, or a process whose incorrect or malicious execution is capable of violating system security policy),

Trusted product: an evaluated and approved product.

Trusted software; the software portion of a system that can be relied upon to enforce security policy.

Trusted computing base (the set of all protection mechanisms within a computing system, including hardware, firmware, and software, that together enforce a unified security policy over a product or system), or **trusted system** (a system that employs sufficient hardware and software integrity measures to allow its use for processing sensitive information). Common to these definitions are the concepts of

- Enforcement of security policy
- Sufficiency of measures and mechanisms
- Evaluation

In studying trusted operating systems, we examine closely what makes them trustworthy.

2- Security Policies

To know that an operating system maintains the security we expect, we must be able to state its security policy.

A **security policy** is a statement of the security we expect the system to enforce. An operating system (or any other piece of a trusted system) can be trusted only in relation to its security policy; that is, to the security needs the system is expected to satisfy.

We begin our study of security policy by examining military security policy because it has been the basis of much trusted operating system development and is fairly easy to state precisely. Then, we move to security policies that commercial establishments might adopt. As shown in figure 1, we assume these five sensitivity levels.

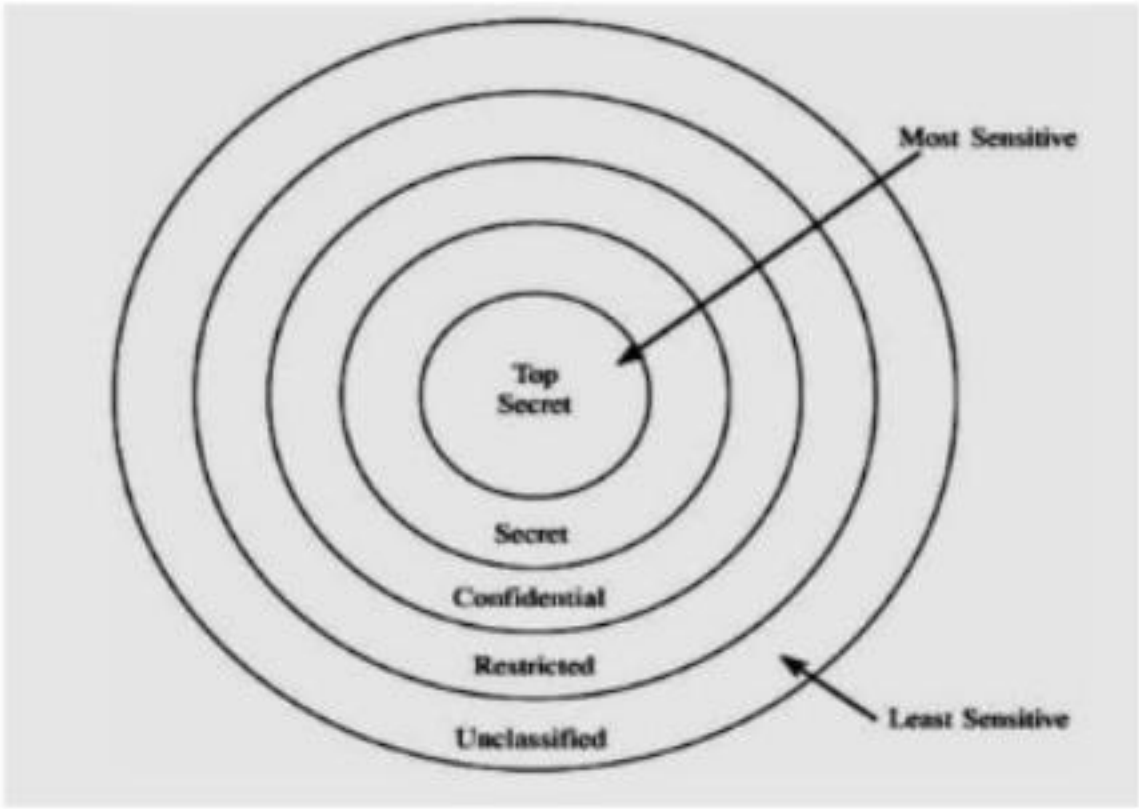

Figure 1: Hierarchy of Sensitivities

3- Trusted Operating System Design

Operating systems by themselves (regardless of their security constraints) are very difficult to design. They handle many duties, are subject to interruptions and context switches, and must minimize overhead so as not to slow user computations and interactions. Adding the responsibility for security enforcement to the operating system substantially increases the difficulty of designing an operating system.

