Imperial College London

Operating Systems

OS Security
Authentication and Authorisation

Peter Pietzuch
prp@imperial.ac.uk

Based on slides by
Daniel Rueckert, Cristian Cadar

Security Goals

- **Prevent unauthorised access to system**
- **Permit authorised sharing of resources**

- Data **confidentiality**
  - Attack: theft of data
- Data **integrity**
  - Attack: destruction or alteration of data
- System **availability**
  - Attack: denial of service

Tutorial Question

1. Why are security and protection important even for computers that do not contain sensitive data?

2. Sharing and protection are conflicting goals. Give 3 examples of sharing in OSs and explain what protection mechanisms are necessary.

Imperial College London

Policy vs. Mechanism

- **Security policy** specifies what security is provided:
  - what is protected
  - who has access
  - what access is permitted

- **Security mechanisms**
  - how to implement security policy
  - same mechanisms can support different policies

Security Aspects

- **People security**
  - Insider, social engineering attacks
- **Hardware security**
  - E.g., steal hard disk to get at data
- **Software security**
  - E.g., exploit bug to become superuser

- **System is as secure as weakest link!**

People Security

- A large number of computer crime by **insiders**
  - Employees need privileges to carry out duties
  - Tempting to abuse privileges for own gain
- Social engineering
  - People often not security conscious: phishing attacks, people tailgating into building, etc.
- People working around security measures for convenience
  - E.g., reusing passwords, providing insecure way for resetting passwords, etc.
- People with wrong security expectations
  - E.g. “one cannot forge a sender’s email address”

Imperial College London

Hardware Security

- With physical access to computer/peripherals one can:
  - Read contents of memory/disks
  - Listen to network traffic including (unencrypted) passwords
  - Alter contents of memory/disks
  - Forge messages on network
  - Steal machine or set it on fire
- Hardware itself can contain exploitable security flaws
  - Eg consider side-channel attacks on CPUs
  - Incorrectly implemented access control checks (Meltdown)

Software Security

- Software bugs may allow attackers to compromise system
  - Gain root privileges
  - Crash application
  - Steal data
  - Compromise data integrity
  - Deny access to the system

- Attacks may exploit
  - Buffer overflows
  - Integer overflows
  - Format string vulnerabilities

Imperial College London

Access Control

Access Control

- Authentication:
  - Verify identity of users (**principals**)

- Authorisation:
  - Allow principals to perform action only when authorised

Authentication

- Verification of identity of principal based on:
  - Personal characteristics
  - Possessions
  - Knowledge

Authentication: Personal Characteristics

- Authentication based on hard to forge, personal characteristics:
    - Fingerprints
    - Voiceprints
    - Retina patterns
    - Signature analysis
    - Signature motion analysis
    - Typing rhythm analysis

- Can suffer from:
    - High equipment cost
    - False positives / negatives

Authentication: Possessions

- Authentication based on securely-kept possessions
- Possession of keys most widely used system
  - Can ensure physical security of computers and other things
  - Keys being superseded by coded magnetic cards, RFID cards, implanted sensors, …

- Can suffer from:
  - Impersonation attacks if key lost
  - High equipment costs

Authentication: Knowledge

- Authentication based on secret knowledge (***password***):
  - Very cheap to implement

- Limitations:
  - ***Dictionary attacks*** can find most passwords:
    - Good guesses include login name, first names, street names, dictionary words, any of these reversed or doubled
  - ***Password reuse***
    - Users tend to reuse passwords
    - *Security as good as the security of weakest system*

Limitations of Passwords

- Password turnover:
  - Password vulnerable to guessing attacks throughout lifetime
  - Well-chosen password (with good encryption algorithm) can only be cracked by exhaustive search
- Change password regularly (every n weeks/months)
  - Crackers has to begin search anew
  - But people get lazy: **`mypasswd1`**, **`mypasswd2`**, …

Password Protection: One-Way Cryptographic Hash

- Some OSs used to store user passwords in protected file
  - Vulnerable to data theft, accidental disclosure/abuse by system administrators

