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

Lecture 6: Access Control

6.1 Access Control model

6.2 Access Control Matrix

6.3 Intermediate control

6.4 Unix

6.5 Elevated privilege and controlled invocation

6.6*   Controlled invocation in Unix

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

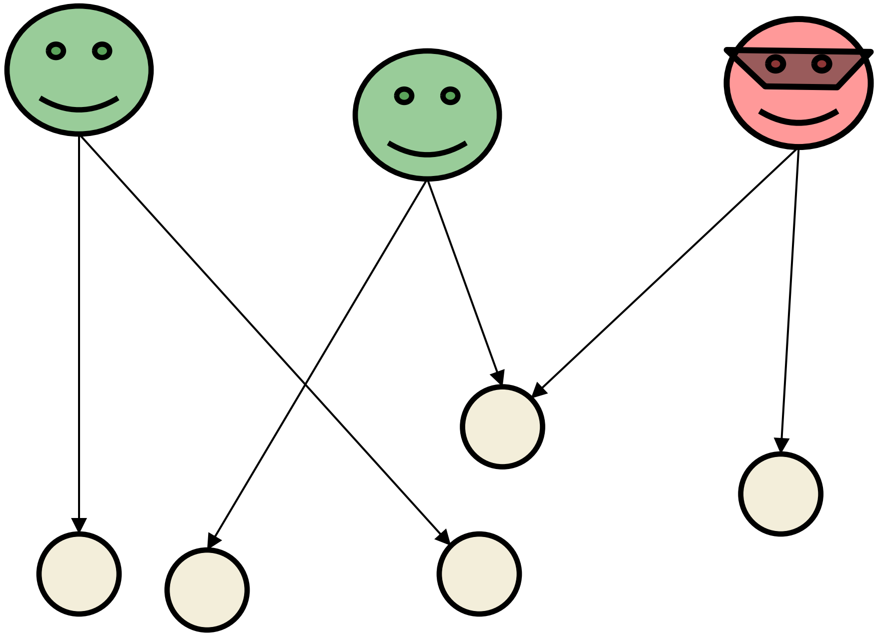
6.1 Access Control model

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

- file system (which files can a user read),
- LumiNUS (who can upload files to the workbin),
- Facebook (which posts can a user read), etc

- 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 (document classified as "restricted", "confidential", "secret", etc), physical access to different part of the building.
- Access control provides security perimeter/boundary, and segregation of the environment. Such segregation confines and localize damage caused by attacks.

- We want to restrict ***operations*** on ***objects*** by ***subjects.***



subjects.

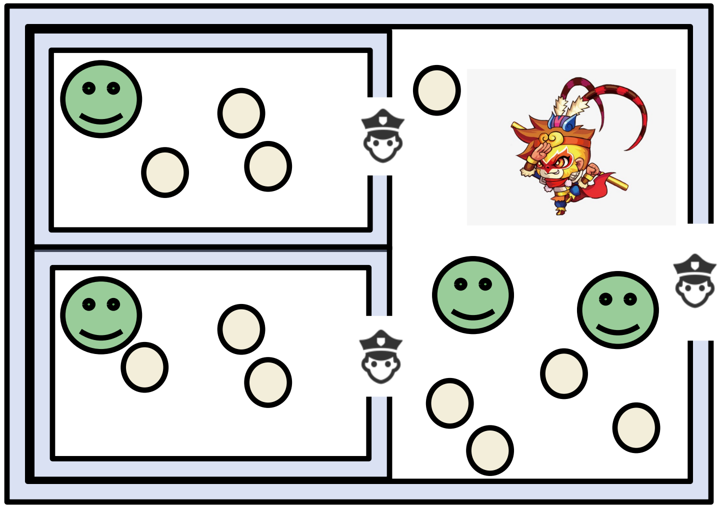
operations

objects

Security Boundary

- Access control gives a well-defined security perimeter/boundary.