3-1 Trusted System Design Elements

Good design principles are always good for security, as we have noted above. But several important design principles are quite particular to security and essential for building a solid, trusted operating system. The design principles are:

- *Least privilege.* Each user and each program should operate by using the fewest privileges possible. In this way, the damage from an inadvertent or malicious attack is minimized.
- *Economy of mechanism.* The design of the protection system should be small, simple, and straightforward. Such a protection system can be carefully analyzed, exhaustively tested, perhaps verified, and relied on.
- *Open design.* The protection mechanism must not depend on the ignorance of potential attackers; the mechanism should be public, depending on secrecy of relatively few key items, such as a password table. An open design is also available for extensive public scrutiny, thereby providing independent confirmation of the design security.

- *Complete mediation.* Every access attempt must be checked. Both direct access attempts (requests) and attempts to circumvent the access checking mechanism should be considered, and the mechanism should be positioned so that it cannot be circumvented.
- *Permission based.* The default condition should be denial of access. A conservative designer identifies the items that *should* be accessible, rather than those that should *not.*
- *Separation of privilege.* Ideally, access to objects should depend on more than one condition, such as user authentication plus a cryptographic key. In this way, someone who defeats one protection system will not have complete access.
- *Least common mechanism.* Shared objects provide potential channels for information flow. Systems employing physical or logical separation reduce the risk from sharing.
- *Ease of use.* If a protection mechanism is easy to use, it is unlikely to be avoided.

3-2 Security Features of Ordinary Operating Systems

An ordinary multiprogramming operating system performs several functions that relate to security. To see how, **Figure 2**, which illustrates how an operating system interacts with users, provides services, and allocates resources.

