The University of Sydney
School of Computer Science
Dr Suranga Seneviratne
Senior Lecturer - Security

THE UNIVERSITY OF SYDNEY

Access Control

Recommended Reading

Security Engineering - 3^{rd} Edition by Ross Anderson

- **Chapter 6:** - Access Control
- **Chapter 9:** - Multilevel Security

1 Access Control

Access control is an important element of a security policy, aiming to control access to resources and assets. Its function is to control which **principals** (e.g., users, user groups, processes, machines etc.) have access to which resources in the system – which files they can read, which programs they can execute, how they share data with other principals, and so on. For instance, it addresses whether a bank's customer can access the vault or if a program can access other users' passwords.

We need to differentiate access control from **authentication** and **authorization**. Authentication is the process of acquiring evidence of the identity of another party. Authorization is the allocation of a privilege to a party.

Within our Reference Framework access control is specified in the policy, countering the incentives of attackers who aim to access resources or assets. It is implemented by one or more mechanisms, providing a certain level of assurance. A good example of access control is the use of passwords: Passwords serve as a mechanism, and their strength directly correlates with the assurance provided.

Example of Access Control Access controls can often be modelled as a matrix of access permissions, with columns for files and rows for users. We'll write `'r'` for permission to read, `'w'` for permission to write, `'x'` for permission to execute a program, and - for no access at all, as shown in Figure 1.
In this simplified example, Sam is the system administrator and has universal access (except to the audit trail, which even he should only be able to read). Alice, the manager, needs to execute the operating system and application, but only through the approved interfaces – she mustn't have the ability to tamper with them. Therefore, she doesn't have write access to the operating system and programs. She also needs to read and write the data. Bob, the auditor, can read everything.

User	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	rw	r
Alice	x	x	rw	-
Bob	rx	r	r	r

Table 1: Naive access control matrix

2 Spectrum of Access Control: DAC and MAC

Objects (files, processes, etc.) are owned by subjects (people, groups, processes, etc.).

Discretionary Access Control (DAC) In the old days, anyone with physical access to a computer controlled all of it: you could load whatever software you liked, inspect everything in memory or on disk and change anything you wanted to. This is the model behind discretionary access control (DAC): you start your computer in supervisor mode and then, as the administrator, you can make less-privileged accounts available for less-trusted tasks – such as running apps written by companies you don't entirely trust, or giving remote logon access to others.

Access to objects is determined by the identity of the subject that owns them. Subjects can transfer privileges over their objects to other subjects, unless constraints are in place, such as those defined by Mandatory Access Control (MAC). DAC offers a high level of flexibility and considerable control to the subjects.

Mandatory Access Control (MAC) However DAC can make things hard to manage at scale, and in the 1970s the US military started a huge computer-security research program whose goal was to protect classified information: to ensure that a file marked 'Top Secret' would never be made available to a user with only a 'Secret' clearance, regardless of the actions of any ordinary user or even of the supervisor. In such a multilevel secure (MLS) system, the sysadmin is no longer the boss: ultimate control rests with a remote government authority that sets security policy. The mechanisms started to be described as mandatory access control (MAC). The supervisor, or root access if you will, is under remote control.

An external entity (security policy administrator) defines access. 'Security kernel' checks via security attributes whether a subject is allowed to interact with an object. MAC offers a high level of rigour.

DAC and MAC: Microsoft OneDrive Example

Definition: DAC allows resource owners to control access based on their discretion.

OneDrive Example:

- **File and Folder Ownership:**
  - The creator of a file or folder is the owner by default.
  - Owners can transfer ownership.
- **Sharing Options:**
  - *Private:* Files are private by default.
  - *Specific People:* Owners can share with specific individuals via email.
  - *Anyone with the Link:* Owners can generate a shareable link for viewing or editing.
  - *Public:* Owners can make files accessible to anyone.
- **Permission Levels:**
  - *View:* Can see the file.
  - *Edit:* Can see and modify the file.

Mandatory Access Control (MAC)

Definition: MAC uses centralized policies to control access, limiting individual discretion.

OneDrive Example:

- **Administrative Controls:**
  - In Microsoft 365, admins set sharing and access policies.
  - Example: Restricting sharing to specific domains.
- **Data Loss Prevention (DLP):**
  - Automatically protects sensitive information.
  - Restricts access based on policies.

Summary

- **DAC in OneDrive:** Users control access, providing flexibility.
- **MAC in OneDrive:** Admins enforce policies in enterprise settings, ensuring security and compliance.

OneDrive primarily uses DAC for personal use, with MAC elements in enterprise environments for stricter control.

3 Operating System Access Control

The access controls provided with an operating system typically authenticate principals using a mechanism such as passwords or fingerprints in the case of phones, or passwords or security protocols in the case of servers, then authorise access to files, communications ports and other system resources.

