

# Information Security (BTCS06001)

By

Prof. Pranita Binnar (Research Scholar, Ph.D. Pursuing)

Visiting Faculty, Dept. of Computer Engineering

NMIMS, Navi Mumbai Campus

Email Contact – [pranitasadgir@gmail.com](mailto:pranitasadgir@gmail.com)

Contact No. - 9699502992

# Myself

- Ex-Scientific Officer, Cyber-Crime Forensic Lab, DFSL, Mumbai, GoM
- CEH, CHFI Certified by EC-council, USA
- PhD. Research Scholar research area Cyber security and Forensic
- CTF Player in IT/OT Domain
- Identifying vulnerability and reporting to respected vendor i.e Device VAPT.
- 02 best scientific research paper award securing 1<sup>st</sup> placed rank

# Topics To Be Covered

- Introduction
- Security in Practices
- Confidentiality
- Integrity
- Availability
- Security violation and threats.

# Learning Objectives

- Describe the key security requirements of confidentiality, integrity and availability
- Discuss the types security threats and attacks that must be dealt with
- Summarize the functional requirements for computer security



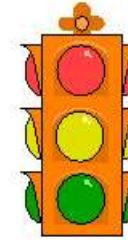
# Security in Practices- From CSI/FBI Report

- 90% detected computer security breaches within the last year
- 80% acknowledged financial losses
- 44% were willing and/or able to quantify their financial losses.
- The most serious financial losses occurred through theft of proprietary information and financial fraud
- 34% reported the intrusions to law enforcement.
- 40% detected external penetration
- 40% detected denial of service attacks.
- 78% detected employee abuse of Internet access privileges
- 85% percent detected computer viruses.
- 38% suffered unauthorized access or misuse on their Web sites.  
[includes insider attacks]
- 12% reported theft of transaction information.
- 6% percent reported financial fraud

# Critical Infrastructure Areas

- Include:

- Telecommunications
- Electrical power systems
- Water supply systems
- Gas and oil pipelines
- Transportation
- Government services
- Emergency services
- Banking and finance
- ...



# Introduction - Information Security

- The protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability. By NIST

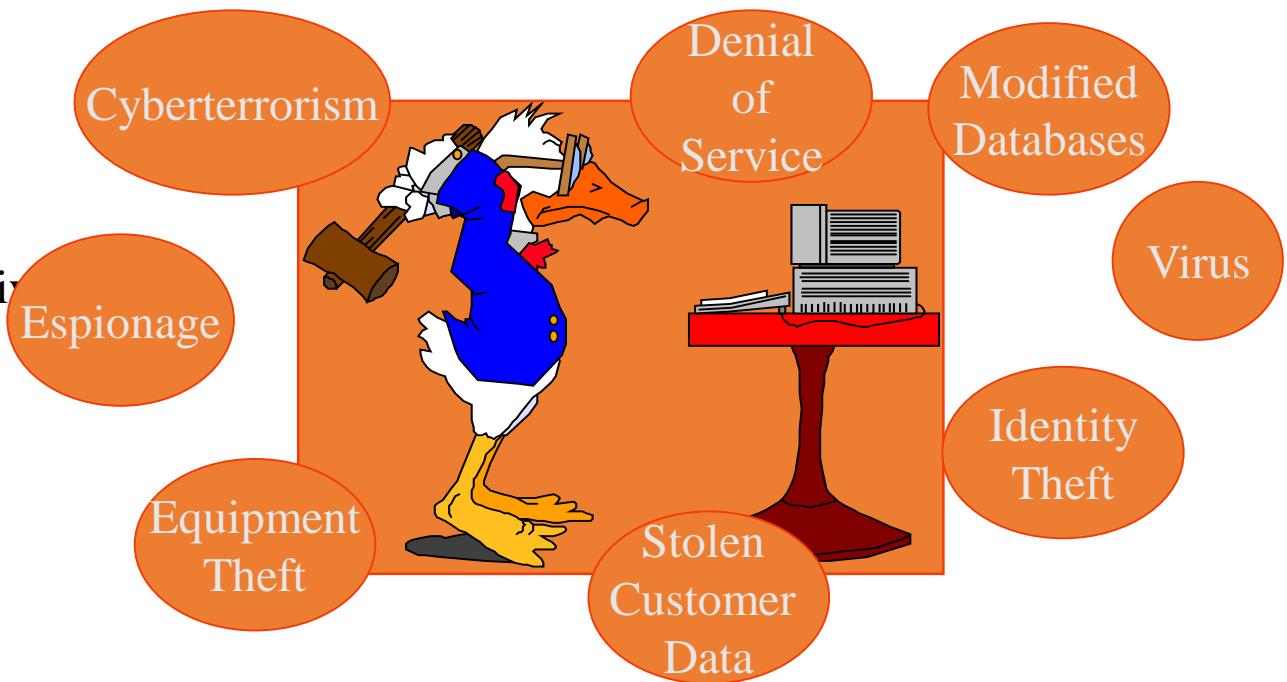
# What is a “Secure” Computer System?

- To decide whether a computer system is “secure”, you must first decide what “secure” *means to you*, then identify the threats you care about.

**You Will Never Own a Perfectly Secure System!**

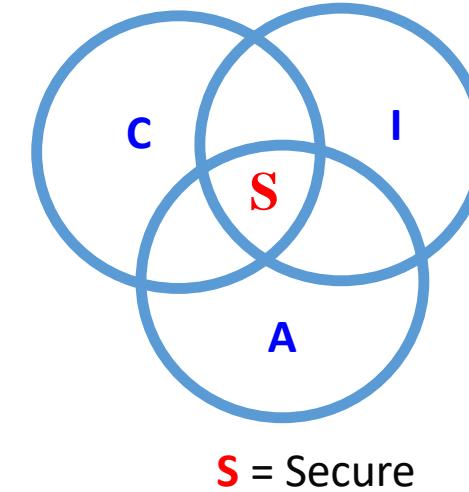
- Threats(in Oval) - examples

- Viruses, trojan horses, etc.
- Denial of Service
- Stolen Customer Data
- Modified Databases
- Identity Theft and other threats to personal privacy
- Equipment Theft
- Espionage/spying in cyberspace
- Hack-tivism
- Cyberterrorism
- ...



# Basic Components of Security: Confidentiality, Integrity, Availability (CIA)

- CIA
  - Confidentiality: Who is authorized to use data?
  - Integrity: Is data „good?”
  - Availability: Can access data whenever need it?



## ■ CIA or CIAAAN... ☺

(other security components added to CIA)

- Authentication
- Authorization
- Non-repudiation
- ...

Ref: Security In Computing - Charles Pfleeger

# Need to Balance CIA

- Example 1: C vs. I+A
  - Disconnect computer from Internet to increase confidentiality
  - Availability suffers, integrity suffers due to lost updates
- Example 2: I vs. C+A
  - Have extensive data checks by different people/systems to increase integrity
  - Confidentiality suffers as more people see data, availability suffers due to locks on data under verification)

# Confidentiality

- “Need to know” basis for data access
  - How do we know who needs what data?  
Approach: **access control** specifies *who* can access *what*
  - How do we know a user is the person she claims to be?  
Need her **identity** and need to **verify** this identity  
Approach: **identification** and **authentication**
- Analogously: “**Need to access/use**” basis for physical assets
  - E.g., access to a computer room, use of a desktop i.e Confidentiality - concerned with *access* to assets
- **Confidentiality**
  - **Data confidentiality:** Assures that confidential information is not disclosed to unauthorized individuals
  - **Privacy:** Assures that individual control or influence what information may be collected and stored

# Risks

- Types Of Risk
  - Legal Risks
    - Fines, liability lawsuits, criminal prosecution
  - Financial Risks
    - Numerous costs involved including losing customer's trust, legal fees, fines
  - Reputational Risks
    - Loss of trust
  - Operational Risks
    - Failed internal processes – insider trading, unethical practices, etc.
  - Strategic Risks
    - Financial institutions future, mergers, etc.

# Threats to Confidentiality

- Access to confidential information by any unauthorized person
- Intercepted data transfers
- Physical(HDD, Pendrive, etc) loss of data
- Privileged access of confidential information by employees
- Social engineered methods to gain confidential information
- Transfer of confidential information to unauthorized third parties
- Compromised machine where attacker is able to access data thought to be secure



# Cont...

- Scheduling information regarding national level speakers/sensitive private meetings highly restricted
- Concerns over unauthorized access as a result of leaks – includes leaks to press as well as opposition/protest groups
- Concerns over “leaks” via IT from opposition groups within the national organization
- Loss of trust in decisions made at event
  - Can include public exposure of sensitive data

## Cont...

- Common theme: leaking private data
- Strict access controls are crucial to protecting the confidential information
- Those who should have access to the confidential information should be clearly defined
  - These people must sign a very clear confidentiality agreement
  - Should understand importance of keeping the information private

# Integrity

- Concerned with unauthorized *modification* of assets (= resources)
- **Integrity**
  - **Data integrity:** assures that information and programs are changed only in a specified and authorized manner
  - **System integrity:** Assures that a system performs its operations in unimpaired manner(Not weakened or damaged)

# Availability

- **Availability:** assure that systems works promptly and service is not denied to authorized users

# Revision

- What is Information Security?
- What are the Key elements of security?
- Explain the key elements ?
- What is the threads ? List some threads

# Vulnerabilities, Threats, and Controls

- Understanding **Vulnerabilities, Threats, and Controls**
  - **Vulnerability** = a weakness in a security system (i.e., in procedures, design, or implementation), that might be exploited to *cause loss or harm*.
  - **Threat** = circumstances that have a *potential* to cause harm  
-a potential violation of security
  - **Controls** = means and ways to block a threat, which tries to exploit one or more vulnerabilities.
- How do we address these problems?
  - We use a **control** as a protective measure.
  - That is, a control is an action, device, procedure, or technique that removes or reduces a vulnerability.

# Cont...

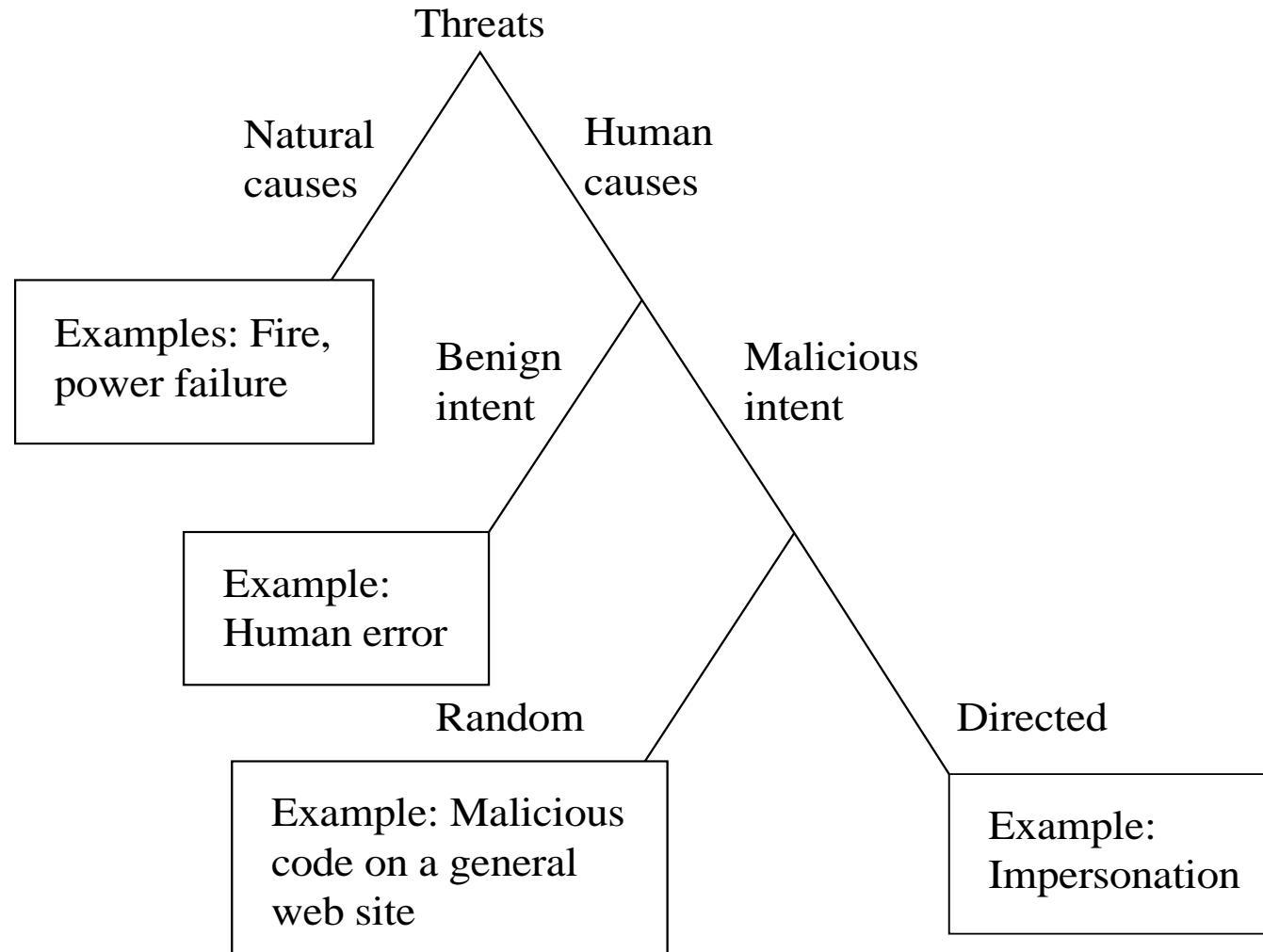
- Relationship among threats, controls, and vulnerabilities:
  - A threat is blocked by control of a vulnerability.
  - To devise controls, we must know as much about threats as possible.

# Visual explanation of basic access control terms



From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

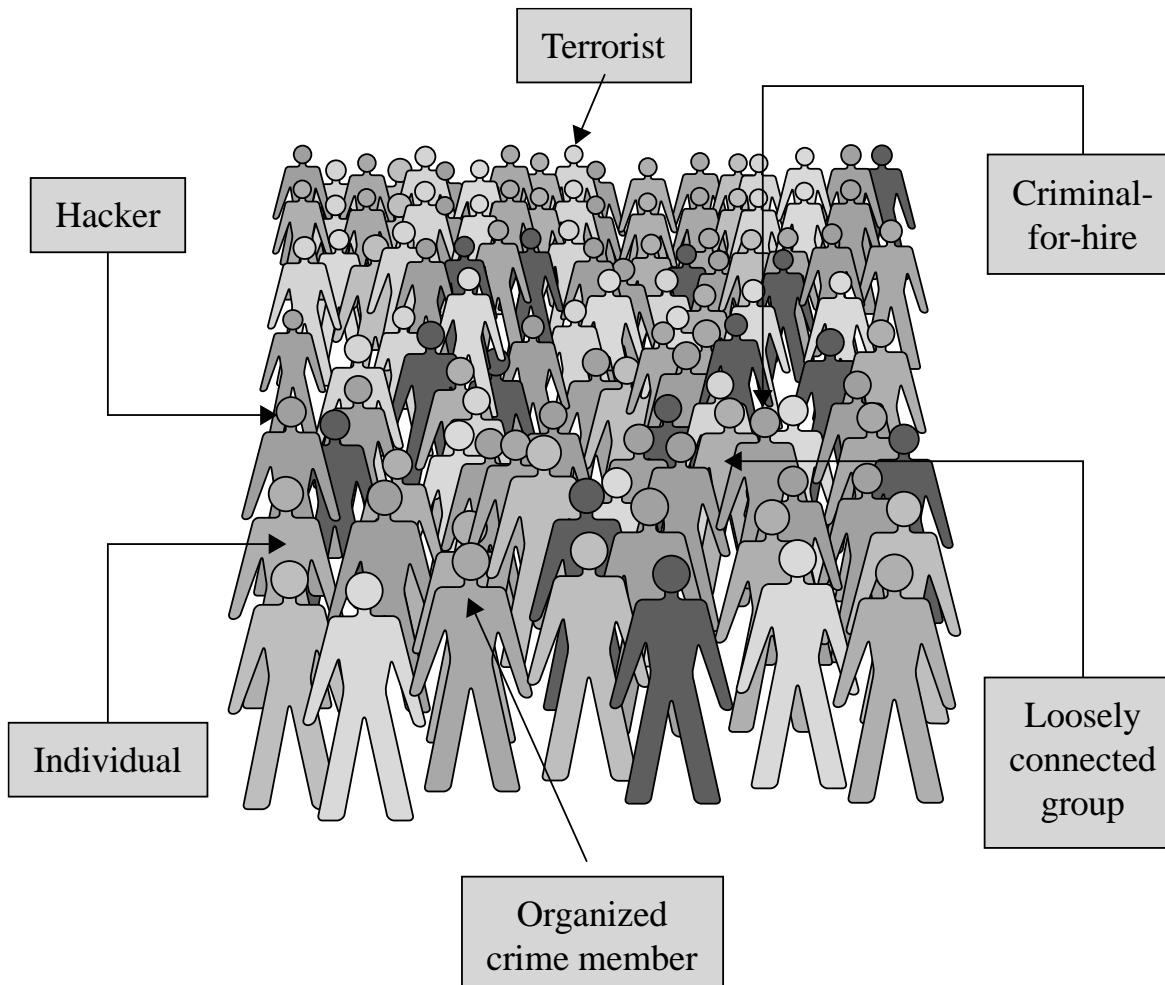
# Types of Threats



# Advanced Persistent Threat (APT)

- APT is a special type of threat that has only been taken seriously by the broad security community over the past decade. In general, security experts believe that no one who becomes a high-priority target can truly be safe from APT.
- Types of APT
  - Organized
  - Directed
  - Well financed
  - Patient
  - Silent

# Types of Attackers



From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

- **Attack**
  - = exploitation of one or more vulnerabilities by a threat; tries to defeat controls
    - Attack may be:
      - *Successful*
        - resulting in a breach of security, a system penetration, etc.
      - *Unsuccessful*
        - when controls block a threat trying to exploit a vulnerability

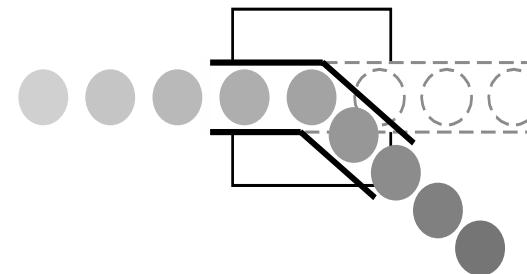
[Pfleeger & Pfleeger]

# Kinds of Threats

- Kinds of threats:

- **Interception**

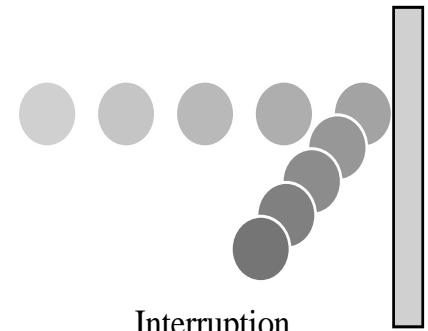
- an unauthorized party (human or not) gains access to an asset



Interception

- **Interruption**

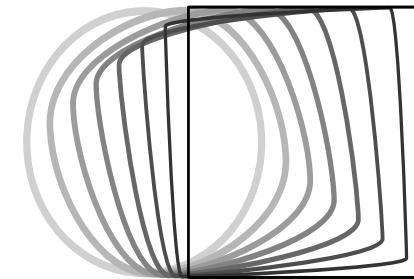
- an asset becomes lost, unavailable, or unusable



Interruption

- **Modification**

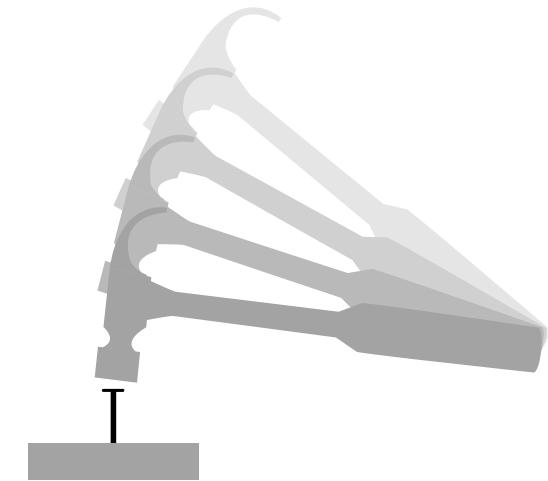
- an unauthorized party changes the state of an asset



Modification

- **Fabrication**

- an unauthorized party counterfeits(imitate something authentic, with the intent to steal, destroy, or replace the original,) an asset



Fabrication

[Pfleeger & Pfleeger]

# Levels of Vulnerabilities / Threats

- D) for other assets (resources)
  - including. people using data, s/w, h/w
- C) for data
- B) for software
- A) for hardware

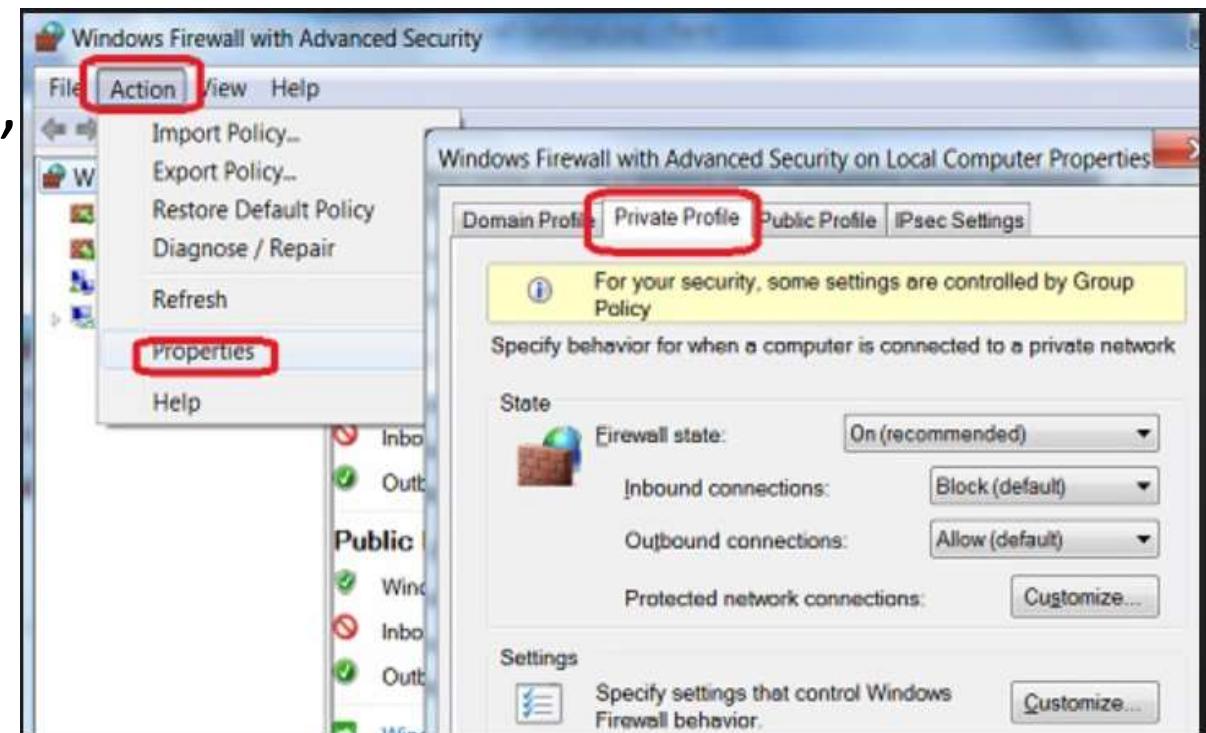
[Pfleeger & Pfleeger]

# Topics to be covered

- Security policy and procedure
- Assumptions and Trust
- Security Assurance
- Implementation and Operational Issues
- Security Life Cycle

# Security policy and Procedures

- A *security policy* is a statement of what is, and what is not, allowed.
- A *security procedure* is a method, tool, or mechanism for enforcing a security policy.
- Mechanisms can be nontechnical, such as requiring proof of identity before changing a password; in fact, policies often require some procedural mechanisms that technology cannot enforce.



