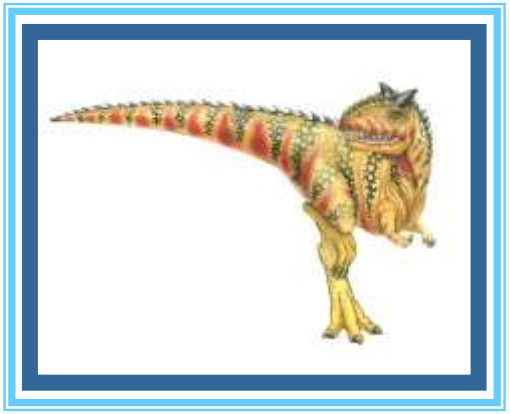
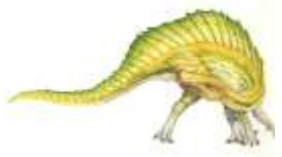
Protection and Security



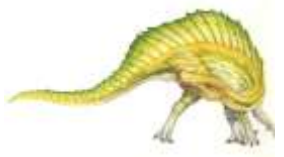
Goals of Protection

- In one protection model, computer consists of a collections of objects, hardware or software
- Each object has a unique name and 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



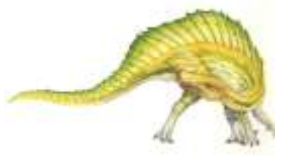
Security Violation Categories

- **Breach of Confidentiality**
  - Unauthorized reading of data
- **Breach of Integrity**
  - Unauthorized modification of data
- **Breach of Availability**
  - Unauthorized destruction of data
- **Theft of Service**
  - Unauthorized use of resources
- **Denial of Service (DoS)**
  - Prevention of legitimate use



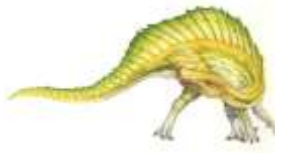
Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Limits damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
  - "Need to know" a similar concept regarding access to data



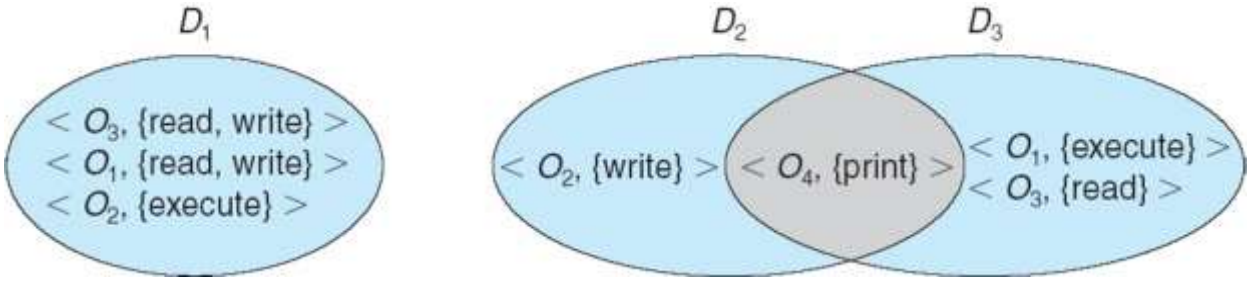
Principles of Protection (Cont.)

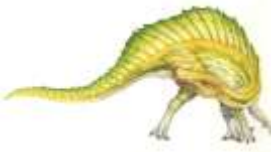
- Must consider "grain" aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists (An ACL is a list of permissions that are associated with a directory or file), RBAC (Role based Access Control)
- Domain can be user, process, procedure



Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights

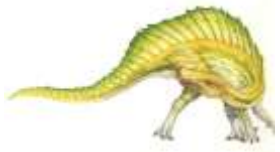




Access Matrix

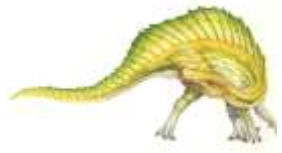
- View protection as a matrix (**access matrix**)
- Rows represent domains
- Columns represent objects
- **Access(i, j)** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