3.1 Access Control Lists (ACLs)

ACLs define the access of subjects to objects. Objects are associated with an ACL, which can be defined globally or be more fine-grained. An ACL entry consists of a user ID and a mapping to allowed operations, which can range from coarse (read/write/execute) to very fine-grained permissions. ACLs are a widespread mechanism, common in all UNIX systems (POSIX standards) and exist in Windows. They are also used in social networking sites, content management systems, and enterprise resource management, etc. An example access control list is shown in Table 2.

User	Accounting Data
Sam	rw
Alice	rw
Bob	r

Table 2: An example access control lists (ACL)

3.2 Unix/Linux Operating Systems

The Linux/Unix permission model is based on a hierarchical structure that assigns different levels of access to files and directories. Each file and directory has three types of permissions: read (r), write (w), and execute (x), which can be assigned to three categories of users: the owner, the group, and others. This model allows for fine-grained control over who can view, modify, or execute files, enhancing security and ensuring that only authorized users can perform specific actions. Permissions are typically displayed using a symbolic notation (e.g., `rwxr-xr-x`) or an octal notation (e.g., `755`) and can be modified using commands like `chmod`, `chown`, and `chgrp`.

User Each user has a User ID (UID) and a Group ID (GID). E.g. the user could also belong to the `sudo` group.

Example of `/etc/passwd` entry, which states login shell:
`sam:x:1000:1000:Sam Jones,,,:/home/sam:/usr/bin/zsh`

Example of `etc/group` entry, which states group name, GID, users in group, etc.:
`sudo:x:27:sam,pnappa`

File Permissions

- **Owner ID**: the user that owns the file.

- **Group ID**: the group that 'relates to' the file.
- **Owner R/W/X**: three bits that determine what the 'owner' can do
  - Bit R set means read, W means write, and X means execute.
  - Directories that have X set can be traversed.
- **Group R/W/X**: bits determine what the group can do.
- **Other R/W/X**: bits determine what everyone else can do

Example: Owner, group and others all can read. Only the owner (sam) can write.
```
-rw-r-r- 1 sam sam 24K Jun 12 10:20 email.png
```

Files also have three more bits

- **Set-user-ID bit (SUID)**: The owner of a program can mark the file representing that program as suid, which enables it to run with the privilege of its owner rather than the privilege of the user who has invoked it. This is useful for running specific programs with additional permissions without being a privileged user. `mount` is an example.
- **Set-group-ID bit (SGID)**: If set on a file, it allows the file to be executed as the group that owns the file (similar to SUID). If set on a directory, any files created in the directory will have their group ownership set to that of the directory owner.
- **Sticky bit**: This permission does not affect individual files. However, at the directory level, it restricts file deletion. Only the owner (and root) of a file can remove the file within that directory. A common example of this is the `/tmp` directory.

Some examples for SUID, SGID, and Sticky bit are shown in Figure 1.

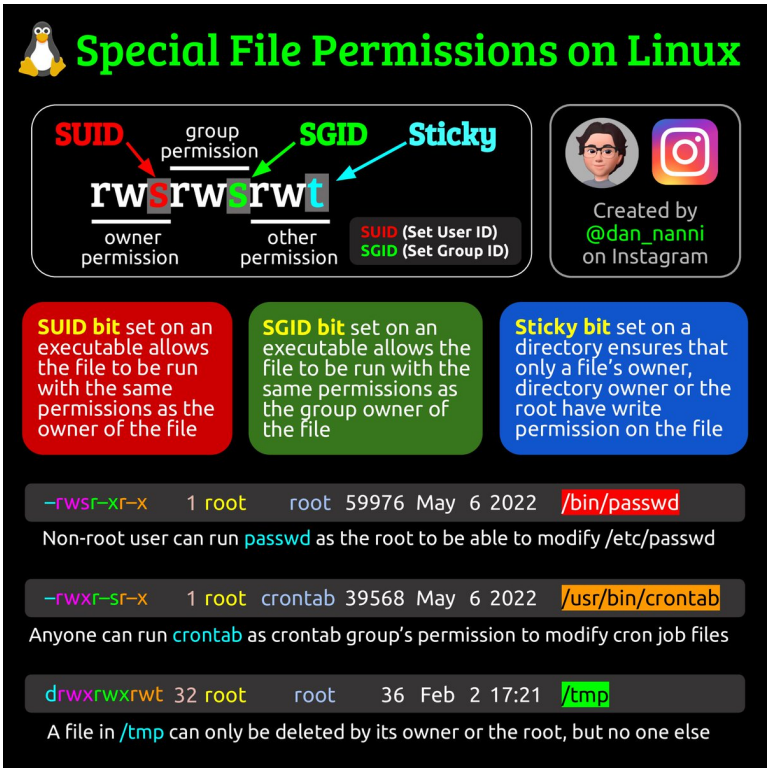

Figure 1: Examples of SUID, SGID, and the Sticky Bit [?]