# Security policy and Procedures

- Policies may be presented mathematically, as a list of **allowed (secure)** and **disallowed (non secure)** states.
- In practice, policies are rarely so precise; they normally describe in English what users and staff are “allowed” to do or “disallowed” to do.
- security policy’s specification of “secure” and “nonsecure” actions, these security mechanisms can prevent the attack, detect the attack, or recover from the attack.

# Security policy and Procedures

- Policy correctly divides world into secure and insecure states.
- Mechanisms prevent transition from secure to insecure states.
- The strategies may be used together or separately.
- *Prevention* means that an attack will fail. For example, if one attempts to break into a host over the Internet and that host is not connected to the Internet, the attack has been prevented.

# Example

- Suppose a university's computer science laboratory has a policy that prohibits any student from copying another student's homework files.
- The computer system provides mechanisms for preventing others from reading a user's files.
- Anna fails to use these mechanisms to protect her homework files, and Bill copies them.
- A breach of security has occurred, because Bill has violated the security policy. Anna's failure to protect her files does not authorize Bill to copy them.
- In this example, Anna could easily have protected her files. In other environments, such protection may not be easy. For example, the Internet provides only the most rudimentary security mechanisms, which are not adequate to protect information sent over that network.

# Assumptions and Trust

- How do we determine if the policy correctly describes the required level and type of security for the site?
- This question lies at the heart of all security, computer and otherwise.
- Assumption Example
- Opening a door lock requires a key. The assumption is that the lock is secure against lock picking. This assumption is treated as an axiom and is made because most people would require a key to open a door lock. A good lock picker, however, can open a lock without a key. Hence, in an environment with a skilled, untrustworthy lock picker, the assumption is wrong and the consequence invalid.

# Assumptions and Trust

- Trust Example
- If the lock picker is trustworthy, the assumption is valid. The term “trustworthy” implies that the lock picker will not pick a lock unless the owner of the lock authorizes the lock picking. This is another example of the role of trust.

# Assurance

- Evidence of how much to trust a system
- Evidence can include
  - System specifications
  - Design
  - Implementation

# Assurance

EXAMPLE: In the United States, aspirin from a nationally known and reputable manufacturer, delivered to the drugstore in a safety-sealed container, and sold with the seal still in place, is considered trustworthy by most people. The bases for that trust are as follows.

1. The testing and certification of the drug (aspirin) by the Food and Drug Administration(FDA).
2. The manufacturing standards of the company and the precautions it takes to ensure that the drug is not contaminated.
3. The safety seal on the bottle.
  - The three technologies (certification, manufacturing standards, and preventative sealing) provide some degree of assurance that the aspirin is not contaminated.
  - The degree of trust the purchaser has in the purity of the aspirin is a result of these three processes.

# Assurance

- Assurance in the computer world is similar. It requires specific steps to ensure that the computer will function properly.
- The sequence of steps includes detailed **specifications** of the desired (or undesirable) behavior; an analysis of the **design** of the hardware, software, and other components to show that the system will not violate the specifications; and arguments or proofs that the **implementation**, operating procedures, and maintenance procedures will produce the desired behavior.

# Specification

- A *specification* is a (formal or informal) statement of the desired functioning of the system.
- The specification can be low-level, combining program code with logical and temporal relationships to specify ordering of events.
- EXAMPLE: A company is purchasing a new computer for internal use. They need to trust the system to be invulnerable to attack over the Internet. One of their (English) specifications would read “The system cannot be attacked over the Internet.”
- Specifications are used not merely in security but also in systems designed for safety.

# Design

- The *design* of a system translates the specifications into components that will implement them.
- The design is said to *satisfy* the specifications if, under all relevant circumstances, the design will not permit the system to violate those specifications.
- EXAMPLE: A design of the computer system for the company mentioned above had no network interface cards, no modem cards, and no network drivers in the kernel. This design satisfied the specification because the system would not connect to the Internet. Hence it could not be attacked over the Internet.

# Implementation

- Given a design, the *implementation* creates a system that satisfies that design. If the design also satisfies the specifications, then by transitivity the implementation will also satisfy the specifications.
- The difficulty at this step is the complexity of proving that a program correctly implements the design and, in turn, the specifications.
- A program is *correct* if its implementation performs as specified.

# Security Assurance

- Furthermore, **testing** relies on test procedures and documentation, errors in either of which could invalidate the testing results.
- Although assurance techniques do not guarantee correctness or security, they provide a firm basis for assessing what one must trust in order to believe that a system is secure.

# Security Assurance

- Trust cannot be quantified precisely.
- System specification, design, and implementation can provide a basis for determining “how much” to trust a system. This aspect of trust is called *assurance*.
- It is an attempt to provide a basis for bolstering (or substantiating or specifying) how much one can trust a system.

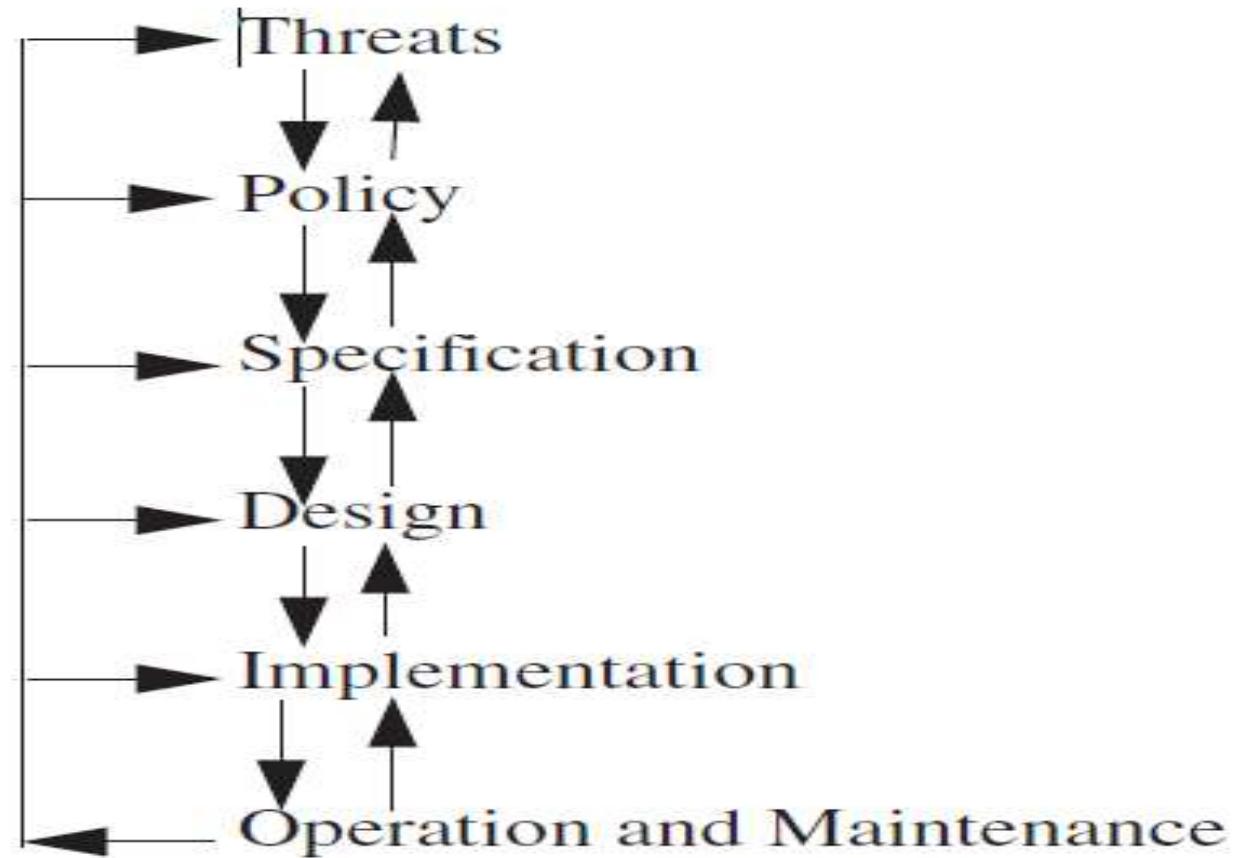
# Operational Issues

- Any useful policy and mechanism must balance the benefits of the protection against the cost of designing, implementing, and using the mechanism. This balance can be determined by analyzing the risks of a security breach and the likelihood of it occurring.
- Such an analysis is, to a degree, subjective, because in very few situations can risks be rigorously quantified.

# Cost-Benefit Analysis

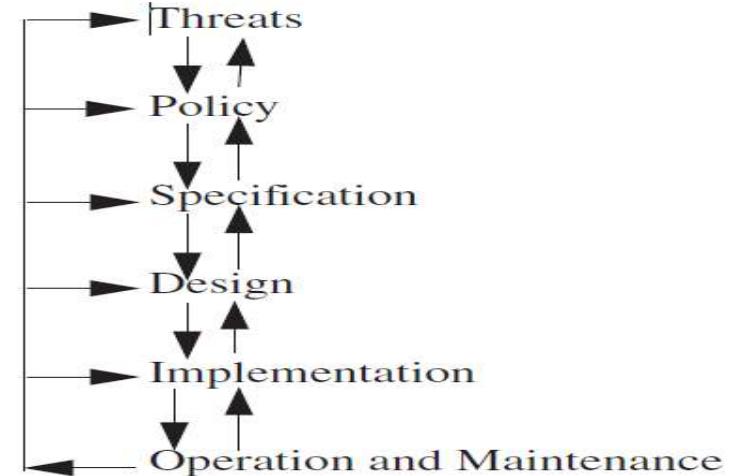
- If the data or resources cost less, or are of less value, than their protection, adding security mechanisms and procedures is not cost-effective because the data or resources can be reconstructed more cheaply than the protections themselves. Unfortunately, this is rarely the case.
- EXAMPLE: A database provides salary information to a second system that prints checks. If the data in the database is altered, the company could suffer grievous financial loss; hence, even a cursory cost-benefit analysis would show that the strongest possible integrity mechanisms should protect the data in the database.

# The security life cycle



# Cont...

- The operation and maintenance stage is critical to the life cycle.
- EXAMPLE: A major corporation decided to improve its security. It hired consultants, determined **the threats**, and created a policy.
- From **the policy**, the consultants derived several specifications that the security mechanisms had to meet.
- They then developed a design that would meet **the specifications**.
- During **the implementation** phase, the company discovered that employees could connect modems to the telephones without being detected.
- **The design** required all incoming connections to go through a firewall. The design had to be modified to divide systems into two classes: systems connected to “the outside,” which were put outside the firewall; and all other systems, which were put behind the firewall. The design needed other modifications as well.
- When the system was deployed, **the operation and maintenance** phase revealed several unexpected threats.



# Thank you

# Unit 2 Authentication

By

Prof. Pranita Binnar (Research Scholar, Ph.D. Pursuing)

Visiting Faculty, Dept. of Computer Engineering

NMIMS, Navi Mumbai Campus

Email Contact – [pranitasadgir@gmail.com](mailto:pranitasadgir@gmail.com)

Contact No. - 9699502992

# Topics to be covered

- Authentication basics
- Password
- Challenge response
- Biometrics
- SSO

# Authentication basics

- **Definition - *Authentication* is the binding of an identity to subject.**
  - Subjects act on behalf of **some other, external entity**.
  - The identity of that entity **controls the actions** that its associated subjects may take.
  - Hence, the subjects must **bind** to the identity of that external entity.

# Authentication basics

- The external entity must provide information **to enable the system to confirm its identity.** This information comes from one (or more) of the following.
  1. What the entity knows (such as passwords or secret information such as PIN, Social security number, Mother's maiden name, Date of birth, Name of your pet, etc.)
  2. What the entity has (such as a badge or card or tokens)
  3. What the entity is (such as fingerprints or retinal characteristics)
  4. Where the entity is (such as in front of a particular terminal)

# Authentication Process

- The authentication process consists of **obtaining the authentication information from an entity, analyzing the data, and determining** if it is associated with that entity.
- This means that the computer must store some information about the entity. It also suggests that mechanisms for managing the data are required. We represent these requirements in an *authentication system* consisting of five components.

# Authentication System

1. The set  $A$  of *authentication information* is the set of specific information with which entities prove their identities.
  2. The set  $C$  of *complementary information* is the set of information that the system stores and uses to validate the authentication information.
  3. The set  $F$  of *complementation functions* that generate the complementary information from the authentication information. That is, for  $f \in F$ ,  $f: A \rightarrow C$ .
  4. The set  $L$  of *authentication functions* that verify identity. That is, for  $I \in L$ ,  
 $I: A \times C \rightarrow \{\text{true, false}\}$ .
  5. The set  $S$  of *selection functions* that enable an entity to create or alter the authentication and complementary information.
- 
- The goal of an authentication system is to ensure that entities are correctly identified. If one entity can guess another's password, then the guesser can impersonate the other.

# Example

- A user authenticates himself by entering a password, which the system compares with the cleartext passwords stored online. Here,  $A$  is the set of strings making up acceptable passwords,

$$C = A, F = \{ I \}, \text{ and } L = \{ \mathbf{eq} \}$$

where  $I$  is the identity function and **eq** is **true** if its arguments are the same and **false** if they are not.

# Passwords

Definition : A *password* is **information associated with an entity that confirms the entity's identity.**

- Passwords are an example of an authentication mechanism based on what people know: the user supplies a password, and the computer validates it.
- If the password is the one associated with the user, that user's identity is authenticated.
- If not, the password is rejected and the authentication fails.

# Password

- The simplest password is some sequence of characters. In this case, the *password space* is the set of all sequences of characters that can be passwords.
- EXAMPLE: One installation requires each user to choose a sequence of 10 digits as a password. Then A has  $10^{10}$  elements (from “0000000000” to “9999999999”).

# Why Passwords?

- Why is “something you know” more popular than “something you have” and “something you are”?
- **Cost**: passwords are free
- **Convenience**: easier for sysadmin to reset pwd than to issue a new thumb

# Good and Bad Passwords

- Bad passwords
  - frank
  - Fido
  - Password
  - incorrect
  - Pikachu
  - 102560
  - AustinStamp
- Good Passwords?
  - jflej,43j-EmmL+y
  - 09864376537263
  - P0kem0N
  - FSa7Yago
  - OnceuP0nAt1m8
  - PokeGCTall150

# Attacks on Passwords

- Attacker could...
  - Target one particular account
  - Target any account on system
  - Target any account on any system
  - Attempt denial of service (DoS) attack
- Common attack path
  - Outsider → normal user → administrator
  - May only require **one** weak password!

# Password Attacks

- Phishing. Phishing is when a hacker posing as a trustworthy party sends you a fraudulent email, hoping you will reveal your personal information voluntarily. ...
- Man-in-the-middle attack. Man-in-the middle (MitM) attacks are when a hacker or compromised system sits in between two uncompromised people or systems and deciphers the information they're passing to each other, including passwords.
- Brute force attack. a brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found.

# Password Attacks

- Dictionary attack. A dictionary attack is a systematic method of guessing a password by trying many common words and their simple variations
- Credential stuffing. Credential stuffing is a type of cyberattack in which stolen account credentials, typically consisting of lists of usernames and/or email addresses and the corresponding passwords (often from a data breach), are used to gain unauthorized access to user accounts through large-scale automated login requests directed against a web application
- Keyloggers. a computer program that records every keystroke made by a computer user, especially in order to gain fraudulent access to passwords and other confidential information.

# Attacking a Password System

1. A dictionary attack is the guessing of a password
2. Counteracting Password Guessing
3. Random Selection of Passwords
4. Pronounceable and Other Computer-Generated Passwords
5. User Selection of Passwords
6. Reusable Passwords and Dictionary Attacks
7. Password Aging

# Attacking a Password System

- The simplest attack against a password-based system is to guess passwords.
- **Definition :** A *dictionary attack* is the **guessing of a password by repeated trial and error.**
- The name of this attack comes from the list of words (a “dictionary”) used for guesses.
- The dictionary may be a set of strings in random order or (more usually) a set of strings in decreasing order of probability of selection.

# Countering Password Guessing

- Password guessing requires either the set of complementation functions (F) and complementary information (C) or access to the authentication functions (L).
- In both approaches, the goal of the defenders is to maximize the time needed to guess the password.
- Let  $P$  be the probability that an attacker guesses a password in a specified period of time T. Let  $G$  be the number of guesses that can be tested in one time unit. Let  $T$  be the number of time units during which guessing occurs. Let  $N$  be the number of possible passwords. Then,

$$P \geq \frac{TG}{N}$$

# Random Selection of Passwords

- **Random Selection of Passwords**
- **Pronounceable and Other Computer-Generated Passwords**
- Pronounceable passwords are based on **the unit of sound** called a *phoneme*. In English, phonemes for constructing passwords are represented by the character sequences *cv*, *vc*, *cvc*, or *vcv*, where *v* is a vowel and *c* a consonant.
  - EXAMPLE: The passwords “helgoret” and “juttelon” are pronounceable passwords; “przbqxdf” and “zxrptglfn” are not.

# User Selection of Passwords

- Rather than selecting passwords for users, one can constrain what passwords users are allowed to select. This technique, *called proactive password selection*, enables **users to propose passwords they can remember, but rejects any that are deemed too easy to guess.**
1. Passwords based on account names
    - a. Account name followed by a number
    - b. Account name surrounded by delimiters
  2. Passwords based on user names
    - a. Initials repeated 0 or more times
    - b. All letters lower- or uppercase
    - c. Name reversed
    - d. First initial followed by last name reversed
  3. Passwords based on computer names
  4. Dictionary words
  5. Reversed dictionary words
  6. Dictionary words with some or all letters capitalized

# User Selection of Passwords

- Conjugations or declensions of dictionary words
- 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
- Concatenations of dictionary words
- Dictionary words preceded or followed by digits, punctuation marks, or spaces
- Dictionary words with all vowels deleted
- Dictionary words with white spaces deleted

# Reusable Passwords and Dictionary Attacks

- Reusable passwords are quite susceptible to dictionary attacks. The goal of random passwords, pronounceable passwords, and proactive password checking is to maximize the time needed to guess passwords.
- **Password Aging**
- Guessing of passwords requires that access to the complement, the complementation functions, and the authentication functions be obtained. If none of these have changed by the time the password is guessed, then the attacker can use the password to access the system.

# Password Cracking Tools

- Popular password cracking tools
  - [Password Crackers](#)
  - [Password Portal](#)
  - [LOphtCrack and LC4](#) (Windows)
  - [John the Ripper](#) (Unix)
- Admins should use these tools to test for weak passwords since attackers will
- Good articles on password cracking
  - [Passwords - Conerstone of Computer Security](#)
  - [Passwords revealed by sweet deal](#)

# Challenge-Response

- Passwords have the fundamental problem that they are *reusable*. If an attacker sees a password, she can later *replay* the password.
- The system cannot distinguish between the attacker and the legitimate user, and allows access.
- An alternative is to authenticate in such a way that the transmitted password changes each time. Then, if an attacker replays a previously used password, the system will reject it.

# Definition

1. Let user  $U$  desire to authenticate himself to system  $S$ .
2. Let  $U$  and  $S$  have an agreed-on secret function  $f$ .
3. A *challenge-response* authentication system is one in which  $S$  sends a random message  $m$  (the challenge) to  $U$ , and  $U$  replies with the transformation  $r = f(m)$  (the response).
4.  $S$  validates  $r$  by computing it separately.

# Pass Algorithms

- **Definition:** Let there be a challenge-response authentication system in which the function  $f$  is the secret. Then  $f$  is called a *pass algorithm*.
- Under this definition, no cryptographic keys or other secret information may be input to  $f$ .
- The algorithm computing  $f$  is itself the secret.

# One-Time Passwords

- **Definition :** A *one-time* password is a password that is invalidated as soon as it is used.
- The ultimate form of password aging occurs when a password is valid for exactly one use. In some sense, challenge-response mechanisms use one-time passwords.
- Think of the response as the password. As the challenges for successive authentications differ, the responses differ.
- Hence, the acceptability of each response (password) is invalidated after each use.
- The challenge is the number of the authentication attempt; the response is the one-time password.

# Hardware-Supported Challenge-Response Procedures

- One-time passwords are considerably simpler with hardware support because the passwords need not be printed on paper or some other medium.
- Hardware support comes in two forms: Both perform the same functions.
  1. A program for a general-purpose computer.
  2. Special-purpose hardware support.

# A program for a general-purpose computer.

- The first type of hardware device, informally called a *token*, provides mechanisms for hashing or enciphering information.
- With this type of device, the system sends a challenge.
- The user enters it into the device.
- The device returns the appropriate response.
- Some devices require the user to enter a personal identification number or password, which is used as a cryptographic key or is combined with the challenge to produce the response.

# Special-purpose hardware support.

- The second type of hardware device is temporally based. Every 60 seconds, it displays a different number. The numbers range from 0 to  $10n - 1$ , inclusive.
- A similar device is attached to the computer. It knows what number the device for each registered user should display.
- To authenticate, the user provides his login name. The system requests a password.
- The user then enters the number shown on the hardware device, followed by a fixed (reusable) password.
- The system validates that the number is the one expected for the user at that time and that the reusable portion of the password is correct.

# Identification vs Authentication

- **Identification** — Who goes there?
  - Compare **one-to-many**
  - Example: FBI fingerprint database
- **Authentication** — Are you who you say you are?
  - Compare **one-to-one**
  - Example: Thumbprint mouse
- Identification problem is more difficult
  - More “random” matches since more comparisons
- We are (mostly) interested in authentication

# Biometrics

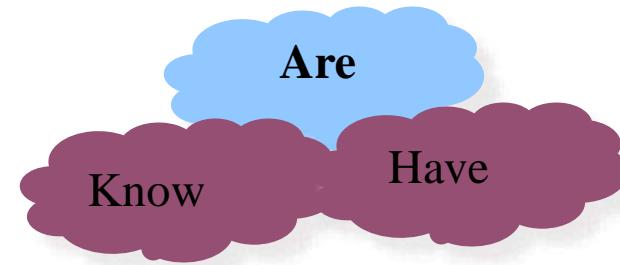


# Something You Are

- Biometric
  - “**You are your key**”—Schneier

## □ Examples

- Fingerprint
- Handwritten signature
- Facial recognition
- Speech recognition
- Gait (walking) recognition
- “Digital doggie” (odor recognition)
- Many more!



# Biometrics Authentication

- May be better than passwords
- But, cheap and reliable biometrics needed
  - Today, an active area of research
- Biometric authentication involves using some part of your physical makeup to authenticate you.
- Biometrics **are** used in security today
  - Thumbprint mouse
  - Palm print for secure entry
  - Fingerprint to unlock car door, etc.

# Biometrics Authentication

- A single characteristic or multiple characteristics could be used.
- It all depends on the infrastructure and the level of security desired.
- With biometric authentication, the physical characteristic being examined is usually mapped to a username. This username is used to make decisions after the person has been authenticated.
- But biometrics not really that popular

# Single Sign on (SSO)

- Authenticating to multiple systems is unpopular with users.
- Left on their own, users will reuse the same password to avoid having to remember many different passwords.
- For example, users become frustrated at having to authenticate to a computer, a network, a mail system, an accounting system, and numerous web sites.
- The panacea for this frustration is called **single sign-on**.
- A user authenticates once per session, and the system forwards that authenticated identity to all other processes that would require authentication.

# Security Assertion Markup Language (SAML)

- SAML is a standard for logging users into applications based on their sessions in another context. This single sign-on (SSO) login standard has significant advantages over logging in using a username/password:
  - No need to type in credentials
  - No need to remember and renew passwords
  - No weak passwords
- Most organizations already know the identity of users because they are logged in to their Active Directory domain or intranet.
- It makes sense to use this information to log users in to other applications, such as web-based applications, and one of the more elegant ways of doing this is by using SAML.

# SSO Protocol

- Kerberos --- example single sign-on protocol
- Kerberos is a computer-network authentication protocol that works on the basis of tickets to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.
- Its designers aimed it primarily at a client–server model, and it provides mutual authentication—both the user and the server verify each other's identity.