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



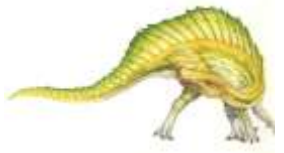
Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *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
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object



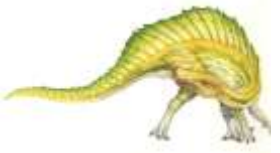
Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
- But doesn't solve the general confinement problem



Access Matrix of Figure A 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			



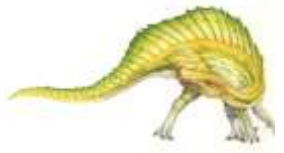
Access Matrix with *Copy* Rights

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)



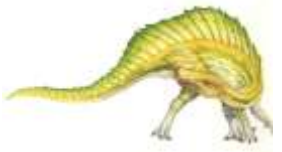
Access Matrix With *Owner* Rights

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

(a)

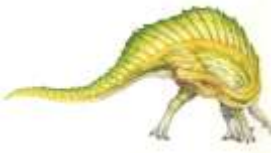
object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)



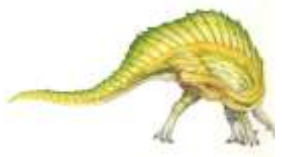
Modified Access Matrix of Figure B

object / domain	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			



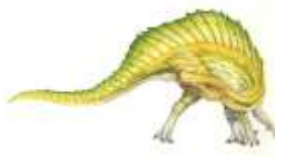
The Security Problem

- System secure if resources used and accessed as intended under all circumstances
  - Unachievable
- Intruders (crackers) attempt to breach security
- Threat is potential security violation
- Attack is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