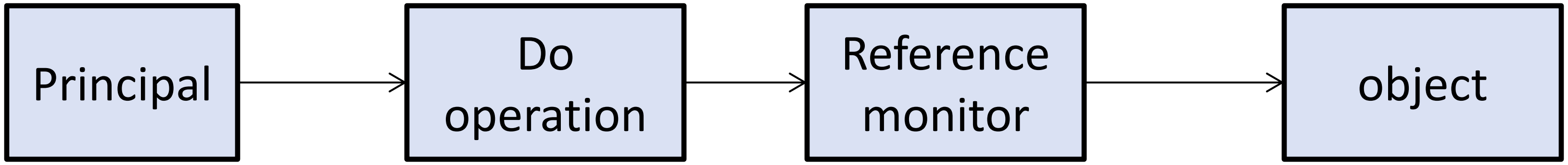
- With the boundary, even if there are malfunctions outside of the boundary, resources within the perimeter remain intact. Furthermore, effect of malware in the boundary stay within the boundary. E.g
  - SQL injection attack target at the SQL Database management system. The OS password management should remain intact even if an SQL injection attack has been successfully carried out.
  - Even if a camera app is a malware, the confidentiality of contact list still preserved. (boundary between contact list and camera)



Design of the boundary is guided by

- Principle of least privilege
- Compartmentalization
- Layered defense
- …

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.

LumiNUS:

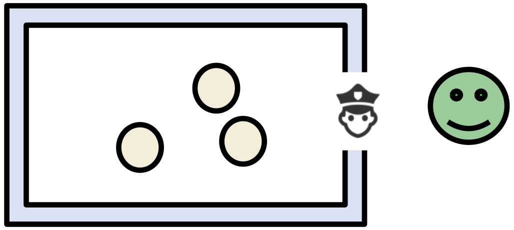
- a ***student*** wants to ***submit*** a ***forum post***.
- a ***TA*** wants to ***read*** the ***grade of student in another group***.

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.



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. (Luminus, downloading a file from workbin)
- ***Alter***: e.g. writing a file, deleting a file, changing properties. (Luminus, 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.

6.2 Access Control Matrix

Access Control Matrix [PF] pg 79

How do we specific the access right of a particular principal to a particular object? Using a table.

object

principals

	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. (see [PG]pg82 on the description of "capability")

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

(Question, does Unix file system adopt ACL or capabilities?)

	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, in unix, suppose the system admin wants to generate the list of files that user **alice0012** has “r” access to. How to quickly generate this list?)

For both methods, the size of the lists can still to be too large to manage.   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.

6.3 Intermediate Control

The main challenging is on how to specify a policy that is ***fine grain*** (e.g. in Facebook, allow user to specify which friend can view a particular photo), meets the requirements of ***security boundary***, and yet ***easy to manage***.

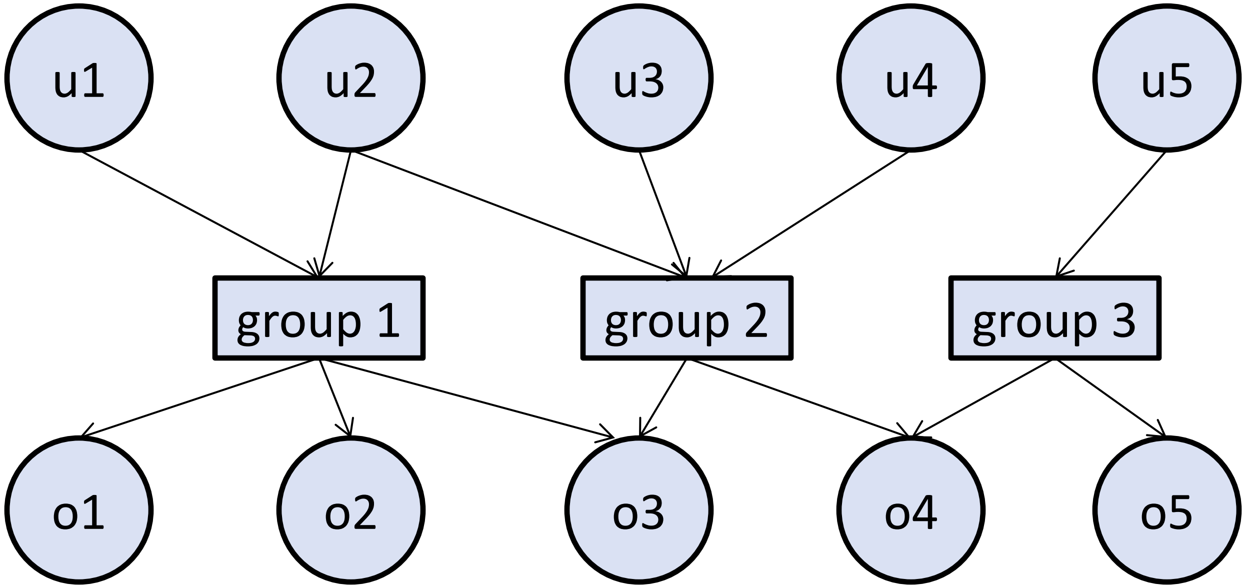
Intermediate Control: Group

In Unix file permission, the ACL specifies the rights for the *owner, group, world.*

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

Subjects in a same group have the same access rights.   Some systems demand that a subject can be in a single group, but some don’t have the restriction.

