CSI354 – OPERATING SYSTEMS

CHAPTER 5: PROTECTION AND SECURITY

We will look at :

- **Protection**
  - goals, principles, domain of access, access matrix, access control
- **Security**
  - program threads, system and network threads, user authentication, implementing security defenses, firewalling

5.1. PROTECTION

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Access Control

5.1.1 Goals of Protection

- OS contains many ***objects*** (hardware or software) which need to be protected
- each object
  - has a unique name
  - can be accessed through a well-defined set of operations

➢ ***protection problem*** - ensure that each object is **accessed correctly** and only by those processes that are allowed to do so

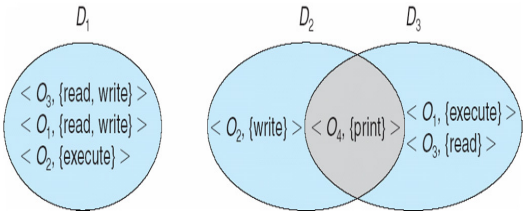
- need a way to
  - prohibit processes from accessing objects that they are not allowed to access
  - restrict processes to a set of legal operations

5.1.2 Principles of Protection

- Guiding principle for protection – principle of least privilege
  - dictates that programs, users and systems should be given just enough privileges to perform their tasks
  - helps produce a more secure computing environment

5.1.3 Domain Structure

- Protection domain
  - specifies *resources* that the process may access
  - collection of *access rights* usually corresponding to a single user
- Access-right = <*object-name*, *rights-set*>
  - each pair specifies an object and a set of operations that can be performed on it
  - where *rights-set* is a subset of all valid operations that can be performed on the object
  - association between process and domain may be
    - **static** – set of resources are fixed through the lifetime of the process
    - **dynamic** – resources change, therefore a process can switch from one domain to another



- e.g. UNIX
- system consists of two domains: user and supervisor
  - Domain = *user-id*; each file has a user id
  - Domain switch accomplished via file system
    - each file has associated with it a domain bit (*setuid* bit) and a *user-id*
    - when file is executed and *setuid* = on, then *user-id* is set to owner of the file being executed
    - if *setuid* is off, then the *user-id* does not change

5.1.4 Access Matrix

- view protection as a matrix (*access matrix*)
- rows represent **domains**
- columns represent **objects**
- *Access(i, j)* is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- **Use of Access matrix**
- if a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- can be expanded to dynamic protection
  - operations to add or delete access rights
  - special access rights:
    - ***owner*** *of* O_i, *controls rights of that object*
    - ***copy*** *op from* O_i *to* O_j, *denoted by* *
    - ***control*** – D_i *can modify* D_j *access rights*
    - ***transfer*** – *switch from domain* D_i *to* D_j

Example

Assume there are 4 objects : F1, F2, F3 and Printer, and four domains D1 to D4

- A user in D1 is allowed to read files F1 and F3
- F2 can be read in D3
- A user can execute F3 in D3
- A process in D4 can read/write both F1 and F3
- The printer can only be used by a process executing in Domain D2

Draw an access matrix

Now, assume that

- A process executing in D1 can transfer to D3
- A process in D2 can transfer to D3 or D4
- A process in D4 can switch to D1
- A process in D2 can read F2, and is allowed to copy this operation
- A process in D1 is allowed to copy the read operation on F3

- Also assume that D1 is the owner of F1 and D2 is the owner of F2 and F3
- A process executing in D2 is allowed to modify the access rights in D4

Access Matrix With Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix with *Copy* Rights

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

A process in D2 can copy the read operation to any entry associated with file F2

Access Matrix With *Owner* Rights

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

D1 is the owner of F1, therefore can delete/add any rights in column F1

D2 is owner of F2 and F3, and can add/delete rights in columns F2 and F3

Modified Access Matrix of Figure B

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Control : A process in D2 can modify access rights for a process in D4

- access matrix can be very large, but sparse
- storing the whole thing is rarely done
- two practical methods :

- Each column = **Access-control list** for one object
  - ➢ defines who can perform what operation on the object

e.g. Object 1: Domain 1 = Read, Write
Domain 2 = Read
Domain 3 = Read
⋮

- Each row = **Capability List**
  - ➢ for each domain, what operations allowed on what objects

e.g. Domain 1 : Object 1 – Read
Object 4 – Read, Write, Execute
Object 5 – Read, Write, Delete, Copy

Example

Given the following information, draw an access matrix

- There are three files named F1, F2 and F3, a printer, and four domains D1 to D4
- A process in D1 can read and write file F1 and can switch to domain D3. Domain D1 is also the owner of F1
- A process in D2 can read F2, access the printer, and switch to D3 or D4. It can also modify domain D4
- A process in D3 can read F2 and execute F3
- In D4, a process can read/write F1 and F3 and can switch to D1. The process is also allowed to copy the right to read F3
- Give the access control lists for each object and capability lists for each domain in the matrix

5.2 SECURITY

- Security Problem
- System Threats
- User Authentication
- Implementing Security Defenses
- Firewalling to Protect Systems and Networks
- Overview of cryptography