Processes inherit permissions from the running user (UID and GID) In Unix/Linux-like operating systems, processes inherit permissions from the user who runs them. These permissions are controlled by user IDs (UIDs) and group IDs (GIDs). Here are the three types of IDs involved:

- Real UID/GID: The actual user and group that initiated the process. It identifies the user who owns the process.
- Effective UID/GID (eUID/eGID): Determines the permissions the process runs with. If a process is set with SUID (Set User ID) or SGID (Set Group ID), it can execute with the privileges of the file's owner/group rather than the user who launched it.
- Saved UID/GID: Used by the kernel to allow the process to revert back to its original UID/GID after temporarily assuming different privileges.

Root is an all-powerful user

- UID and GID of 0.
- Root can do what it likes – access any file, become any user, or whatever. What's more, there are certain things that only root can do, such as starting certain communication processes. The root userid is typically made available to the system administrator in systems with discretionary access control.
- The `sudo` command lets you run a command as root. This only can be done if the invoking user is part of the sudo group and usually requires password. Non-sudo users can still run some commands using `sudo`, if specified via the sudoers file.

3.3 macOS

Apple's macOS operating system is based on the FreeBSD version of Unix. Thus, it has similar permissions. The BSD layer provides memory protection; applications cannot access system memory (or each others') unless running with advanced permissions. This means, for example, that you can kill a wedged application using the 'Force Quit' command without having to reboot the system.

At the file system level, macOS is almost a standard Unix. Files in macOS have attributes and ACLs. Additionally, macOS features a system called Keychain, which stores extra passwords and some ACLs. The operating system also integrates built-in network authentication capabilities, such as Kerberos.

The default installation has the root account disabled, but users who may administer the system are in a group 'wheel' that allows them to su to root. If you are such a user, you can install programs (you are asked for the root password when you do so). Since version 10.5 (Leopard), it has been based on TrustedBSD, a variant of BSD that incorporates mandatory access control mechanisms, which are used to protect core system components against tampering by malware.

3.4 Windows

Windows is very powerful and shares similarities with Unix; however, instead of the RWX (read, write, execute) permissions, Windows features 13 kinds of permissions, such as Traverse Folder/Execute File, List Folder/Read Data, and Read Attributes, among others. However, this intricately complex access control system could lead to errors in secure default settings. Windows is enforced via a Security Reference Monitor, a trusted component that provides fine-grained security.

3.5 Apple iOS

iOS is based on Unix. Apps have many user permissions, often for privacy; they request a capability to access device services such as the mobile network, the phone, SMSes, the camera, and the first time the app attempts to use such a service. This is granted if the user consents. The many device services open up possible side-channel attacks; for example, an app that's denied access to the keyboard could deduce keypresses using the accelerometer and gyro.

Sensor values are read-only, reflecting principles similar to the Biba model. iOS implements extensive access control mechanisms, including hardware support, 'signed software', and encryption. It employs sandboxing to prevent apps from accessing files of other apps. Additionally, apps are granted specific capabilities through APIs and execute in a non-privileged mode. The OS itself is read-only to apps.

3.6 Android

Android is based on Linux; apps from different vendors run under different userids. The Linux mechanisms control access at the file level, preventing one app from reading another's data and exhausting shared resources such as memory and CPU. As in iOS, apps have permissions, which are in effect capabilities: they grant access to device services such as SMSes, the camera and the address book. Android supports a wide range of hardware platforms. How-

ever, one of it's Concessions is that developers cannot assume uniform features across all devices

Apps in Android come in signed packages, as .apk files, and while iOS apps are signed by Apple, the verification keys for Android come in self-signed certificates and function as the developer's name. This supports integrity of updates while maintaining an open ecosystem. Each package contains a manifest that demands a set of permissions, and users have to approve the 'dangerous' ones – roughly, those that can spend money or compromise personal data. In early versions of Android, the user would have to approve the lot on installation or not run the app. But experience showed that most users would just click on anything to get through the installation process, and you found even flashlight apps demanding access to your address book, as they could sell it for money. So Android 6 moved to the Apple model of trust on first use; apps compiled for earlier versions still demand capabilities on installation.

There are problematic issues, particularly with storage, which is well-known. There are two forms: app-internal and 'public storage.' Evidence suggests that developers often misuse public storage because it's easier to work with, allowing write access to everyone. Unfortunately, this also means that everyone can read from it. The API is more permissive; for example, apps can send SMS messages on your behalf. While the Google Play Store is less tightly controlled than Apple's App Store, apps are still screened for malware.

4 Middleware Access Control

Middleware, whether located on a single host or distributed across several, encompasses a range of technologies such as message brokering (e.g., AMQP) and various forms of Remote Procedure Calls (RPCs), including Java RMI, CORBA, XML-RPC, among others. These components are often exposed to probing by external entities, increasing security risks.

The complexity involved in managing middleware is substantial, necessitating precise definition of access granularity, meticulous monitoring and enforcement of permissions, and presenting significant challenges in configuration.