Thank you

# Unit 3

# Access Control

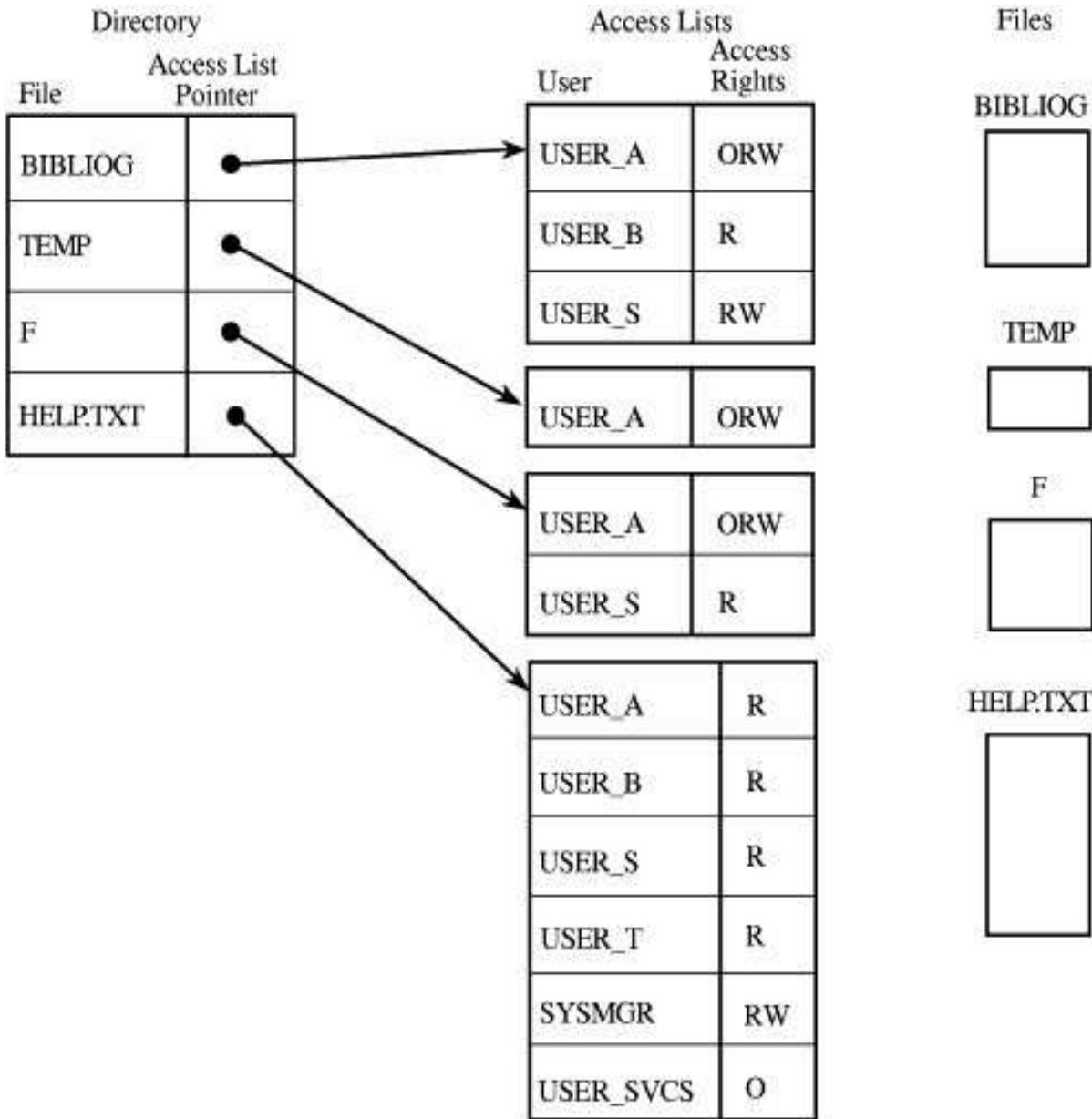
# Access Control

- The mechanism which defines user access is called *access control*.
- When the server receives a request, it uses the authentication information provided by the user and the access control instructions (ACIs) to allow or deny access to directory information.
- The server can allow or deny permissions for actions on entries like read, write, search, and compare.

# Access Control Principles

- Access control is not a stand alone component of a security system
- Access control coexists with other security services
- Access control works closely with audit control
- Access matrix is a good tool to specify permissions
- Access Control List (ACL) details are placed in Access Matrix

# Access Control List



- There is one such list for each object, and the list shows all subjects who should have access to the object and what their access is.

# Discretionary Access Control

- Discretionary Access Control (DAC) is a type of access control in which a user has complete control over all the programs it owns and executes, and also determines the permissions other users have to those files and programs.
- Restricts access to objects based solely on the identity of users who are trying to access them.
- Because DAC requires permissions to be assigned to those who need access, DAC is commonly described as a "need-to-know" access model.

# Discretionary Access Control

- Relies on the object owner to control access.
- Strength of DAC: Flexibility
- Limitations of DAC:
  - **Global policy:** DAC lets users to decide the access control policies on their data, regardless of whether those policies are consistent with the global policies.
  - **Information flow:** information can be copied from one object to another, so access to a copy is possible even if the owner of the original does not provide access to the original copy.
  - **Malicious software:** DAC policies can be easily changed by owner, so a malicious program (e.g., a downloaded untrustworthy program) running by the owner can change DAC policies on behalf of the owner.

# Mandatory Access Control

- Mandatory Access Control (MAC) is a type of access control in which only the administrator manages the access controls.
- The administrator defines the usage and access policy, which cannot be modified or changed by users, and the policy will indicate who has access to which programs and files.
- MAC is most often used in systems where priority is placed on confidentiality.

# Role-based Access Control

- Role-based access control (RBAC) is a type of access control in which access is based on the roles of individual users within an enterprise.
- Roles are defined according to job competency, authority, and responsibility within the enterprise.
- A user has access to an object based on the assigned role.
- The object is concerned with the user's role and not the user.
- Role of the user in the organization determines the access level for the database
- Roles can define specific individuals allowed access or extent of access to resources for multiple individuals

# Role-based Access Control

- RBAC supports the following security principles:
  - Least privilege (only the needed permissions are assigned to roles)
  - Separation of duties (use of mutually exclusive roles – e.g., accountant writes cheque and manager signs the cheque)
  - Data Abstraction (instead of read/write/execute permissions such as credit/debit are established)
- RBAC is independent of MAC and DAC
- RBAC can support MAC and DAC separately

# Access control matrix model

- The *access control matrix model* is the most precise model used to describe a protection state.
- It characterizes the rights of each *subject* (active entity, such as a process) with respect to every other entity.

# Matrix Model

- Matrix model consists of:
  - Objects (data)
  - Subjects (user processes like queries)
  - Rights (permissions for read, etc)
- Rows of the matrix are objects and columns are subjects and the content of each cell is the rights
- Protection domain consists of a collection of access rights

Table 4-1. Access Control Matrix.

	BIBLIOG	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	-	-	R	X	X	R	W
USER S	RW	-	R	R	X	X	R	W
USER T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

# Matrix Model

- Matrix model consists of:

- Access lists

Access list identifies people who have access to a particular object

- Capability lists

Capability list identifies each object and its operations

- A **capability** is an unforgeable token that gives the possessor certain rights to an object.

- The algebra allows policies to be restricted (by posing constraints on their authorizations) and closed with respect to inference rules

# Unit 4

# Program Security

# Topics to be covered

- Secure programs
- Non malicious Program
- Errors
- Viruses and other malicious code
- Types of viruses
- Attack mechanism of viruses
- Targeted Malicious Code
- Controls Against Program Threats.

# Program Security

- It's our first step on how to apply security to computing
- Protecting programs is the heart of computer security
  - All kinds of **programs, from apps via OS, DBMS, networks**
- Issues:
  - How to keep programs free from flaws
  - How to protect computing resources from programs with flaws

# What is Program Security?

- Depends on whom you ask
  - user
  - programmer
  - manager - conformance to all specifications
- Developmental criteria for program security include:
  - Correctness of security & other requirements
  - Correctness of implementation
  - Correctness of testing

# Fault tolerance terminology

- Error - may lead to a fault
  - Fault - cause for deviation from intended function
  - Failure - system malfunction caused by fault
- 
- Faults - seen by "insiders" (e.g., programmers)
  - Failures - seen by "outsiders" (e.g., independent testers, users)

# Fault tolerance terminology

- Error/fault/failure example:
  - Programmer's indexing error, leads to buffer overflow fault
  - Buffer overflow fault causes system crash (a failure)
- Two categories of faults w.r.t. duration
  - Permanent faults
  - Transient faults – can be much more difficult to diagnose

# Secure Programs

- What is a secure program?
- Everyone has their own requirement of “being secure”
- Part of assessing software quality
- Does it meet security requirements in specification?
- In general, we often look at quantity and types of faults for evidence of security.

# Fixing Faults

- Finding lots of faults in software early.
  - NOT GOOD.
- Early approaches were “Dig” and then “Patch”
  - NOT GOOD.
- Repairing with a patch is a narrow focus area.
- Patches can cause other problems.
  - Non obvious side effects
  - Fix one place – fails another
  - Performance or function suffers

# Flaws

- Comparing program requirements with behavior to identify any unexpected behavior is called as **program security flaw**
- Flaw is either a fault or failure
- Vulnerability is a class of flaws (e.g. buffer overflow)
- Need to determine how to prevent harm caused by possible flaws
- Hindrances for eliminating program security flaws
  - How do we test for what a program shouldn't do?
  - Programming and software engineering techniques evolve more rapidly than computer security techniques

# Types of Flaws

- **Intentional**
  - Malicious
  - Nonmalicious
- **Inadvertent**
  - Validation error (incomplete / inconsistent): permission checks
  - Domain error: controlled access to data
  - Serialization and aliasing: program flow order
  - Inadequate identification and authentication: basis for authorization
  - Boundary condition violation: failure on first or last case
  - Other exploitable logic errors

# Non-malicious Program Errors

- These errors cause program malfunctions but do not lead to more serious security vulnerabilities
- We consider three classic error types:
  1. Buffer overflows
  2. Incomplete mediation
  3. Time-to-check to Time-to-Use Errors

# Nonmalicious Program Errors

- **Buffer Overflows**
- A buffer is a space in which data can be held. A buffer's capacity is finite.
- The programmer must declare the buffer's max. size so that the compiler can set aside that amount of space.

# Nonmalicious Program Errors

- Buffer may overflow into (and change):
  - User's own data space
  - User's program area
  - System data space
  - System program area

# Nonmalicious Program Errors

- **Buffer Overflows:**



a) Affects user's data



a) Affects user's code

# Nonmalicious Program Errors

- **Buffer Overflows:**



a) Affects system data



a) Affects system code

# Nonmalicious Program Errors

- Buffer Overflows Security Implication
  - Attacker replaces code in the system space and takes control back from the operating system
  - Attacker uses the stack pointer or return register to execute other codes

# Nonmalicious Program Errors

- Incomplete Mediation (data checking)
- Attackers are exploiting it to cause security problems.
  - Supplying the wrong type of data being requested.
  - Supplying the wrong length of data being requested.
  - Problem
    - System Fails
    - Supply of Bad Data
  - Must be checked by programmer
  - Client side v/s Server Side
    - [http://www.somesite.com/subpage/data&parm1=\(808\)555-1212&parm2=2004Jan01](http://www.somesite.com/subpage/data&parm1=(808)555-1212&parm2=2004Jan01)
    - What if parm2 is 1800Jan01 or 2004Feb30...
    - the user could send incorrect data to the server
- Security Implication
  - Easy to exploit

# Nonmalicious Program Errors

## Time-of-Check to Time-of-Use Errors

- 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.

# Nonmalicious Program Errors

## Time-of-Check to Time-of-Use Errors

- 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.

- This flaw concerns mediation that is performed with a "bait and switch" in the middle. It is also known as a serialization or synchronization flaw.

# Nonmalicious Program Errors

- Bait-and-switch is a form of fraud used in retail sales but also employed in other contexts.
- First, customers are "baited" by merchants' advertising products or services at a low price, but when customers visit the store, they discover that the advertised goods are not available, or the customers are pressured by salespeople to consider similar, but higher-priced items ("switching").
- Security Implication
  - to avoid checking one action and performing another – use digital signatures and certificates

# Time of Check, Time of Use

- ToCToU conditions
  - Can occur during file I/O
  - first checking some object and then using it

# Nonmalicious Program Errors

## Combinations of Nonmalicious Program Flaws

- These three vulnerabilities are bad enough when each is considered on its own.
- But perhaps the worst aspect of all three flaws is that they can be used together as one step in a multistep attack.

# Viruses and Other Malicious Code

- When was the last time you saw a bit?
- Do you know in what form a document file is stored?
- Can you find where a document resides on a disk?
- 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?

*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.*

# Malicious Code

- Malicious code is the general name for **unanticipated or undesired effects in programs** or program parts caused by an agent intent on damage.
- The agent is the writer of the program or a person who causes its distribution.

# Kinds of Malicious Code

- Virus – code that attaches to another program and copies itself to other programs
  1. Transient virus – life depends on life of its host
  2. Resident virus – locates inside memory
- Trojan Horse – malicious effect is hidden from user (Ex: login script)
- Logic viruses – triggered by an event and goes off when specific condition occur
- Time viruses – triggered by a time or date

# Kinds of Malicious Code

- Trapdoor (backdoor) – feature that allows access to program other than through normal channels
- Worm – program that spreads copies of itself through a network, can be stand alone program
- Rabbit – virus/worm that self-replicates without bound

# Kinds of Malicious Code

<b>Code type</b>	<b>Characteristics</b>
Virus	Attaches itself to program and propagate 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(Back door )	Allows unauthorized access to functionality
Worm	Propagates copies of itself through network
Rabbit	Replicates itself without limit to exhaust resources

**Study Assignment**  
Based on the kinds of MALICIOUS  
CODE identify two examples of  
each type and explain in details.

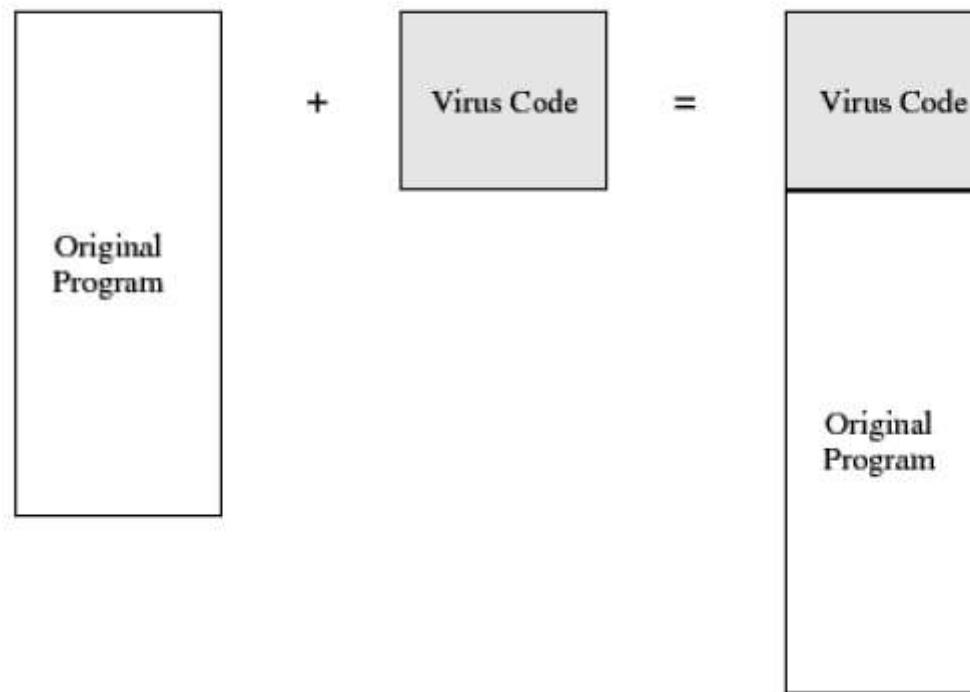
# How do Viruses Attach

- A virus is attached to a “program”
- The virus is activated by executing the program
- Most common viruses today are attached to e-mail; when the attachment is opened, virus is active
- Three ways:
  1. Appended
  2. Surrounds programs
  3. Integrated viruses and replacements

# Appended Viruses

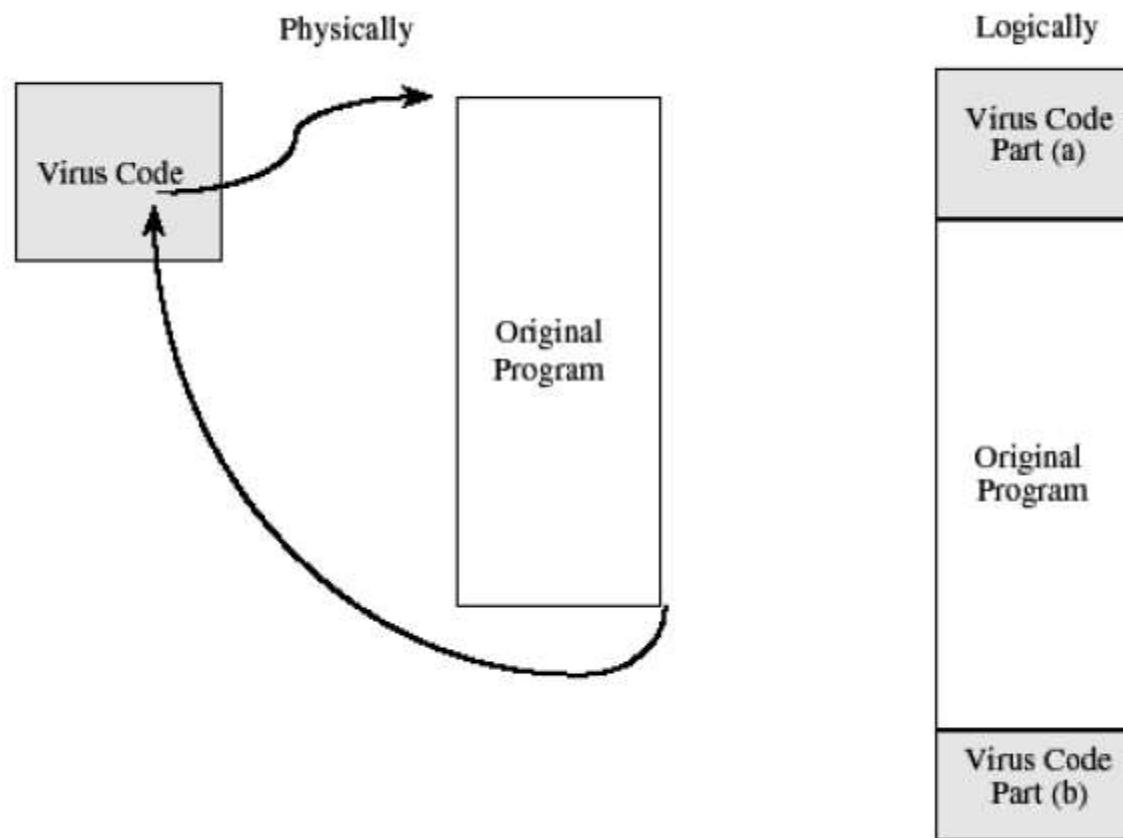
- A program virus attaches itself to a program; then, whenever the program is run, the virus is activated. This kind of attachment is usually easy to program.
- This kind of attachment is simple and usually effective.
- The virus writer does not need to know anything about the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus.

# Appended virus example



# Viruses That Surround a Program

- An alternative to the attachment is a virus that runs the original program **but has control before and after its execution.**



# Integrated Viruses and Replacements

- A third situation occurs when the virus replaces some of its target, integrating itself into the original code of the target.
- The virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.
- 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

Original  
Program

+

Virus  
Code

=

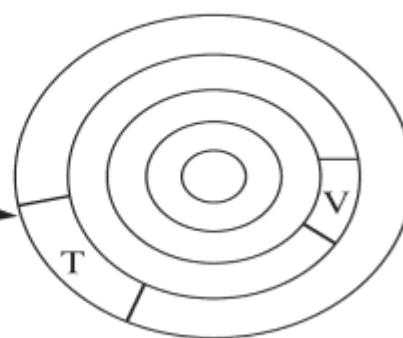
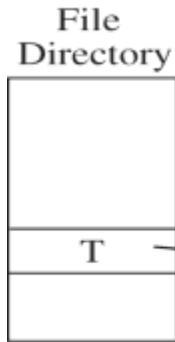
Modified  
Program

# How viruses gain control

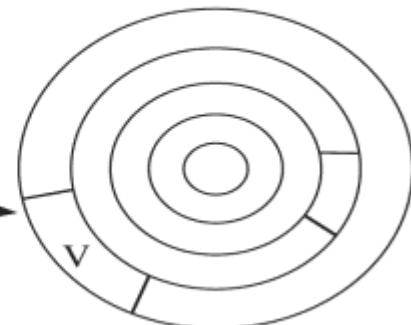
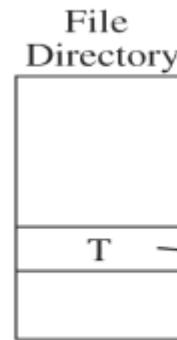
- The virus V has to be invoked instead of the target (T).
- Virus (V) modify/overwrite the program (T) in storage.
- Virus can change file pointer in file table for itself to be located instead of T.

# How viruses gain control

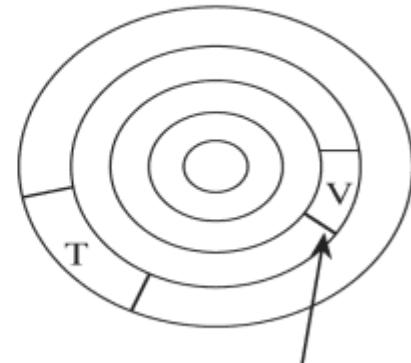
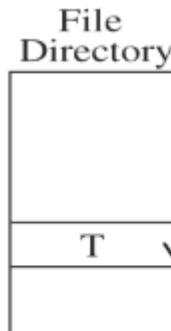
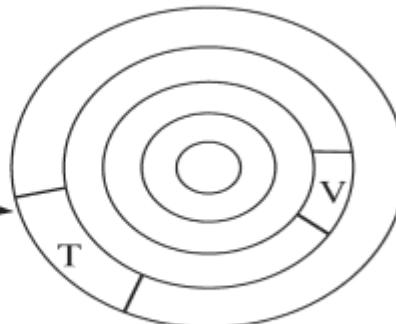
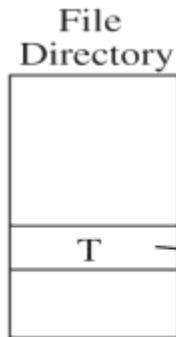
Before



After



(a) Overwriting T



(b) Changing Pointers

# Home for viruses

- 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

# Types of viruses

1. One time execution
2. Boot sector viruses
3. Memory resident viruses
4. Document viruses

# One-Time Execution

- The majority of viruses today execute only once, spreading their infection and causing their effect in that one execution.
- A virus often arrives as an e-mail attachment of a document virus.
- It is executed just by being opened.

# Boot Sector Viruses

- A special case of virus attachment
- 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.
- 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.

# Boot Sector Viruses

- 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.
- The virus writer simply breaks the chain at any point, inserts a pointer to the virus code and reconnects the chain after the virus has been inserted.

# 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 is needed.
- Such **code remains in memory** and is called "**resident**" code.
- Virus writers 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.

# Document Viruses

- Currently, the **most popular** virus type is what we call the document virus, which **is implemented** within a **formatted document**, such as a written document, a database, a slide presentation, or a spreadsheet.
- These documents are **highly structured files** that contain both **data** (words or numbers) and **commands** (such as formulas, formatting controls, links).

# 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/characteristics to spread.
- Each of these characteristics yields a pattern, called a signature, that can be found by a program that knows to look for it.