Question: *Is it possible that an owner does not has read access, but others have?* Strangely, yes.

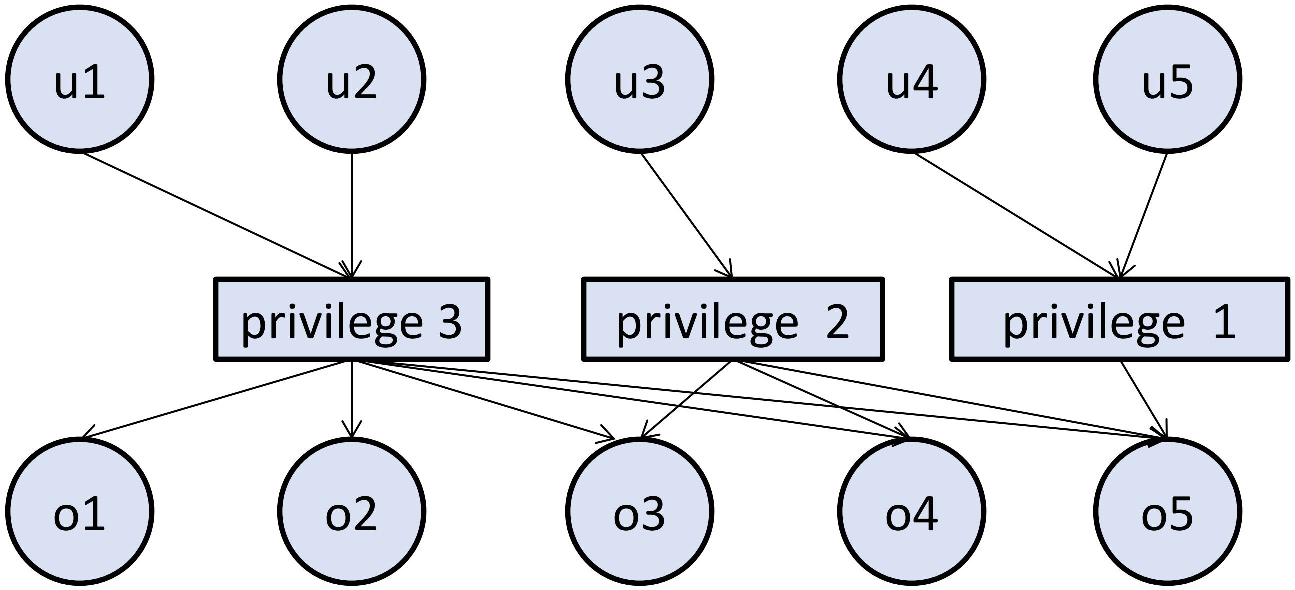


- In Luminus, project groups can be created by lecturer. Objects created in a group can only be read by members in the group + lecturer.
- In Unix, groups can only be created by root. The groups information is stored in the file

```
/etc/group
```

Intermediate Control: privileges

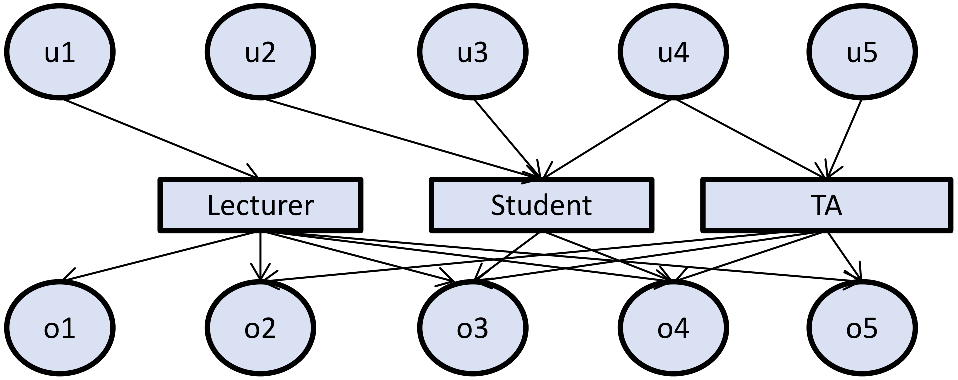
We sometime use the term *privilege* to describe the access right. Privilege can also be viewed as an intermediate control.



