Next Week

- Exam briefing. Summary/Review.

- Demo of cybersecurity kill-chain on a virtual enterprise system by
  - Shen Jiamin
  - Arshdeep Singh Kawatra

Change log v1:
1. Slide 48. Correct the arrows.
2. Improving presentation here and there.
3. Remove references ([pf]) to the reference book.

see [PF] Chapter 2.2 (pg 72 – 85)
see [Andersen] Chapter 4 up to 4.2.4
read wiki http://en.wikipedia.org/wiki/File_system_permissions

Topic 9: Access Control

9.1 Access Control model

9.2 Access Control Matrix

9.3 Intermediate control

9.4 Unix

9.5 Elevated privilege and controlled invocation

9.6* Controlled invocation in Unix

"Does TikTok access the home Wi-Fi network?"

- Senator to Shou

*: These steps are complicated. Complexity is bad for security.
see https://www.schneier.com/news/archives/2012/12/complexity_the_worst.html

Summary & takeaways

- How access control is specified: operation, objects, subjects (principle).

Access Control Matrix, Intermediate control to simplify representation. Examples of intermediate control. Representation: ACL vs Capability.

- Guideline.

Security perimeter, security boundary, principle of least privilege, segregation, compartment, segmentation, firewall.

- How to share info across security perimeter: “Bridge” and “privilege elevation” (aka “privilege escalation” in the context of attack).

Example in Unix.

- File system: ACL. Intermediate control (user, group, world).
- *Objects*: program, resources (e.g keyboard, display, network) treated as files.
- *Subjects*: processes. Each process has (1) **real uid**, specifying its owner (principle) (2) **effective uid** that is used by the reference monitor to check permission.
- *Operations*: read, write, execute.
- Ring: root (0) vs user(1).
- Example of “bridge”.

Example: Apps in Android vs Users in Linux/Window

Maybe the following is a more sensible question.

- “Does TikTok discover other nodes in the local area network?”

In the context of Access control,

- “Does Android’s access control allows an app such as TikTok to discover all the nodes in the same local area network?”

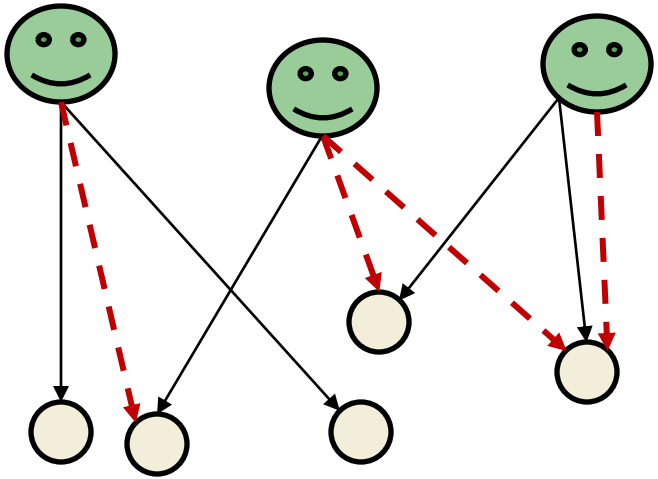
9.1 Access Control model

Access control model is relevant in many systems. E.g.

- File system (which files can a user read),
- Canvas (who can upload files),
- Facebook (which posts can a user read), etc

Access control

About controlling ***operations*** on ***objects*** by ***subjects.***



Subjects: users

Operations: read, write

Objects: file

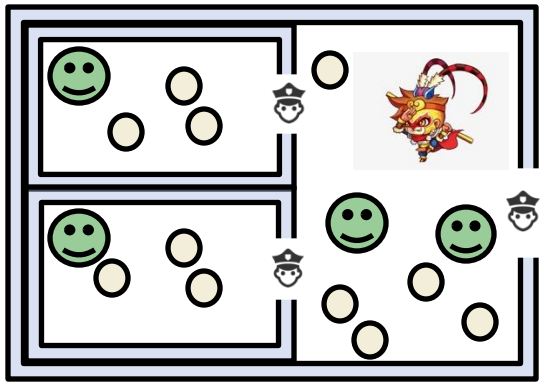
- IT and computer system handles resources such as files, printers, network, etc. Certain resources can only be accessed by certain entities. Access control is about controlling such accesses. Different application have different requirements. Generally, it is about “selective restriction of access to a place or other resource” (wiki). An access control system specifies and enforces such restriction on the subject, objects and actions.

- E.g. OS, Social media (e.g. Facebook), documents in an organization, physical access to different part of the building.

- Access control provides security perimeter which in turn facilitates segregation of accesses. Such segregation confines and localize damage caused by attacks.

Security Perimeter

- Access control setup security perimeter/boundary.
- With the boundary, malicious activities (e.g. malware) outside of the boundary would not affect resources within the perimeter. Furthermore, malicious activities within the boundary stays within the boundary.



compartmentalization

Design of the boundary is guided by

- Principle of least privilege
- Compartmentalization
- Defense in depth / Swiss Cheese Model
- Segregation of duties
- etc

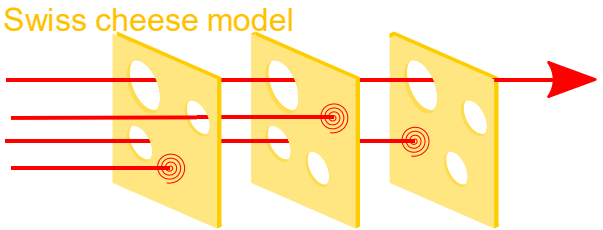


image from https://en.wikipedia.org/wiki/Swiss_cheese_model

Security Perimeter

Examples:

- A calculator apps shouldn't need to have access to the contact list to carry out its task. So, it is better not to grant the calculator apps access to the contact list. With that, even if the app is malicious or vulnerable, the Confidentiality/Integrity of contact still preserved. *(Principle of Least privilege)*
- A school website hosts two services: (1) course's fee payment and (2) exam result. With the perimeter between them, the exam result system would remain intact even if an SQL injection attack has been successfully carried out on the fee payment system. *(Compartmentalization)*
- Colonial Pipeline's ransomware attack compromised the IT's system that handle client's database. If proper perimeter being setup between the IT and OT (Operation Technology) system, the failure of the client's database system should not affect OT system that manages the fuel pipeline. *(Compartmentalization).*
- A company deploys a firewall separating their server from DMZ. An IDS (intrusion Detection System) reside in the firewall to detect malicious traffic based on known attack signature. In addition, processes in the server are monitored for abnormal behavior. Attack that evade the firewall might be caught by the process monitor, and vice versa. *(Defence in depth) (Swiss Cheese Model)*
- A company keeps backup of is business-critical data. The company implements a policy: *a single person must not have access to both the production copy and the backup copy.* Assigning different components to different person is aka *Segregation of Duties*. The goal is to eliminate *single-point-of-failure.* With that, a single rogue system admin (insider) is unable to corrupt all. *(Segregation of duties).*

Eg of Security Perimeter on Android

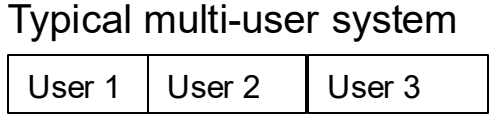
"The purpose of a *permission* is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

A central design point of the Android security architecture is that no app, by default, has permission to perform any operations that would adversely impact other apps, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another app's files, performing network access, keeping the device awake, and so on."

From https://developer.android.com/guide/topics/permissions/overview

In Android, each app is associated with a "manifest file" which listed down permissions the app wishes to have. (this is the requests by the app, not the actual granted permission). During runtime, the os would grant the request based on default setting or prompt the user.

Essentially:



no boundary between two apps running by the same user.

Eg on Android: Implications

- Comparing two applications (e.g. a game ***G*** and an image editing tool ***T***), both implemented for Window and Android. Alice installed both ***G*** and ***T*** in a desktop running Window, and a devices running Android.

- Question:
  - Can ***T*** read/write files generated by ***G***? Example, the game G generated an image and the user wanted to edit the image using T.

    (yes for window, no for android)
  - While ***G*** is executing, can ***T*** access the memory space of ***T***?