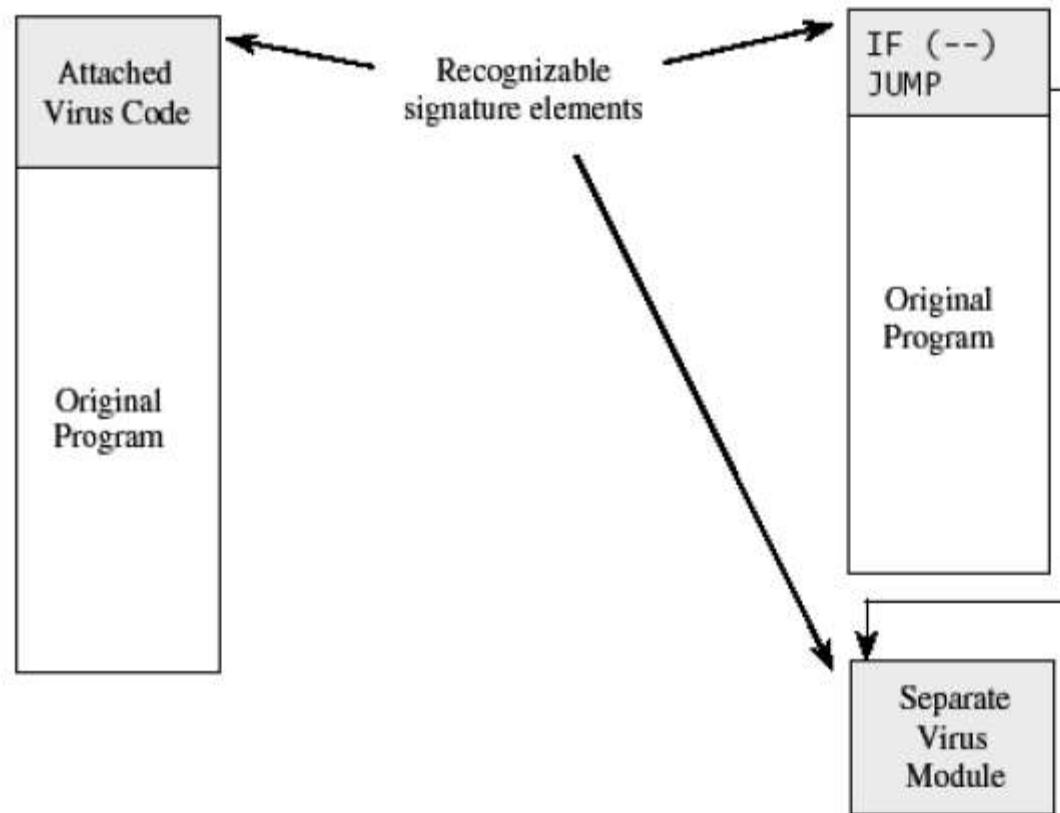
# Virus Signatures

- The virus's signature **is important** for creating a program, called a **virus scanner**, that can automatically detect and, in some cases, remove viruses.
- The scanner searches memory and long-term storage, monitoring execution and watching for the signatures of viruses.
- When the scanner recognizes a known virus's pattern, it can then block the virus, inform the user, and deactivate or remove the virus.
- However, a **virus scanner is effective** only if it has been kept up-to-date with the **latest information** on current viruses

# Storage Patterns

- Most viruses attach to programs that are stored on media such as **disks**.
- The attached **virus piece is invariant**, so that the start of the virus code becomes a detectable **signature**.
- The attached piece is **always located at the same position relative to its attached file**.

- A virus may attach itself to a file, in which case the file's size grows.
- Or the virus may obliterate/destroyed all or part of the underlying program, in which case the program's size does not change but the **program's functioning will be impaired**. The virus writer has to choose one of these detectable effects.
- The virus scanner can use a code or checksum to detect changes to a file. It can also look for **suspicious patterns**, such as a **JUMP** instruction as the first instruction of a system program



# Execution Patterns

- A virus writer may want a virus to do several things at the same time, namely, spread infection, avoid detection, and cause harm.
- Unfortunately, many of these behaviours are perfectly normal and might otherwise go undetected. For instance, one goal is modifying the file directory; many normal programs create files, delete files, and write to storage media. Thus, there are no key signals that point to the presence of a virus.

<b>Virus effect</b>	<b>How it is caused</b>
Attach to executable program	Modify file directory Write to executable program file
Attach to data or control file	Modify directory, rewrite data, append to data, append data to self
Remain in memory	Intercept/ interrupt by modifying interrupt handler address table. Load self in nontransient memory area
Infect disks	Intercept interrupt Intercept OS call (eg., to format disk) Modify system file Modify ordinary executable program
Conceal self	Intercept system calls that would reveal self and falsify result
Spread infection	Infect boot sector, infect systems program, Infect ordinary program Infect data ordinary program reads to control its execution
Prevent deactivation	Activate before deactivating program and block deactivation Store copy to reinfect after deactivation

# Transmission Patterns

- A virus is effective only if it has **some means** of transmission from one location to another. As we have already seen, viruses can travel during the boot process, by attaching to an executable file or travelling within data files.
- The travel itself occurs during execution of an already infected program. Since a virus can execute any instructions a program can, virus travel is not confined to any single medium or execution pattern.

# Polymorphic Viruses

- A virus that can change its appearance is called a polymorphic virus. (*Poly* means "many" and *morph* means "form".) *A two-form polymorphic virus can be handled easily as two independent viruses.*
- Therefore, the virus writer intent on preventing detection of the virus will want either a large or an unlimited number of forms so that the number of possible forms is too large for a virus scanner to search for.

# Prevention of Virus Infections

- Use only commercial software acquired from reliable, well-established vendors
- Test all new software on an isolated computer
- Open attachments only when you know them to be safe
- Make a recoverable system image and store it safely
- Make and retain backup copies of executable system files.
- Use virus detectors daily and update them regularly

# Truths and Misconceptions about viruses

- Viruses can infect only Microsoft Windows systems – False
- Viruses can modify “hidden” or “read-only” files – True
- Viruses can appear only in data files, or only in Word documents, or only in programs – False
- Viruses spread only on disks or only in e-mail – False
- Viruses cannot infect hardware – True
- WINE

# EXAMPLES

1. **Brain Virus** - Brain (computer virus)
  - **Brain** is the industry standard name for a computer virus that was released in its first form in 19 January 1986, and is considered to be the first computer virus for the IBM Personal Computer (IBM PC) and compatibles.  
Designed by Pakistan
2. Internet Worm - The **Morris worm** or **Internet worm of November 2, 1988**, was one of the oldest computer worms distributed via the Internet, and the first to gain significant mainstream media attention.

# EXAMPLES

- Code RED Worm - **Code Red** was a computer worm observed on the Internet on July 15, 2001. It attacked computers running Microsoft's IIS web server. Type – Server jamming worm.
- SQL-Server Slammer - **SQL Slammer** is a 2003 computer worm that caused a denial of service on some Internet hosts and dramatically slowed general Internet traffic.
- Web Bugs (spyware) - Web bugs are tiny (usually a single pixel) transparent image files on web pages that are used to monitor user's online habits.

# TARGETED MALICIOUS CODE

- Another class of malicious code is written for a particular system, for a **particular application, and for a particular purpose.**
- Examples:
- Trapdoor – undocumented entry point to a module
- Salami Attack (Ex. Interest computation). An attack on a computer network which involves the **intruder siphoning off small amounts** of money from a file and placifile that holds their bank account detailsng them in another file that he or she can access; for example, a.
- Covert Channels: programs that **leak information** (Ex. Hide data in output). Type of computer security attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate

# Trapdoors

A trapdoor is an **undocumented entry point** to a module. The trapdoor is inserted during code development, perhaps to test the module, to provide "hooks" by which to connect future modifications or enhancements or to allow access if the module should fail in the future.

In addition to these legitimate uses, **trapdoors can allow a programmer access to a program once it is placed in production.**

# Trapdoors

Causes of trapdoors:

- Forget to remove them
- Intentionally leave them in the program for testing
- Intentionally leave them in the program for maintenance of the finished program
- Intentionally leave them in the program as a **covert** means of access to the component after it becomes an accepted part of a production system.

# Salami Attack

- A salami attack merges bits of seemingly insignificant data to yield powerful results.
- For example, programs often disregard small amounts of money in their computations, as when there are fractional pennies as interest or tax is calculated.
- These small amounts are shaved from each computation and accumulated elsewhere, it is unlikely to be noticed for an individual case.
- Salami attack uses siphoning technique. siphoning is a technique used to **“steal” traffic that would normally be directed to another website in search engine results**

# Covert Channels

## Storage channels:

- They pass information using the presence/absence of objects in storage.
- Example: file lock channel
  - In multiuser systems, files can be locked to prevent two people from writing to the same file at the same time.
  - A covert channel can signal one bit of information by whether or not a file is locked.

## Timing channels:

- They pass information by using the speed at which things happen.
- These are shared resource channels in which the shared resource is time.

# Controls against program threats

- There are many ways a program can fail and many ways to turn the underlying faults into security failures.
- It is better to focus on prevention than cure; how do we use controls during s/w development to find and eliminate the faults.
- 3 types of controls:
  - Developmental controls
  - Operating system
  - administrative

# Controls against program threats

- **Developmental controls:**
- Many controls can be applied during s/w development to fix problems.
- The nature of s/w development:
  - specify
  - Design
  - Implement
  - Test
  - Review
  - Document the system
  - Manage
  - maintain

# Controls against program threats

- **Modularity, Encapsulation and information hiding:**
- A key principle of SE is to **create a design or code in small, self-contained units** called components/modules, when a system is written this way, it is called **modular**
- If a **component is isolated from the effects of other components**, then
  - it is easier to trace a problem to the fault that caused it
  - easier to maintain the system
  - Easier to see where vulnerabilities may lie
- This isolation is called **encapsulation**
- **Information hiding** is another characteristic of modular s/w.
- When information is hidden, each component hides its implementation from others, so that when a change is needed, the overall design can remain intact.

# Controls against program threats

- **Modularity:**
- Modularization is a process of **dividing a task into subtasks**.
- The goal is to have each component **meet 4 conditions**:
- **Single-purpose:** performs one function
- **Small:** consists of an amount of information for which a human can readily grasp both structure and content.
- **Simple:** is of a low degree of complexity so that a human can readily understand the purpose and structure of the module.
- **Independent:** performs a task isolated from other modules.

# Controls against program threats

- **Modularity:**
- Advantages:
  - Maintenance
  - Understandability
  - Reuse
  - Correctness
  - Testing
- A modular component has high cohesion and low coupling

# Controls against program threats

- **Modularity:**
- **Cohesion:** all elements of a component have a logical and functional reason; every aspect of the component is tied to the component's single purpose.
- A high cohesive component **has high degree of focus on purpose**
- Low degree cohesion means **that component's are unrelated.**
- **Coupling:** refers to the **degree with which a component depends on other components in the system**
- Thus, **low coupling is better than tight coupling.**

# Controls against program threats

- **Encapsulation:**
  - It hides a component's implementation details, but it does not necessarily mean complete isolation.
  - The sharing is documented so that a component is affected only in known ways by others.
  - Sharing is minimized so that fewest interfaces possible are used.
- **Information hiding:**
  - Component is a black box with certain well-defined inputs and outputs and well-defined function.
  - Other components do not need to know how the module completes its function.

# Controls against program threats

- Peer Reviews
- Hazard Analysis – set of systematic techniques to expose potentially hazardous system states
- Testing – unit testing, integration testing, function testing, performance testing, acceptance testing, installation testing, regression testing
- Good Design
  - Using a philosophy of *fault tolerance*
  - Have a consistent *policy* for handling failures
  - Capture the *design rationale* and history
  - Use design patterns

# Controls against program threats

- Prediction – predict the risks involved in building and using the system
- Static Analysis – Use tools and techniques to examine characteristics of design and code to see if the characteristics warn of possible faults
- Configuration Management – control changes during development and maintenance
- Analysis of Mistakes
- Proofs of Program Correctness – Can we prove that there are no security holes?

# Operating System Controls on Usage of Programs

- Trusted Software – code has been rigorously developed and analyzed
  - Functional correctness
  - Enforcement of integrity
  - Limited privilege
  - Appropriate confidence level
- Mutual Suspicion – Suspicion or a suspicion is a belief or feeling that someone has **committed a crime or done something wrong.** Assume other program is not trustworthy
- Confinement – limit resources that program can access
- Access Log – list who access computer objects, when, and for how long

# Administrative Controls

- Standards of Program Development
  - Standards of design
  - Standards of documentation, language, and coding style
  - Standards of programming
  - Standards of testing
  - Standards of configuration management
  - Security Audits
- Separation of Duties according to administrative and operational level managers and workers.

# Unit 5

# Systems Design

# Topics

- Design principles
- Representing identity
- Control of access and information flow
- Confinement problem
- Assurance: Building systems with assurance, formal methods,  
Evaluating systems.

# Systems design

- **Systems design** is the process of defining the architecture, product design, modules, interfaces, and data for a system to satisfy specified requirements.
- Systems design could be seen as the application of systems theory to product development.

# Secured Design properties

- **Simplicity** makes designs and mechanisms easy to understand.
- Simplicity reduces the potential for inconsistencies within a policy or set of policies.
- Minimizing the interaction of system components minimizes the number of sanity checks on data being transmitted among components.
- A **sanity check** is an essential procedure **to check** the presence of any **errors** in the initial process. A sanity check focuses on possible errors that may appear in the initial process of setting up.

# The Importance of Good Design Principles

- Every design, whether it be for hardware or software, must begin with a **design philosophy and standard guiding principles**.
- These principles cover the design, are **built in from the beginning, and are preserved** (according to the design philosophy) as the design evolves.

# Design principles

- There are eight principles for the design and implementation of security mechanisms.
  1. Principle of least privilege.
  2. Principle of Fail-Safe Defaults.
  3. Principle of Economy of Mechanism
  4. Principle of Complete Mediation
  5. Principle of Open Design
  6. Principle of Separation of privilege
  7. Principle of least common mechanism
  8. Principle of Psychological acceptability

# Principle of least privilege

- Restricts how privileges are granted.
- Principle states that “**a subject should be given only those privilege that it needs in order to complete its task.**”
- If a specific action requires that **a subject’s access rights be augmented**, those extra rights be relinquished/claimed immediately on completion of the action.

# Principle of Fail-safe Defaults

- Restricts how privileges are initialized when a subject or object is created.
- “The principle of Fail-Safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object”.
- This principle requires that the default access to an object is none.

# Principle of economy of mechanism

- Simplifies the design and implementation of security mechanisms.
- “The principle of economy of mechanism states that security mechanism should be as simple as possible”
- If a design and implementations are simple, fewer possibilities exist for errors.
- Simpler means less can go wrong And when errors occur, they are easier to understand and fix

# Principle of complete mediation

- Complete mediation meaning **every access to every object must be checked for authority.**
- This principle restricts the caching of information, which often leads to simpler implementations of mechanisms.
- “**The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed.**” Whenever a subject attempts to read an object, the OS should mediate the action.
- **First**, it determines if the subject is allowed to read the object. If so, it provides the resources for the read to occur.
- If the subject tries to read it again, the system should check that the subject is still allowed to read the object.
- Most systems would not make the **second check**.
- They would cache(future requests for that data can be served faster) the results of the first check and base the second access on the cached results.

# Principle of open design

- The principle suggests that complexity does not add security.
- “The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation”.
- Designers of a program must not depend on secrecy of the details of their design and implementation to ensure security.
- If the strength of the program’s security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism.
- Does not apply to information such as passwords or cryptographic keys

# Principle of separation of privilege

- This principle is restrictive because it limits access to system entities
- “The principle of separation of privilege states that a system should not grant permission based on single condition.”
- Company cheques for more than \$75,000 must be signed by two officers of the company. If either does not sign, the cheque is not valid. The two conditions are the signatures of two officers.
- This provides a **fine-grained control** over the resource as well as additional assurance that the access is authorized.

# Principle of least common mechanism

- This principle is restrictive because it limits sharing.
- “The principle of least common mechanism states that mechanisms used to access resources should not be shared.”
- Sharing resources provides a channel along which information can be transmitted, and so such sharing should be minimized.
- Covert channels

# Principle of psychological acceptability

- This principle **recognizes the human element** in computer security
- “The Principle of psychological acceptability states that security mechanism should not make the resource more difficult to access than if the security mechanisms were not present.”
- If security related software is too complicated to configure, system administrators may unintentionally set up software in a non secure manner.
- Security-related user programs must be easy to use and must output understandable messages.
- Captcha

**Which principle(s) you would like to incorporate while designing Security systems for following:**

- 1)Research Lab for developing COVID-19 vaccine
- 2)Library
- 3)Shopping mall

# Representing Identity

- Identity is simply a computer's representation of an entity.
- **Definition**

A *principal* is a unique entity. An *identity* specifies a principal.

- Identities are used for several purposes.
- accountability
  - Accountability requires an identity that tracks principals across actions and changes of other identities, so that the principal taking any action can be unambiguously identified.
- access control
  - Access control requires an identity that the access control mechanisms can use to determine if a specific access (or type of access) should be allowed.

# Files and objects

- The identity of a file or other entity (here called an “object”) depends on the system that contains the object.
- Local systems identify objects by assigning names.
- The name may be intended for **human use** (such as a file name), for **process use** (such as a file descriptor or handle), or **for kernel use** (such as a file allocation table entry).
- Each name may have different semantics(meaning, or truth.).

# Users

- In general, a *user* is an identity tied to a single entity.
- Specific systems may add additional constraints.
- Systems represent user identity in a number of different ways.
- Indeed, the same system may use different representations of identity in different contexts.
- EXAMPLE:
- Versions of the UNIX operating system usually represent *user identity* as an integer between 0 and some large integer (usually 65,535). This integer is called the *user identification number*, or **UID**. Principals (called *users*) may also be assigned *login names*. Each login name corresponds to a single UID (although one UID may have many different login names)

# Groups and Roles

- The “entity” may be a set of entities referred to by a single identifier.
- Principals often need to share access to files.
- Most systems allow principals to be grouped into sets called, logically enough, *groups*.
- Groups are essentially a shorthand tool for assigning rights to a set of principals simultaneously.
  - EXAMPLE: UNIX users are assigned membership to a group when they log in. Each process has two identities, a “user identification” and a “group identification.”
  - A *role* is a type of group that ties membership to function.
  - When a principal assumes a role, the principal is given certain rights that belong to that role.
  - When the principal leaves the role, those rights are removed.

# Naming

- The identifier corresponds to a principal.
- The identifier must uniquely identify the principal to avoid confusion.
- Suppose the principals are people. The identifiers cannot be names, because many different people may have the same name.
- The identifiers must include supportive information to distinguish the “Matt Bishop” who teaches at UC Davis from the “Matt Bishop” who works at Microsoft Corporation.

# Certificates

- Certification authorities (CAs) vouch, at some level, for **the identity of the principal to which the certificate is issued**. Every CA has two policies controlling how it issues certificates.
  1. A *CA authentication policy* describes the level of authentication required **to identify the principal to whom the certificate is to be issued**.
  2. A *CA issuance policy* describes the principals **to whom the CA will issue certificates**

# The Meaning of the Identity

- The authentication policy defines the way in which principals prove their identities.
- Each CA has its own requirements.
- All rely on non electronic proofs of identity, such as biometrics (fingerprints), documents (driver's license, passports), or personal knowledge.
- If any of these means can be compromised, the CA may issue the certificate in good faith to the wrong person

# Trust

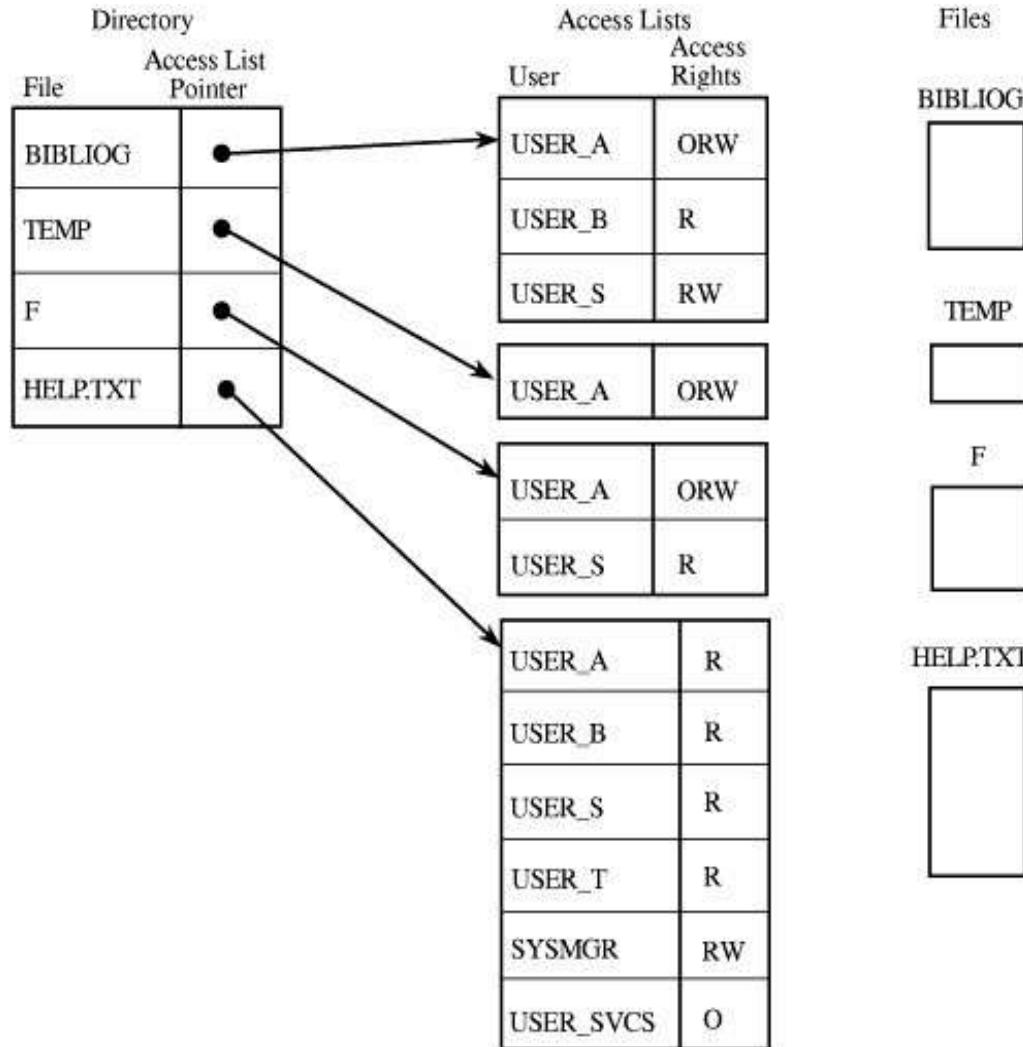
- The goal of certificates is to bind the correct identity to the public key.
- When a user obtains a certificate, the issuer of that certificate is vouching, to some degree of certainty, that the identity corresponds to the principal owning the public key.
- EXAMPLE: Consider the CA that requires a passport to issue a certificate. The certificate will have the name in the passport, the name of the country issuing the passport, and the passport number.

# Identity on the Web

- Certificates are not common on the Internet.
- Several other means attach identity to information, even though the binding may be very transient.
- The Internet requires every host to have an address. The address may be fixed or may change, and without cryptography the binding is weak.

# Control of Access

## Access Control List



- There is one such list for each object, and the list shows all subjects who should have access to the object and what their access is.

# Creation and Maintenance of ACL

Specific implementations of ACLs differ in details. Some of the issues are as follows.

- 1. Which subjects can modify an object's ACL?
- 2. If there is a privileged user (such as *root* in the UNIX system or *administrator* in Windows NT), do the ACLs apply to that user?
- 3. Does the ACL support groups or wildcards?
- 4. How are contradictory access control permissions handled? If one entry grants read privileges only and another grants write privileges only, which right does the subject have over the object?
- 5. If a default setting is allowed, do the ACL permissions modify it, or is the default used only when the subject is not explicitly mentioned in the ACL?

# Which Subjects Can Modify an Object's ACL?

- When an ACL is created, rights are instantiated. Chief among these rights is the **one we will call *own***.
- Possessors of the *own* right can modify the ACL.
- Creating an object also creates its ACL, with some initial value.
- By convention, the subject with *own* rights is allowed to modify the ACL.
- However, some systems allow anyone with access to manipulate the rights.

# Does the ACL Support Groups and Wildcards?

- In its classic form, **ACLs do not support groups or wildcards**.
- In practice, systems support one or the other (or both) to limit the size of the ACL and to make manipulation of the lists easier.
- A group can either refine the characteristics of the processes to be allowed access or be a synonym for a set of users (the members of the group).