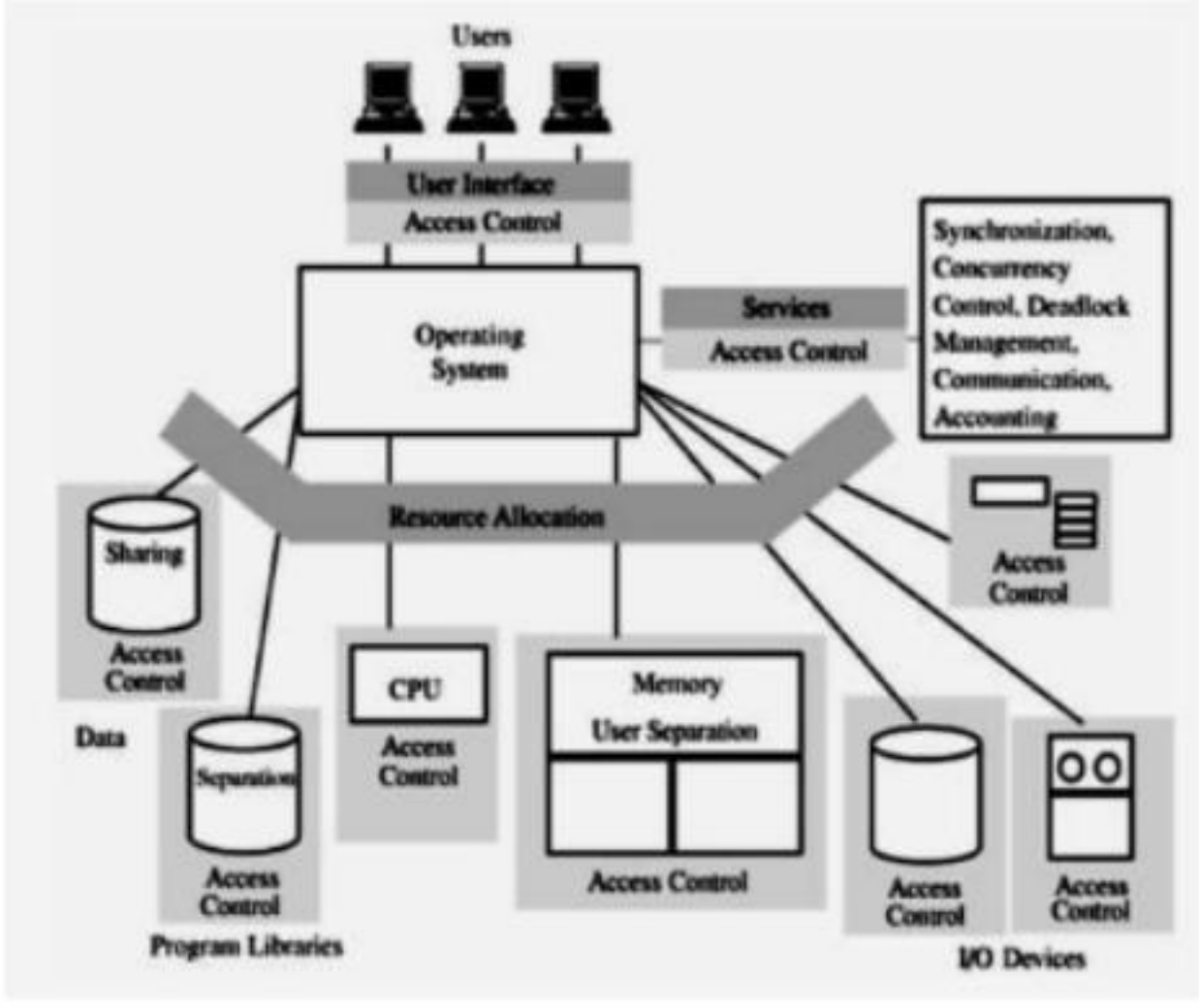

Figure 2: Overview of an Operating System's Functions

A more detail the key features of a trusted operating system, including:

- User identification and authentication
- Mandatory (by force of law) access control
- Discretionary (decided by people authority) access control
- Object reuse protection
- Complete mediation
- Trusted path
- Audit
- Audit log reduction
- Intrusion detection

4-Kernelized Design

A **kernel** is the part of an operating system that performs the lowest-level functions. In standard operating system design, the kernel implements operations such as synchronization, interprocess communication, message passing, and interrupt handling. The kernel is also called a **nucleus** or **core**.

A **security kernel** is responsible for enforcing the security mechanisms of the entire operating system. The security kernel provides the security interfaces among the

hardware, operating system, and other parts of the computing system. Typically, the operating system is designed so that the security kernel is contained within the operating system kernel. There are several good design reasons for why security functions may be isolated in a security kernel:

- *Coverage.* Every access to a protected object must pass through the security kernel. In a system designed in this way, the operating system can use the security kernel to ensure that every access is checked.
- *Separation.* Isolating security mechanisms both from the rest of the operating system and from the user space makes it easier to protect those mechanisms from penetration by the operating system or the users.
- *Unity.* All security functions are performed by a single set of code, so it is easier to trace the cause of any problems that arise with these functions.
- *Modifiability.* Changes to the security mechanisms are easier to make and easier to test.

- *Compactness.* Because it performs only security functions, the security kernel is likely to be relatively small.
- *Verifiability.* Being relatively small, the security kernel can be analyzed rigorously. For example, formal methods can be used to ensure that all security situations (such as states and state changes) have been covered by the design.

4- Reference Monitor

The most important part of a security kernel is the **reference monitor**. A reference monitor is not necessarily a single piece of code; rather, it is the collection of access controls for devices, files, memory, interprocess communication, and other kinds of objects. As shown in **Figure 3**, a reference monitor acts like a brick wall around the operating system or trusted software.