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. In order to carry out these procedures, access rights to certain objects are required.



E.g. In LumiNUS, 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 Luminus’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.

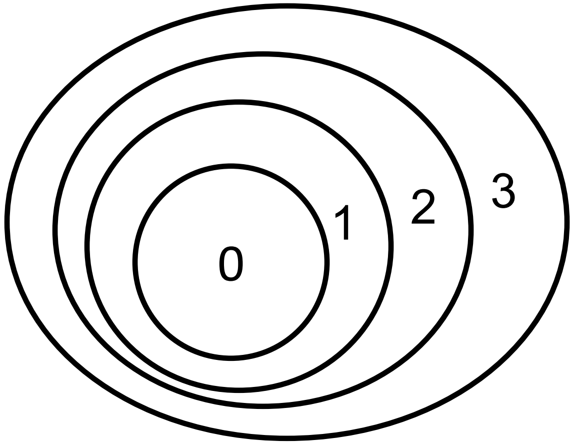
Intermediate control: Protection rings

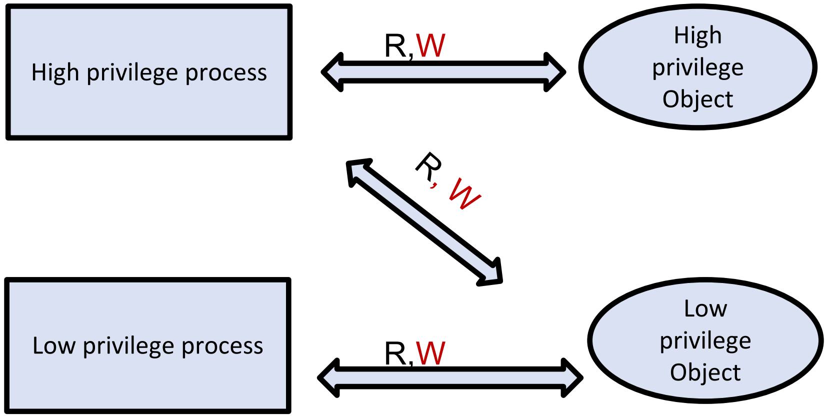
Here, each object (data) and subject (process) is assigned a number. Whether a subject can access an object can be determined by their respective assigned number. Object with smaller number are more important.   If a process is assigned a number *i*, we say that the process runs in ring *i*.   Very often,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

A subject cannot access (both read/write) an object with smaller ring number. It can only do so if its privilege is "elevated".

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

e.g.





High privilege process

R,W

High privilege Object

R, W

Low privilege process

R,W

Low privilege Object

Two Examples of intermediate control: Bell-LaPadula vs Biba

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

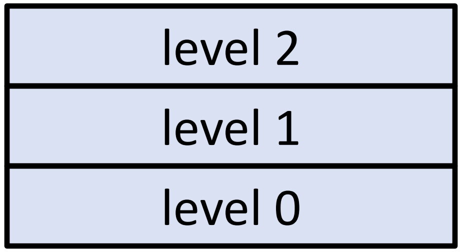
Here are two well-known models: Bell-LaPadula and Biba. Although they are rarely implemented as-it-is in computer system, they serve as a good guideline.

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).



level 2

level 1

level 0

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 has only read access to objects whose security level is below the subject's current clearance level. This prevents a subject from getting access to information available in security levels higher than its current clearance level.

