University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

Protection & Security

Dr. Vivek Nallur (vivek.nallur@ucd.ie)

Protection

- Protection: controlling the access of programs, processes, or users to the resources defined by a computer system (files, memory segments, CPU. . .)
  - protection is a necessary condition (but not sufficient) to achieve security in a computer system
  - protection was originally conceived as an adjunct to multiprogramming OSs:
    - allow untrustworthy users to safely share a logical or physical name space (such as a directory or memory, respectively)

Goals of Protection

- Modern protection concepts address two basic goals:
- to prevent mischievous/intentional violation of access to a resource by a user
  - e.g.: distinction between authorised and unauthorised use of a resource
- to ensure that each active process uses resources only in ways consistent with OS policies -> reliability increase
  - e.g.: error detection at the interfaces between subsystems
  - early detection at the interface level can avoid malfunction propagation from one subsystem to another, improving system reliability

OS Protection

- OS protection: mechanisms for the enforcement of the system policies governing resource use
  - policy: what will be done about resource use (dictated by administrators, users, etc)
  - mechanism: how a policy will be implemented and enforced
- General mechanisms are more desirable, as policies may change from place to place or from time to time

Formal Model for Protection in Computer System

- Any computer system is a collection of:
  1. Objects: entities to which access must be controlled
     - hardware: CPU, memory segments, printers. . .
     - software: files, semaphores. . .
  2. Subjects: entities that access objects (processes, users)
  3. Rules: manner in which subjects may access objects
     - the operations depend on the object
       - CPU can be executed
       - memory can be read or written
       - files can be read, written or executed

Principles of Protection

- A process should only be allowed access to those resources

1. for which it has authorisation
   - example: even if any file can be read per se, we allow certain files to be read only by certain processes/users
2. and that are currently needed to complete its task: need-to-know principle, also called least privilege principle
   - it limits the amount of damage by a faulty process
   - example: a kernel mode process should have its access rights limited when doing everyday unimportant tasks

Domain of Protection

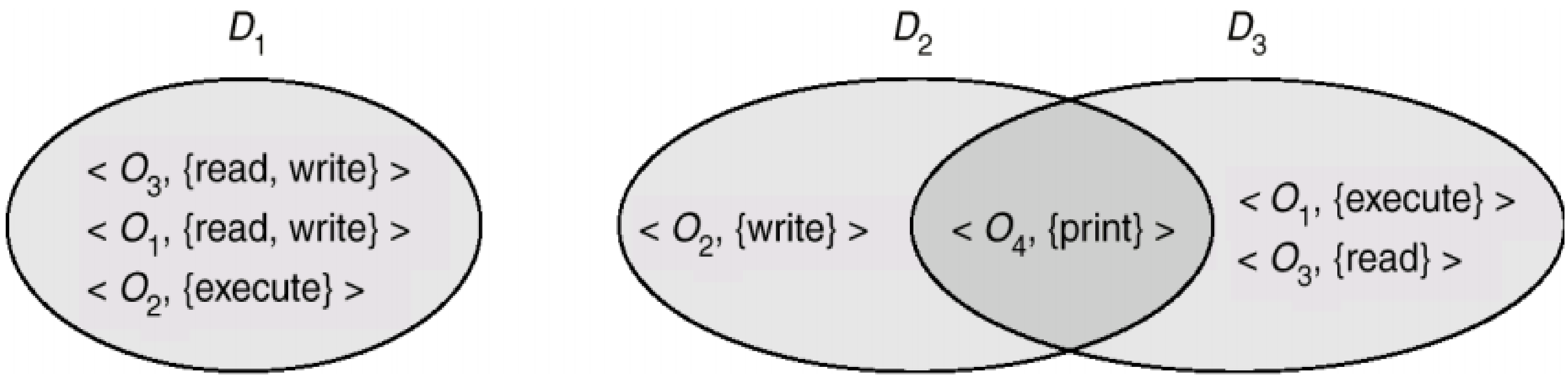
Definition

- *A process operates within a protection domain, which specifies the resources that the process may access*

Formal Definition

- *Protection domain is a set of ordered pairs <object-name, right-set>*
- *Example: D = <file F, {read, write}>. Any process operating in D can both read and write file F*

Domain example



D2 and D3 share < O4, {print}> so domains do not need to be disjointed

Association Between Processes & Domains

- Depending on whether it is fixed or not during the process lifetime, the association between process and domain may be:
  - Static: fixed association
    - however, if we are to enforce the need-to-know principle, we need a mechanism to change the content of a domain
    - <u>example</u>: if a process needs read and write access in two different phases and the domain is static with both r/w access, the principle is violated unless we modify the domain dynamically to reflect the minimum necessary rights
  - Dynamic: variable association
    - a mechanism to allow a process to switch from one domain to another must be available