# ACLs and Default Permissions

- When ACLs and abbreviations of access control lists or **default access rights coexist** (as on many UNIX systems), there are two ways to determine access rights.
  1. The first **is to apply the appropriate ACL entry**, if one exists, and to apply the default permissions or abbreviations of access control lists otherwise.
  2. The second way is **to augment the default permissions** or abbreviations of access control lists with those in the appropriate ACL entry.

# Revocation of Rights

- *Revocation*, or the prevention of a subject's accessing an object, requires that the subject's rights be deleted from the object's ACL.
- Preventing a subject from accessing an object is simple.
- The entry for the subject is deleted from the object's ACL.
- If only specific rights are to be deleted, they are removed from the relevant subject's entry in the ACL.

# Capabilities

- Conceptually, a capability is like **the row of an access control matrix**.
- Each subject has associated with it a set of pairs, with **each pair containing an object and a set of rights**.
- The subject associated with this list can access the named object in any of the ways indicated by the named rights.

# Locks and Keys

- The locks and keys technique combines features of access control lists and capabilities.
- A piece of information (the lock) is associated with the object and a second piece of information (the key) is associated with those subjects authorized to access the object and the manner in which they are allowed to access the object.
- When a subject tries to access an object, the subject's set of keys is checked. If the subject has a key corresponding to any of the object's locks, access of the appropriate type is granted.
- locks and keys are dynamic in nature. An access control list is static in the sense that all changes to it are manual; a user or process must interact with the list to make the change.
- Locks and keys, on the other hand, may change in response to system constraints, general instructions about how entries are to be added, and any factors other than a manual change.

# Ring-Based Access Control

- To understand its simplicity and elegance, one must realize that files and memory are treated the same from the protection point of view.
- For example, a procedure may occupy a segment of the disk. When invoked, the segment is mapped into memory and executed.
- Data occupies other segments on disk, and when accessed, they are mapped into memory and accessed.
- In other words, there is no conceptual difference between a segment of memory and a segment on a disk.

# Propagated Access Control Lists(PACL)

- It provides the creator of an object with control over who can access the object.
- The creator (originator) is kept with the PACL, and only the creator can change the PACL.
- When a subject reads an object, the PACL of the object is associated with the subject.
- When a subject creates an object, the PACL of the subject is associated with the object.

# Information flow

- Information flow policies define the way information moves throughout a system.
- Typically, these policies are designed to preserve confidentiality of data or integrity of data.
- In the former, the policy's goal is to prevent information from flowing to a user not authorized to receive it.
- In the latter, information may flow only to processes that are no more trustworthy than the data.

# Compiler-Based Mechanisms

- Compiler-based mechanisms check that information flows throughout a program **are authorized**.
- The mechanisms determine if the information flows in a program **could violate a given information flow policy**.
- **Definition :** A set of statements is *certified* with respect to **an information flow policy** if the information flow within that set of statements **does not violate the policy**.

# Declarations

- For our discussion, we assume that the allowed flows are supplied **to the checking mechanisms through some external means**, such as from a file.
- The specifications of allowed flows involve **security classes of language constructs**.
- The program involves variables, so some language construct must relate variables to security classes.
- One way is to assign each variable to exactly one security class.

# Program Statements

- A program consists of several types of statements. Typically, they are
  - 1. Assignment statements
  - 2. Compound statements
  - 3. Conditional statements
  - 4. Iterative statements
  - 5. Goto statements
  - 6. Procedure calls
  - 7. Function calls
  - 8. Input/output statements.
- We consider each of these types of statements separately, with two exceptions.
  1. **Function calls can be modeled as procedure calls** by treating the return value of the function as an output parameter of the procedure.
  2. **Input/output statements** can be modeled as assignment statements in which the value is assigned to (or assigned from) a file.
- Hence, we do not consider function calls and input/output statements separately.

# Execution-Based Mechanisms

- The goal of an execution-based mechanism is to prevent an information flow that violates policy.
- Checking the flow requirements of explicit flows achieves this result for statements involving explicit flows.
- EXAMPLE: Let  $x$  and  $y$  be variables. The requirement for certification for a particular statement  $y \text{ op } x$  is that  $x \leq y$ .
- The conditional statement

if  $x = 1$  then  $y = a$ ;

causes a flow from  $x$  to  $y$ .

Now, suppose that when  $x \neq 1$ ,  $x = High$  and  $y = Low$ .

If flows were verified only when explicit, and  $x \neq 1$ , the implicit flow would not be checked.

# Example Information Flow Controls

- Like the **program-based information flow** mechanisms discussed in last slide, both special purpose and general-purpose computer systems have information flow controls at the system level.
- File access controls, integrity controls, and other types of access controls are mechanisms that **attempt to inhibit the flow of information within a system, or between systems.**
  - The first example is a special-purpose computer that checks I/O operations between a host and a secondary storage unit. It can be easily adapted to other purposes.
  - A mail guard for electronic mail moving between a classified network and an unclassified one follows.
  - The goal of both mechanisms is to prevent the illicit flow of information from one system unit to another.

# The Confinement Problem

*The confinement problem deals with prevention of processes from taking disallowed actions. OR*

According to Lampson:

*The confinement problem is the problem of preventing a server from leaking information that the user of the service considers confidential.*

- Consider a client and a server. When the client issues a request to the server, the client sends the server some data.
- The server then uses the data to perform some function and returns a result (or no result) to the client.
- **Access control affects** the function of the server in two ways.
  - 1. The server must ensure that the resources it accesses **on behalf of the client** include only those resources that the client is authorized to access.
  - 2. The server must ensure **that it does not reveal the client's data** to any other entity not authorized to see the client's data.

- According to Lampson:
- **Definition :**The *confinement problem* is the problem of preventing a server from leaking information that the user of the service considers confidential.
- **Definition:** A *covert channel* is a path of communication that was not designed to be used for communication.
- **Definition:** The *rule of transitive confinement* states that if a confined process invokes a second process, the second process must be as confined as the caller.

# Isolation

- Systems isolate processes in two ways.
  1. In the first, the process is presented with an environment that appears to be a computer running only that process or those processes to be isolated.
  2. In the second, an environment is provided in which process actions are analyzed to determine if they leak information.
- The first type of environment prevents the process from accessing the underlying computer system and any processes or resources that are not part of that environment.
- The second type of environment does not emulate a computer. It merely alters the interface between the existing computer and the process(es).

# Virtual Machines

- The first type of environment is called a *virtual machine*.
- **Definition :** A *virtual machine* is a program that simulates the hardware of a (possibly abstract) computer system.
- A virtual machine uses a special operating system called a *virtual machine monitor* to provide a virtual machine on which conventional operating systems can run.

# Sandboxes

- The computer sandbox provides a safe environment for programs to execute in.
- If the programs “leave” the sandbox, they may do things that they are not supposed to do.
- Both types of sandboxes restrict the actions of their occupants.

**Definition :** A *sandbox* is an environment in which the actions of a process are restricted according to a security policy.

- Systems may enforce restrictions in two ways.
- First, the sandbox can limit the execution environment as needed.
- The second enforcement method is to modify the program (or process) to be executed.

# Covert Channels

- Covert channels use shared resources as paths of communication.
- This requires sharing of space or sharing of time.
- **Definition :** A *covert storage channel* uses an attribute of the shared resource. A *covert timing channel* uses a temporal or ordering relationship among accesses to a shared resource.
- **Definition :** A *noiseless covert channel* is a covert channel that uses a resource available to the sender and receiver only. A *noisy covert channel* is a covert channel that uses a resource available to subjects other than the sender and receiver, as well as to the sender and receiver.

# Mitigation of Covert Channels

- Covert channels convey information by varying the use of shared resources.
- An obvious way to eliminate all covert channels is to require processes to state what resources they need before execution and provide these resources in such a manner that only the process can access them.
- This includes runtime, and when the stated runtime is reached, the process is terminated and the resources are released.
- The resources remain allocated for the full runtime even if the process terminates earlier.

# Mitigation .....

- An alternative approach is to obscure the amount of resources that a process uses.
- This can be done in two ways.
  - First, the resources devoted to each process can be made uniform. In essence, the system eliminates meaningful irregularities in resource allocation and use.
  - Second, a system can inject randomness into the allocation and use of resources. The goal is to make the covert channel a noisy one and to have the noise dominate the channel.

# Introduction to Assurance

---

Assurance for secure and trusted systems must be **an integral part of the development process.** Confidence gained as result of evidence.

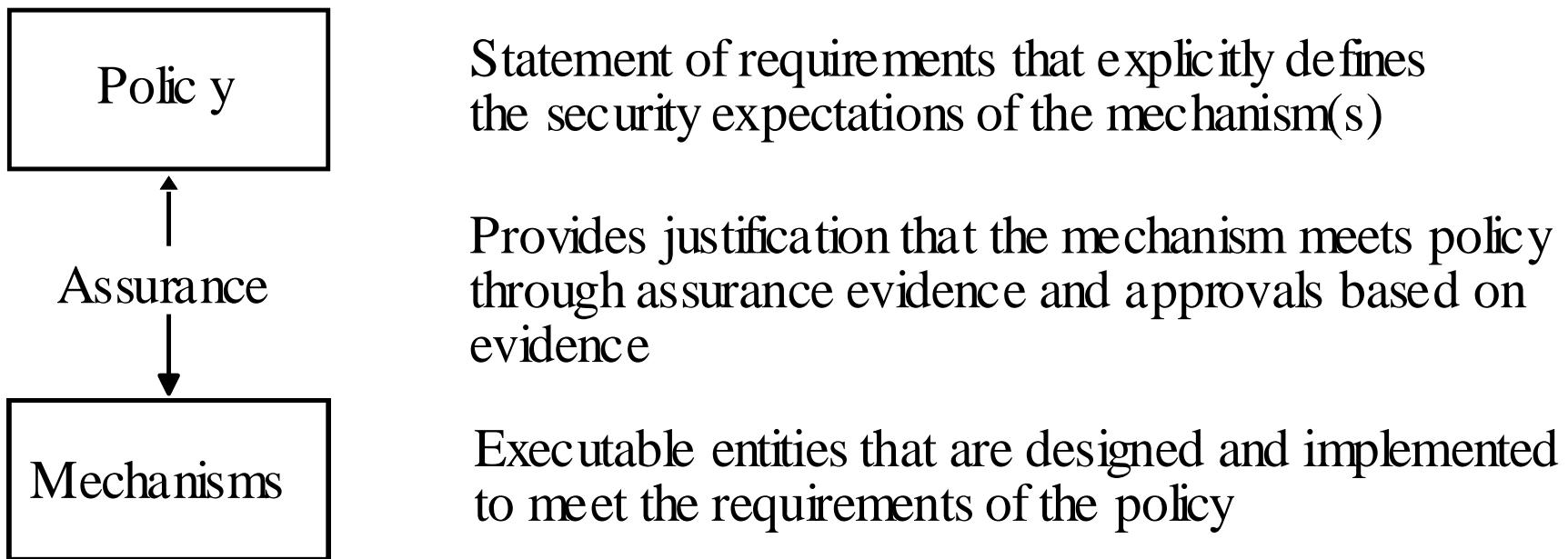


# Trust

- *Trustworthy* entity has sufficient credible evidence leading one to believe that the system will meet a set of requirements.
- *Trust* is a measure of trustworthiness **relying on the evidence**.
  - To trust makes one vulnerable to violations trust.
- *Assurance* is **confidence** that an entity **meets its security requirements** based on evidence provided by applying assurance techniques.
  - Meets security requirements” == Enforces policy

# Trusted System

- A *trusted system* is a system that has been shown to meet well-defined requirements under an evaluation by a credible body of experts who are certified to assign trust ratings to evaluated products and systems.



# The need of Assurance

- Applying assurance techniques is time-consuming and expensive.
- Accidental or unintentional failures of computer systems, as well as intentional compromises of security mechanisms, can lead to security failures.

# Problem Sources -- Neumann's list

1. Policy Flaws
2. Requirements definitions, omissions, and mistakes
3. System design flaws
4. Hardware implementation flaws, such as wiring and chip flaws
5. Software implementation errors, program bugs, and compiler bugs
6. System use and operation errors and inadvertent mistakes
7. Willful system misuse (A serious or high degree of negligence and unmistakable abuse of duty of legal right towards others)
8. Hardware, communication, or other equipment malfunction
9. Environmental problems, natural causes, and acts of God
10. Evolution, maintenance, faulty upgrades, and decommissions

- Implementation assurance deals with hardware and software implementation errors (items 3, 4, and 7),
- errors in maintenance and upgrades (item 9),
- willful misuse (item 6), and environmentally induced problems (item 8).
- Thorough security testing as well as detailed and significant vulnerabilities assessment find flaws that can be corrected prior to deployment of the system.
- Operational assurance can address system use and operational errors (item 5)
- as well as some willful misuse issues (item 6).
- Neumann's list is not exclusive to security

# Examples

- Space Shuttle Challenger explosion
  - Sensors removed from booster rockets to meet accelerated launch schedule
- Deaths from faulty radiation therapy system
  - Hardware safety interlock removed
  - Flaws in software design
- Intel 486 chip
  - Bug in trigonometric functions
  - Intel's public reputation was damaged, and replacing the chips cost Intel time and money.

# Role of Requirements

- *Requirements* are statements of goals that must be met.
  - Vary from high-level, generic issues to low-level, concrete issues.
- *Security objectives* are high-level security issues.
- *Security requirements* are specific, concrete issues.

# Types of Assurance

1. *Policy assurance is evidence establishing security requirements in policy is complete, consistent, technically sound.*
2. *Design assurance is evidence establishing design sufficient to meet requirements of security policy.*
3. *Implementation assurance is evidence establishing implementation consistent with security requirements of security policy*

# Types of Assurance

4. *Operational assurance* is evidence establishing system sustains the security policy requirements during installation, configuration, and day-to-day operation

- Also called *administrative assurance*
- One fundamental operational assurance technique is a thorough review of product or system documentation and procedures, to ensure that the system cannot accidentally be placed into a non-secure state.
- This emphasizes the importance of proper and *complete* documentation for computer applications, systems, and other entities.

# Assurance Throughout the Life Cycle

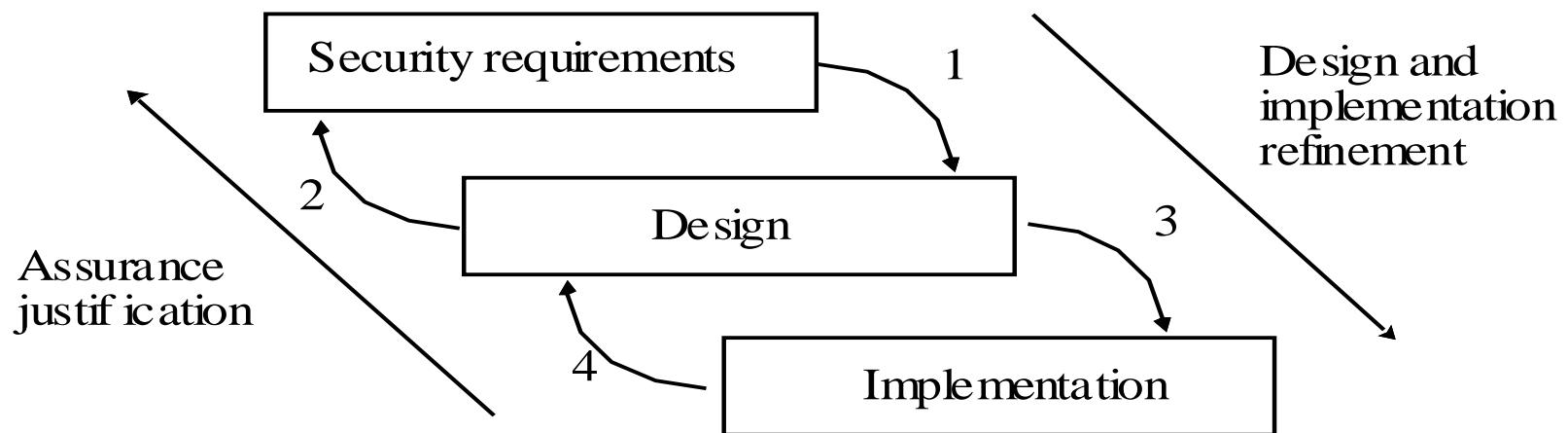
- The goal of assurance is to show that **an implemented and operational system meets its security requirements** throughout its life cycle.
- Because of the **difference in the levels of abstraction** between **high-level security requirements and low-level implementation details**, the demonstration is usually done in stages.

# Building Secure and Trusted Systems

- Building secure and trusted systems depends on standard software engineering techniques augmented with specific technologies and methodologies.
- Hence, a review of the life cycles of systems will clarify much of what follows.
  - Life cycle

# Life Cycle

- This process is usually iterative, because assurance steps identify flaws that must be corrected. When this happens, the affected steps must be rechecked.



- Assurance must continue throughout the life of the system. Because maintenance and patching usually affect the system design and implementation.
- **Note that the refinement steps alternate with the assurance steps.**

# Life Cycle Assurance

## 1. Conception

- Initial focus is on policy and requirements

## 2. Manufacture

- Select mechanisms to enforce policy
- Give evidence that mechanisms are appropriate

## 3. Deployment

- Prepare operational plans that realize policy goals
- Provide mechanism for distribution and delivery that assures product integrity
- Support appropriate configuration

## 4. Fielded Product Life

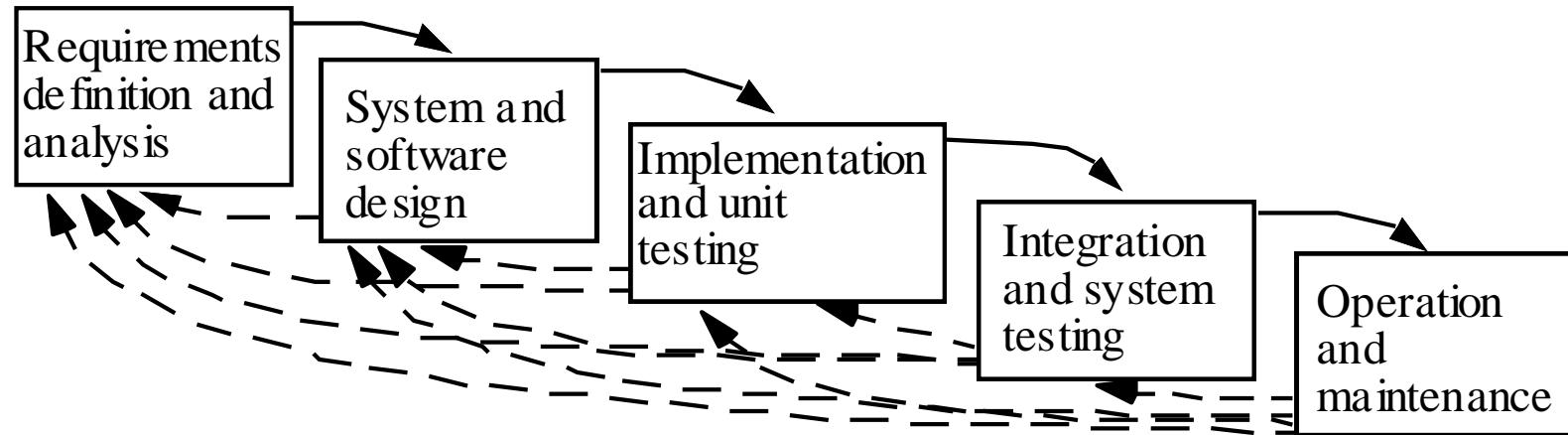
- Update and patch mechanism
- Customer support
- Product decommissioning and end of life

# Waterfall Life Cycle Model

- The *waterfall life cycle model* is the model of building in stages, whereby one stage is completed before the next stage begins.
- Five Steps are :
  1. Requirements definition and analysis
    - Functional and non-functional
    - General (for customer), specifications
  2. System and software design
  3. Implementation and unit testing
  4. Integration and system testing
  5. Operation and maintenance

# Relationship of Stages

- The waterfall life cycle model. The solid arrows represent the flow of development in the model. The dashed arrows represent the paths along which information about errors may be sent.



# 1. Requirements Definition and Analysis

- In this phase, a feasibility study may examine whether or not the requirements are correct, consistent, complete, realistic, verifiable, and traceable.
- It is likely that there will be **some iteration** between the **requirements definition step** and the **architecture step** before either can be completed.
  - Functional requirements describe **interactions between the system and its environment**.
  - Nonfunctional requirements are **constraints or restrictions** on the system that **limit design or implementation choices**.

# System and Software Design

- This stage is sometimes broken into the two phases *system design*, in which the system as a whole is designed, and *program design*, in which the programs of the system are individually designed.
- Software design further partitions the requirements into specific executable programs.
  - The external functional specifications describe the inputs, outputs, and constraints on functions that are external to the entity being specified,
  - whereas the internal design specifications describe algorithms to be used, data structures, and required internal routines.

# Implementation and Unit Testing

- *Implementation* is the development of software programs based on the software design from the previous step. Typically, the work is divided into a set of programs or program units.
- *Unit testing* is the process of establishing that the unit as implemented meets its specifications. It is in this phase that many of the supporting processes described earlier come into play.

# Integration and System Testing

- *Integration* is the process of combining all the unit-tested program units into a complete system.
- *System testing* is the process of ensuring that the system as a whole meets the requirements. System testing is an iterative step because invariably bugs and errors are found that have to be corrected.

# Operation and Maintenance

- Once the system is finished, it is moved into production. This is called *fielding the system*.
- Maintenance involves correction of errors that have been reported from the field and that have not been corrected at earlier stages.
- This stage also involves routine maintenance and the release of new versions of the system.
- Finally, *retirement/deployment* of the system also falls under this phase.

# Other Models

1. Exploratory programming
  - Develop working **system quickly** and then modified until it **performs adequately**.
  - No requirements or design specification, **so difficulty in assurance**.
  - Therefore, this model is **not particularly useful** for building secure and trusted systems because such systems need precise requirements and detailed verification that they meet those requirements as implemented.
2. Prototyping
  - Objective is **rapid development to establish system requirements**

# Other Models

## 3. Formal transformation

- Create formal specification
- Translate it into program using correctness-preserving transformations
- Very conducive to assurance methods

## 4. System assembly from reusable components

- Depends on whether components are trusted
- Must assure connections, composition as well
- Very complex, difficult to assure

# Other Models

## 5. Extreme programming

- Rapid prototyping and “best practices”
- Project driven by business decisions
- Requirements open until project complete
- Programmers work in teams
- Components tested, integrated several times a day
- Objective is to get system into production as quickly as possible, then enhance it
- Evidence adduced *after* development needed for assurance

# Key Points

- Assurance critical for determining trustworthiness of systems
- Different levels of assurance, from informal evidence to rigorous mathematical evidence
- Assurance needed at all stages of system life cycle
- Building security in is more effective than adding it later

# Evaluation

---

- Evaluation is a process in which the evidence for assurance is gathered and analyzed against criteria for functionality and assurance. Perfect security is an ultimate, but unachievable, goal for computer systems
- A *formal evaluation methodology* is a technique used to provide measurements of trust based on specific security requirements and evidence of assurance.

# An evaluation methodology provides the following features.

- A set of requirements defining the security functionality for the system or product.
- A set of assurance requirements that delineate the steps for establishing that the system or product meets its functional requirements. The requirements usually specify required evidence of assurance.
- A methodology for determining that the product or system meets the functional requirements based on analysis of the assurance evidence.
- A measure of the evaluation result (called a *level of trust*) that indicates how trustworthy the product or system is with respect to the security functional requirements defined for it.

# Evaluation system are :

1. TCSEC(Trusted Computer System Evaluation Criteria, also known as the Orange Book): 1983–1999
2. FIPS( Federal Information Processing Standard) 140: 1994—Present
3. The Common Criteria(CC): 1998—Present
4. SSE-CMM: 1997—Present

# SSE-CMM : 1997- Present

- System Security Engineering Capability Maturity Model (SSE-CMM) is a process-oriented methodology for developing secure systems.
- The SSE-CMM became ISO Standard 21827 in 2002.
- The SSE-CMM is organized into processes and maturity levels.
- Generally speaking, the processes define what needs to be accomplished by the security engineering process and the maturity levels categorize how well the process accomplishes its goals.