- **no write down**: A subject has append (can add but not delete or modify) access to objects whose security level is higher than its current clearance level. This prevents a subject from passing information to levels lower than its current level. *(e.g. a clerk working in the highly classified department should not gossip with other staff, in order to prevent leakage).*

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:

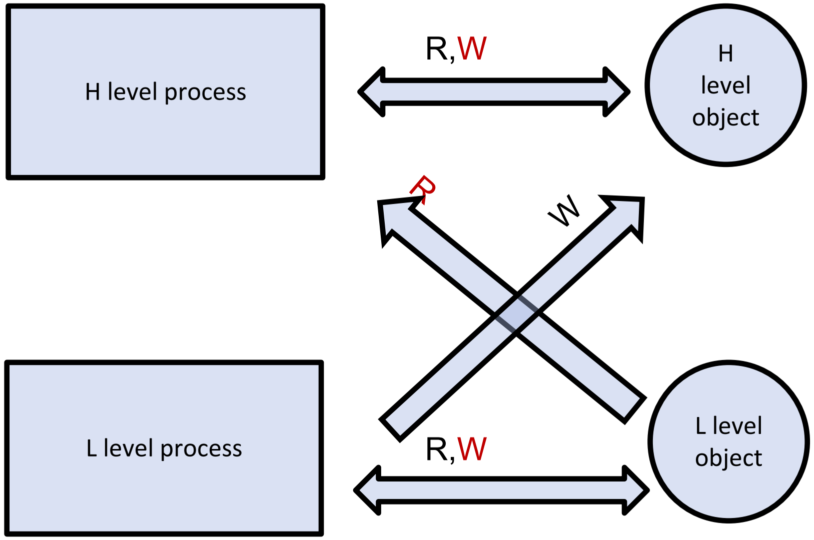
The following restrictions are imposed by the model:

- **no write up**: A subject has only write access to objects whose security level is below the subject's current clearance level. This prevents a subject from compromising the integrity of objects with security levels higher than its current clearance level.
- **no read down**: A subject has only read access to objects whose security level is higher than its current clearance level. This prevents a subject from reading forged information from levels lower than its current level.

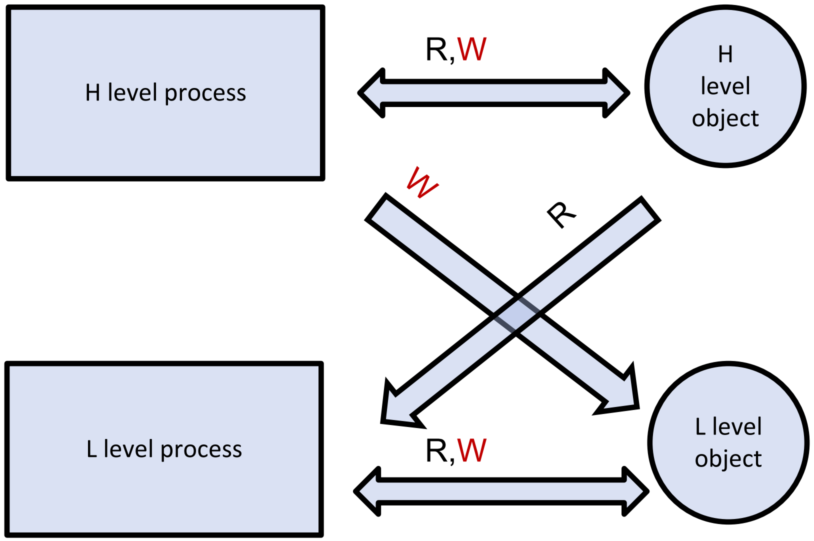
For "Integrity".

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 information coming down



Biba (Integrity)
No information going up.