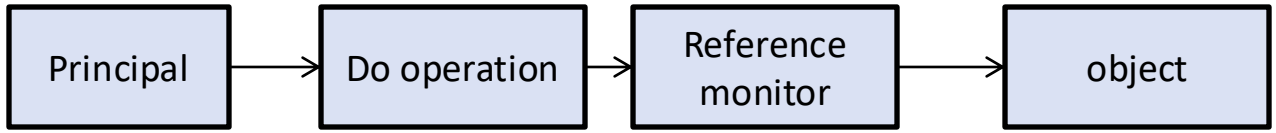
    (no for both)
  - Can ***T*** modify the installation of ***G***?

    (No, but yes for early versions of Window)
  - In Android/ios, how to pass information/data from one app to another? (user need to explicitly give permission by indicating in "share")

- ***Isolation*** of apps is a key design consideration in Android.

- Newer versions of OS (e.g. Window and Mac) starts to impose boundary between apps.

Definitions: Principal/Subject, Operation, Object



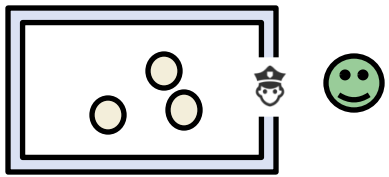
- A *principal* (or *subject*) wants to access an *object* with some *operation*. The *reference monitor* either grants or denies the access.

E.g.

Canvas:
- a ***student*** wants to ***submit*** a ***forum post***.
- a ***TA*** wants to ***read*** the ***grade of student in another group***.

File system:
- a ***user*** wants to ***delete*** a ***file***.
- a ***user*** wants to ***change the mode*** of a ***file*** so that it can be read by another user Bob.

OS:
- An ***app*** (e.g. touch light) wants ***access*** to the ***phone.***
- An ***app*** wants to ***read files*** generated by another app.



Some terminologies

Principals vs Subjects:

- ***Principals***: the human users.
- ***Subjects***: The entities in the system that operate on behalf of the principals.

Accesses to objects can be classified to the following:

- ***Observe*****:** e.g. Reading a file. (Canvas, downloading a file from workbin)
- ***Alter***: e.g. writing a file, deleting a file, changing properties. (Canavs: uploading a file to the workbin).
- ***Action*****:** e.g executing a program.

Definitions: Ownership

Every object has an “owner”.

Who decides the access rights to an object?

There are two options:

(1) The *owner* of the object decides the rights. (known as ***discretionary access control***)

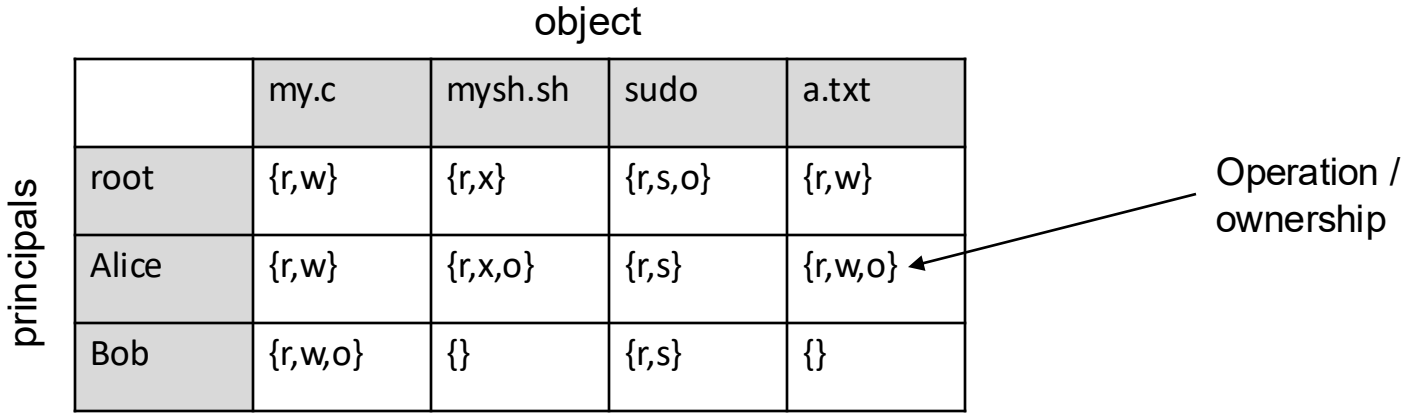
(2) A system-wide policy decides. (known as ***mandatory access control***).

Mandatory access control are strict rules that everyone must follow.

9.2 Access Control Matrix

Access Control Matrix

How to specify the access right of a particular principal to a particular object? Using a table.



	my.c	mysh.sh	sudo	a.txt
root	{r,w}	{r,x}	{r,s,o}	{r,w}
Alice	{r,w}	{r,x,o}	{r,s}	{r,w,o}
Bob	{r,w,o}	{}	{r,s}	{}

Although the above *access control matrix* can specify the access right for all pairs of principals and objects, the table would be very large, and thus different to manage.

Hence, it is seldom explicitly stored.

r:read, w:write, x:execute, s: execute as owner, o: owner

Access Control List (ACL) & Capabilities

The access control matrix can be represented in two different ways: ACL or capabilities.

ACL:

An ACL stores the access rights to an object as a list.

Capabilities:

A subject is given a list of capabilities, where each capability is the access rights to an object.

> "a capability is an unforgeable token that gives the possessor certain rights to an object"

	my.c	mysh.sh	sudo	a.txt
root	{r,w}	{r,x}	{r,s,o}	{r,w}
Alice	{}	{r,x,o}	{r,s}	{r,w,o}
Bob	{r,w,o}	{}	{r,s}	{}

- ACL

my.c	→ (root, {r,w}) → (Bob, {r,w,o})
mysh.sh	→ (root, {r,x}) → (Alice, {r,x,o})
sudo	→ (root, {r,s,o}) → (Alice, {r,s}) → (Bob, {r,s})
a.txt	→ (root, {r,w}) → (root, {r,w,o})

- Capability

root	→ (my.c, {r,w}) → (mysh.sh, {r,x}) → (sudo, {r,s,o}) → (a.txt, {r,w})
Alice	→ (mysh.sh, {r,x,o})→ (sudo, {r,s}) → (a.txt, {r,w,o})
Bob	→ (my.c, {r,w,o}) → (sudo, {r,s})

For ACL, it is difficult to obtain the list of objects a particular subject has access to. Conversely, for capabilities, it is difficult to get the list of subjects who have access to a particular object.

- To illustrate, note that Unix file system employs ACL. Suppose the system admin wants to generate the list of files that user **alice0012** has “r” access to. How to generate this list?

9.3 Intermediate Control

We want an intermediate control that is ***fine grain*** (e.g. in Facebook, allow user to specify which friend can view a particular photo) and yet ***easy to manage***.

It is not practical for an owner to specific each single entries in the access control matrix. So, we need some ways to simplify the representation. One method is to "group" the subjects/objects and define the access rights on the group. This is called intermediate control.

Most access control systems have such grouping. Next few slides are examples.

Intermediate Control: Grouping

Example:

Unix File system. In Unix file permission, subjects are divided into groups. Unix file permission uses ACL.