Although policy languages exist to define access control, they are challenging to use effectively. To mitigate some risks, it is advisable to protect middleware from external access, which at least guards against random, remote exploitation. However, it's crucial to note that malware originating from within one of the participating systems can still attempt to exploit the middleware, presenting a persistent security challenge.

4.1 Database Access Controls

Database Management Systems (DBMSes) with support for multiple users and processes incorporate their own access control mechanisms. Despite this, the actual data often resides on disk or in RAM, making Operating System (OS) protections essential. Accessing a database typically requires a username and password, emphasizing the importance of internal privilege management. This includes mapping users on the OS to roles within the database **(role-based access control)**. The SQL standard outlines fine-grained methods for access control, theoretically enabling any kind of object to be under access control. However, this level of granularity can lead to an overcomplication of the system, potentially making it cumbersome to manage and understand. Additionally, DBMSes provide specific integrity controls, which,

while crucial for data accuracy, can be computationally intensive to verify.

4.2 Browsers

The web browser is another middleware platform on which we rely for access control and whose complexity often lets us down. The main access control rule is the *same-origin policy* whereby JavaScript or other active content on a web page is only allowed to communicate with the IP address that it originally came from; such code is run in a sandbox to prevent it from altering the host system. We will discuss more about it during our Web Security Lecture.

By now there's a realisation that we should probably have treated browsers as access control devices all along. After all, the browser is the place on your laptop were you run code written by people you don't want to trust and who will occasionally be malicious; as we discussed earlier, mobile-phone operating systems run different apps as different users to give even more robust protection. Even in the absence of malice, you don't want to have to reboot your browser if it hangs because of a script in one of the tabs. (Chrome tries to ensure this by running each tab in a separate operating-system process.)

Bugs in browsers are exploited in drive-by download attacks, where visiting an attack web page can infect your machine, and even without this the modern web environment is extremely difficult to control. Many web pages are full of trackers and other bad things, supplied by multiple ad networks and data brokers, which make a mockery of the intent behind the same-origin policy. Malicious actors can even use web services to launder origin: for example, the attacker makes a mash-up of the target site plus some evil scripts of his own, and then gets the victim to view it through a proxy such as Google Translate. A prudent person will go to their bank website by typing in the URL directly, or using a bookmark; unfortunately, the marketing industry trains everyone to click on links in emails.

Confused Deputy Problem

The “confused deputy problem” in browsers occurs when a web browser is tricked into performing actions on behalf of a malicious website, leveraging the browser’s higher-level permissions to access resources or execute actions that the malicious website itself would not have permission to perform. This happens because the browser is “confused” about who the real requester of the action is, mistakenly attributing the authority to the malicious site. This can lead to unauthorized actions and data breaches.

The confused deputy problem can occur in web browsers through Cross-Site Scripting (XSS) attacks, where a malicious script is injected into a trusted website. We will discuss this further in our web security lecture, but here is a quick rundown.

Malicious Script Injection: A user visits a trusted website that has an XSS vulnerability. The attacker injects malicious JavaScript into the site.

Executing Malicious Actions: When the user interacts with the site, the injected script runs in the user’s browser with the website’s permissions.

Exploiting Trust: The script can now perform actions on behalf of the user, like sending authenticated requests or accessing sensitive data, because the browser believes these actions originate from the trusted site.

5 Other Access Control

5.1 Sandboxing

Sandboxing creates a shielded environment where malicious applications are prevented from interfering with the surrounding system. This environment usually offers a limited API for interactions, striking a balance between functionality and security.

For instance, Java applets, though now largely obsolete, serve as a historical example of sandboxing in action. sandbox provides a restricted environment in which the code has no access to the local hard disk (or at most only temporary access to a restricted directory), and is only allowed to communicate with the host it came from (the same-origin policy). Other examples of sandboxing include JavaScript running within browsers, and Google Chrome’s approach of isolating tabs in separate processes. However, securing the sandbox presents challenges, as there are methods by which it can be "broken out of," compromising the intended isolation and protection.

5.2 Virtualisation

The primary use case of virtualization involves Virtual Machines (VMs) that emulate a computer system. Virtualisation was invented in the 1960s by IBM; a single machine could be partitioned using VM/370 into multiple virtual machines. VMs allow software to run even if the original environment it was designed for is unavailable, with the operating system within the VM perceiving only its designated "own" hardware. At the client end, virtualisation allows people to run a guest operating system on top of a host (for example, Windows on top of

macOS), which offers flexibility.

Virtualization comes in varying degrees: Full virtualization entails complete hardware emulation, even across different hardware architectures. Para-virtualization requires the guest operating system to support virtualization. Containers and jails represent a form of lightweight virtualization, offering an isolated environment for running applications.

The management and operation of VMs are handled by a hypervisor, which has hardware support at the ring -1 level. This management includes both full and para-virtualization and is exemplified by tools such as VirtualBox, VMWare, Xen, QEMU, and others.