6.4 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/

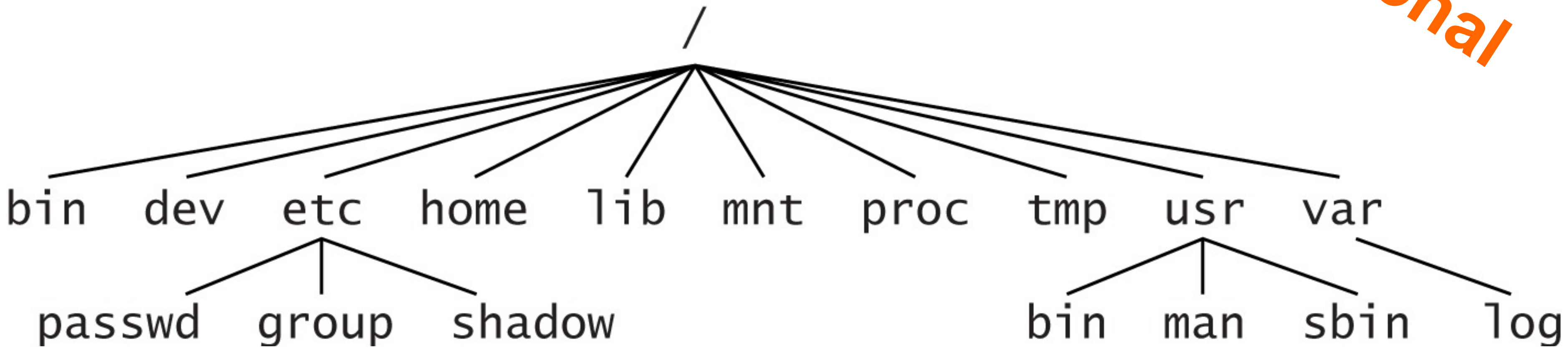
UNIX/Linux: Some Background

Optional

- History from 1970s
- Many versions:
  Solaris, AIX, Linux, Android, OS X + iOS
- Linux is open source (`http://www.kernel.org`)
- Many available tools (usually also open source)
- Many Linux distributions (distros):
  - Vary in setup, administration, kernel.
  - A popular choice: Ubuntu desktop

UNIX/Linux File System Structure

Optional



`/etc/passwd`: user database

`/etc/shadow`: user database containing hashed user passwords.

`/etc/group`: group database

`/bin/ls`

`man`

UNIX Manual Pages

Optional

- UNIX documentation using the **man** command
  - man is your friend!
  - Note: small variations in man with different UNIX

```
$ man ls
$ man man
```

- Organized in sections

```
$ man printf
$ man 1 printf
$ man 3 printf
```

- A free good resource to learning Linux commands: W. Shotts, "*The Linux Command Line*", `http://linuxcommand.org`

Unix Access control.

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

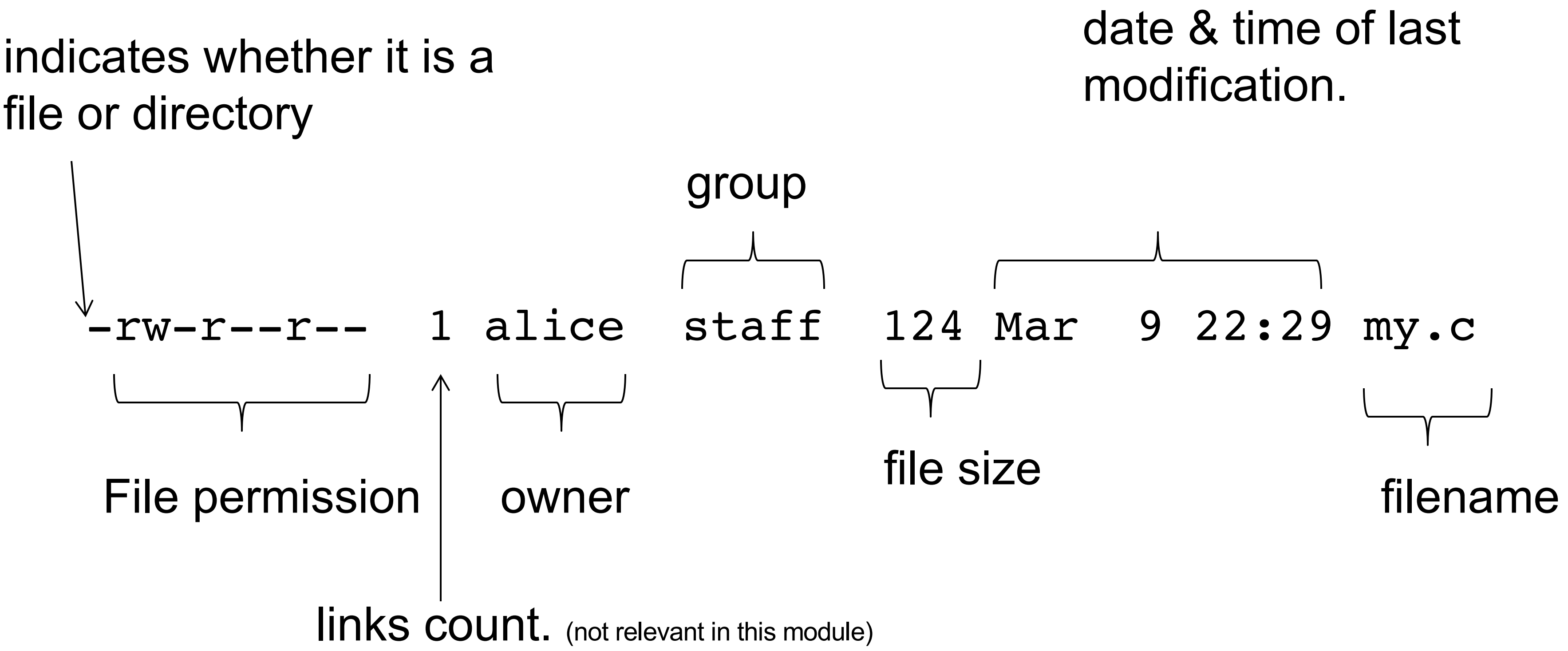
read wiki http://en.wikipedia.org/wiki/File_system_permissions

```
%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



indicates whether it is a file or directory

group

date & time of last modification.