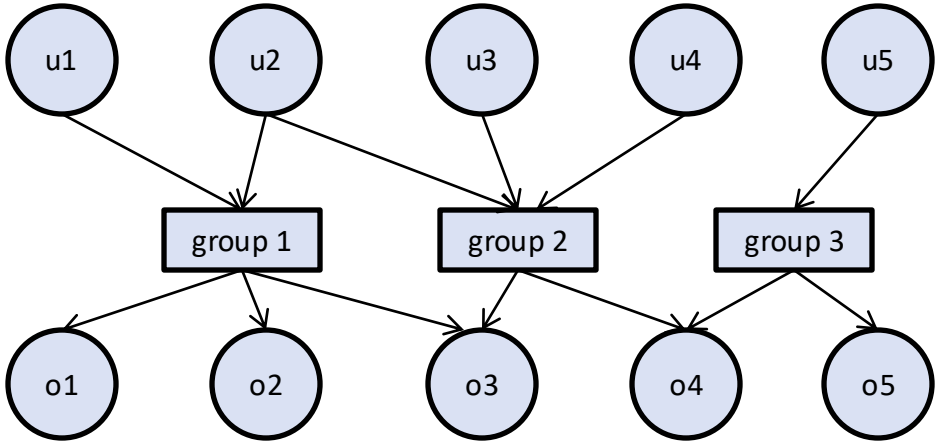
For each object, the owner specific the rights for

- *owner,*
- *group,*
- *world (everyone).*

(for the precise definition, see next section)

```
--w-r--r--   1 alice  staff       3 Mar 13 00:27 temp
```

Example of grouping



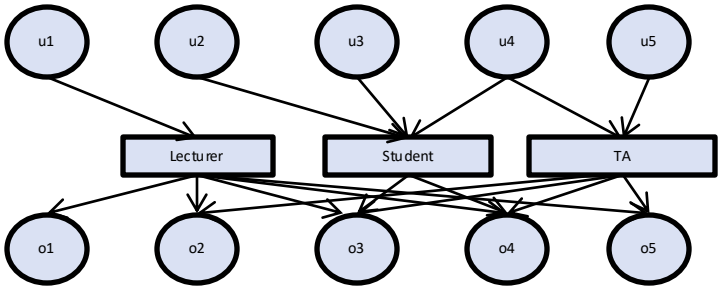
- In Canvas, project groups can be created by lecturer. Objects created in a group can only be read by members in the group + lecturer.
- In Unix, groups can only be created by root. The groups information is stored in the file

```
/etc/group
```

Intermediate control: Role-based access control

The grouping can be determined by the “role” of the subject.

A role associates with a collection of procedures. To carry out these procedures, access rights to certain objects are required.



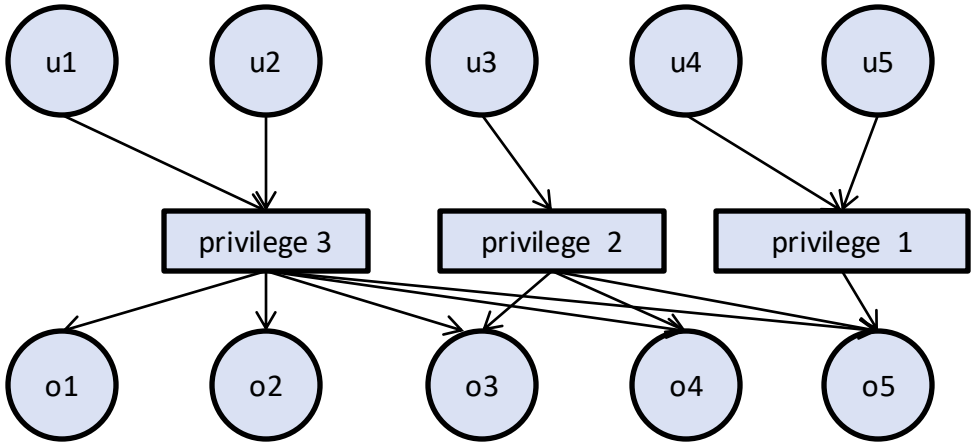
E.g. In Canvas, there are predefined rights for different roles: “Lecturer”, “TA” and “Student”. When Alice enrolled to CS2107 as student, her rights are inherited from the role “Student”. When Alice volunteered as TA in CS1010, her rights are inherited from the role “TA” in CS1010.

To design the access right of a role, we should follow the ***least privilege principle***, i.e. access rights that are not required to complete the role will not be assigned.

e.g. Consider Canvas’s gradebook. The task of a student Teaching Assistance include entering the grade for each students. So we should give the TA “write” access to the gradebook so that the TAs can complete their task. Should we give the TAs the right to delete a gradebook? Since this access right is not required for the TA to complete their task, by the ***least privilege principle,*** the TA should not be given this access right.

Intermediate Control: privileges

We sometime use the term *privilege* to describe the access right. Privilege can also be viewed as an intermediate control. It can be represented by a number, e.g. 1,2,3. if a subject can access an object, another subject with higher privilege can also access the object.



Intermediate control: Protection rings

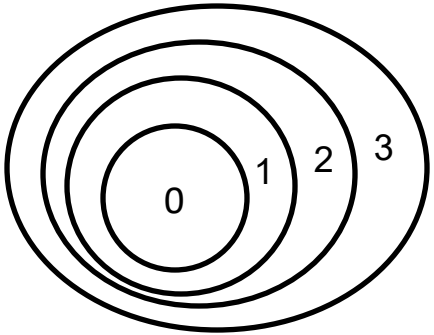
In OS, “privilege” is often called protection rings. “Privilege” and “Rings” are essentially the same but with different names.

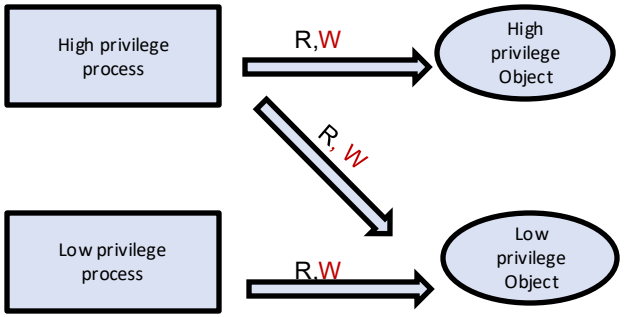
In protection rings, each object (data) and subject (process) is assigned a number. Whether a subject can access an object is determined by their respective assigned number. Object with smaller number are more important. If a process is assigned a number *i*, we say that the process runs in ring *i*. We call processes with lower ring number as having “***higher privilege***”.

A subject cannot access (both read/write) an object with smaller ring number.

Unix has only 2 rings, *superuser* and *user*.

e.g.





Two non-intuitive examples of intermediate control: Bell-LaPadula vs Biba

In protection rings, a subjects has read/write access to objects that are classified with the same or lower privilege. Are there reasonable alternatives?

Here are two well-known models: Bell-LaPadula and Biba. Although they are rarely implemented as-it-is in computer system, they illustrate the differences of Integrity vs Confidentiality in access control.

In both models, objects and subjects are divided into linear levels.

Level 0, level 1, level 2, ...

higher level corresponds to higher "security".

(The numbering is opposite of protection ring. This is just a different choice of notations).

Bell-LaPadula Model (for data confidentiality)

Read https://en.wikipedia.org/wiki/Bell%E2%80%93LaPadula_model

Restrictions imposed by the Bell-Lapadula Model:

The following restrictions are imposed by the model:

- **no read up**: A subject does not have read access to object in higher level. This prevent a lower level from getting info in the higher level.

- **no write down**: A subject does not have append-right to object in lower level. This prevents a malicious insider from passing information to lower levels. *(e.g. a clerk working in the highly classified department is forbidden to gossip with other staff).*

For "Confidentiality".

(A subject can append to objects at higher security level. Is it possible that, by appending to an object, one could distort its original content? Yes. See e.g. in renegotiation attack.)

Biba Model (for process integrity)

Restrictions imposed by the Biba Model:

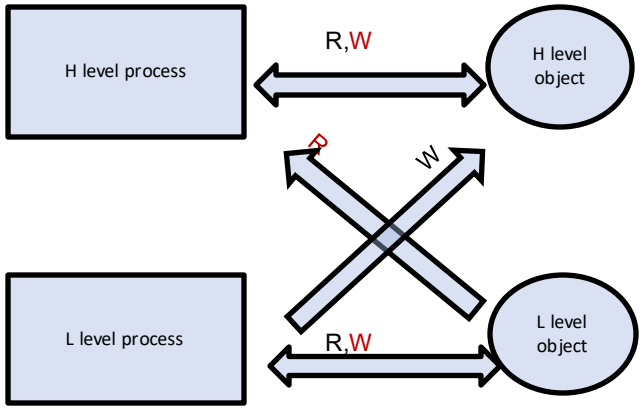
The following restrictions are imposed by the model:

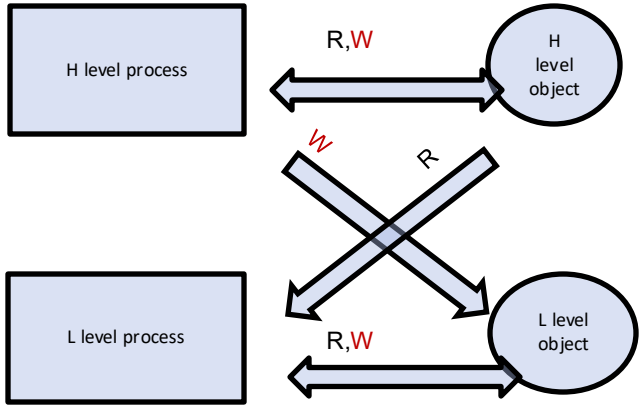
- **no write up**: A subject does not has "write" access to objects in higher level. This prevent a malicious subject from poisoning upper level data, and thus ensure that a process will not get compromised by lower level subjects.
- **no read down**: A subject does not has read access to objects lower level. This prevents a subject from reading data poisoned by lower level subjects.