Although not primarily designed for access control, the significant effort required to "break out" of a VM inherently increases security by raising the bar for attackers. However, it's important to note that VM execution can be detected; VMs do not perfectly mimic real environments, allowing malware to identify its presence within a VM and potentially alter its behavior. Additionally, hypervisors themselves can have vulnerabilities that may be exploited. It's crucial to keep hypervisor software up to date and not to rely on virtualization as the sole defense mechanism against attacks.

Example: Docker It was initially developed for continuous development and deployment processes. It offers an abstraction layer over containers, facilitating support for various operating systems through their native jail implementations. Docker enables the creation of images that package applications along with their supporting libraries, highlighting its efficiency and resource-friendly nature. Containers created by Docker can be quickly spun up or deleted, and management systems such as Kubernetes further enhance its utility by simplifying container orchestration.

Docker containers sandbox applications, preventing direct communication between apps in different containers. However, the Docker daemon, which acts as a broker, runs with root privileges. This presents a potential risk, as any user with the capability to start or stop containers can interact with the daemon, thus expanding the attack surface.

Although Docker and similar container technologies are not primarily designed for access control, they inherently provide an additional layer of separation between applications. This separation effectively raises the security threshold, making it more challenging for attackers to compromise system integrity.

6 Hardware Access Control

6.1 Hardware Protections

Hardware protections enhance the security of computer systems through various mechanisms, including memory protection, privileged execution, and isolated cryptographic components.

Memory Protection CPUs support the operating system in maintaining distinct memory allocations for various processes. This is primarily achieved through segment addressing, where memory is accessed using two distinct registers: a segment register, which identifies a specific segment of memory, and an address register, which pinpoints a location within that segment.

If a process attempts to access memory outside its designated address space, a segmentation fault (SIGSEGV) is triggered, protecting against unauthorized read and write operations. This mechanism ensures that processes operate within their allocated memory spaces, maintaining the integrity and security of the system's memory.

Privileged execution CPUs have layered modes, or rings to support the operating system. These rings are organized into distinct privilege classes, each delineating the permissions and types of operations permissible within that layer. Further enhancing this security model, the ARM architecture incorporates separate registers for each privilege class, allowing for more nuanced control and security management. This system ensures critical operations are safeguarded by restricting access based on privilege level, thus bolstering the system's defense against unauthorized access and potential vulnerabilities.

Isolated cryptographic components Storing cryptographic keys within hardware are involved to enhance security. This approach restricts access to the keys, allowing only authorized functionalities to use them through a dedicated interface. An example of this is the Intel Trusted Platform Module (TPM), which was initially designed for Digital Rights Management (DRM) purposes. The TPM provides a secure environment for cryptographic operations, ensuring that sensitive information is stored and processed in a manner that minimizes the risk of exposure to malicious software and attacks.

6.2 Intel Processors

Intel processors have evolved significantly over time, introducing advanced security and memory management features. The Intel 80286 processor introduced segment addressing and protection rings, establishing a foundation for modern computing security measures. Following this, the Intel 80386 added built-in virtual memory and expanded memory segments to 4Gb, greatly enhancing the processor's capability.

The rings of protection are supported by a number of mechanisms. The current privilege level can only be changed by a process in ring 0 (the kernel). Procedures cannot access objects in lower-level rings directly but there are gates that allow execution of code at a different privilege level and manage the supporting infrastructure, such as multiple stack segments.

Modern Intel CPUs now have nine rings: ring 0–3 for normal code, under which is a further set of ring 0–3 VMM root mode for the hypervisor, and at the bottom is system management mode (SMM) for the BIOS. In practice, the four levels that are used are SMM, ring 0 of VMX root mode, the normal ring 0 for the operating system, and ring 3 above that for applications.

6.3 Kernel and Hardware

In the interaction between the kernel and hardware, code execution occurs within designated CPU modes, known as **rings**, which are hierarchical levels of privilege. CPUs enforce strict access control, preventing processes running in higher privileged rings from directly interacting with those in lower rings.

This architecture embodies the **principle of least privilege**, asserting that all code should operate only with the minimal required privilege and no more. Furthermore, the number of

pathways for communication with the kernel should be meticulously limited to the absolute minimum and rigorously controlled. This approach ensures a secure computing environment by minimizing potential attack vectors and restricting the scope of actions that processes can perform, thereby enhancing overall system security.

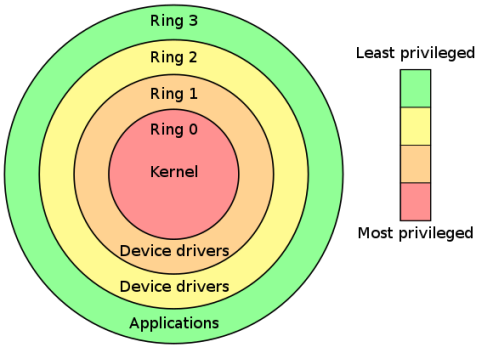

Figure 2: Privilege rings for the x86 available in protected mode (Wikipedia)

6.4 ARM Processors

The Arm is the processor core most commonly used in phones, tablets and IoT devices. Unlike Intel, it licenses a range of processor cores, which chip designers include in their products. The core initially contained no memory management, so Arm-based designs could have their hardware protection extensively customized; there are now variants with memory protection units (MPUs), and others with memory management units (MMUs) that handle virtual memory as well.