- Modern OSs store only **encrypted** versions of passwords
  - Use one-way cryptographic hash function for encryption
  - Compare encrypted version of the string entered by user A with the encrypted password stored for A

Password Encryption

- Encryption based on one-way hash functions
  - One-way function: function that is easy to compute, but computationally hard to invert
  - Pre-image resistance: Given hash value h, it should be infeasible to find M s.t. H(M) = h
  - UNIX's is based on Data Encryption Standard (DES)

```
$ sudo cat /etc/shadow | grep prp
prp:$8r$O2/aRmG$1CF14YdqweF9iAf0SQ1tXJy:13453:0:99999:7:::
```

(Yes, I changed the hash…)

- **Guessing** is the only feasible way to find cleartext password from encrypted password
  - Choose inherently slow encryption function to limit number of guesses

Rainbow tables

- Given one-way function H, compute a **rainbow table** of H(k)’s, for many popular passwords k
- If H(password) leaks, compare it with all available H(k) in the rainbow table
- Continue to improve the rainbow table over time

- Is it possible to prevent this attack?

Password Protection: Salt

- Salt s: random value, often based on time
- Triple `(userid, s, E(s, P))` stored in password file
- At login, `E(s, p)` re-computed and compared with stored value
- Use of salt prevents:
  - Rainbow table attacks/reuse of dictionary attacks
  - Duplicate passwords from being visible

Imperial College London

Adobe – Leaked passwords

Nov 2013: 130,324,429 leaked passwords, no salt, hints not encrypted

#	Count	Ciphertext	Plaintext
1)	1,911,938	EQ7fIpT7i/Q=	123456
2)	446,162	j9p+HwtWWT86aMjgZFLzYg==	123456789
3)	345,834	L8qbAD3jl3jioxG6CatHBw==	password
4)	211,659	BB4e6X+b2xLioxG6CatHBw==	adobe123
5)	201,580	j9p+HwtWWT/ioxG6CatHBw==	12345678
6)	130,832	5djv7ZCI2ws=	qwerty
7)	124,253	dQi0asWPYvQ=	1234567
8)	113,884	7LqYzKVeq8I=	111111
9)	83,411	PMDTbP0LZxu03SwrFUvYGA==	photoshop
10)	82,694	e6MPXQ5G6a8=	123123

Authorisation

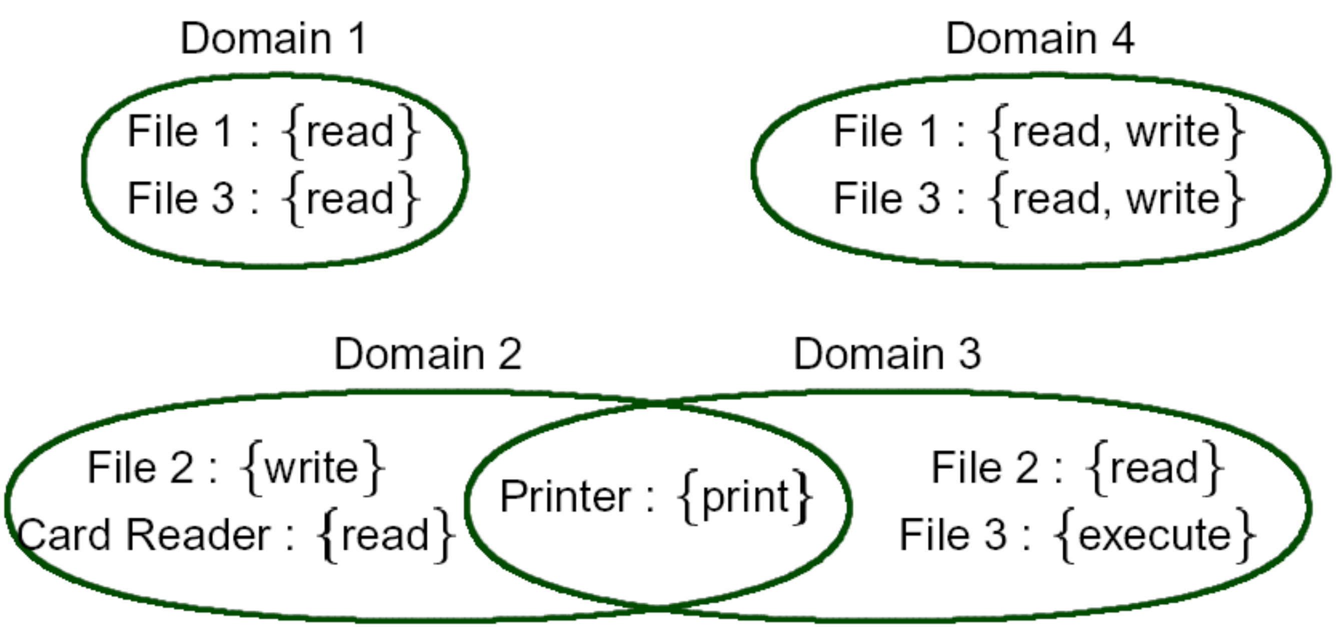
- Specifies:
  - who can access
  - what they can access
  - how they access can (what operations)