Realisation of a Domain

- Depending on the OS, a domain is realised in a number of ways;

1. Each user may be a domain: domain switching occurs when the user identity is changed
2. Each process may be a domain: domain switching is implemented by a process sending a message to a process in a different domain and waiting for a response
3. Each procedure may be a domain: domain switching occurs when a procedure call is made

Examples of OS Protection

- Unix: domain associated with the user
  - domain switching corresponds to changing user identity temporarily, which is done through the file system
  - any file has two special values associated:
    - owner identification: user-id
    - domain bit: setuid
  - when user A executes a file owned by user B:
    - if setuid==0, then user-id of process is set to A
    - if setuid==1, then user-id of process is set to B
  - another way to switch domain: send a message to a more privileged process to do something on one's behalf

Access Matrix

- Access matrix (AM): abstract view of protection model
  - i-th row of AM: domain D_i
  - j-th column of AM: object O_j
  - Entry $a_{i,j}$ in AM: set of operations that a process executing in D_i can invoke on O_j
- **Example: three files F1, F2, F3 and a printer**

	F1	F2	F3	Printer
D_1	read		read	
D_2				print
D_3		read	exec	
D_4	Read write		Read write	

Enforcement of Access Constraints

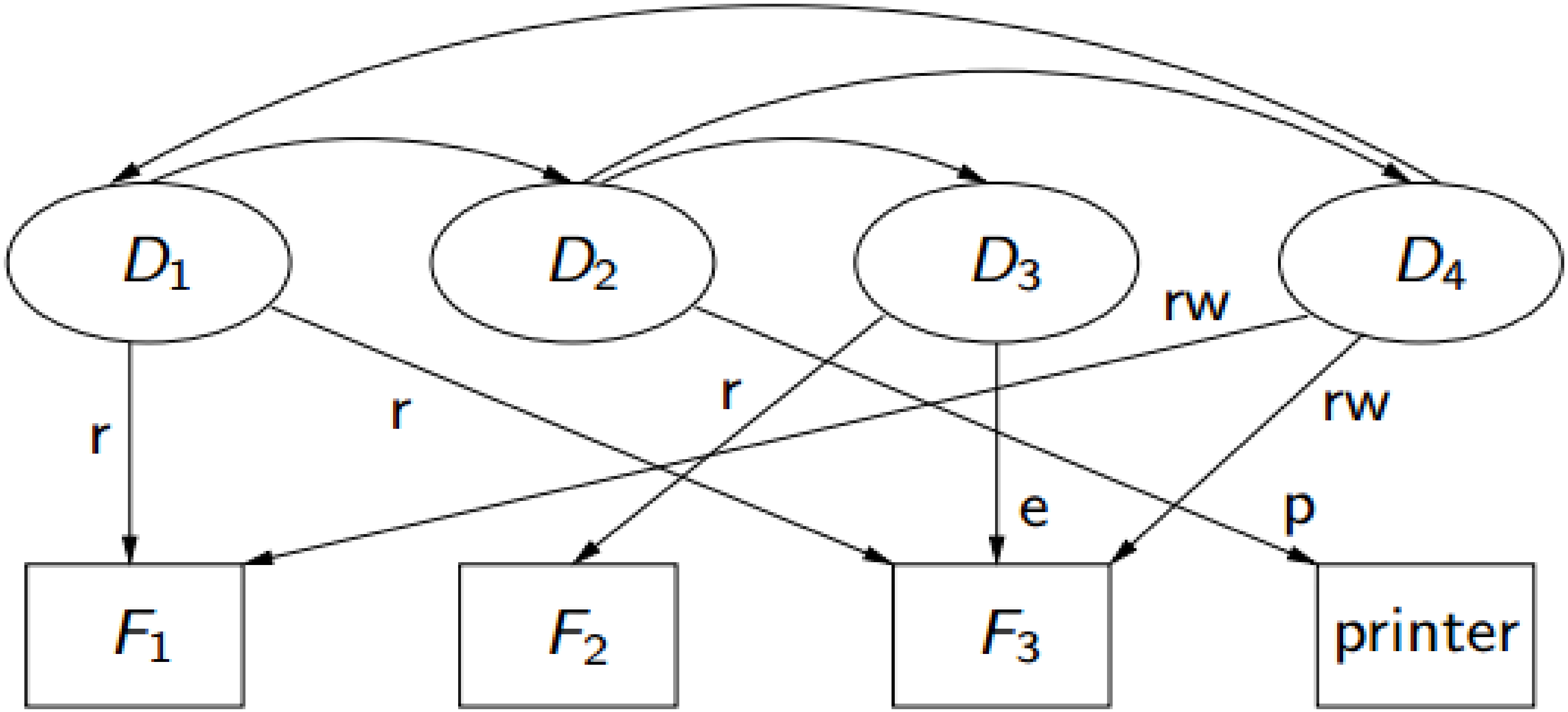
- Protection mechanism: AM implementation ensuring that its semantic properties hold
- In order to enforce the policy established in the AM, we need a monitor that controls access to objects
  - when a process executing in domain Di attempts to perform an operation M on an object O_j
    - the triple (D_i, O_j, M) is formed by the system and passed to the object monitor
    - the object monitor returns the Boolean value $\{M \in a_{i,j}\}$
    - if true, the operation is allowed to proceed