For "Integrity".

If a model imposes both Biba and Bell-LaPadula, subjects can only read/write to objects in the same level (not practical).

Direction of information flow





Bell-LaPadula (confidentiality)
No “sensitive” information leaking down

no read-up
no write-down

Biba (Integrity)
No ”malicious” information going up.

no write-up
no read-down

9.4 Unix File system

To get Unix or Unix-like environment in Windows:

- (Virtual Machine as in the assignment): Install a hypervisor or virtual machine monitor (VMM) such as **VirtualBox** (https://www.virtualbox.org), or **VMWare** (https://www.vmware.com). Then install Linux (e.g. **Ubuntu desktop**).

- A Unix-like environment in Windows is cygwin
  https://www.cygwin.com/

- Another method: Bash shell in Window 10.
  https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/

Unix file system Access control.

In Unix, objects includes files, directories, memory devices and I/O devices. All these resources are treated as files.

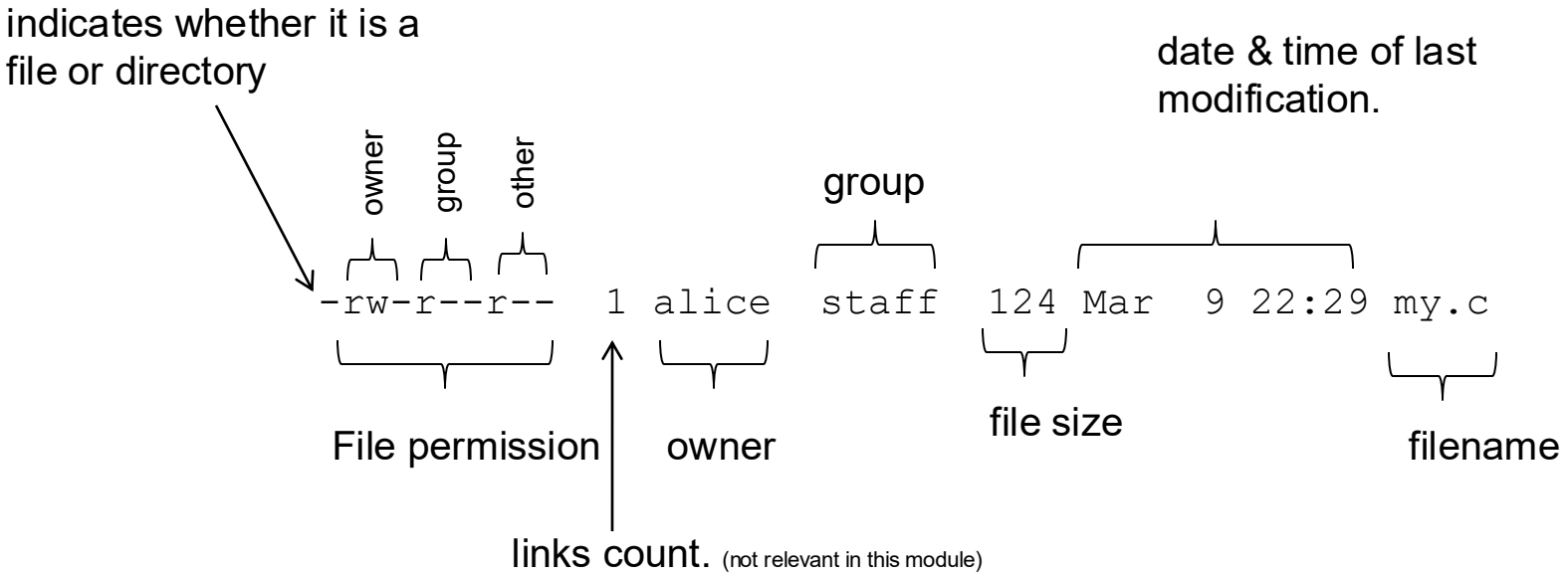
read wiki http://en.wikipedia.org/wiki/File_system_permissions

```
%ls -al
-r-s--x--x  1 root  wheel   164560 Sep 10  2014 sudo
-rwxr-xr-x  2 root  wheel    18608 Nov  7 06:32 sum
-rw-r--r--  1 alice staff      124 Mar  9 22:29 myprog.c
lr-xr-xr-x  1 root  wheel        0 Mar 12 16:29 stdin
```

Question: what are the files in the following directories?

```
/dev
/dev/stdin
/dev/stdout
/dev/urandom
```

File system permission



The file permission are grouped into 3 triples, that define the *read*, *write*, *execute* access for *owner, group, other (also called the "world").*

A '-' indicates access not granted. Otherwise
r: read
w: write (including delete)
x: execute (s: allow user to execute with the permission of the owner)

Principals, Subjects

- Principals are *user-identities* (UIDs) and *group-identities* (GIDs)

- Information of the user accounts are stored in the "password" file

  /etc/passwd

e.g.

```
root:*:0:0:System Administrator:/var/root:/bin/sh
```

read wiki page for details of these fields.
https://en.wikipedia.org/wiki/Passwd

- The subjects are processes. Each process has a process ID (PID). (e.g. in Unix, the commend ps –alx display a list of processes).

Optional Side Remarks on the "Password file" /etc/passwd

- The file is made world-readable because the account information in /etc/passwd should be accessible by other users. *In earlier version of Unix*, the "*" in the file was the hashed password H(pw), where H() is some cryptographic hash, and pw the password of the user. Hence, previously all users can see the hashed passwords of others.

- With knowledge of the hashed password, the attackers can carry out offline dictionary attack. To prevent that, it is now replaced as "*", and the actual hashed passwords are stored somewhere else and not world-readable. (You may wonder why can't the we completely remove the field. This is for backward compatibility. If the field is completely removed, some existing programs might not work.)

- Hence, unlike in original design of Unix-based system, the file /etc/passwd is now not storing information of the password. It is just due to the legacy issue that it keeps the name "passwd".

superuser(root)

- A special user is the superuser, with UID 0 and usually with the username root. All security checks are turned off for root.

(Unix's protection rings consists of 2 rings: superuser, user)

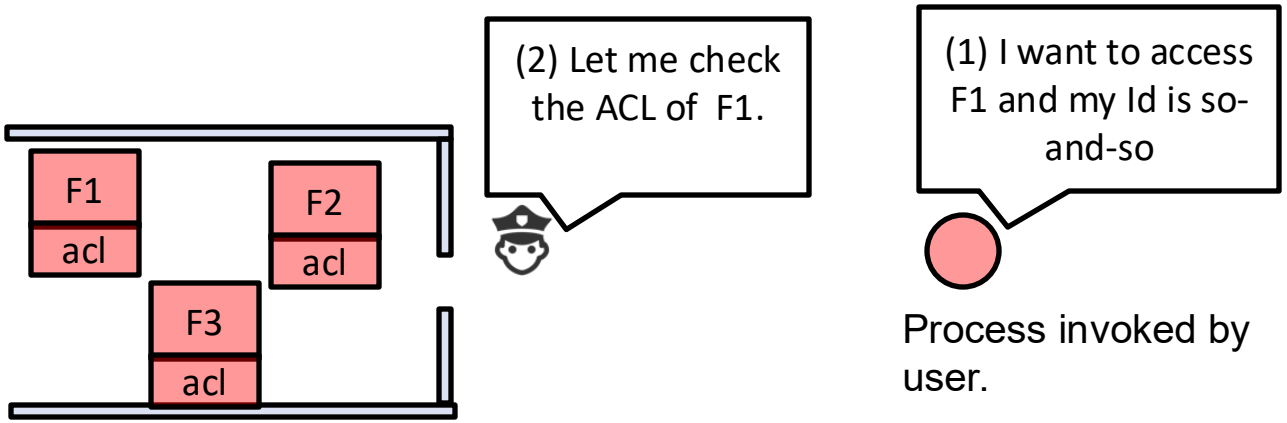
Checking rules for file access

- The objects are files. Each file is associated with a 9-bit permission. Each file is owned by a user, and a group.