Arm's latest offering is CHERI (Capability Hardware Enhanced RISC Instructions) which adds fine-grained capability support to Arm CPUs. CHERI enables a process spawning a subthread to allocate it read and write accesses to specific ranges of memory, so that multiple sandboxes can run in the same process. The long-term promise of this technology is that, if it were used thoroughly in operating systems such as Windows, Android, or iOS, it would have the potential to prevent most of the zero-day exploits of recent years.

7 Security Policy Models

Multilevel security (MLS) policy MLS is a foundational access control model designed to handle information across different levels of classification, such as Top Secret, Secret, and Confidential. This model holds enormous importance in military and government sectors, where the need to strictly control access to information based on its classification is critical. Despite its foundational status, implementing MLS in a way that maintains high utility and operational efficiency is surprisingly challenging. For example, how can a system provide automated means to reduce the classification of a document from top secret to secret, so that other, dependent principals at lower classification can read it?

Security Policy Models When analyzing a system's security, the approach typically follows an order starting with the analysis of incentives and the creation of a *threat model* to understand the motivations and capabilities of potential attackers. This is followed by an examination of the *security policy*, which outlines the goals and objectives the system aims to achieve, detailing

the standards and requirements for protection. The final step involves scrutinizing the *security mechanisms* put in place to enforce this policy, assessing the technical controls and procedures designed to mitigate, detect, and respond to threats. A Security Policy Model provides a precise and concise statement regarding the protection properties of a system. This model can be highly formal, serving as the foundation for driving engineering decisions and actions.

7.1 Bell-LaPadula Model

The classic multilevel security policy model that gained wide acceptance was proposed in 1973. In the 1970s, there was a growing recognition that operating systems (OSes) would inherently contain vulnerabilities that could potentially be circumvented by users. This realization underscored the necessity to implement Mandatory Access Control (MAC) mechanisms that could not be easily bypassed.

Its basic property is that information cannot flow downwards. More formally, the Bell-LaPadula (BLP) model enforces **two properties**, which are two forms of MAC:

- The *simple security property*: no subject may read data at a higher level. This is also known as **no read up**.
- The ** (star)-property*: no subject may write data to a lower level. This is also known as **no write down**.

The ** (star)-property* is crucial. It specifically addresses the scenario where a user at a lower security level attempts to exfiltrate data by writing a program and then waits for an administrator or a user at a higher security level to accidentally execute it. The ** (star)-property* enforces a "no write down" rule, which prevents higher-level entities from writing to lower-level objects, thereby blocking the avenue for such indirect data exfiltration attempts. Implementing Mandatory Access Control (MAC) is essential to enforce this property effectively, ensuring that attempts to bypass security controls through clever manipulation of higher-privileged users are thwarted.

It is also important to understand the implicit properties of the model

- **No read up** - This means the subject can read their own level or down.
- **No write down** - This means the subject can write to their own level or up.

- Bell-LaPadula has one discretionary form of access control, where an external entity defines the access of subjects to objects in a matrix (Table 3).

	o_1	o_2	o_3	o_4
Sam	read	append	$\emptyset$	read
Bert	$\emptyset$	read-write	append	execute

Table 3: Bell-LaPadula Access Control Matrix

The Bell-LaPadula model is notable for its innovations, particularly its suitability for formal analysis and its integration of both discretionary access control (DAC) and mandatory access

control (MAC). However, it faces several criticisms. The model relies on administration by a single, omnipotent Trusted Principal, raising questions about the security of this central authority. Designed primarily with confidentiality in mind, it does not adequately address data integrity. Furthermore, the model overlooks the potential for covert channels, such as encoding data through variations in CPU load, allowing information to be inferred by programs at lower security levels. Additionally, a system that combines two Bell-LaPadula-secure systems does not necessarily maintain Bell-LaPadula security, highlighting complexities in its application and potential limitations in ensuring comprehensive security.

7.2 Biba Model

The Biba model focuses on data integrity rather than confidentiality, aiming to prevent malicious modifications to data. Essentially, it can be seen as a reverse of the Bell-LaPadula model, with its core principles including:

- The *simple integrity property:* A subject may only read data at own level or higher **(no read down)**.
- The **-(star) integrity property:* A subject may only write data at own level or lower **(no write up)**.
- The *invocation property:* A process from below cannot request higher access; only with subjects at an equal or lower level.