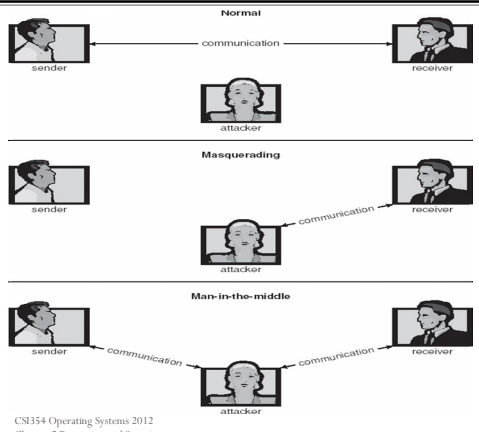
5.2.1 Security Problem

- security must consider external environment of the system, and protect the system resources
- *intruders* (crackers) attempt to breach security
- *threat* is potential security violation
- *attack* is attempt to breach security
  - accidental or malicious
- easier to protect against accidental than malicious misuse
- some security goals
  - data confidentiality – secret data remains secret
  - data integrity – no tampering of data
  - system availability – system always usable
  - privacy – protect misuse of user info

5.2.2 Security Violations

- categories
  - breach of confidentiality
    - unauthorized reading of secret data
  - breach of integrity
    - unauthorized modification of data
  - breach of availability
    - destruction of data
  - theft of service
    - unauthorized use of resources
  - denial of service
    - preventing legitimate usage of system

- methods
  - masquerading (breach authentication)
    - pretending to be someone else in order to gain access
  - replay attack
    - message modification to repeat valid data transmission
  - (wo)man-in-the-middle attack
    - attacker sits in data flow communication masquerading as sender or receiver
  - session hijacking
    - intercepting communication



5.2.3 Security Measure Levels

- security must occur at four levels to be effective:
  - **Physical** – site physically secured against intruders
  - **Human** – only legitimate users access the system
    - avoid **social engineering, phishing, dumpster diving**
  - **Operating System** – system protects itself against malicious processes, queries
  - **Network** – protect data travelling over the network from being intercepted

➢ security is as good as the weakest chain

- all the above aspects need to be addressed for security to be maintained

5.2.4 User Authentication

- major security problem
- establish the identity of user/machine by
  - something you know (password, secret)
  - something you have (credit card, smart card)
  - something you are (retinal scan, fingerprint)
- in the case of an OS this is done during login

- two factor authentication – use two forms of user verification
- multifactor – use multiple forms

- **Passwords**
- most widely used form of authentication
- secret known only to the subject
- can be generated by the system or selected by the user
- usually only one required
- simplest OS implementation keeps (login, password) pair
- easy to understand and use
- authenticates user on login by checking the password
- require user to change their passwords regularly
  - the extreme is the *one time password*
- variation is the *challenge-response* scheme
- they could be guessed, exposed accidentally, sniffed, or shoulder surfed

- **Physical identification**
- check to see if the user has some item
  - usually plastic card with magnetic strip
  - inserted into the reader
  - can be combined with password (two factor id)
- physical characteristics
  - finger print, voice print, finger length, signature analysis

5.2.5. System Threats

- **Viruses**
  - code fragment embedded in legitimate program
  - very specific to CPU architecture, operating system, applications
  - usually borne via email or as a macro
  - *virus dropper* inserts virus onto the system
  - reproduces itself
  - but require human intervention to spread
  - can be used to cause *denial of service*

- many categories of viruses, literally many thousands of viruses
  - program file
  - boot sector
  - macro
  - source code
  - polymorphic
  - encrypted
  - stealth
  - tunneling
  - multipartite
  - armored

- **Worms**
  - similar to a virus
  - use *spawn* mechanism to replicate without a helper
  - standalone program
  - use networks to transmit copies of itself to other computers
  - not necessarily destructive

e.g Internet worm (1988)
  - exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs
  - *Grappling hook* program uploaded main worm program

- **Port scanning**
  - not really an attack
  - automated attempt to connect to a range of ports on one or a range of IP addresses
- **Denial of Service**
  - overload the targeted computer preventing it from doing any useful work
  - distributed denial-of-service (DDOS) come from multiple sites at once

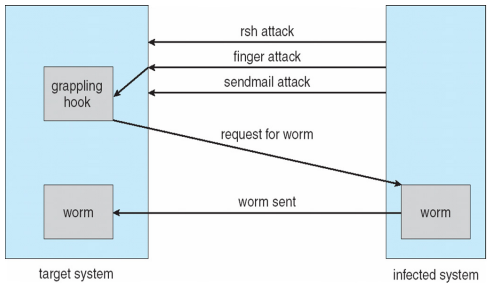
- **Trojan horse**
  - malicious program disguised as an innocent one
  - could modify/delete user's file, send important info to cracker, etc
  - cracker hides it as a new game, e-card, windows update site, etc.
  - when run, Trojan Horse executes with user's privileges
  - examples:
    - hide program in path directory as a common typo: *la* for *ls*
    - malicious user puts malicious *ls* in directory, and attracts superuser
      - ➢ malicious *ls* could make user the superuser