- When a user (subject) wants to access a file (object), the following are checked in the order:
  1. If the user is the owner, the permission bits for ***owner*** decide the access rights.
  2. If the user is not the owner, but the user's group (GID) owns the file, the permission bits for ***group*** decide the access rights.
  3. If the user is not the owner, nor member of the group that own the file, then the permission bits for ***other*** decide.

  The owner of a file, or superuser can change the permission bits.

Unix's Access Control and Reference Monitor



checking rules.

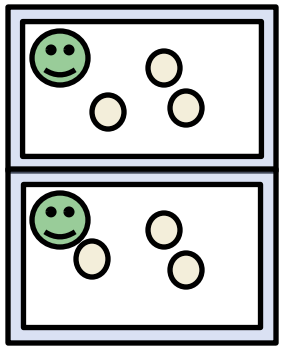
- When a user (subject) wants to access a file (object), the following are checked, in the following order:
  1. If the user is the owner, the permission bits for ***owner*** decide the access rights.
  2. If the user is not the owner, but the user's group (GID) owns the file, the permission bits for ***group*** decide the access rights.
  3. If the user is not the owner, nor member of the group that own the file, then the permission bits for ***other*** decide.

  The owner of a file, or superuser can change the permission bits.

9.5 Controlled Invocation & privilege elevation

Two separated environments ensure segregation for security. However, to be useful, we need interaction between them.

Solution: no free-flow of interactions but “controlled” interactions.



Controlled invocation

Eg in Unix: Some sensitive resources (such as network port 0 to 1023, printer) should be accessible only by the superuser. However, users sometime need those resources.

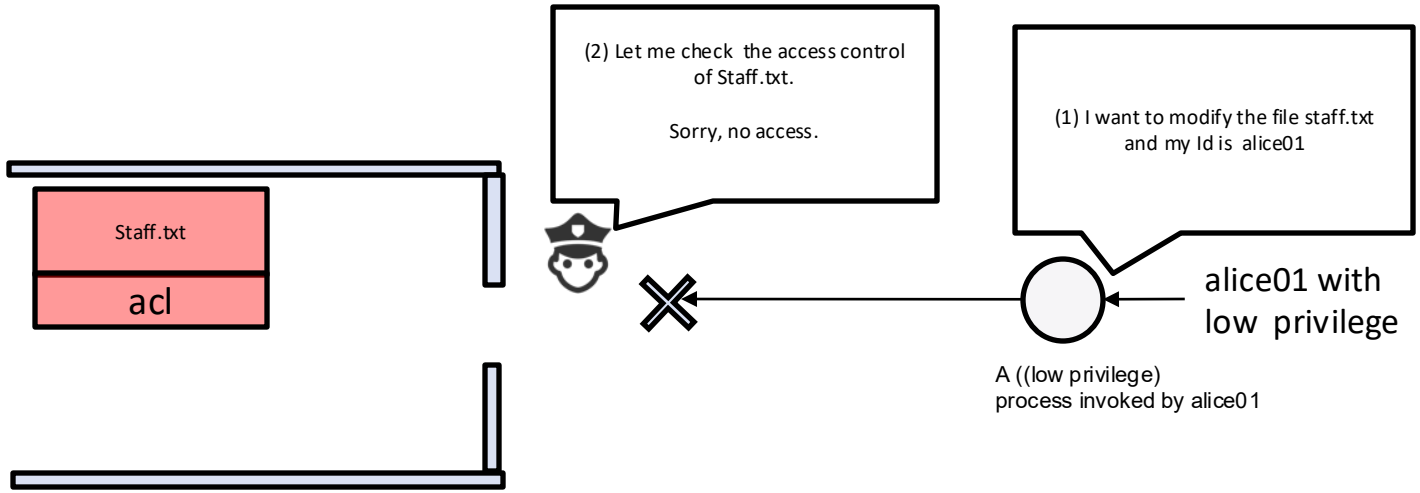
Eg: Consider a file **F** that contains home addresses of all staffs. Clearly, we cannot grant any user to read **F**. However, we must allow a user to read/modify his/her address and thus need to make it readable/writeable to that user. The polarized setting where either a process can read or cannot read a file would get stuck!

Solution: ***controlled invocation***.

- The system provides a predefined set of applications that have access to **F.**
- These application is granted "elevated privilege" so that they can freely access the file, and any user can invoke the application. Now, any user can access **F** via the application.
- The programmer who write the application bear the responsibility to make sure that the application only performed intended limited operation.

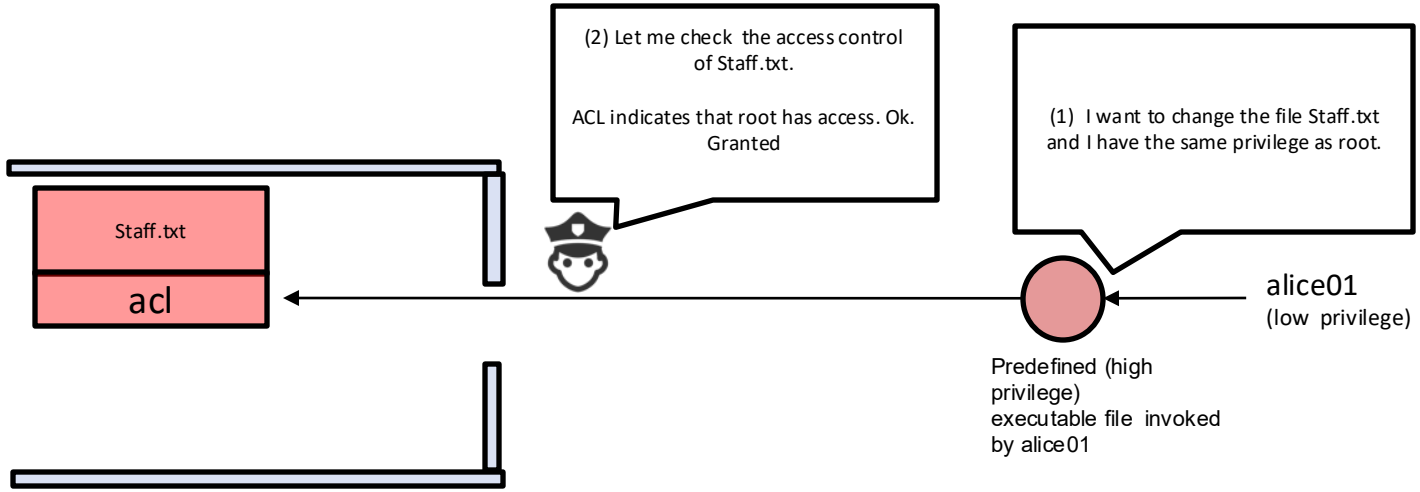
Without controlled escalation

alice01 doesn't has access right to Staff.txt.
Any processes (subject) invoked by alice01 inherit alice01's right



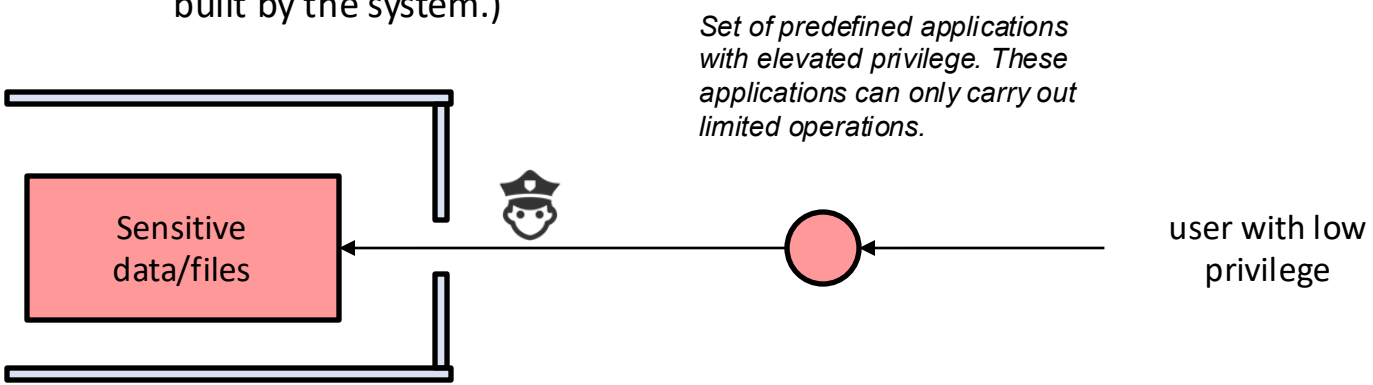
With Controlled escalation

There is a set of predefined applications with “elevated” privilege. A normal user alice01 can’t create applications with high privilege. However, any user can invoke these predefined applications.