It is important to understand the Invocation property of the Biba model here. Usually, a subject refers to any users, programs, or processes in security policy models. In the Biba model, the first two rules use the word *subject*, and the third rule uses the word *processes*. While the Simple and * (Star) Integrity Properties control direct read and write access to data, the Invocation Property controls the ability of processes to call or execute other processes. It avoids scenarios such as a low-integrity process trying to invoke a high-integrity process to perform operations on its behalf, potentially leading to integrity breaches (i.e., prevents lower integrity processes from invoking higher integrity processes).

Examples:

- An aircraft's in-flight entertainment system being able to read data from the avionics, such as airspeed, but unable to alter any avionics data, ensuring that entertainment system vulnerabilities cannot compromise flight controls.
- An odometer being able to display mileage without the capability to alter it, ensuring the integrity of the vehicle's mileage data.

7.3 Example of Security Policy Models

Table 4 shows a mapping between users and clearances, and between required clearances and objects. Three files are protected, each holding a code needed to access, respectively, the fridge, the TV, and the PlayStation. Only these mappings are defined; no other rule sets exist.

User	Clearance
Bart	None
Homer	Low
Lisa	Medium
Marge	Ultimate

Required clearance	Object
Ultimate	fridgecodes.txt
Medium	tvlockcodes.txt
Low	playstationcodes.txt

Table 4: Access Tables

In a Bell-LaPadula model:

- **Can Homer obtain the TV lock codes to watch TV?**
  No. Homer with "Low" clearance can't read "Medium level" tvlockcodes.txt.
- **Can Homer enlist the help of Marge to obtain the TV lock codes?**
  No, because Marge can't write down.
- **Can Bart change all codes as he wishes?**
  Yes. Bart, with "No" clearance can write up. That means Bart can change any document above their clearance. This is an extreme example but highlights the fact that the Bell LaPadula model focuses only on confidentiality, not integrity.

In a Biba model:

- **Can Homer obtain the TV lock codes to watch TV?**
  Yes, read up is allowed.
- **Can Marge change all codes as she wishes?**
  Yes, write down is allowed.

7.4 Use of Models

The use of pure Bell-LaPadula or any single security model in practice is rare, given the complexity and varied security needs of modern systems. However, variants of the Biba model, which emphasizes data integrity over confidentiality, find more application in contemporary operating systems. Examples include Windows Vista and certain UNIX systems like SELinux, some versions of Red Hat, and a FreeBSD module. These implementations underscore the relevance of the Biba model's principles in real-world scenarios.

Despite the rarity of pure model implementations, the concepts underlying both Bell-LaPadula and Biba models are crucial for understanding and discussing real-world access control mechanisms. These theoretical frameworks provide valuable insights into the design and evaluation of access control systems in various environments. In the broader context of this lecture, the focus will shift towards exploring these real-world concepts as they apply to mobile, desktop, and server operating systems and hardware, highlighting the practical application of theoretical security models in enhancing system security.

8 Practice Quiz

Question 1: Which Bell-LaPadula property keeps lower-level subjects from accessing objects with a higher security level?

a) No write-up property
b) *(star) security property
c) No read-up property
d) No read-down property

Explanation: *Bell-LaPadula model has three properties; no read up, no write down (also called the * property), and a discretionary access control matrix. No read-up means any principal can't access objects above their clearance level. No write-down means principles can't write to documents below their clearance level. Based on that, the correct answer is c) No read-up.*

Also note that the properties a) No write up and c) No read down are actually properties of the Biba model

Question 2: What is a covert channel?

a) Use of CPU heat or noise to identify what message the CPU is processing.

b) A method that is used to pass information and that is not normally used for communication.

c) Any form of encryption used to transmit secret messages.

d) Any communication used to transmit secret or top-secret data.

Explanation: *A covert channel is to use a method to transmit data that is not usually used for communication. For example, Steganography is the use of images for communications. It is covert in the sense that the attackers expect messages to be text rather than images. Therefore the answer is b).*

What a) explain is a side-channel, not a covert channel. In a side channel, the attacker infers what is inside a protected communication by observing other artefacts such as packet lengths and timing or the heat signature of the CPU.

c) and d) are incorrect because it talks about the usual encrypted communication. In both cases, the attacker knows the message payload is something important, but they can't decrypt it.

Question 3: Which security model(s) address(es) data confidentiality??

a) Both Biba and Bell-LaPadula models
b) Security Policy Models
c) Bell-LaPadula Model
d) Biba Model

Explanation: *As explained in class Bell-LaPadual model addresses data confidentiality, and*

the Biba model addresses data integrity. That also means the Bell-LaPadual model does not provide data integrity, and the Biba model does not provide data confidentiality.

Security policy models are generic terms. You already know about two example models, Bell-LaPadual and Biba; they do not always provide data confidentiality.

Therefore the answer is c) the Bell-LaPadual model.