Problems with Access Control

- It must be enforced at every step
  - e.g.: what happens if a process opens and begins reading a file for which it has access rights, but then the access is revoked?
- It does not dictate information propagation, only initial access
  - e.g.: what happens if a process A copies a file to a location accessible by process B, which could not initially access it?
  - this is the confinement problem, which is in general unsolvable

Dynamic Protection State

- The dynamic protection state of a process can also be represented by means of a directed graph with labelled edges



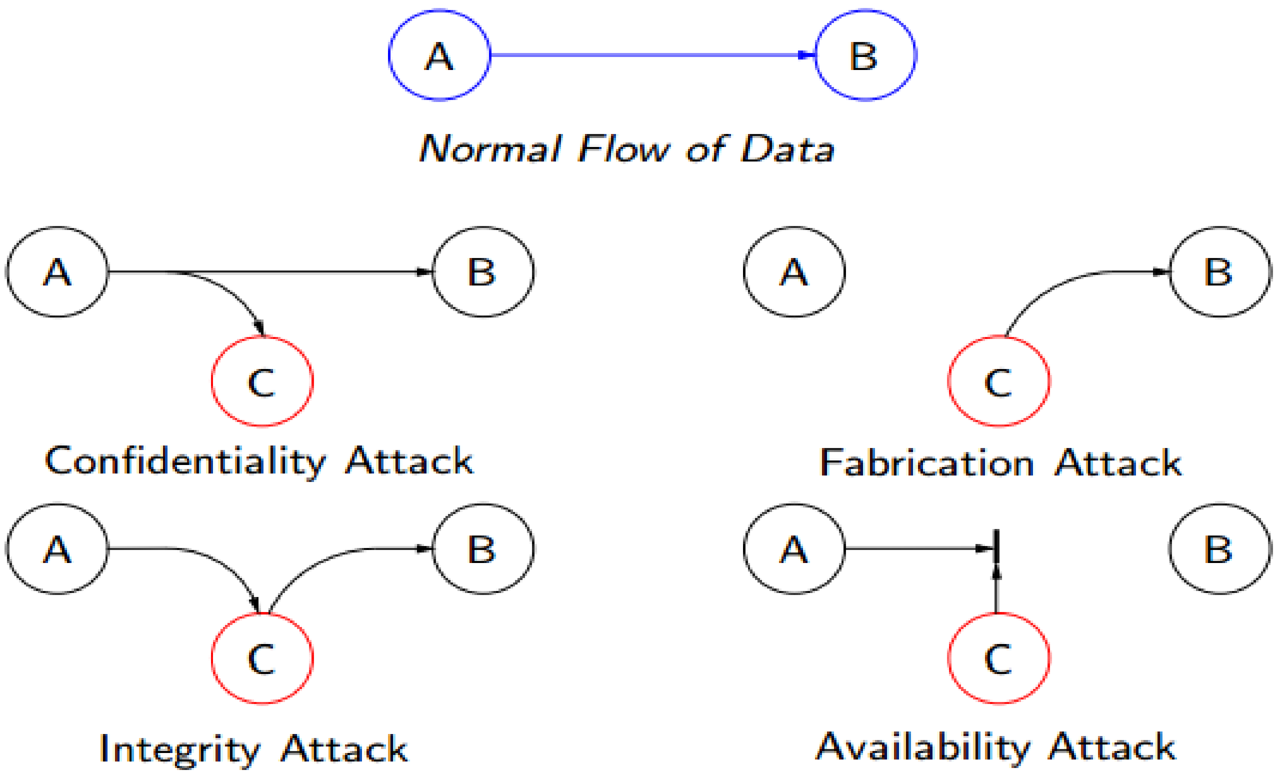
Security

- A protection mechanism is useless if, for instance,
  - an unauthorised person is able to log into the system
  - vulnerabilities enable the protection system to be bypassed by someone legally logged in
- An OS is secure if its resources are used and accessed as intended under all circumstances
  - total security cannot be achieved, but there are mechanisms that make security breaches a rare occurrence