Bridges with Elevated Privilege

- We can view the predefined application as a predefined "bridges" for the user to access sensitive data. (note that the "bridge" can only be built by the system.)



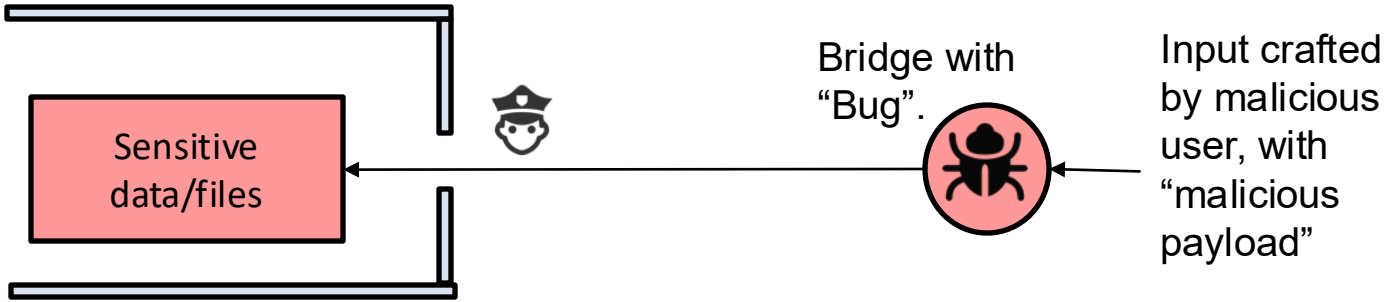
Firewall ensure that users can't directly send query to the SQL database server. Users has controlled access to SQL server.

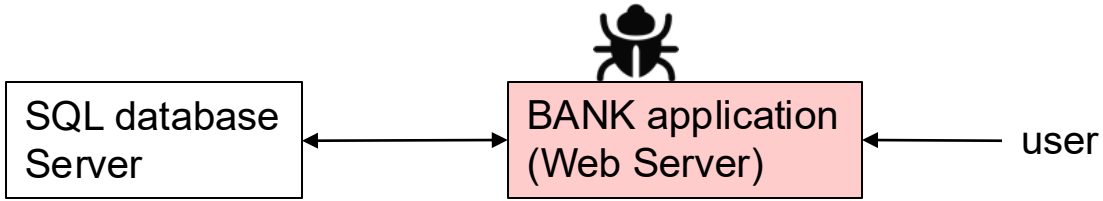
- Suppose a “bridge” is not implemented correctly and contains exploitable vulnerabilities. In some vulnerabilities, an attacker can trick the bridge to perform “illegal” operations not expected by the programmer/designer. This would have serious implication, since the process is now running with “***elevated privilege***”.

- Attacks of such form is also known as “***privilege escalation***”.

> ***“Privilege escalation*** *is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.”*
>
> - https://en.wikipedia.org/wiki/Privilege_escalation

Bugs in bridge leads to Privilege escalation.





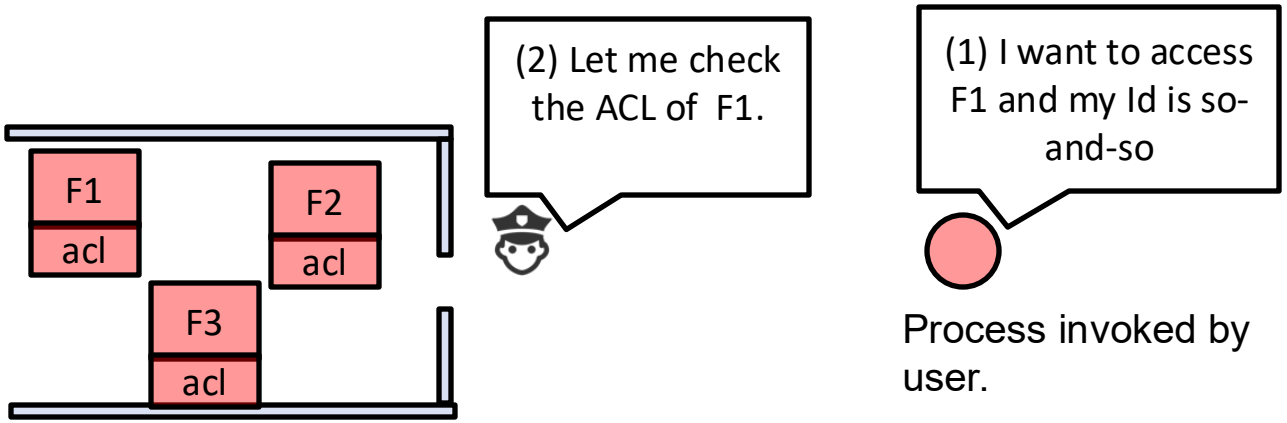
A bug in BANK that allows the user to send in arbitrary query. (SQL injection)

9.6 Controlled Invocation in UNIX

Real UID, Effective UID, privilege escalation

The definitions, steps are very complicated with many exceptions. Complexity is bad for security. Programmers often get confused which leads to buggy implementation.

Let's recap Unix's Access Control and Reference Monitor



checking rules.

- When a user (subject) wants to access a file (object), the following are checked, in the following order:
  1. If the user is the owner, the permission bits for ***owner*** decide the access rights.
  2. If the user is not the owner, but the user's group (GID) owns the file, the permission bits for ***group*** decide the access rights.
  3. If the user is not the owner, nor member of the group that own the file, then the permission bits for ***other*** decide.

  The owner of a file, or superuser can change the permission bits.

More definitions before we move on: Process and Set userID (SUID)

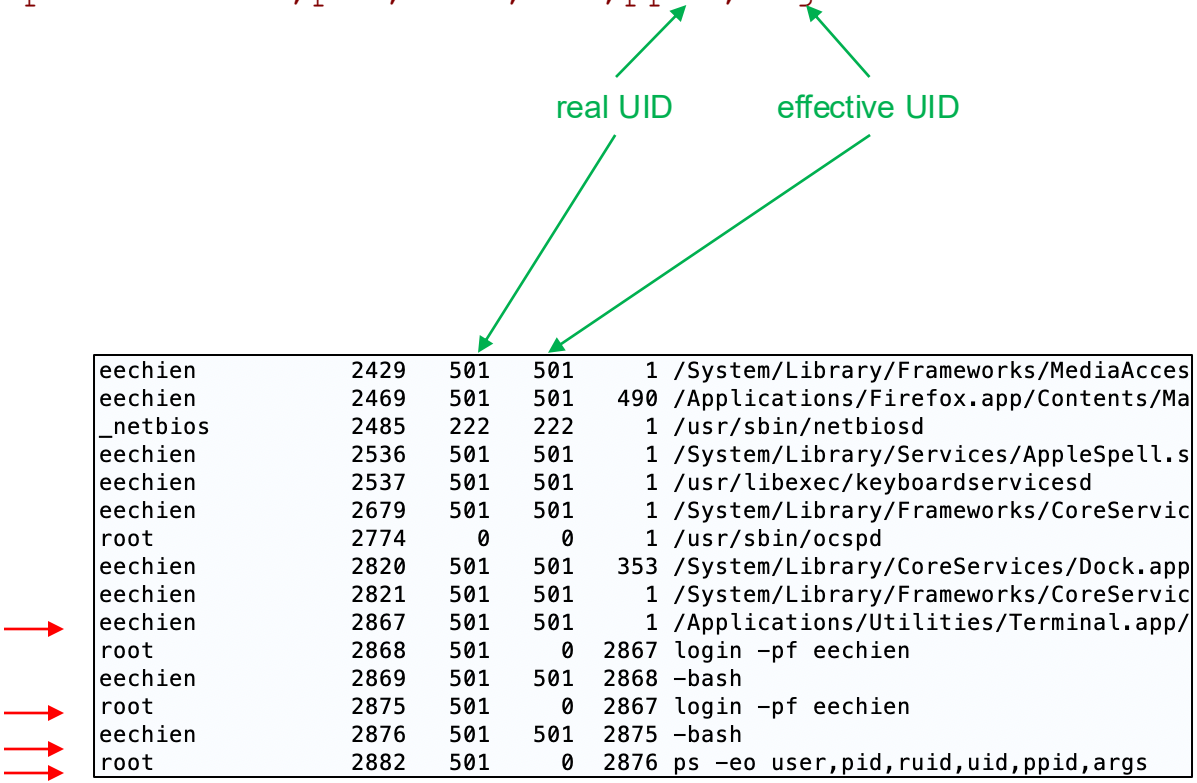
(the unix command `ps` list the current running processes)