- Policy decision: what should be the default authorisation?
  - no access?
  - all access?

- **Principle of Least Privilege (PoLP)**
  - Gives user minimum rights required to carry out assigned task
  - Unfortunately, often more rights given by default for convenience

Protection Domains

- Set of **access rights** defined as:
  - Set of **objects**
  - **Operations** permitted on them
- **Principal** executing in **domain** D has access rights specified by D



Access Control Matrix

- Specifies **authorisation policy**
  - Rows represent principals
    - e.g. users, user groups, …
  - Columns represent target objects
    - e.g. files, devices, processes, …

	Object 1	Object 2	Object 3	Object 4	Object 5
Principal 1	read		read		read
Principal 2		execute		read, print	
Principal 3	read	read, print		execute	read
Principal 4	read, write		read, write		

Access Control Matrix: Implementation

- Expensive to implement matrix as global 2D array
- Two options:
  - Access-Control Lists (ACLs)
  - Capabilities
- Both options have pros and cons
  - In practice, most operating systems implement ACLs

Access Control List

- Each column of access matrix stored as **access control list** (ACL)
- An ACL ***stores with each object***:
  - The principals that can access it
  - The operations each principal can perform on it

Case Study: UNIX/Linux

Accessing Files on UNIX/Linux

- Users are the principals
  - Each user has a unique user ID (uid)
  - Superuser **root** has UID 0 and can access any resource
- Files are the objects
  - In UNIX, “everything is a file” (sockets, pipes, block and character devices, etc.) and can be accessed via file system interface using same access control mechanism
- Groups
  - Each user can belong to one or more groups
  - Each file can only belong to one group
- Access rights are **read (R)**, **write (W)**, **execute (X)**

Access Rights/Operations

- Only three domains for each file:
  - **read (R) : can read the file**
  - **write (W): can write the file**
  - **execute (X): can execute the file**

- For directories, the access rights mean:
  - **read (R) : can list contents of directory**
  - **write (W): can create/delete (owned) files**
  - **execute (X): can enter the directory & get access to files**

```
$ ls -l test.txt
-rw-r--r-- 1 prp prp 2517522 Mar  9 09:14 test.txt
```

Example

File owner

Group name

```
$ ls -l test.txt
-rw-r--r--   1 prp   lsds 2517522 Mar   9 09:14 test.txt
```

Access rights of everybody else

Access rights of group members

Access rights of file owner

File type

- \- regular file
- d directory
- b block file
- c character file
- l symbolic link
- p pipe
- s socket

Permission bits

- \- none
- r read
- w write
- x execute
- s setuid, setgid
- t sticky bit

Tutorial Question: Unix Permissions

Represent the ownerships and permissions shown in this UNIX directory listing as an access control matrix. Treat each of the two users and two groups as principals.