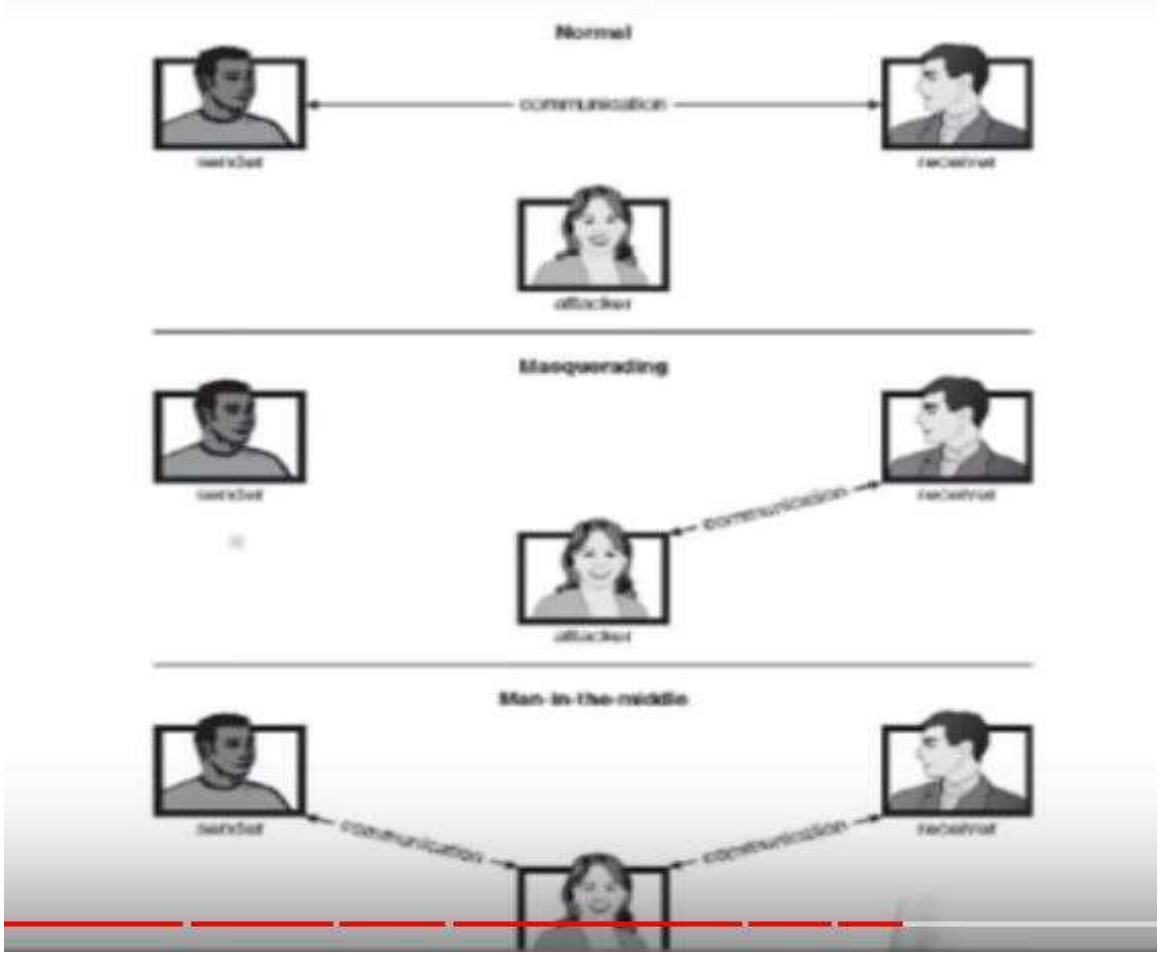


Security Violation Methods

- Masquerading (breach authentication)
  - Pretending to be an authorized user to escalate privileges
- Replay attack
  - As is or with message modification
- Man-in-the-middle attack
  - Intruder sits in data flow, masquerading as sender to receive and vice versa
- Session hijacking
  - Intercept an already-established session to bypass authentication

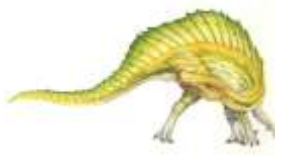


Standard Security Attacks



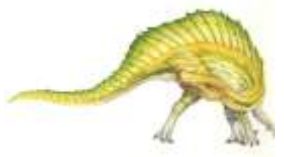
Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - Physical
    - Data centers, servers, connected terminals
  - Human
    - Avoid social engineering, phishing, dumpster diving
  - Operating system
    - Protection mechanisms, debugging
  - Network
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But, can be too much security a problem?



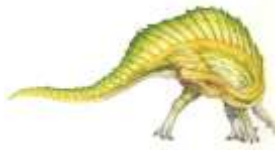
Program Threats

- Many variations, many names
- Trojan Horse – it is not a virus. It is a destructive program that looks as a genuine application. Unlike viruses, Trojan horses do not replicate themselves but they can be just as destructive. Trojans also open a backdoor entry to your computer which gives malicious users/ programs access to your system, allowing confidential and personal information to be theft.
- Spyware, pop-up browser windows, covert channels
  - Upto 80% of spam delivered by spyware-infected systems
- Trap Door
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?



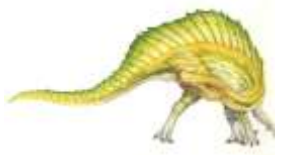
Program Threats (Cont.)

- Viruses
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications
    - A computer virus attaches itself to a program or a file so it can spread from one computer to another, leaving infections as it travels. Much like human viruses, computer viruses can range in severity: some viruses cause only mildly annoying effects while others can damage your hardware, software or files. Almost all viruses are attached to an executable file, which means the virus may exist on your computer but it can not affect your computer unless you run or open the malicious program. Virus can not spread without the human action, (such as running an infected program) to keep it going. People continue the spread of a computer virus, mostly unknowingly, by sharing infecting files or sending emails with viruses as attachments in the email.



WORM

- A WORM is similar to a virus by its design, and is considered to be a sub-class of a virus. Worms spread from computer to computer, but unlike a virus, it has the capability to travel without any help from a person. A worm takes advantage of a file or information transport features on your system, which allows it to travel unaided. The biggest danger with a worm is its capability to replicate itself on your system, so rather than your computer sending out a single worm, it could send out hundreds or thousands of copies of itself, creating a huge devastating effect. One example would be for a worm to send a copy of itself to everyone listed in your e-mail address book.



End of Chapter