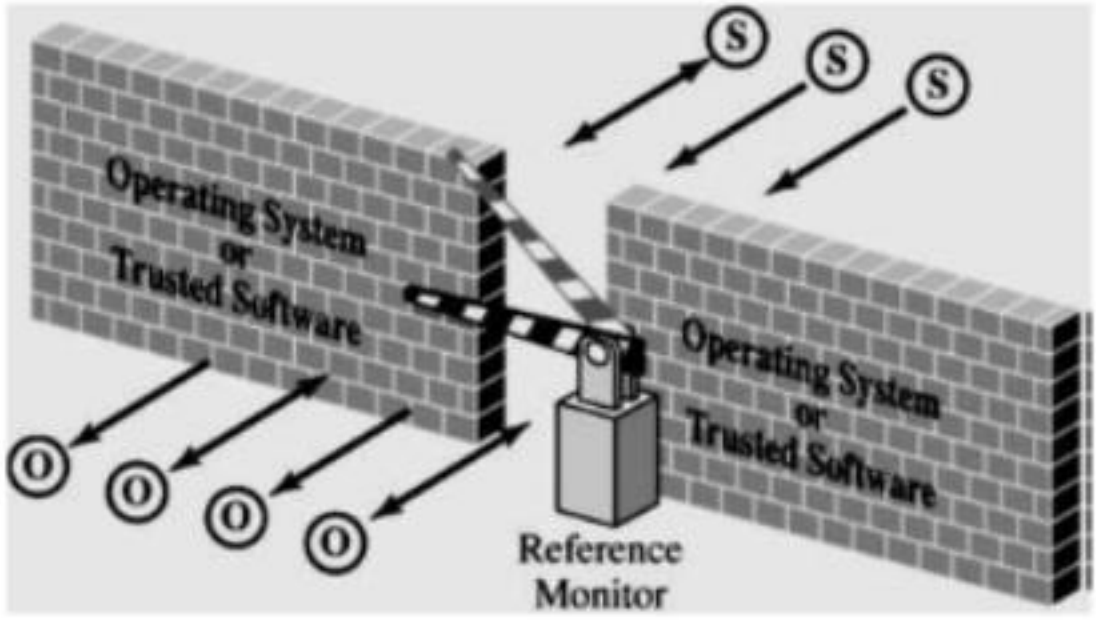

Figure 3. Reference Monitor.

A reference monitor must be

- **Tamperproof**, that is, impossible to weaken or disable
- **Unbypassable**, that is, always invoked when access to any object is required
- **Analyzable**, that is, small enough to be subjected to analysis and testing, the completeness of which can be ensured

A reference monitor can control access effectively only if it cannot be modified or circumvented by a rogue process,

and it is the single point through which all access requests must pass. Furthermore, the reference monitor must function correctly if it is to fulfill its crucial role in enforcing security. Because the likelihood of correct behavior decreases as the complexity and size of a program increase, the best assurance of correct policy enforcement is to build a small, simple, understandable reference monitor.

The reference monitor is not the only security mechanism of a trusted operating system. Other parts of the security suite include audit, identification, and authentication processing, as well as the setting of enforcement parameters, such as who the allowable subjects are and which objects they are allowed to access. These other security parts interact with the reference monitor, receiving data from the reference monitor or providing it with the data it needs to operate.

The reference monitor concept has been used for many trusted operating systems and also for smaller pieces of trusted software. The validity of this concept is well supported both in research and in practice.

5- Separation/Isolation

There are four ways to separate one process from others: physical, temporal, cryptographic, and logical separation. **physical separation:** two different processes use two different hardware facilities. For example, sensitive computation may be performed on a reserved computing system; nonsensitive tasks are run on a public system. Hardware separation offers several attractive features, including support for multiple independent threads of execution, memory protection, mediation of I/O, and at least three different degrees of execution privilege. **Temporal separation:** occurs when different processes are run at different times. For instance, some military systems run nonsensitive jobs between 8:00 a.m. and noon, with sensitive computation only from noon to 5:00 p.m.

cryptographic separation, so two different processes can be run at the same time because unauthorized users cannot access sensitive data in a readable form.

Logical separation, also called **isolation**, is provided when a process such as a reference monitor separates one user's objects from those of another user. Secure

computing systems have been built with each of these forms of separation.

Multiprogramming operating systems should isolate each user from all others, allowing only carefully controlled interactions between the users. Most operating systems are designed to provide a single environment for all. In other words, one copy of the operating system is available for use by many users, as shown in **Figure 4.** The operating system is often separated into two distinct pieces, located at the highest and lowest addresses of memory.

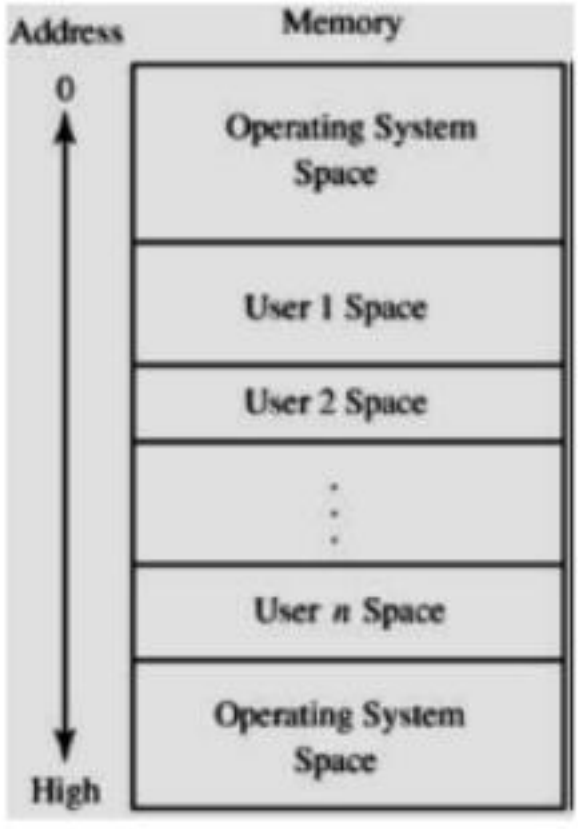

Figure 4. Conventional Multiuser Operating System Memory.