- A process is a subject.
- A process has an identification (PID). New process can be created by executing a file or by "forking" an existing process.
- A process is associated with a ***Real UID*** and an ***Effective UID***.
- The ***real UID*** is inherited from the user who invokes the process. For e.g. if the user is `alice`, then the real UID is `alice`.

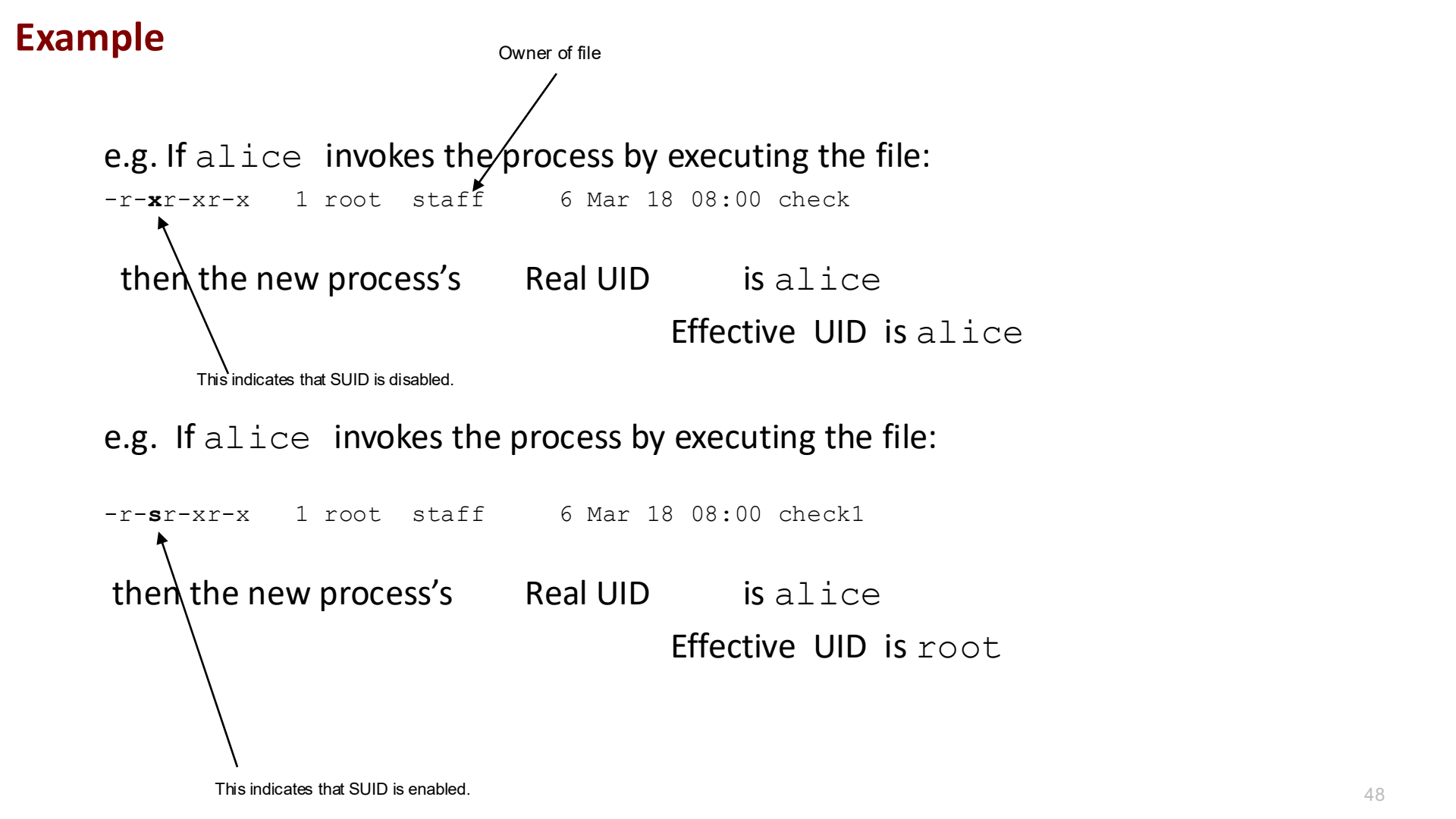
- Processes can be created by executing a file. Each executable file has a SUID flag. There are two cases:
  - If the Set User ID (SUID) is disabled (the permission will be displayed as "**x**"), then the process' ***effective UID*** is same as real UID.
  - If the Set User ID (SUID) is enabled (the permission will be displayed as "**s**"), then the process' ***effective UID*** is inherited from the UID of the *file's owner*.

Demo:

```
ps -eo user,pid,ruid,uid,ppid,args
```



Example


Owner of file
e.g. If alice invokes the process by executing the file:
-r-xr-xr-x 1 root staff 6 Mar 18 08:00 check
then the new process's Real UID is alice
Effective UID is alice
This indicates that SUID is disabled.
e.g. If alice invokes the process by executing the file:
-r-sr-xr-x 1 root staff 6 Mar 18 08:00 check1
then the new process's Real UID is alice
Effective UID is root
This indicates that SUID is enabled.

When a process (subject) wants to read/write/execute a file (object)

When a process wants to access a file, the effective UID of the process is treated as the “subject” and checked against the file permission to decide whether it will be granted or denied access.

E.g consider a file own by the root.

```
-rw-------   1 root  staff     6 Mar 18 08:00 sensitive.txt
```

- If the effective UID of a process is `alice` then the process is denied access to the file.

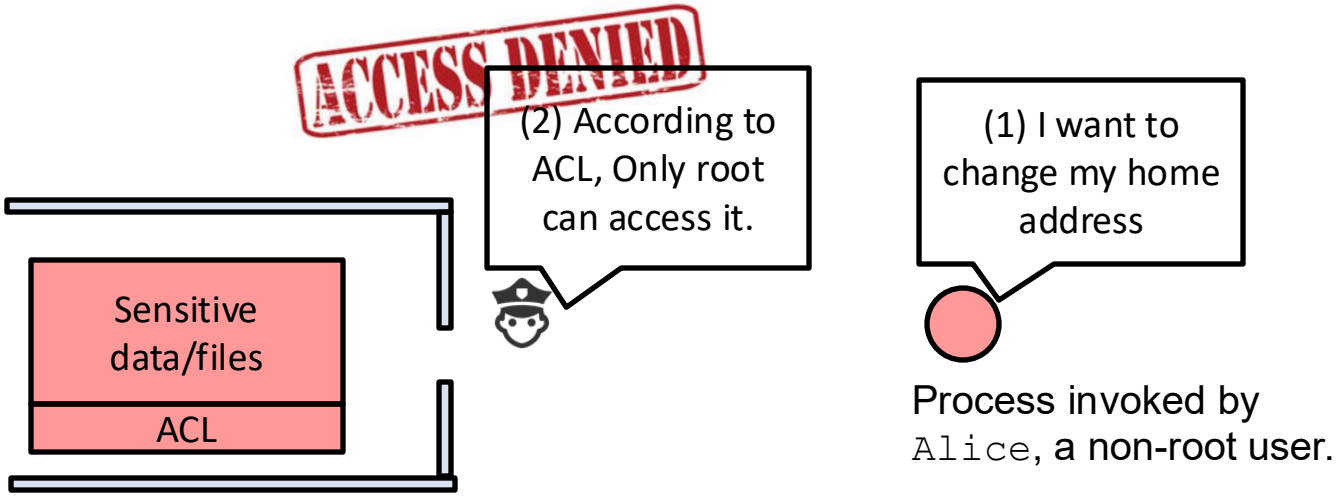
- If the effective UID of a process is `root`, then the process is allowed to access the file.