Question 4: Permissions of a file in a Linux-based system are represented by which of the following characters?

a) r,w,x
b) e,w,x
c) x,w,e
d) e,x,w

Explanation: *There are three permissions associated with a file in a Linux-based system. They are read-'r', write-'w', and execute-'x'. Therefore the answer is a).*

Question 5: A file named abc.txt has the following set of permissions in a Linux-based system.

```
-rwxrwxrwx
```

Is the following statement True or False? "All three operations, i.e. read, write and execute, can be performed on the file by the file owner, group owner and others."

a) True
b) False

Explanation: *Linux provides a three tired file protection system that determines the file access rights, i.e. the permissions are divided into three groups as*

```
rwx rwx rwx
```

The first group has all three permissions, i.e. file is readable, writable and executable by the file owner.

The second group also has all three permissions, i.e. file is readable, writable and executable by the group owner.

The third group also has all three permissions, i.e. file is readable, writable and executable by others who are neither a part of the group nor they are an owner of the file.

Normally, this set of permissions is too dangerous!

Question 6: Consider the following Bell-LaPadula model implementation. An organisation has four security levels.

- Top Secret (Highest level of security)
- Secret
- Confidential
- Unclassified (Lowest level of security)

These security levels are used as "security clearances" for personnel and "security classifications" for objects. You are given the clearances and classifications few people working in the organisation and some objects.

Employee	Security clearance
Tamara	Top secret
Samuel	Secter
Claire	Confidential
Alice	Unclassified

Object	Security classification
Personnel Files	Top secret
E-mail files	Secret
Activity logs	Confidential
Telephone Lists	Unclassified

Figure 3: Document classifications and security clearances

Which of the following statements is TRUE? Select all that apply.

a) Tamara can read all files
b) Claire cannot read Personnel or E-Mail files
c) Alice can only read telephone files
d) Tamara can write to Activity Logs
e) Alice can write to Personnel Files

Explanation: *Bell-LaPadula model is defined by two policies "No read up" and "No write down". There is no mention of the discretionary part, so we do not have to consider it here.*

"Tamara can read all files" TRUE - Tamara has the highest clearance of "Top secret". So she can read any object with any classification.

"Claire cannot read personnel or e-mail files" - TRUE - Claire has the clearance of "Confidential". So she can't read objects classified as "Secret" or "Top-Secret".

"Tamara can write to Activity Logs" - FALSE - Tamara, with the "Top Secret" clearance, can write only at her level of up. "No write down" means she will not be able to "Activity logs" that are "Confidential".

"Alice can write to Personnel Files" - As counter-intuitive as it is, this statement is TRUE. Alice can indeed write to any document at her own level and above, even though she can't read them.

Question 7: Sam works for a start-up in the defence industry. Every time when he creates a file on his computer, he needs to define what access each one of the other employees in the company has. If he did not define access by default, the file would be accessible to everyone in the company.

What is the most likely access control mechanism that is in place in Sam's start-up?

a) Role-based access control
b) Discretionary access control
c) Mandatory access control
d)Bell-LaPadula model

Explanation: *Here, the point to notice is that Sam has discretion in defining access to the objects he creates. And if he did not do anything, there is no access control. Therefore, the correct answer is Discretionary Access Control.*

Role-based access control assigns a role to a given user. For example, it is heavily used in databases. The "Database Administer" has more privileges than the "Database Manager".

No object classification is discussed, so it is not the Bell-LaPadula model.

Question 8: Mobile operating systems such as Android and iOS use sandboxing as access control for mobile apps. Which one of the following is a feature of sandboxing in the context of mobile operating systems? Select all that apply.

a) Each app runs as a separate user in the operating system

b) Apps can communicate with other apps only via controlled APIs

c) The Operating System is only read-only to an app

d) Apps have the ability to access any part of the main storage.

Explanation: *Only "Apps have the ability to access any part of the main storage." is incorrect. All the other three are correct. They are analogous to how user permissions are managed in a Linux-based system. Each app has similar permissions as a user in a Linux system.*

Question 9: Docker container's initial use case is continuous development and deployment. However, they have some properties that make them a good candidate for some access control applications. TRUE or FALSE.

a) TRUE

b) FALSE

Explanation: *Docker allows you to build, test, and deploy applications quickly. Docker packages software into standardised units called containers with everything the software needs to run, including libraries, system tools, code, and run time. Therefore it can act as a sandbox. As a result, it can be used in access control.*

Question 10: Read about the "Confused Deputy Problem" using the below links and answer the following question.

`https://dimosr.github.io/confused-deputy/`

`https://en.wikipedia.org/wiki/Confused_deputy_problem`

Some subcomponents of browsers, such as scripting, add-ons, and cookies, are particularly vulnerable to the confused deputy problem. TRUE or FALSE.

a) TRUE

b) FALSE

Explanation: *When visiting a website, browsers load various resources, such as Javascript, from different locations and that allows the possibility of confusing the browser about what is allowed and what is not for different components. CSRF (Corss-site Request Forgery), which we will discuss later in the course, is a typical example of the "Confused Deputy Problem".*

References