- **Login spoofing**
  - specialized case of Trojan Horse
  - attacker displays a custom screen that user thinks belongs to the system
    - ➢ user responds by typing in user name and password

- **Login bombs**
  - piece of code, in the OS or app
  - dormant until a certain time has elapsed or event has occurred
    - ➢ e.g. missing employee record from payroll
  - could act as a Trojan Horse/virus once triggered
  - also called *slag code* or *time bomb*
- **Trap doors**
  - code in system inserted by programmer to bypass normal check
  - Ken Thompson "Reflections on Trusting Trust"
    - hole in UNIX system utility; enforced by C compiler

- **Accidental Data Loss**
  - acts of God
  - hardware or software errors
  - human errors

The Morris Internet Worm



5.2.6. Implementing Security Defenses

- **Defense in Depth Theory**
  - most common security theory
  - multiple layers of security are better than fewer layers
- first step is to create a **security policy** which describes what is being secured
- then a **vulnerability assessment** is used to compare real state of system to security policy, therefore initiating appropriate responses

- **intrusion detection** then endeavors to detect attempted or successful intrusions
  - **signature-based** detection spots known bad patterns
  - check for specific behavior that are known to indicate attacks
  - **anomaly detection** spots differences from normal behavior
  - can help detect previously unknown methods of behavior
  - **false-positives** and **false-negatives** a problem
  - false alarms and missed intrusions
- **virus protection** – using antivirus programs
- **auditing**, **accounting**, and **logging** of all or specific system or network activities

5.2.7 Firewalling

- a network ***firewall*** is placed between trusted and untrusted hosts
  - firewall limits network access between these two security domains
  - blocks unauthorized access, only permitting authorized communication
  - also limits connectivity based on the source/destination address
  - monitors and log all connections
  - can be hardware, software, or both
- mechanisms used include
  - packet filtering
  - proxy server to hide network addresses and intercept all messages entering and leaving the system

- main problem is that they can be tunneled or spoofed
  - **tunneling**
  - an attack travels within allowed protocols or connections
  - allows disallowed protocol to travel within allowed protocol (i.e. telnet inside of HTTP)
  - **spoofing**
  - an unauthorized host pretends to be authorized by meeting some authorization criteria
  - firewall rules typically based on host name or IP address which can be spoofed

- **personal firewall** is software layer on given host
  - can monitor/limit traffic to and from a particular host
  - either included within the OS or added as an application
- **application proxy firewall**
  - applies security mechanism to specific application
  - understands application protocols and can control them (e.g. SMTP)
- **system-call firewall**
  - monitors all important system calls and apply rules to them (i.e. this program can execute that system call)

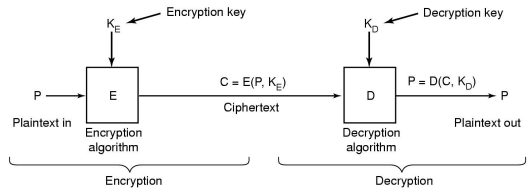
5.2.8. Design Principles

- identified by Saltzer and Schroeder (1975), used as a guide to design secure systems
- system design should be public
- default should be no access
- check for current authority
- give each process the least privilege possible
- protection mechanism should be simple, uniform, and built into the lowest layers of the system
- scheme chosen must be psychologically acceptable

5.2.9. Overview of Cryptography

- ***encrypt*** data so it only makes sense to authorized users
  - needed when communicating over untrusted medium to protect data from theft
  - data is written in secret code
  - also used for user authentication
- input data is a message or file called ***plaintext***
- encrypted data is called ***ciphertext***
- sender
  - encrypts plain text using an encryption key and an algorithm to create ciphertext
  - then send the message over the network

- receiver
  - gets the ciphertext
  - decrypts it using the decryption key and algorithm
  - this produces the original plaintext



i. Secret-key cryptography

- also called *symmetric cryptography*
  - encryption algorithm is publicly known
  - E(message, key) = ciphertext
  - D(ciphertext, key) = message
- uses a single key for both encryption and decryption
  - ➢ key must be known to only sender and receiver
- extremely fast
- suitable for large streams of data
- presumes the two parties have agreed on a key
- biggest challenge is how to distribute the key
- there is also the problem of key management

- another problem is if the two parties wanting to communicate do not trust each other, therefore reluctant to exchange keys
- several solutions
  - an older method is using a third trusted party (TTP) can be used to generate a key and send it to both
  - or the parties can use public key encryption, which allows the two parties to each generate a pair of keys and send exchange them over an unsecure network

ii. Public -key cryptography

- Diffie and Hellman, 1976
- also called *asymmetric cryptography*
- all users get a public key and a private key
  - public key is made available to everyone
  - private key is not known to anyone else, just owner
- uses public key for encryption
- private key used for decryption
- private key linked mathematically to public key
  - difficult to derive by making it computationally infeasible

- cons: slower, useful for transmitting very small amounts of data
- pros:
  - more security as there are more range of public key values
  - convenient as public key is easy to distribute as it does not have to be secured
  - used to create digital signatures