5-4 Virtualization

Virtualization is a powerful tool for trusted system designers because it allows users to access complex objects in a carefully controlled manner. By v**irtualization** we mean that the operating system emulates or simulates a collection of a computer system's resources. We say that a

virtual machine is a collection of real or simulated hardware facilities: a [central] processor that runs an instruction set, an amount of directly addressable storage, and some I/O devices.

These facilities support the execution of programs.

Obviously, virtual resources must be supported by real hardware or software, but the real resources do not have to be the same as the simulated ones. There are many examples of this type of simulation. For instance, printers are often simulated on direct access devices for sharing in multiuser environments. Several small disks can be simulated with one large one. With demand paging, some noncontiguous memory can support a much larger contiguous virtual memory space. And it is common even

on PCs to simulate space on slower disks with faster memory. In these ways, the operating system provides the virtual resource to the user, while the security kernel precisely controls user accesses.

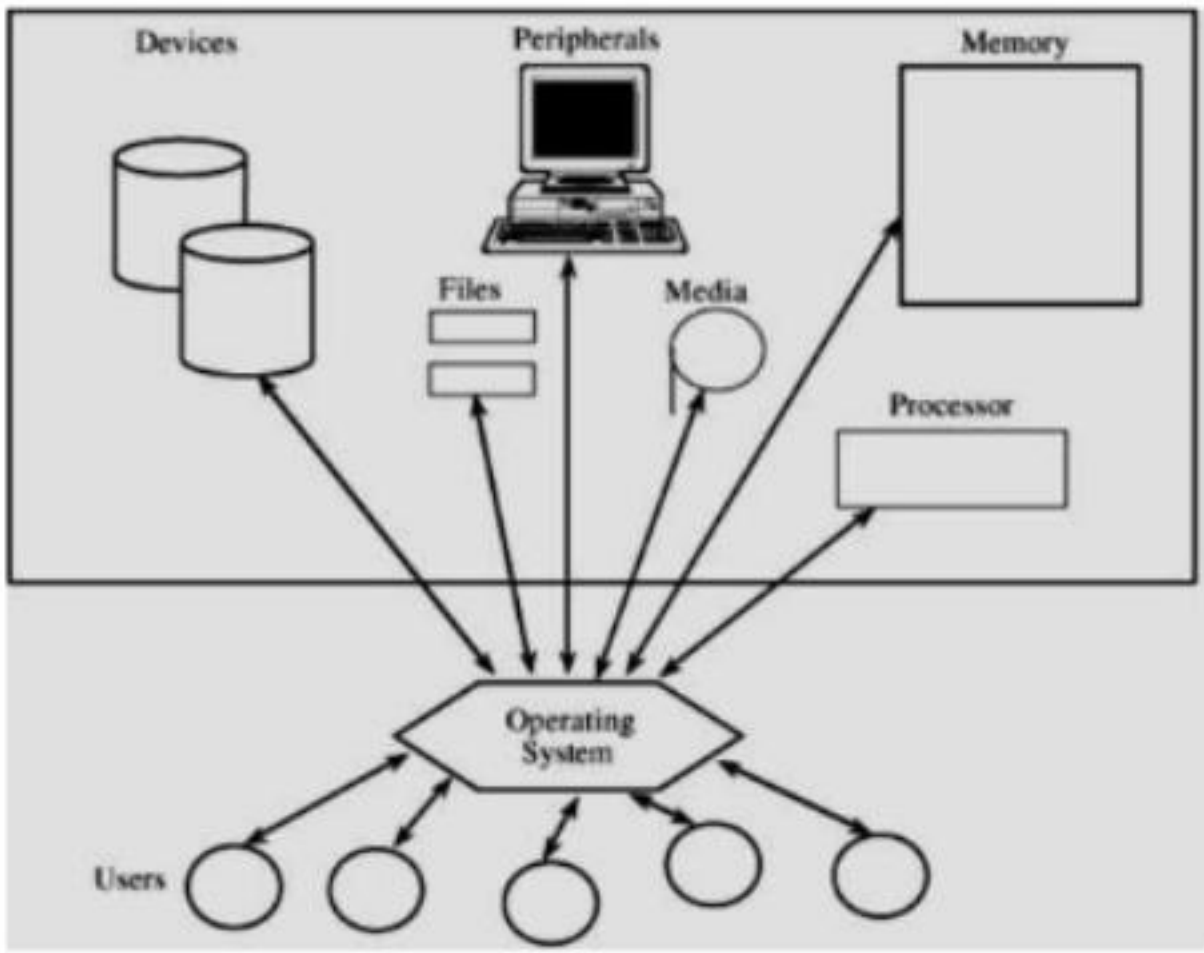

Figure 5: Conventional Operating System.