- A *process capability* is the range of expected results that can be achieved by SSE CMM process. It is a predictor of future project outcomes.
- *Process performance* is a measure of the actual results achieved.
- *Process maturity* is the extent to which a process is explicitly defined, managed, measured, controlled, and effective.

# SSE-CMM contains 11 process areas.

1. Administer Security Controls
2. Assess Impact
3. Assess Security Risk
4. Assess Threat
5. Assess Vulnerability
6. Build Assurance Argument
7. Coordinate Security
8. Monitor System Security Posture
9. Provide Security Input
10. Specify Security Needs
11. Verify and Validate Security

- The definition of the Assess Threat process area contains the goal that threats to the security of the system be identified and characterized. The base processes are :
  - Identify Natural Threats
  - Identify Human-Made Threats
  - Identify Threat Units of Measure
  - Assess Threat Agent Capability
  - Assess Threat Likelihood
  - Monitor Threats and Their Characteristics

The five Capability Maturity Levels that represent increasing process maturity are as follows.

1. *Performed Informally*. Base processes are performed.
2. *Planned and Tracked*. Project-level definition, planning, and performance verification issues are addressed.
3. *Well-Defined*. The focus is on defining and refining a standard practice and coordinating it across the organization.
4. *Quantitatively Controlled*. This level focuses on establishing measurable quality goals and objectively managing their performance.
5. *Continuously Improving*. At this level, organizational capability and process effectiveness are improved.

# James Anderson's “Computer Security Planning Study” provides a blueprint

- Needs analysis:
  - Multi-level operation
  - Systems connected to the world
  - On-line operation
  - Networks
- Vision
  - Security engineering
  - Secure components (hardware & software)
  - Handbook of Computer Security Techniques

# Evaluation Assurance Level

- EAL 1: functionally tested
- EAL 2: structurally tested
- EAL 3: methodically tested and checked
- EAL 4: methodically designed, tested and reviewed
- EAL 5: semiformallly designed and tested
- EAL 6: semiformallly verified design and tested
- EAL 7: formally verified design and tested

# Common Criteria

- International standard
- EAL 1 -- 5 transferred across borders
- EAL 6 and 7 are not

# Unit 6

## Logic Based System

# Topics

- Malicious logic
- Vulnerability analysis
- Auditing
- Intrusion detection
- Applications: Network security, operating system security, user security, program security
- Special Topics: Data privacy, introduction to digital forensics, enterprise security specification.

# Malicious logic

- *Malicious logic* is a set of instructions that cause a site's security policy to be violated.

# Forms of Malicious Logic

- Trojan Horses
- Computer Viruses
  - Boot Sector Infectors
  - Executable Infectors
  - Multipartite Viruses
  - TSR Viruses
  - Stealth Viruses
  - Encrypted Viruses
  - Polymorphic Viruses
  - Macro Viruses
- Computer Worms
- Other Forms of Malicious Logic
  - Rabbits and Bacteria
  - Logic Bombs

# Trojan Horses

- *Trojan horse* is a program with an overt (documented or known) effect and a *covert* (undocumented or unexpected) effect.
- A *propagating Trojan horse* (also called a *replicating Trojan horse*) is a Trojan horse that creates a copy of itself.
- EXAMPLE: The following UNIX script is named *ls* and is placed in a directory.
  - cp /bin/sh /tmp/.xxsh
  - chmod u+s,o+x /tmp/.xxsh
  - rm ./ls
  - ls \$\*
- It creates a copy of the UNIX shell that is setuid to the user executing this program. This program is deleted, and then the correct *ls* command is executed. On most systems, it is against policy to trick someone into creating a shell that is setuid to themselves. If someone is tricked into executing this script, a violation of the (implicit) security policy occurs. This script is an example of malicious logic.

# EXAMPLE

The NetBus program allows an attacker to control a Windows NT workstation remotely. The attacker can intercept keystrokes or mouse motions, upload and download files, and act as a system administrator would act.

In order for this program to work, the victim Windows NT system must have a server with which the NetBus program can communicate. This requires someone on the victim's system to load and execute a small program that runs the server.

This small program was placed in several small game programs as well as in some other “fun” programs, which could be distributed to Web sites where unsuspecting users would be likely to download them.

# Computer Viruses

- *computer virus* is a program that inserts itself into one or more files and then performs some (possibly null) action.
- This type of Trojan horse propagates itself only as specific programs. When the Trojan horse can propagate freely and insert a copy of itself into another file, it becomes a computer virus.
- The first phase, in which the virus inserts itself into a file, is called the *insertion phase*. The second phase, in which it performs some action, is called the *execution phase*. The NEXT pseudocode fragment shows how a simple computer virus works.

Example :As this code indicates, the insertion phase must be present but need not always be executed. It would check for an uninfected boot file (the *spread-condition* mentioned in the pseudocode) and, if one was found, would infect that file (the *set of target files*). Then it would increment a counter and test to see if the counter was at 4. If so, it would erase the disk. These operations were the *action(s)*.

```
beginvirus:  
    if spread-condition then begin  
        for some set of target files do begin  
            if target is not infected then begin  
                determine where to place virus instructions  
                copy instructions from beginvirus to endvirus  
                    into target  
                alter target to execute added instructions  
            end;  
        end;  
    end;  
    perform some action(s)  
    goto beginning of infected program  
endvirus:
```

# Boot Sector Infectors

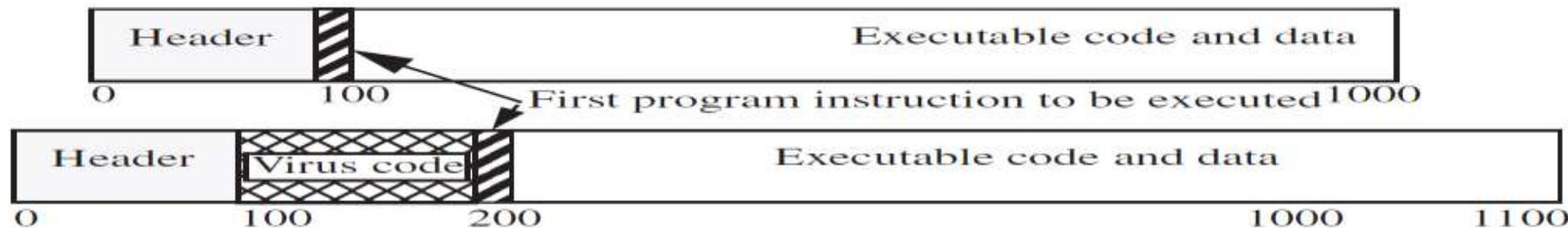
- The *boot sector* is the part of a disk used to bootstrap the system or mount a disk.
- Code in that sector is executed when the system “sees” the disk for the first time. When the system boots, or the disk is mounted, any virus in that sector is executed.
- A *boot sector infector* is a virus that inserts itself into the boot sector of a disk.
- Eg - The Brain virus for the IBM PC is a boot sector infector. When the system boots from an infected disk, the virus is in the boot sector and is loaded. It moves the disk interrupt vector (location 13H or 19) to an alternative interrupt vector (location 6DH or 109) and sets the disk interrupt vector location to invoke the Brain virus now in memory. It then loads the original boot sector and continues the boot.

# Example

- The Brain (or Pakistani) virus, written for IBM PCs, is thought to have been created in early 1986 but was first reported in the United States in October 1987. It alters the boot sectors of floppy disks, possibly corrupting files in the process. It also spreads to any uninfected floppy disks inserted into the system. Since then, numerous variations of this virus have been reported.

# Executable Infectors

- An *executable infector* is a virus that infects executable programs.
- The PC variety of executable infectors are called COM or EXE viruses because they infect programs with those extensions. Figure 19–1 illustrates how infection can occur. The virus can prepend itself to the executable (as shown in the figure) or append itself.



**Figure 19–1** How an executable infector works. It inserts itself into the program so that the virus code will be executed before the application code. In this example, the virus is 100 words long and prepends itself to the executable code.

# Multipartite Viruses

- *multipartite virus* is one that can infect either boot sectors or applications.
- Such a virus typically has two parts, one for each type. When it infects an executable, it acts as an executable infector; when it infects a boot sector, it works as a boot sector infector.

# TSR Viruses

- A *terminate and stay resident* (TSR) virus is one that stays active (resident) in memory after the application (or bootstrapping, or disk mounting) has terminated.
- TSR viruses can be boot sector infectors or executable infectors. Eg – Brain Virus

# Stealth Viruses

- *Stealth* viruses are viruses that conceal the infection of files.
- Example - The Stealth virus (also called the IDF virus or the 4096 virus) is an executable infector. It modifies the DOS service interrupt handler (rather than the interrupt vector; this way, checking the values in the interrupt vector will not reveal the presence of the virus).

# Polymorphic Viruses

- A *polymorphic* virus is a virus that changes its form each time it inserts itself into another program.
- **Macro Viruses**
- A *macro* virus is a virus composed of a sequence of instructions that is interpreted, rather than executed directly.
- For example, a spreadsheet virus executes when the spreadsheet interprets these instructions. If the macro language allows the macro to access files or other systems, the virus can access them, too.

# Computer Worms

- A computer virus infects other programs. A variant of the virus is a program that spreads from computer to computer, spawning copies of itself on each one.
- A *computer worm* is a program that copies itself from one computer to another.

# Other forms of malicious Logic

- **Rabbits and Bacteria**
- Some malicious logic multiplies so rapidly that resources become exhausted. This creates a denial of service attack.
- A *bacterium* or a *rabbit* is a program that absorbs all of some class of resource.
- **Logic Bombs**
- A *logic bomb* is a program that performs an action that violates the security policy when some external event occurs

## EFFECT OF COMPUTER VIRUS

- ✓ It can slow down your computer.
- ✓ It might corrupt your system files.
- ✓ It might make some programs faulty or corrupt.
- ✓ It might damage your boot sector creating problems when you boot into the windows.
- ✓ It might steal important information from your computer and send to some other person.
- ✓ It might change the power ratings of your computer and could blast the system.

# **Vulnerability analysis**

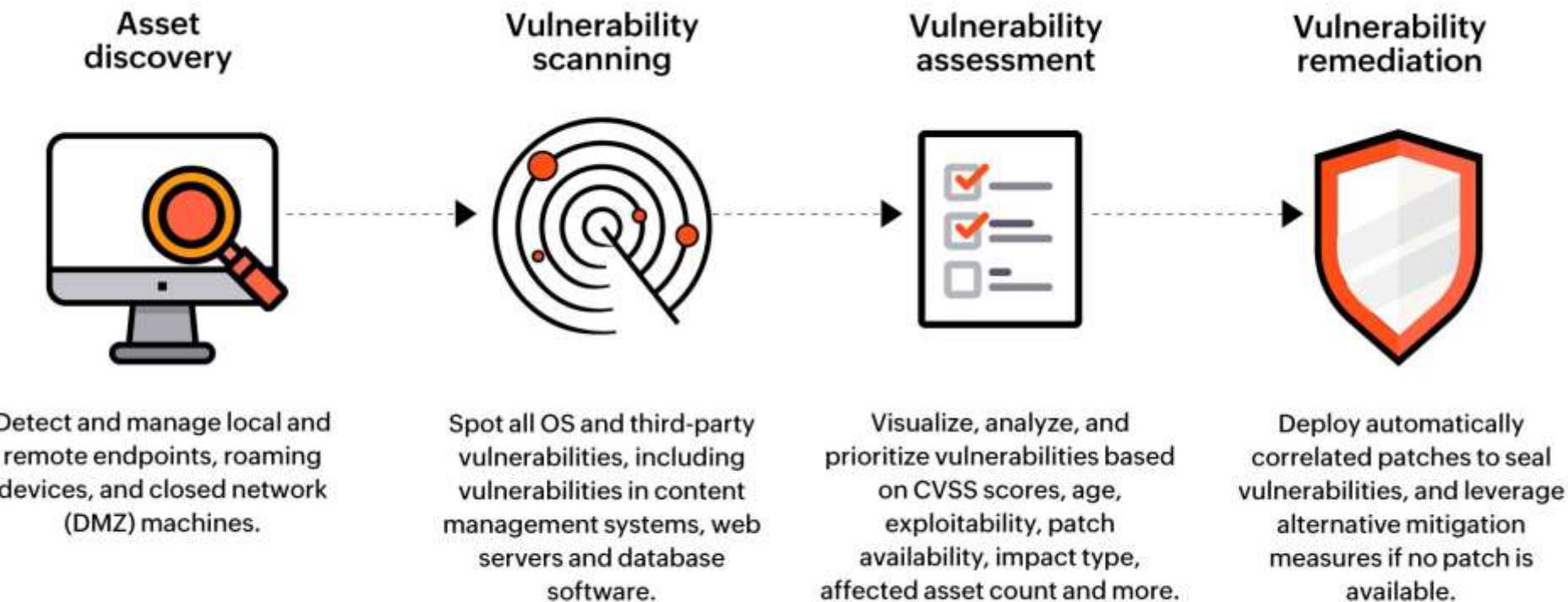
# Vulnerability analysis

- A vulnerability assessment is the process of identifying, quantifying, and prioritizing the vulnerabilities in a system.
- A vulnerability analysis is a review that focuses on security-relevant issues that either moderately or severely impact the security of the product or system.

## **Steps to conducting a proper vulnerability assessment**

- Identify where your most sensitive data is stored.
- Uncover hidden sources of data.
- Identify which servers run mission-critical applications.
- Identify which systems and networks to access.
- Review all ports and processes and check for misconfiguration.

# Steps to vulnerability analysis



# Auditing

- An information technology audit, or information systems audit, is an examination of the management controls within an Information technology infrastructure and business applications.

**First-Party  
Audits**  
Internal  
Audits

## AUDIT LEVELS

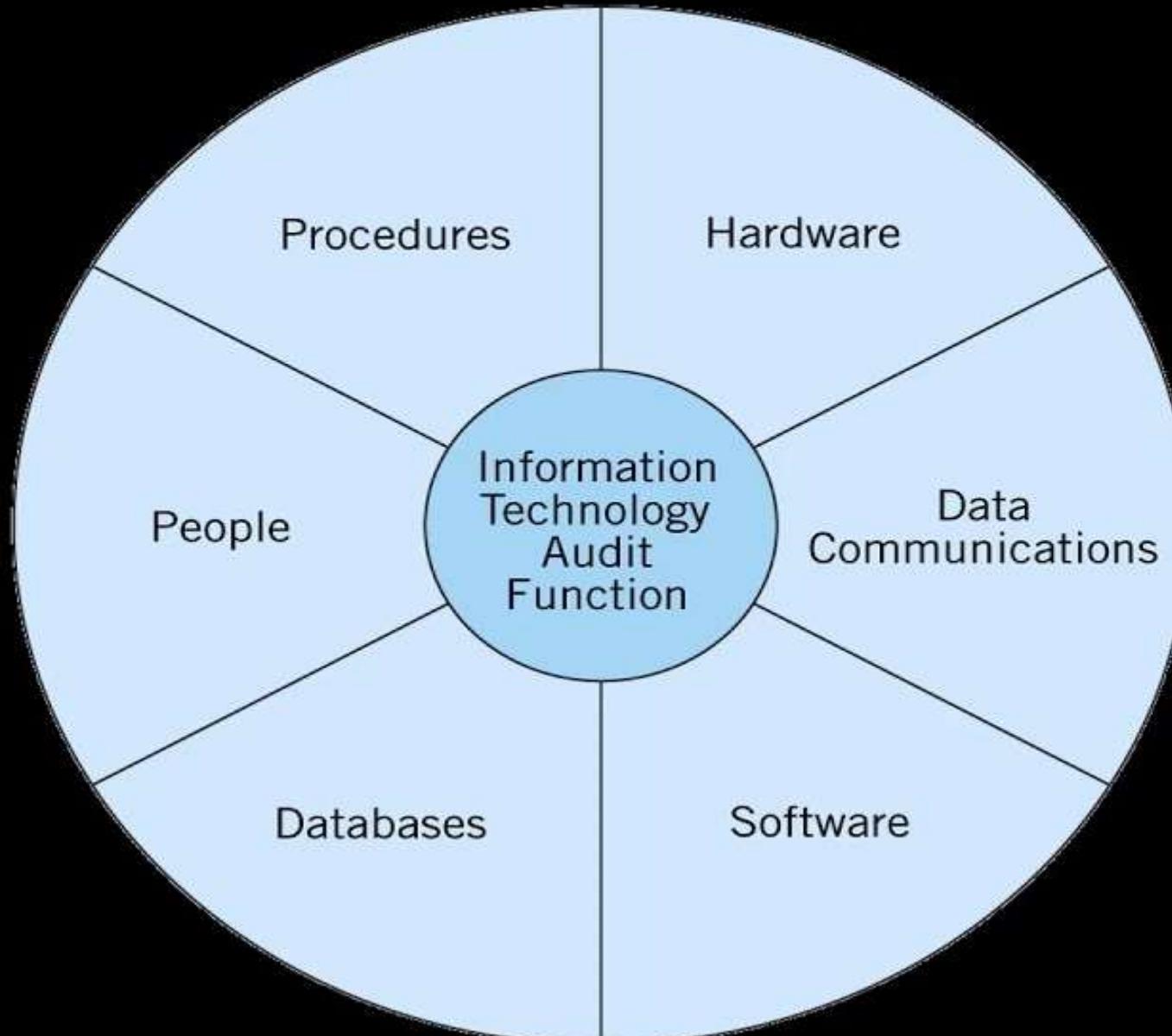
**Third-Party  
Audits**  
Independent  
Examination

**Second-Party Audits**  
Customer Audits

# IT Audit strategies

- Review IT organizational structure.
- Review IT policies and procedures.
- Review IT standards.
- Review IT documentation.
- Review the organization's BIA.
- Interview the appropriate personnel.
- Observe the processes and employee performance.

# The Components of an IT Audit

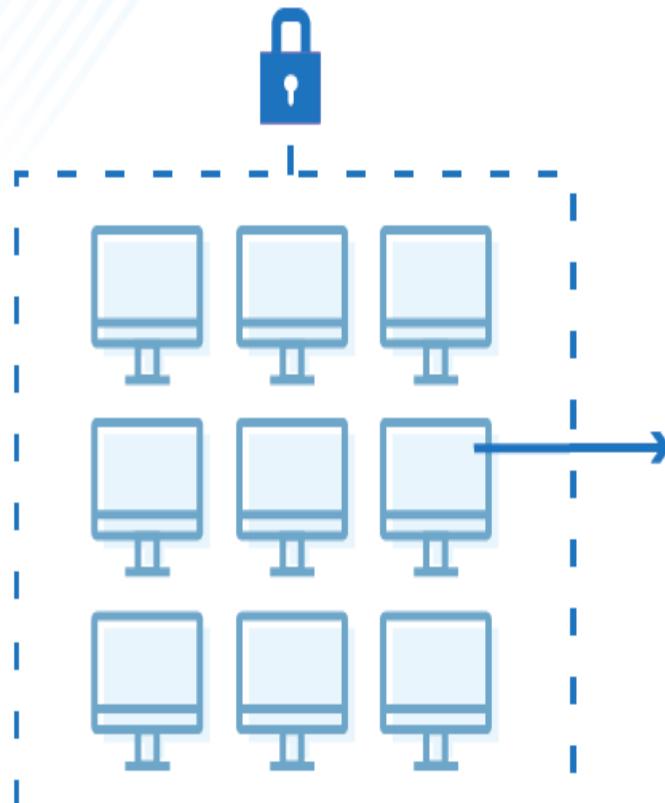


# **Intrusion Detection System**

- An intrusion detection system is a device or software application that monitors a network or systems for malicious activity or policy violations.
- Any intrusion activity or violation is typically reported either to an administrator or collected centrally using a security information and event management system.

- Intrusion detection systems primarily use two key intrusion detection methods: **signature-based intrusion detection and anomaly-based intrusion detection**
- A NIDS system operates at the network level and monitors traffic from all devices going in and out of the network.
- NIDS performs analysis on the traffic looking for patterns and abnormal behaviours upon which a warning is sent.

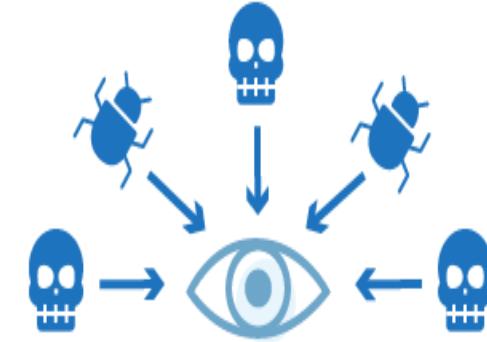
# What Does an Intrusion Detection System Do?



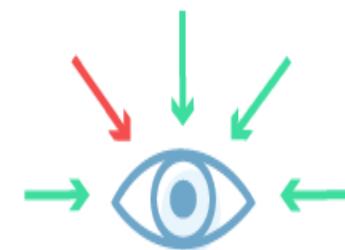
Network Intrusion  
Detection



Host Intrusion  
Detection



Signature-based  
Detection



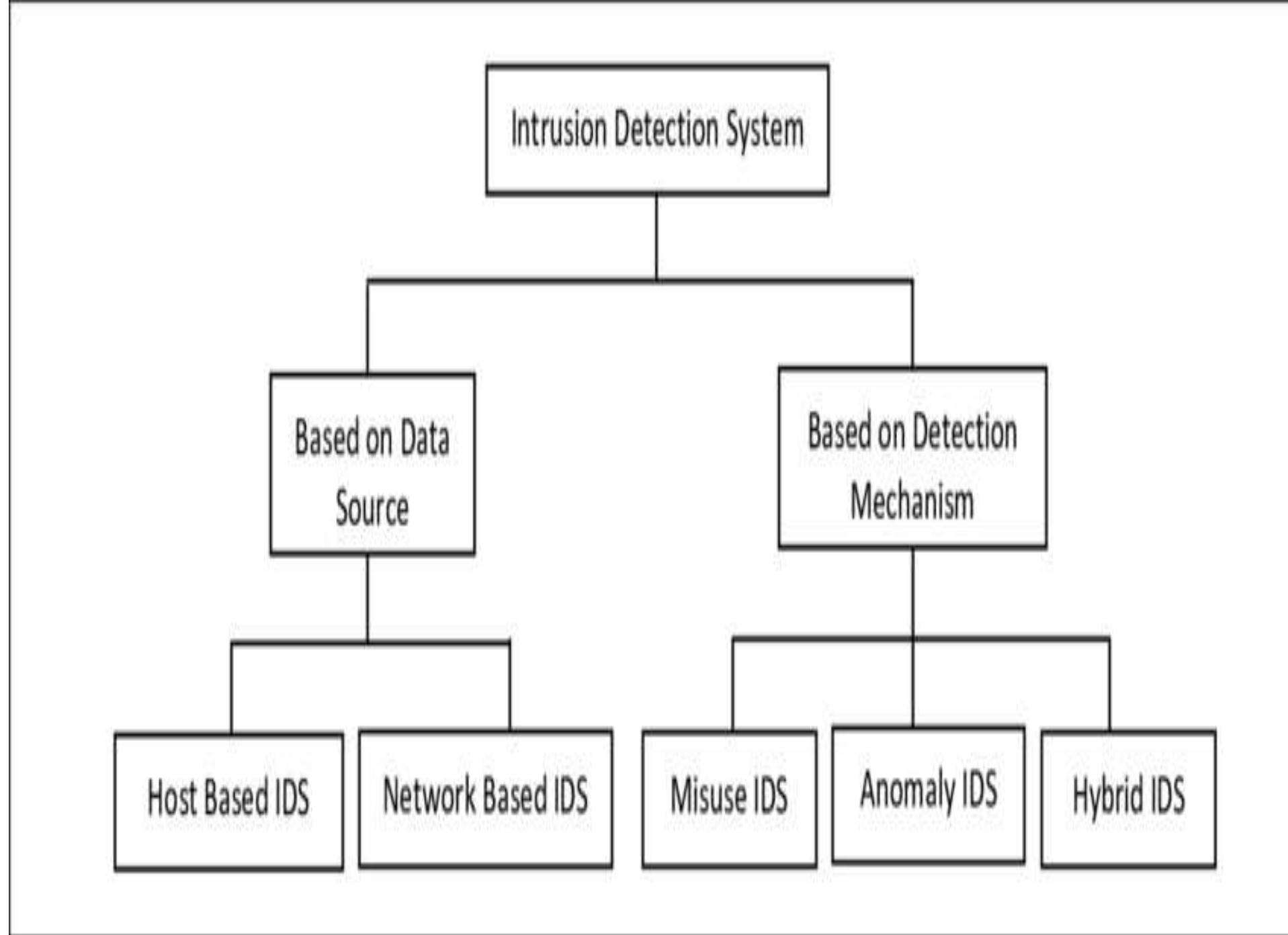
Anomaly-Based  
Detection

# What Does an Intrusion Prevention System Do?



- Identify suspicious activity
- Log security events
- Attempt to block intrusions or limit damage
- Report intrusion attempts





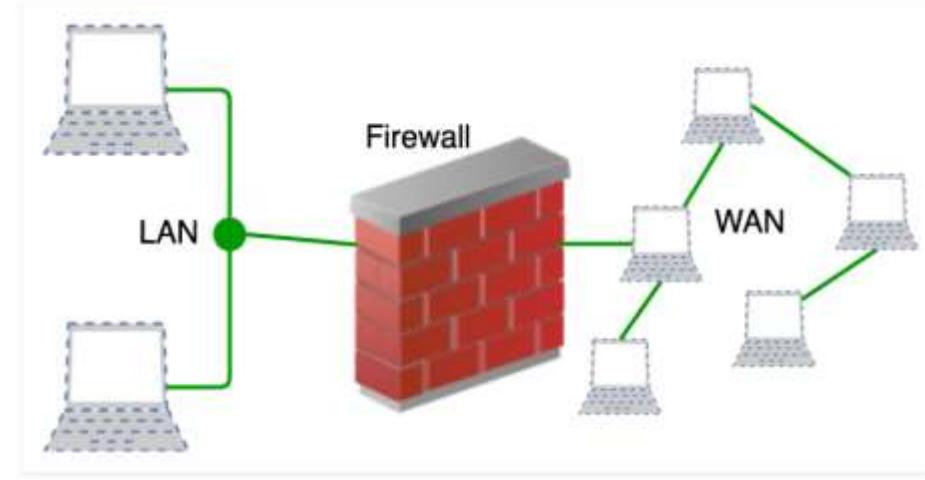
# **Network security**

- Network security is a broad term that covers a multitude of technologies, devices and processes.
- In its simplest term, it is a **set of rules and configurations designed to protect the integrity, confidentiality and accessibility of computer networks and data using both software and hardware technologies.**

# **Types of Network Security Protections**

- Firewall.
- Network Segmentation.
- Remote Access VPN.
- Email Security.
- Data Loss Prevention (DLP)
- Intrusion Prevention Systems (IPS)
- Sandboxing.