Use Case Scenario of “s” (SUID)

- Consider a scenario where the file `employee.txt` contains personal information of the users.
- This is sensitive information, hence, the system administrator set it to non-readable except by root:

```
-rw-------   1 root  staff     6 Mar 18 08:00 employee.txt
```

- However, users should be allowed to self-view and even self-edit some fields (for e.g. postal address) of their own profile.
  Since the file permissible is set to “-” for all users (except root),
  a process created by any user (except root) cannot read/write it.
- Now, we are stuck: there are data in the file that we want to protect, and data that we want the user to access.
- *What can we do?*

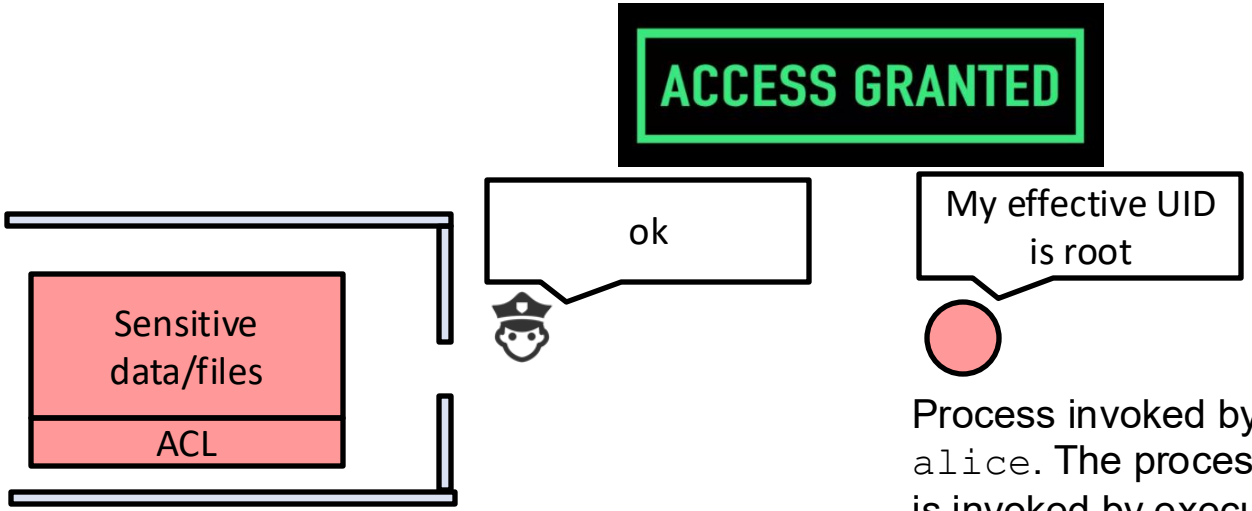

ACCESS DENIED
(2) According to ACL, Only root can access it.
(1) I want to change my home address
Sensitive data/files
ACL
Process invoked by Alice, a non-root user.

Solution

- Create an executable file `editprofile` owned by `root`:

```
-r-sr-xr-x   1 root  staff     6 Mar 18 08:00 editprofile
```

- The program is made world-executable so that *any* user can execute it.
- Furthermore, the permission is set to be “s”:
  when it is executed, its effective UID will be “root”
- Now, if `alice` executes the file, the process’ real UID is `alice`, but its effective UID is `root`.
  Following the checking rule, this process can now read/write the file `employee.txt`



Process invoked by `alice`. The process is invoked by executing a program created by the root, with permission “s”.

Summary: When SUID is Disabled

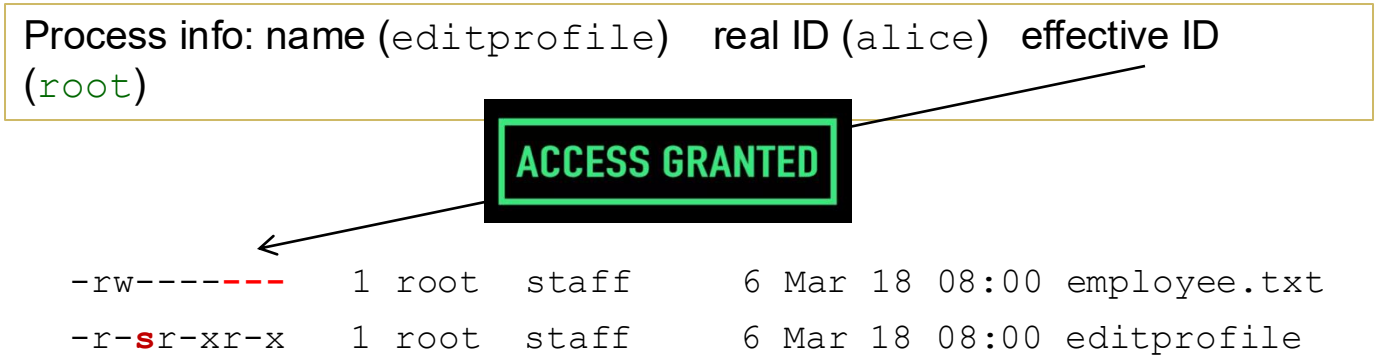
- If the user `alice` invokes the executable, the process will have its ***effective ID*** as `alice`
- When this process wants to read the file `employee.txt`, the OS (reference monitor) will deny the access

Process info: name (`editprofile`) real ID (`alice`) effective ID (`alice`)


```
-rw-------   1 root  staff      6 Mar 18 08:00
employee.txt
-r-xr-xr-x   1 root  staff      6 Mar 18 08:00 editprofile
```

Summary: When SUID is Enabled

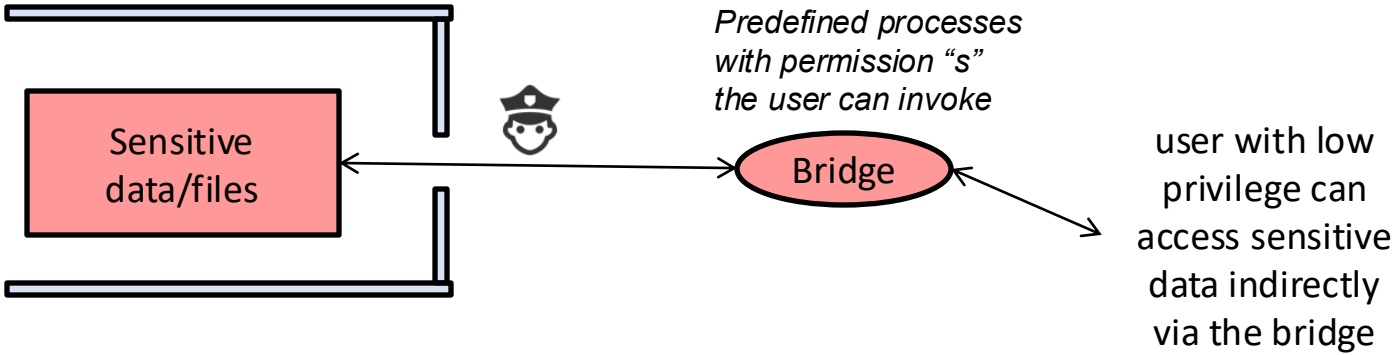
- But if the permission of the executable is “s” instead of “x”, then the invoked process will has `root` as its effective ID
- Hence the OS grants the process to read the file
- Now, the process invoked by `alice` can access `employee.txt`



Elevated Privilege

- In this example, the process `editprofile` is ***elevated*** to superuser (i.e. root), so that it can access sensitive data. We can view the elevated process as the interfaces where a user can access the "sensitive" information.
  - They are the predefined "bridges" for the user to access the data.
  - The "bridge" can only be built by the root.

- These bridges solve the problem. However, it is important that these "bridges" are correctly implemented and do not leak more than required.



- If the bridge is not built securely, there could be *privilege escalation* attack.