Note: `a` is a member of `users` and `systems`, `b` is a member of `users` only.

```
-rw-r--r--  3 a   systems    4137 2010-03-16 14:19 .emacs
-rwxr-xr-x  3 b   users      6420 2010-03-16 14:19 os.pptx
-rw-rw----  1 b   systems    1997 2010-03-16 14:19 notes.txt
-rw-r-----  3 a   users      7442 2010-03-16 14:19 index.html
```

Process Execution

- What happens when user A executes program (for which A has execute privileges)?
  - Program runs with A’s privileges
  - Can access any files to which A has access

- How does passwd work!?
  - Only root has access to password file

Imperial College London

SETUID programs

- **SUID** (set user id) bit
  - File switches effective UID to file owner when executed
  - Increases privileges when using system programs:

```
$ ls -l `which passwd`
-rwsr-xr-x 1 root root 42776 2009-04-04 06:50 /usr/bin/passwd
```

Process IDs

- Each process has three IDs:
  - **real UID:** ID of the user who started the process
  - **effective UID:** effective ID of the process, which is used in access control checks (with very few exceptions)
  - **saved UID:** a saved ID to which the effective ID can be changed to

Process IDs

- When a process starts effective UID = real UID
- If a setuid file, effective UID = ID of the file owner
- Processes with elevated privileges may temporarily drop their privileges changed their EUID to an unprivileged value
  - EUID can be saved as saved UID
- Non-root processes can change their EUID to
  - their real UID or their saved UID

Question: Dropping privileges

- Why would setuid programs need to drop privileges?

Imperial College London

Question: UNIX Permissions

Consider a file with the following UNIX permissions:

```
-rwsrwxrwx 1 root lsds 2240 2016-11-30 20:18 wombat
```

What kind of security implications does this file have?

(a) lsds members have full root access

(b) everyone has full root access

(c) everyone has partial root access

Capabilities

- Row of access matrix can be associated with domain to give **capability list**
- **Capability**
  - Possession of capability gives right to perform operations specified by it
    - Similar to possession of key
- Capabilities are **protected objects**
  - Protected pointer to object specifying permitted operations on object
    - E.g., file descriptor can be seen as a capability
  - Often not directly accessible by users but maintained by OS
    - Only accessed indirectly (e.g. via index into capability list)
    - OS provides procedures to create, delete, modify capabilities
  - Alternatively give encrypted capability to user

ACLs vs. Capabilities

- Principle of least privilege: + capabilities
- Revocation: + ACLs
- Rights transfer: + capabilities
- Persistence: + ACLs

DAC vs. MAC

- **Discretionary Access Control (DAC):**
  - Principals determine who may access their objects
- **Mandatory Access Control (MAC):**
  - Precise system rules that determine access to objects