Classification of Security Threats

- Intentional (malicious)
- unauthorised reading of data, information theft, or traffic analysis (passive threats, affecting data confidentiality)
- unauthorised destruction, tampering or fabrication of data (active threats, affecting data integrity or authenticity)
- prevention of legitimate use of system (active threats, affecting system availability)
- Accidental: human errors, hardware/software errors, natural disasters. . .

Classification of Intentional Security Threats



- A (Alice) and B (Bob) and C (Carol)

Classification of Intentional Security Threats

- “A” and “B” are authorised parties, “C” is the intruder
- This classification model is commonly used to examine security threats
- Additional Characters D,E,F.. Also can exist

Examples of Attacks

- Forcing system calls (fabrication attack)
  - trying illegal system calls, or legal system calls with illegal parameters
- Examining memory information (confidentiality attack)
  - many systems do not erase the space before allocation (remember how deletion with linked blocks works)
- Stack & buffer overflow (fabrication attack)
  - many C programs don't check array boundaries: by giving "wrong" execution parameters it is possible to overwrite the return address of local procedures to execute arbitrary code
  - in Unix, this is a bad problem if the program attacked has setuid==1 and is owned by the administrator (root)
- Denial of service (availability attack)
  - overloading a computer with legal but idle instructions to prevent it from doing useful things

User Authentication

- Protection relies on users being who they say they are
  - user authentication is the first line of security in any OS
  - without being inside the system many attacks are not possible
- Determining a user's identity uses one or more of
  - user knowledge (user identifier and password); most common
  - user possession (key or smart card)
  - user attributes (biometrics)

Passwords

- Passwords can be considered as capabilities (keys)
- Password vulnerabilities:
  - they can be guessed
  - they can be exposed or sniffed
  - they can be illegally transferred
- Secure passwords should
- be strong, which means
  - being long, to avoid brute-force attacks
  - not being frequent or obvious (i.e., unrelated to natural languages, etc), to avoid the use of dictionaries by attackers
- change frequently, to decrease the likelihood of illegal use on interception

Password Strength

- Length: with n bytes, there are 2^{8n} different passwords
- Frequency: a password is stronger if it is less frequent an attacker will have more uncertainty when trying to guess it
- Entropy: rigorous measure of uncertainty
  - Passwords should ideally not use repeated characters e.g if common words are used, then an attacker can use a dictionary attack , just checking a few thousand common passwords instead of having to try all possible combinations.

Changeable Passwords

- To help aid security , changeable passwords can be used
- One-time passwords
  - extreme form of changeable passwords; intercepting them won't give any advantage to an adversary in the future as they are only used once.
  - Challenge-response
    - Most common , using smart cards and very complex functions
      - **Example (simple) a user has x^2 as they password**
      - **Computer generates a random number like 5, user writes 25 back**

Password Protection Measures

1. Limit number/frequency of logins
2. Access control on password file (not enough if an intruder accesses it through some exploit)
3. Encrypt password file: store f (p) instead of p to keep secret even if password file can be accessed
   - e.g.: p =”password” → f (p) =”%s73da*wr”
   - An intruder with access to the password file could still crack passwords using an encrypted dictionary (if f (·) is public)
   - salting: secret random characters are concatenated with p before applying f (·) to avoid this threat
   - e.g. f (”password*8 W&”) instead of f (”password”) n bits of “salt” multiply the size of attacker search space by 2n