# Firewall



- A firewall is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. A firewall typically establishes a barrier between a trusted network and an untrusted network, such as the Internet.
- a defined set of security rules it accepts, rejects or drops that specific traffic.
- **Accept** : allow the traffic  
**Reject** : block the traffic but reply with an “unreachable error”  
**Drop** : block the traffic with no reply

# Network Segmentation

- Network segmentation in computer networking is the act or practice of splitting a computer network into subnetworks, each being a network segment. Advantages of such splitting are primarily for boosting performance and improving security.
- Example : Imagine a large bank with several branch offices. The bank's security policy restricts branch employees from accessing its financial reporting system. Network segmentation can enforce the security policy by preventing all branch traffic from reaching the financial system. And by reducing overall network traffic, the financial system will work better for the financial analysts who use it.

# Remote Access VPN

- In a Remote-access VPNs, individual hosts or clients, such as telecommuters, mobile users, and extranet consumers, are able to access a company network securely over the Internet. Each host typically has VPN client software loaded or uses a web-based client.
- VPNs can be characterized as host-to-network

# Email Security

- Email security refers to various [cybersecurity](#) measures to secure the access and content of an email account or service.
- Email security is important because malicious email is a popular medium for spreading [ransomware](#), [spyware](#), [worms](#), [different types of malware](#), [social engineering attacks](#) like [phishing](#) or [spear phishing](#) emails and other [cyber threats](#).

# Data loss prevention

- Data loss prevention (DLP) software detects potential data breaches/data ex-filtration transmissions and prevents them by monitoring, detecting and blocking sensitive data while in use (endpoint actions), in motion (network traffic), and at rest (data storage). The terms "data loss" and "data leak" are related and are often used interchangeably
- Prevention by :
  - Database Hardening. One of the best ways to prevent data loss is to secure a database by hardening it as much as possible.
  - Manage Database Access Tightly. ...
  - Secure Authentication. ...
  - Secure Communication. ...

# IPS

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

# **Operating system security**



# Operating system security

- Operating system security (OS security) is the process of ensuring OS integrity, confidentiality and availability.
- OS security refers to specified steps or measures used to protect the OS from threats, viruses, worms, malware or remote hacker intrusions



# Protected Objects

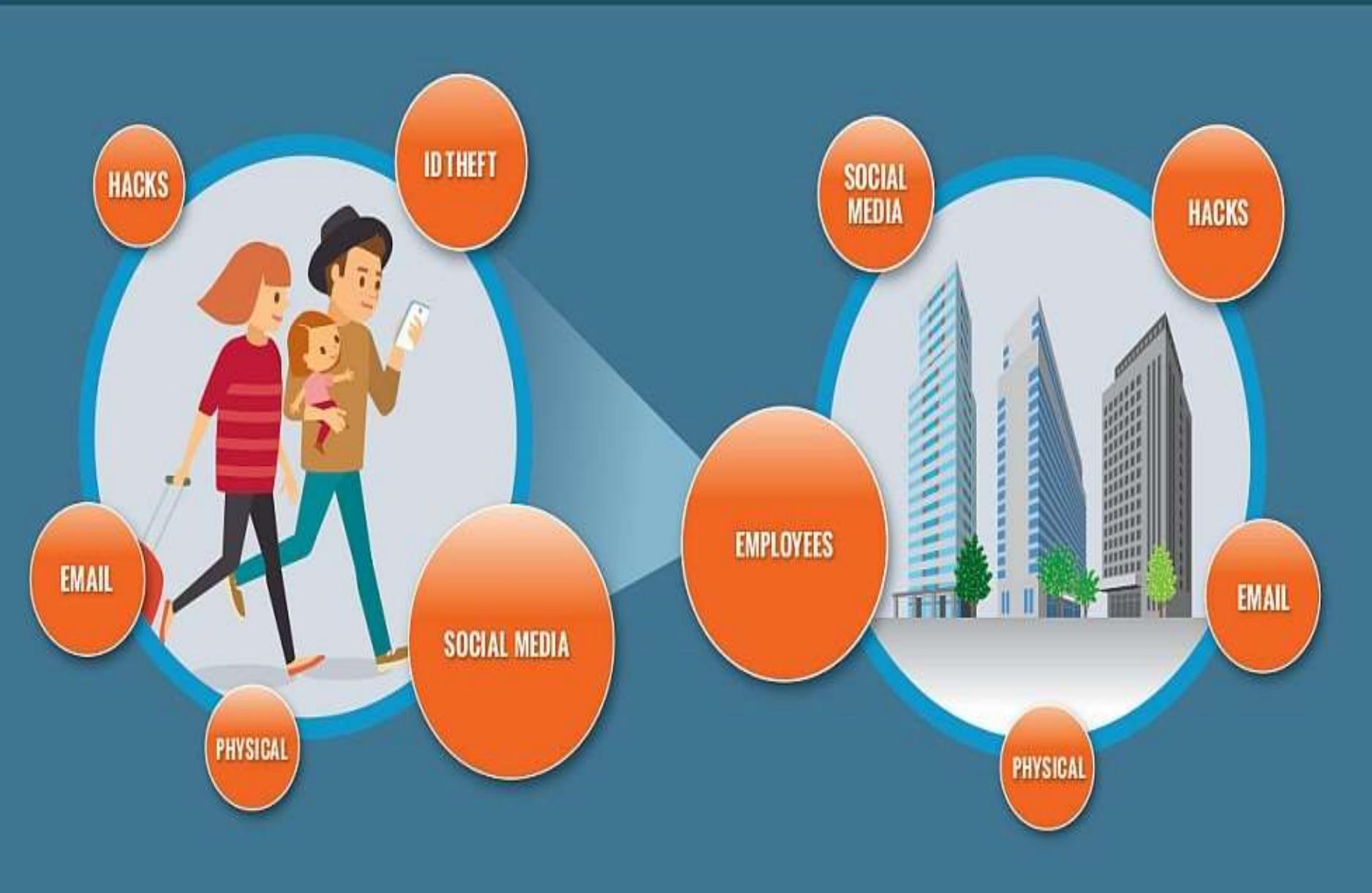
- Hardware, software and data.
  - Memory.
  - Sharable I/O devices.
    - Serially reusable I/O devices.
  - Sharable programs and sub-procedures.
  - Sharable data.

As it assumed responsibility for controlled sharing, the operating system had to protect these objects.

# User Security

- User security awareness training helps every employee in your organization recognize, avoid, and report potential threats that can compromise critical data and systems, including phishing, malware, ransomware, and spyware.

# The Human Factor - Social Engineering Risk Points



# Securing user account

- Use a long/secure password.
- Do not reuse or share passwords.
- Use Two-Factor Authentication (2FA) .
- Use a password management application.
- Check web site security.

# **Program Security**

- A security program is the **entirety of an organization's security policies, procedures, tools and controls.**
- Protection from an unauthorized access to the system.
- Strict allocation of user roles and their access to certain data.
- Protection of the stored and processed data from damage and loss

# Security Program



Awareness &  
Training



Risk  
Assessment



Managed  
Services



Policies &  
Procedures



Design &  
Implementation



Business  
Case

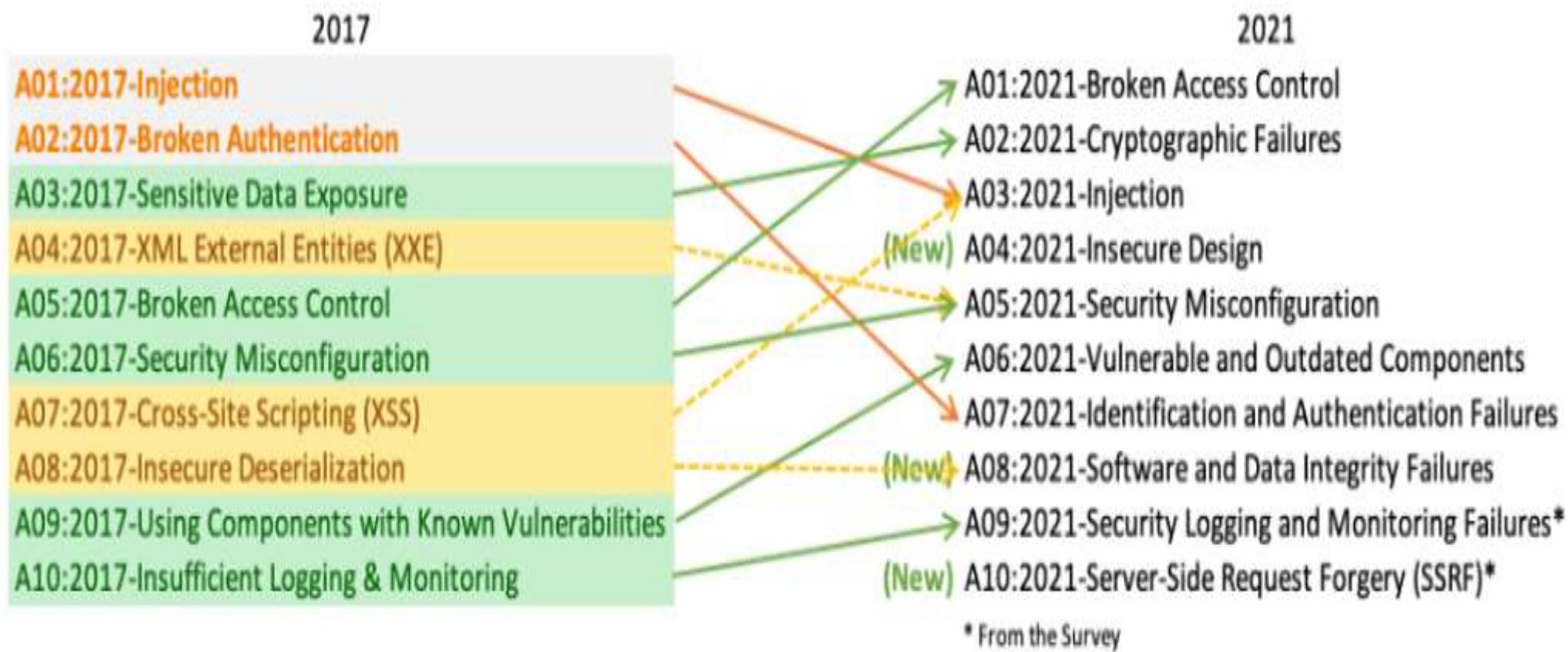
# Application Security

- Application security is the process of developing, adding, and testing security features within applications to prevent security vulnerabilities against threats such as unauthorized access and modification.
- Examples of application security features.
  - Authentication,
  - authorization,
  - encryption,
  - logging,
  - application security testing

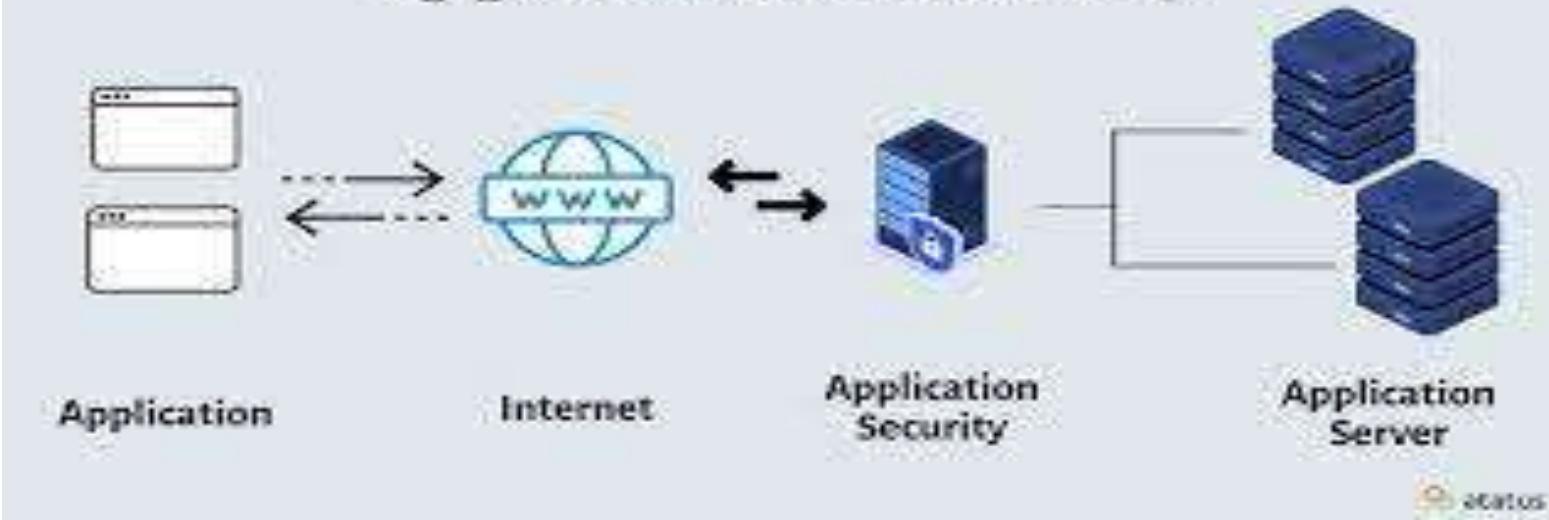
- Application Security Tools are designed to protect **software applications from external threats** throughout the entire application lifecycle.
- The purpose of this class of tools is to protect the many different kinds of application against data theft or other nefarious intent.

- Building secure applications: Top 10 application security best...
  - Follow the OWASP top ten.
  - Get an application security audit.
  - Implement proper logging.
  - Use real-time security monitoring and protection.
  - Encrypt everything.
  - Harden everything. (hardening is usually the process of securing a system by **reducing its surface of vulnerability**, which is larger when a system performs more functions; in principle a **single-function system is more secure than a multipurpose one.**)
  - Keep your servers up to date.
  - Keep your software up to date

# OWASP stands for Open Web Application Security Project.



# Application Security



atatus

# Digital Privacy

- Digital privacy (also internet or online privacy) means the protection of any data a user creates or transmits while navigating the web via a mobile or desktop.



- The concept of digital privacy can best be described as the protection of the information of private citizens who use digital mediums.
- Information such as the date and time of his searches, what browser he used to access websites and even how long he viewed websites can be retained on a search engine's servers

# **Digital Forensics**

# Global Digital Forensics Market Size and Forecast 2015-2024



# What is Forensics ?

- Collection and analysis of evidence.
- Using scientific test or techniques.
- To establish facts against crime.
- For presenting in a legal proceeding.
- Therefore Forensic science is a scientific method of gathering and examining information about the past which is then used in court of law.

# Digital Forensics

- Digital Forensics deals with the **process of finding evidence related to a digital crime.**
- It is a science of finding evidence from digital media like a computer, mobile phone, server, or network.
- It provides the forensic team with the best techniques and tools to **solve complicated digital-related cases.**

- Digital Forensics Specialists are generally consulted to investigate **cyber-crimes, crimes that involve a security breach in a system or network.**
- When a cyber-crime occurs, digital forensics specialists can assist in various ways.



# **Investigation Carried by:**

- Law Enforcement officials.
- Investigation Departments.
- Civil Litigations.
- Insurance Companies.
- Private Corporations.
- Individual/Private Citizens.



# Use-Cases of Digital Forensics

- Banking credit/debit card related crimes.
- Data Recovery.
- Financial Fraud Investigation.
- Damage control in the wake of cybercrime.
- Post Attack Identification.
- Log Analysis.
- Cyber Crime Investigation.
- Malware Analysis and so on....



# **Branches of Digital Forensics:**

- Network Forensics.
- Email Forensics
- Web Forensics.
- Software Forensics.
- Memory Forensics.
- System Forensics.
- Cloud Forensics.
- Mobile Device Forensics
- Blockchain Forensics and so on..



# **Goals of a Digital Forensics Examiner**

- As a digital forensics investigator, you should have a goal for investigation.**

**1. What is the crime and Evidence?**

**2. Where it can be found?**

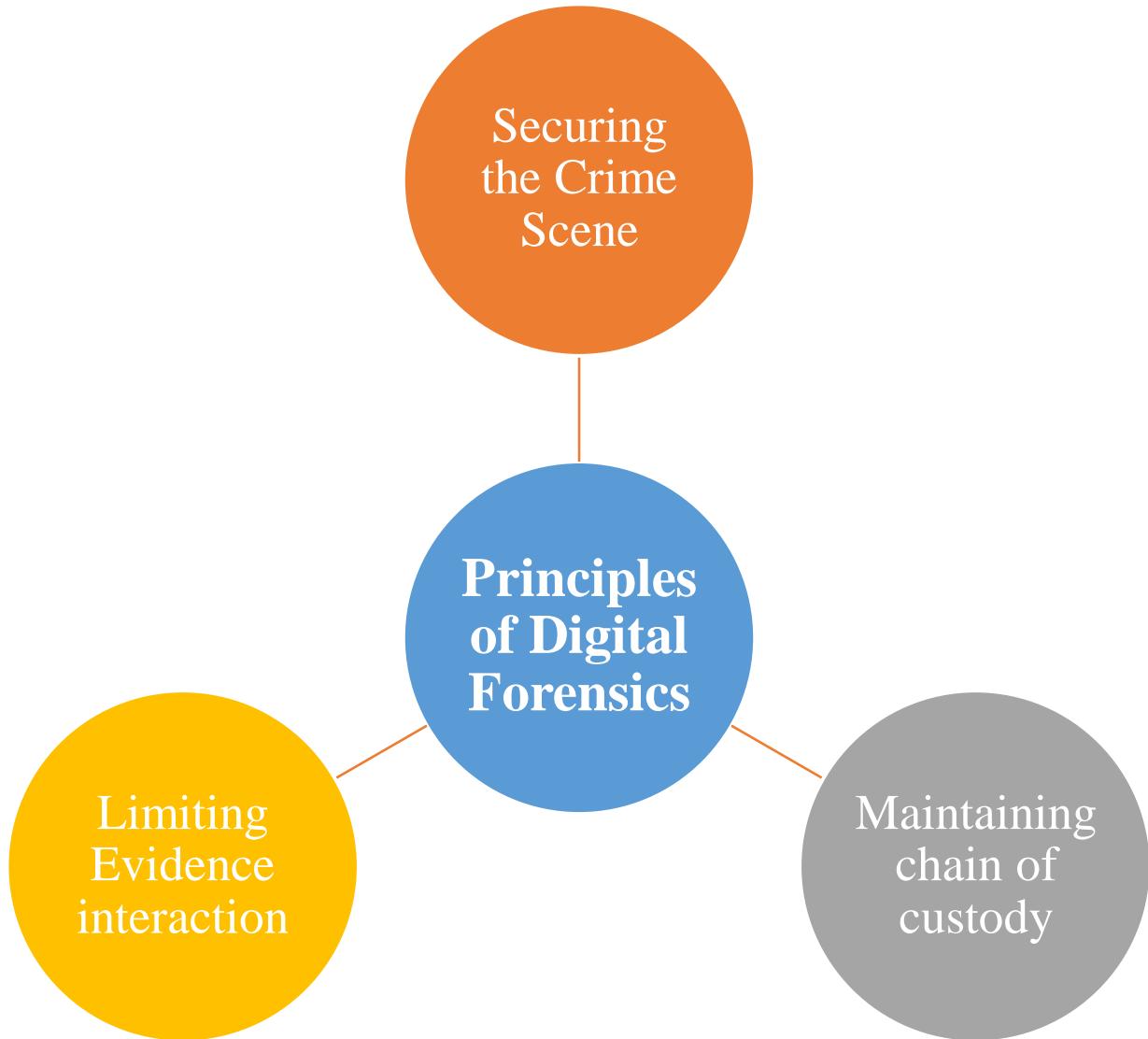
**3. When was the crime committed ?**

**4. Who is the culprit of the crime?**

**5. How was the crime committed?**



# Principles of Digital Forensics



# **Digital Forensics Scientific Process**

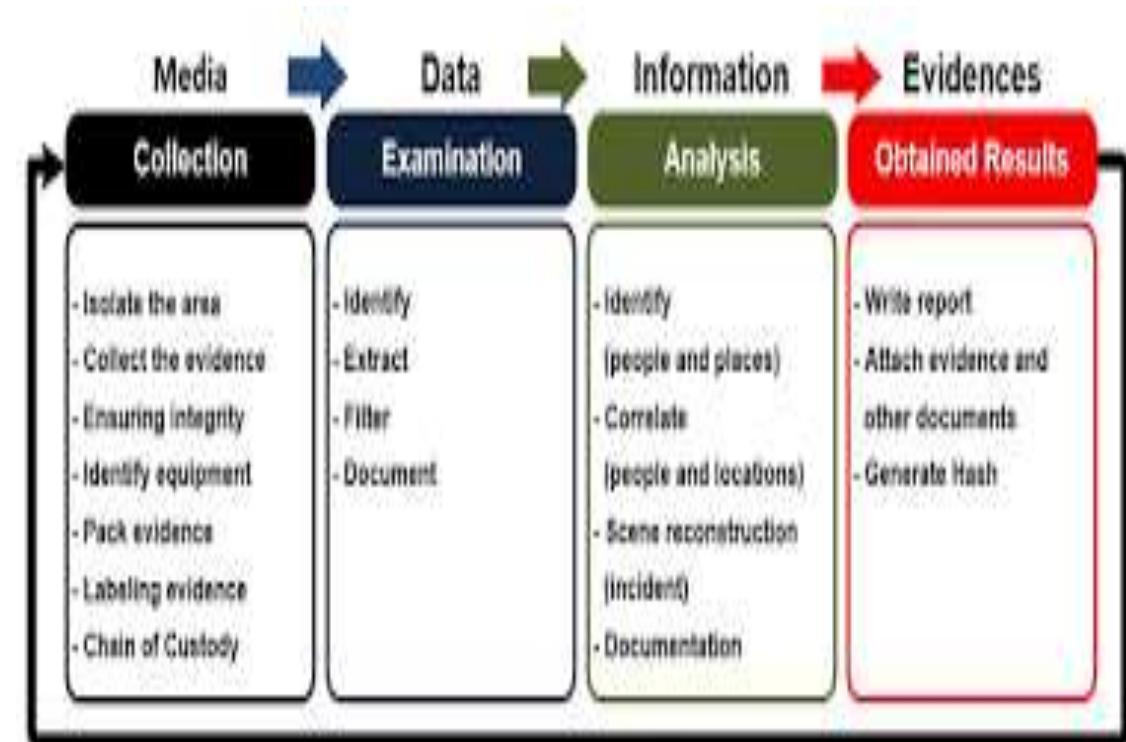
# Digital Forensics Scientific Process:

A. Evidence Identification and collection.

B. Preservation.

C. Examination and Analysis.

D. Reporting.

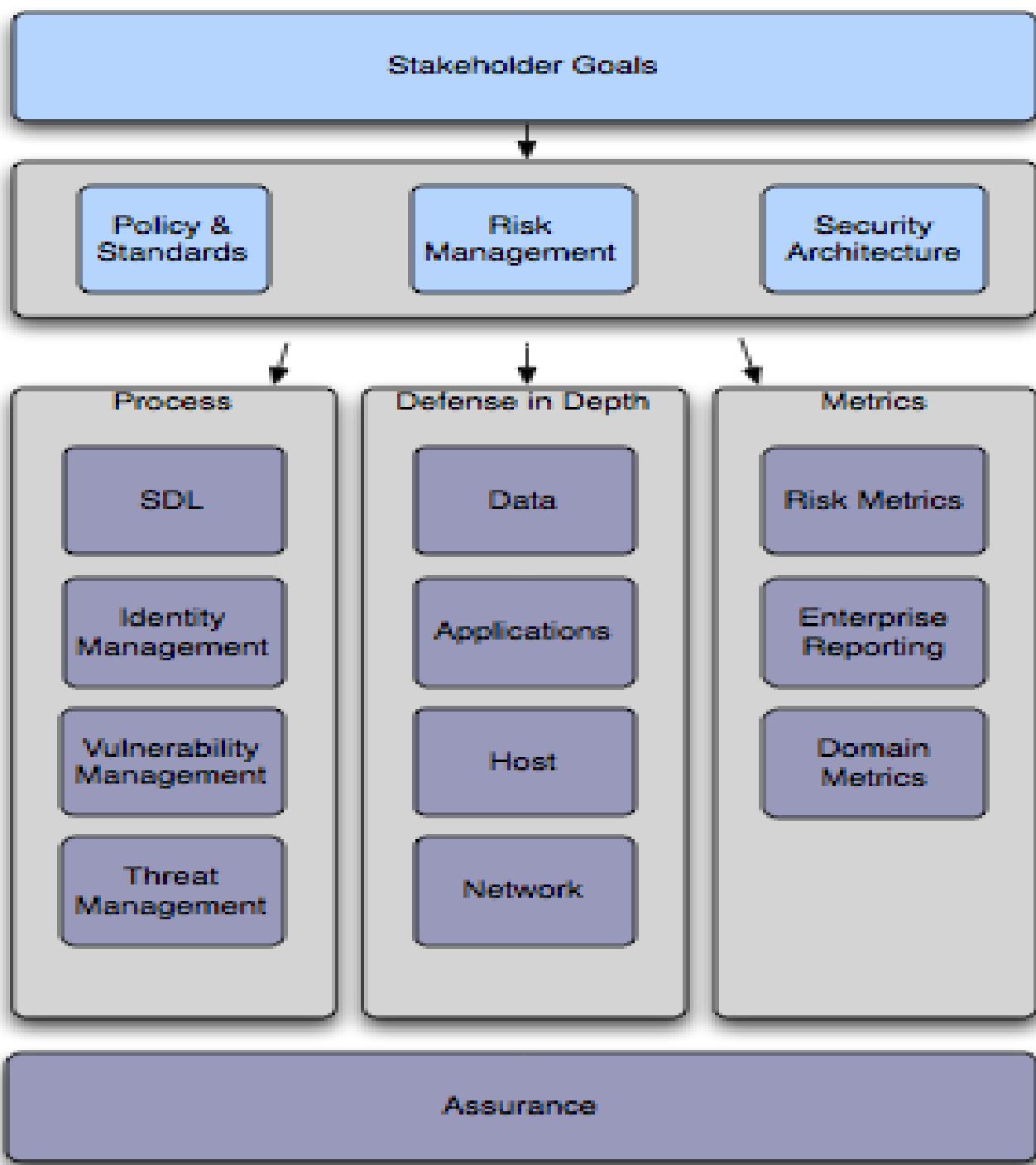


# **Enterprise security specification**

- Enterprise security specification is a specification aimed at fomenting a minimum standard of security for enterprise applications.
- Updated Security Applications (firewalls, proxies, antivirus software, etc.) Network Architecture Design (and review)  
Endpoint Controls & Analysis

# **Major Types of Enterprise CyberSecurity Tools**

- Network Firewall.
- Application Firewall.
- Anti-Virus Software (AV) .
- Network Proxy.
- Endpoint Detection and Response (EDR) .
- Vulnerability Patching.
- Intrusion Detection and Protection Systems (IDS/IPS) .
- Role-Based Access Control (RBAC)



# **NMIMS University**

**Information Security  
Unit -7**

# Unit 7:

- Operating System Security
- Security architecture
- Analysis of security in windows/linux

# **Operating system (OS)**

- An **operating system** (OS) is **system** software that manages computer hardware, software resources, and provides common services for computer programs.

# Function of operating System

- Memory Management.
- Processor Management.
- Device Management.
- File Management.
- Security.

## Functions of the operating system

Process  
Management

Memory  
Management

File  
Management

Device  
management

Security

Job Accounting

Secondary storage  
Management

Process  
Management

Networking

Cordination  
between Other  
software and Users

# Operating system security

- Operating system security (OS security) is the process of ensuring OS integrity, confidentiality and availability.
- OS security refers to specified steps or measures used to protect the OS from threats, viruses, worms, malware or remote hacker intrusions.



# Protected Objects

- Hardware, software and data.
- Memory.
- Sharable I/O devices.
- Serially reusable I/O devices.
- Sharable programs and sub-procedures.
- Sharable data.

As it assumed responsibility for controlled sharing, the operating system had to protect these objects.

# Security Methods

## ❖ Seperation:

- keeping one user's objects separate from other users'
- Physical Separation
- Temporal Separation
- Logical Separation
- Cryptographic Separation

## ❖ Granularity of Control

- The larger the level of the object controlled, the easier it is to implement access control.

# **Memory address protection**

- An operating system is the multiprogramming system allowing multiple users to use concurrently.
- Operating system is designed in such a way that one user's computation cannot be intercepted by malicious user.
- For this purpose, operating system has following facilities.
  1. Memory protection,
  2. File protection,
  3. General control on how objects are accessed,
  4. User authentication

# Memory and Address Protection:

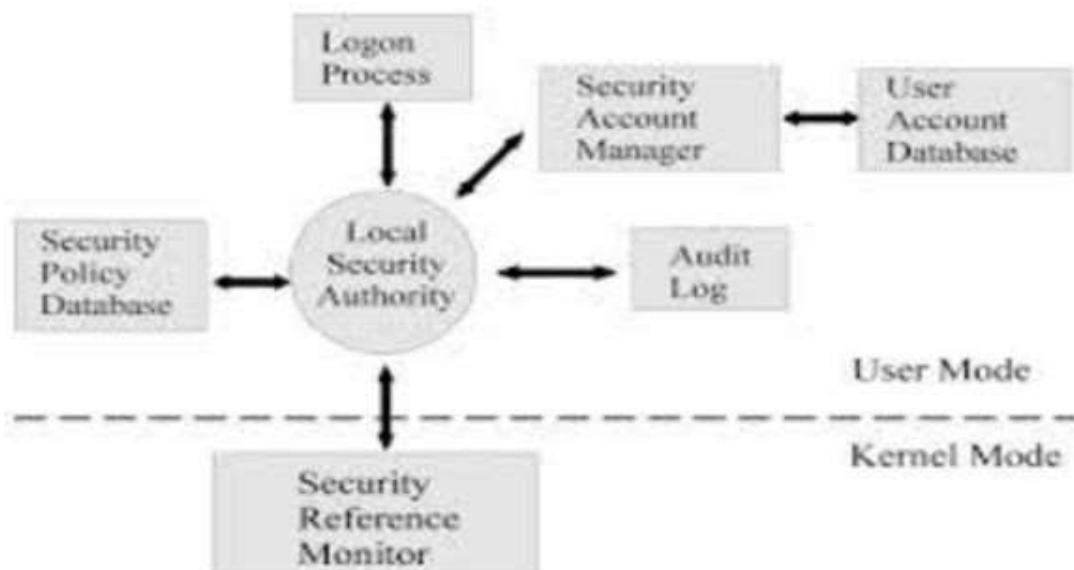
- Memory protection in multiprogramming prevents other programs from interfering to user's program.
- Hardware is designed to provide memory protection.
  1. Fence
  2. Relocation
  3. Segmentation
  4. Paging

# Relocation

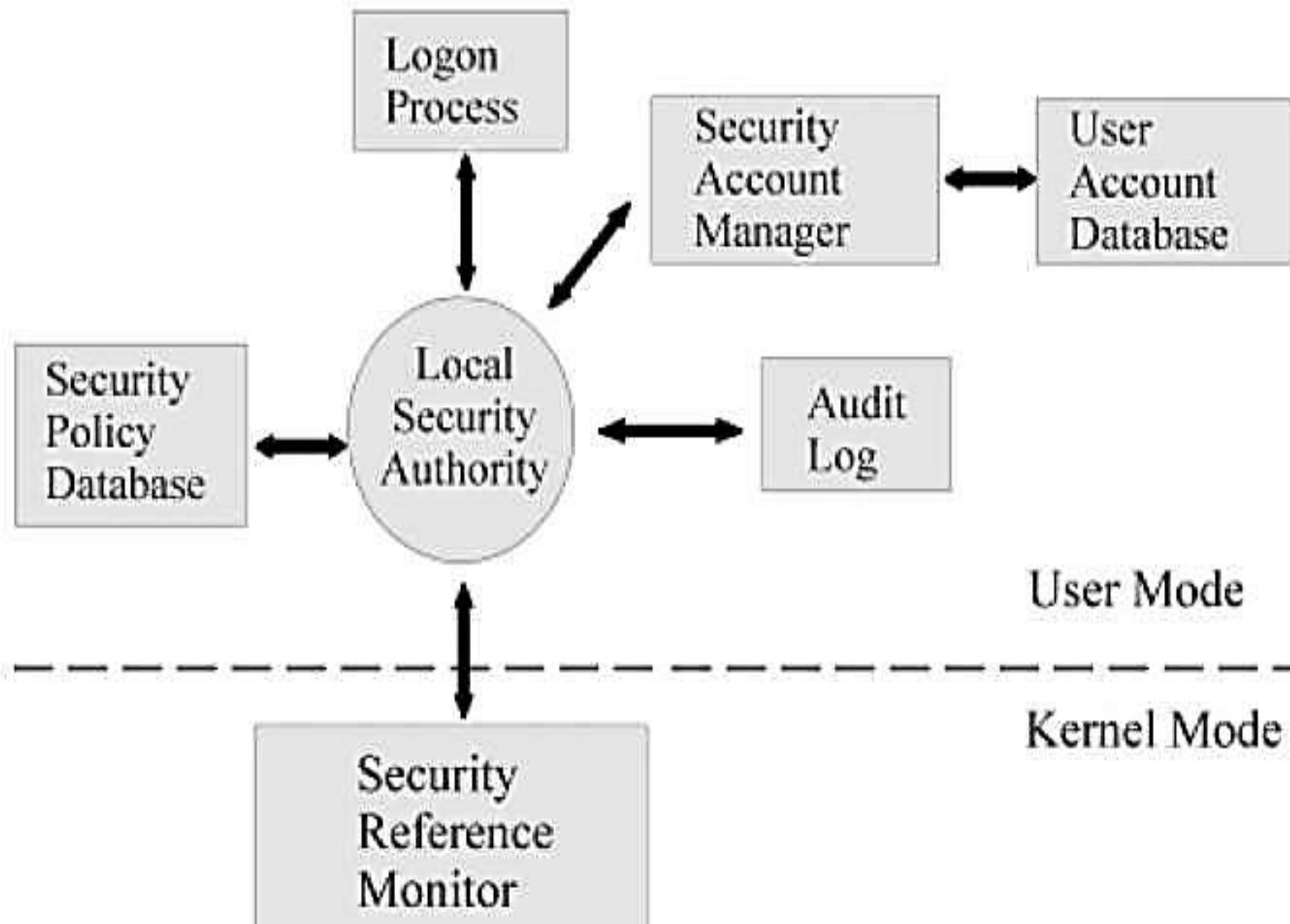
- *Relocatability - the ability to move process around in memory without it affecting its execution*
- Operating systems can manage where each process is stored in memory using a technique called **relocation**.
- With **static relocation**, the process must be put in the same position.
- With **dynamic relocation**, the OS finds a new position in memory for the process and updates the **relocation** and limit registers.

# Security Architecture of Windows

- Security Reference Monitor(SRM)
- Local Security Authority(LSA)
- Security Account Manager(SAM)



- The Security Architecture of the OSI Reference Model (ISO 7498-2) considers five main classes of security services:
  - a) Authentication,
  - b) Access control,
  - c) Confidentiality,
  - d) Integrity
  - e) Non-repudiation.



- The OSI security architecture **focuses on security attacks, mechanisms, and services.**
- These can be defined briefly as follows: Threats and Attacks (RFC 2828) Threat.
  - A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm

# Security in Linux

- The Linux kernel boasts an array of built-in security defenses including **firewalls** that use packet filters in the kernel, the UEFI Secure Boot firmware verification mechanism, the Linux Kernel Lockdown configuration option and the SELinux or AppArmor Mandatory Access Control (MAC) security enhancement systems

# Levels of security in Linux

- 3 levels of security in Linux
- For each level of access control (user, group, other), the 3 bits correspond to three permission types.
- For regular files, these 3 bits control **read access, write access, and execute permission**.
- For directories and other file types, the 3 bits have slightly different interpretations.

- These levels are user, group and others.
- Each permission level has three types of permission; **read, write and execute**.

# How to secure Linux server

1. Only install required packages.
2. Disable the root login.
3. Configure 2FA.
4. Enforce good password hygiene.
5. Server-side antivirus software.
6. Update regularly or automatically.
7. Enable a firewall.

# **NMIMS University**

## **Information Security**

### **Unit -8**

# Unit 8:

- Security requirements.
- Reliability and integrity.
- Sensitive Data.
- Inference.
- Multilevel database.
- Proposal for multilevel security.

# Database

- A DBMS plays a crucial **role** in both the creation and management of data.
- Without a **database** management system, running and managing data effectively is not possible.
- Serving as the intermediary between the user and the **database**, a DBMS provides users access to files stored in a **database**

# Types of Database

- Centralised database.
- **Distributed database.**
- Personal database.
- **NoSQL database.**
- Operational database.
- **Relational database.**

- **Database security** can guard against a compromise of your **database**, which can lead to financial loss, reputation damage, consumer confidence disintegration, brand erosion, and non-compliance of government and industry regulation.



- **Database security** refers to the collective measures used to protect and **secure** a **database** or **database management** software from illegitimate **use** and malicious threats and attacks.
- It is a broad term that includes a multitude of processes, tools and methodologies that ensure **security** within a **database** environment.

# Security Requirements

- Database security best practices that can help keep your databases safe from attackers:
  - Ensure physical database security.
  - Use web application and database firewalls.
  - Harden database to the fullest extent possible.
  - Encrypting data.
  - Minimize value of databases.
  - Manage database access tightly.
  - Audit and monitor database activity.

# Types of Database security

- Many layers and types of information security control are appropriate to databases, including:
  - Access control.
  - Auditing.
  - Authentication.
  - Encryption.
  - Integrity controls.
  - Backups.
  - Application security.
  - Database Security applying Statistical Method.

# **Reliability and integrity**

- **Database reliability** is defined broadly to mean that the **database** performs consistently without causing problems.
- **Reliability** - the ability of software and hardware to work without failure
- **More specifically, it means that there is accuracy and consistency of data.**

- **Integrity** - how 'correct' data within a system is.  
While errors in data may seem minor, their impacts can be significant when major decisions are based upon them.

# Reliability and Integrity

## Reliability :

database guards against loss or damage.

Database concerns about reliability and integrity can be viewed from three dimensions:

1. Database integrity: whole database is protected against damage (e.g. disk failure, corruption of data)
2. Element integrity: specific data value is changed by authorized users.
3. Element accuracy: only correct values are written into the elements of database.

# Maintaining Reliability

- In order to achieve data integrity, all data types and properties must be defined correctly according to business rules and should have proper relationships between data entities.
- There is also need for **error checking** and **validation** function to ensure that only valid data types are stored in a defined field.

# **Database Integrity**

- **Database Integrity:** It concern that the database as a whole is **protected against damage**, as from the failure of a disk drive or the corruption of the master database index.
- The **data integrity** refers to the overall **completeness, accuracy and consistency of data**.
- This concerns are addressed by OS (Operating System) integrity controls and recovery procedures.

- **Element Integrity:** concern that the value of a specific data element is written or changed only by authorized users.
- **Element Accuracy:** concern that only correct values are written into the elements of a database.
- Checks on the values of elements can help prevent insertion of improper values.

# Sensitive Data

- **Sensitive data** is information that must be protected against unauthorized access.
- Access to **sensitive data** should be limited through sufficient **data** security and information security practices designed to prevent unauthorized disclosure and **data** breaches.

# **Example of sensitive data**

- **Customer Information.** Customer information is what many people think of first when they consider **sensitive data**.
- This could include customer names, home addresses, payment card information, social security numbers, emails, application attributes, and more

COMPANY

FINANCIAL

CONFIDENTIAL

Data



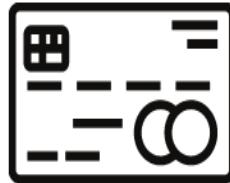
- The **three main types of sensitive** information that exist are:
  1. personal information,
  2. business information
  3. classified information.



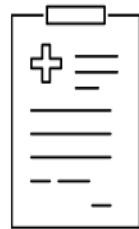
GEOGRAPHIC LOCATION



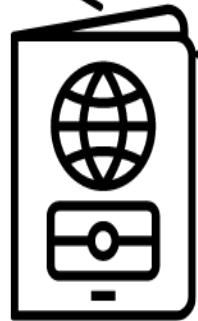
SOCIAL SECURITY NUMBER



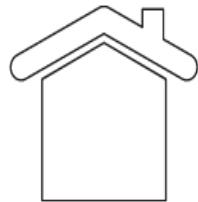
CREDIT CARD NUMBER



MEDICAL RECORDS



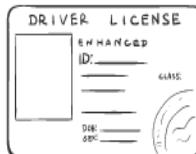
PASSPORT NUMBER



ADDRESS INFORMATION



TELEPHONE NUMBER



DRIVER'S LICENSE  
NUMBER



BIOMETRIC DATA

# Inference Attack

- An Inference Attack is a data mining technique performed by analyzing data in order to illegitimately gain knowledge about a subject or database.
- A subject's sensitive information can be considered as leaked if an adversary can infer its real value with a high confidence.

- A **threat to database security** is the misuses of these databases by the authorized users, for example selling the personal information to outsiders.
- An inference occurs when a user uses legitimate data to infer information without directly accessing it.

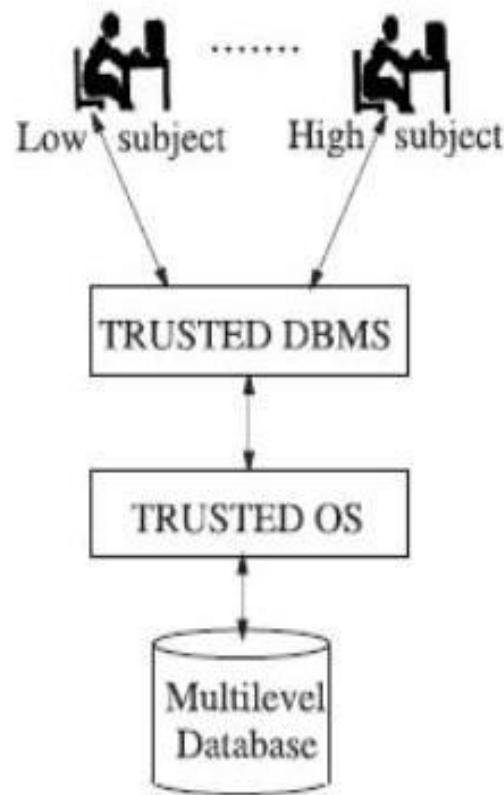
- Inference is a database system technique used to attack databases where malicious users infer sensitive information from complex databases at a high level.
- The more complex the database is, the greater the security implemented in association with it should be.

- The top ten most common database security vulnerabilities
- Deployment Failures. The most common cause of **database** vulnerabilities is a lack of due care at **the** moment they are deployed. ...
- Broken **databases**. ...
- Data leaks. ...
- Stolen **database** backups. ...
- The abuse of **database** features. ...
- A lack of segregation. ...
- Hopscotch. ...
- SQL injections.

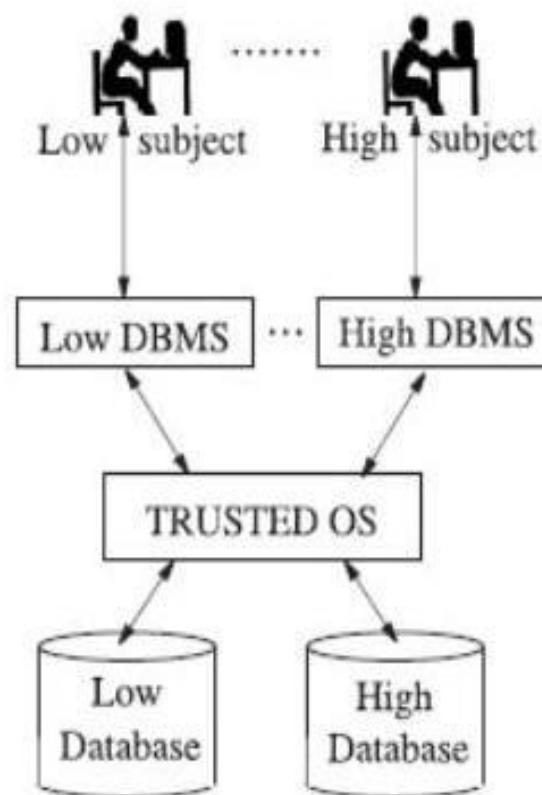
# Multilevel Database

- A **multilevel database** system (MDBMS) supports the application of a **multilevel** policy for regulating access to the **database** objects.
- **Multilevel Databases.** So far, we have considered data in only two categories: either sensitive or non sensitive.

- Trusted subject. The DBMS itself must be trusted to ensure mandatory policy
- Trusted Computing Base: Data are partitioned in different databases, one for each level



(a) Trusted subject



(b) Trusted computing base

# Multilevel Database Security

- **Multilevel security** is a **security** policy that allows you to classify objects and users based on a system of hierarchical **security** levels and a system of non-hierarchical **security** categories. ...
- **Multilevel security** does not rely on special views or **database** variables to provide row-level **security** control.

# Proposals for Multilevel Security

## ◆ Separation

- Partitioning – divide DB into separate DBs with own level of sensitivity
- Encryption (time consuming)
- Integrity Lock – each data item contains a sensitivity label and a checksum
  - ◆ Sensitivity label must be *unforgeable, unique, concealed*
  - ◆ Checksum must be unique
  - ◆ Sensitivity lock