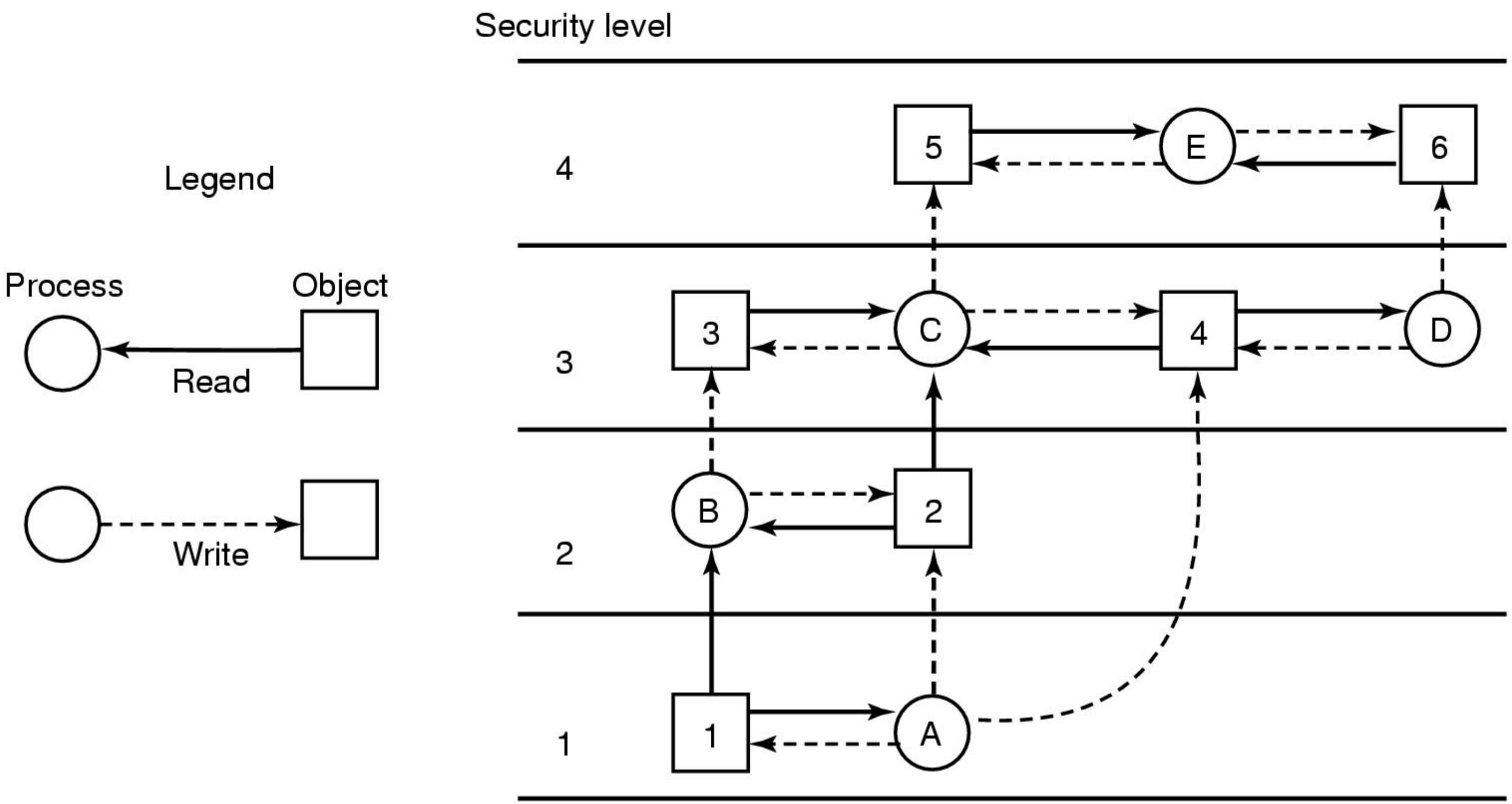
Bell – La Padula Model

- Objects and principals have assigned **security level**
  - E.g., unclassified, confidential, top secret

- Two rules:
  - **The simple security property**: A process running at security level k can read only objects at its level or lower
  - **The * property**: A process running at security level k can write only objects at its level or higher

- No info can leak from a higher level to a lower one
  - Ensures confidentiality, but what about integrity?

Bell – La Padula Model



Biba Model

- Guarantees data integrity:
  - **The simple integrity principle:** A process running at security level k can write only objects at its level or lower (no write up)
  - **The integrity * property:** A process running at security level k can read only objects at its level or higher (no read down)

Imperial College London

Design Principles for Security

- Give each process least privilege possible
    - Default should be no access
- Protection mechanism should be simple and uniform
    - Keep it simple!
- Scheme should be psychologically acceptable
- System design should be public
    - “Security through obscurity” is usually bad idea

Computer Security: Summary

- Security goals:
  - Prevent unauthorized access to system
  - Permit authorized sharing of resources
- Security aspects:
  - People security
  - Hardware security
  - Software security
- Access control:
  - Authentication: personal characteristics, possessions, passwords
  - Authorisation: ACLs, capabilities
  - UNIX cases study
- Discretionary vs mandatory access control