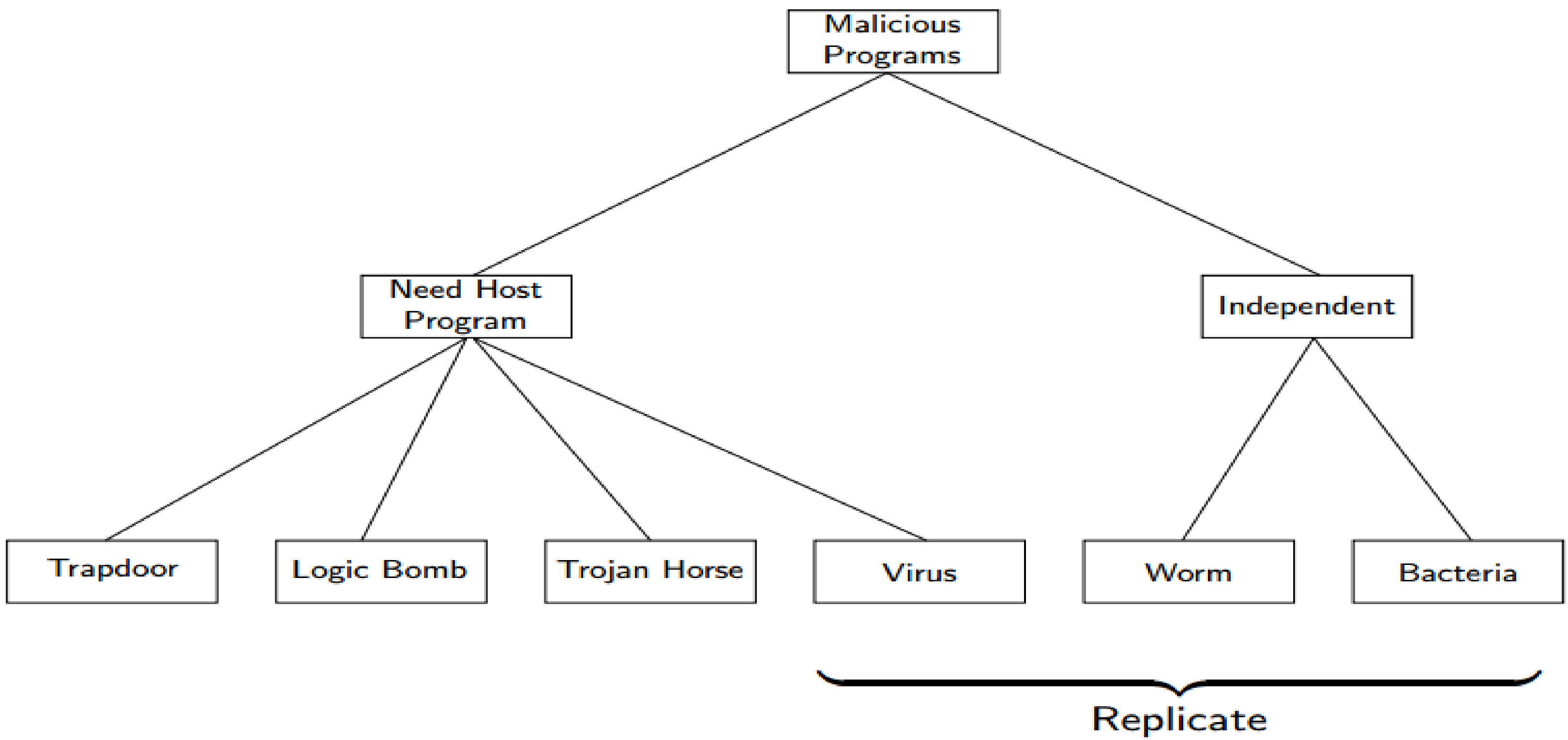
A Taxonomy of Security Threats

- Threats can also be classified by their “modus operandi”
- Bacteria
  - program that consumes system resources by replicating itself
- Logic bomb
  - logic embedded in a program that checks for a certain set of conditions to be present on the system; when conditions are met, it executes some unwanted function
- Trapdoor
  - secret undocumented entry point into a program, used to grant access without normal methods of authentication

A Taxonomy of Security Threats

- Trojan horse
  - secret undocumented routine embedded within a useful program; execution of the program results in execution of the secret routine (example: login spoofing)
- Virus
  - code embedded within a program that causes itself to be inserted in one or more other programs and which performs unwanted functions
- Worm
  - program that can replicate itself and send copies across network connections

Hierarchy of Security Threats Taxonomy



- This is a simplified over view as a Trojan could be part of a virus for example.

Preventative Security

- Install Anti-Virus programs
- Install Anti-Spyware programs
- In mission critical software systems, the design and verification of programs should be done using a system such as Evaluation Assurance Level

Evaluation Assurance Level

- International Standard for a Common Criteria security evaluation
- It does not guarantee security but suggests a system has been rigorously tested and evaluated.
- Starts at EAL1 (functionally test) to EAL 7 (Formally Verified Design and tested)
- Windows 8 is at EAL 4 , very few systems are higher as to get to level 5 or 6 would be incredibly expensive.
- For an OS as complex as Windows 8 to get to EAL 7 would require an inconceivable amount of work as every component would have to be formally verified.

Next week

- Last Lecture topic , Distributed Operating systems
- Study Time
  - Review Chapter 15