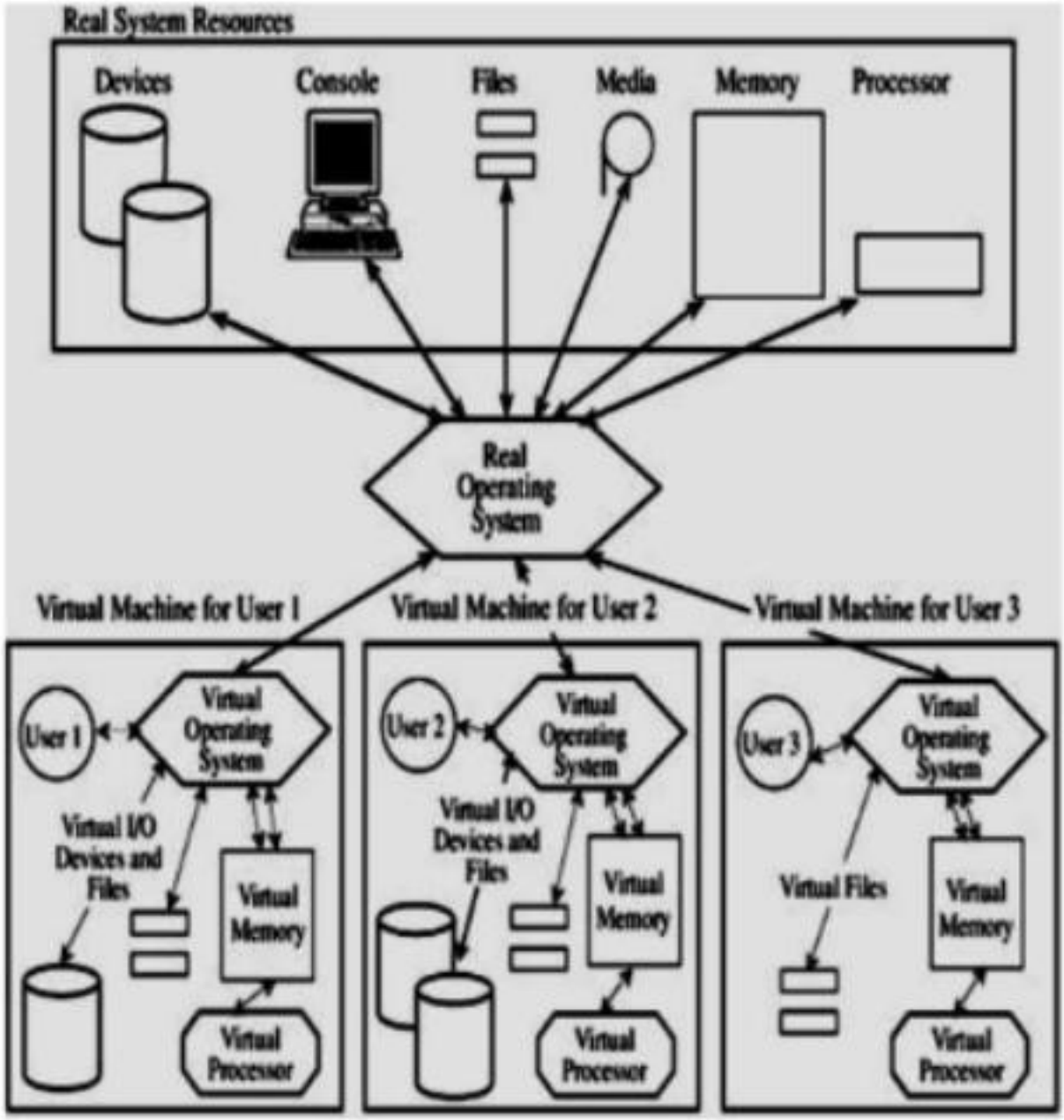

Figure 6Virtual Machine.