```
-rw-r--r--  1 alice  staff  124 Mar  9 22:29 my.c
```

File permission

owner

file size

filename

links count. (not relevant in this module)

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
```

Earliest versions of unix place the hashed password here. Still maintain the field for backward compatibility.

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).

Remarks on the Password file

- The file is made world-readable because some information in /etc/passwd is needed by non-root program. *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 have access to the hashed passwords of others.

- The availability of the hashed password allows attackers to carry out offline dictionary attack. To prevent that, it is now replaced as "*", and the actual password is stored somewhere else and not world-readable (actual location depends on different versions of Unix).

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)

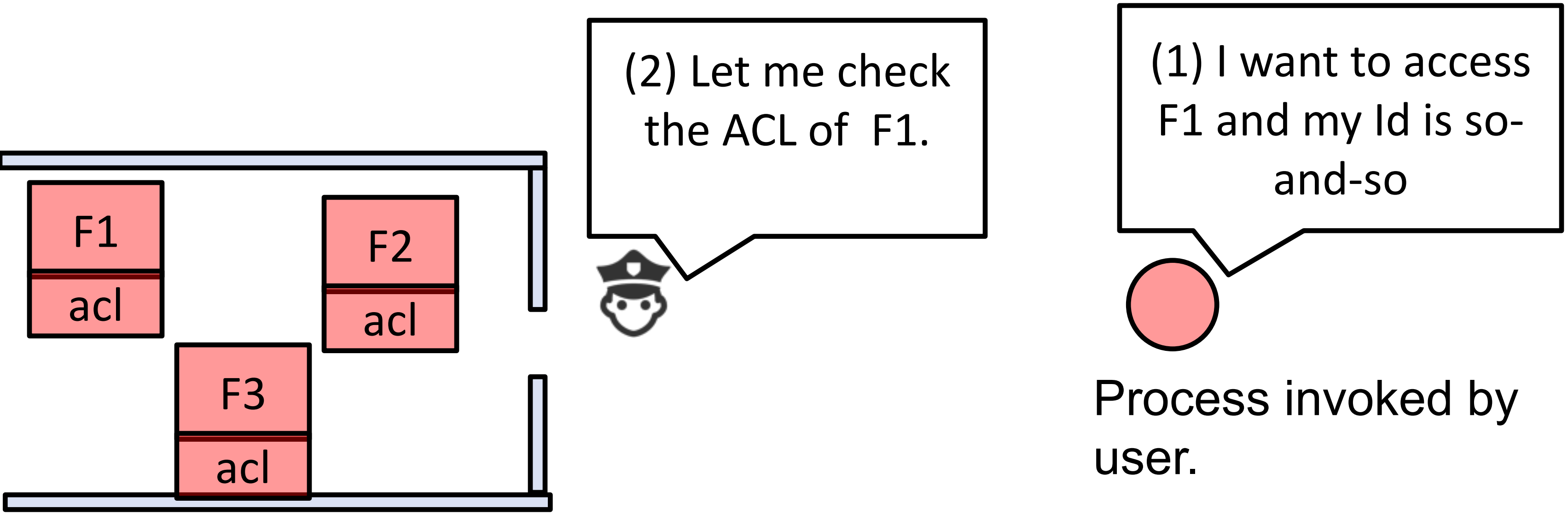
Checking rules for file access

- The objects are files. Recall that 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



(2) Let me check the ACL of F1.

(1) I want to access F1 and my Id is so-and-so

Process invoked by user.

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.

6.5 Controlled Invocation & privilege elevation

Controlled invocation

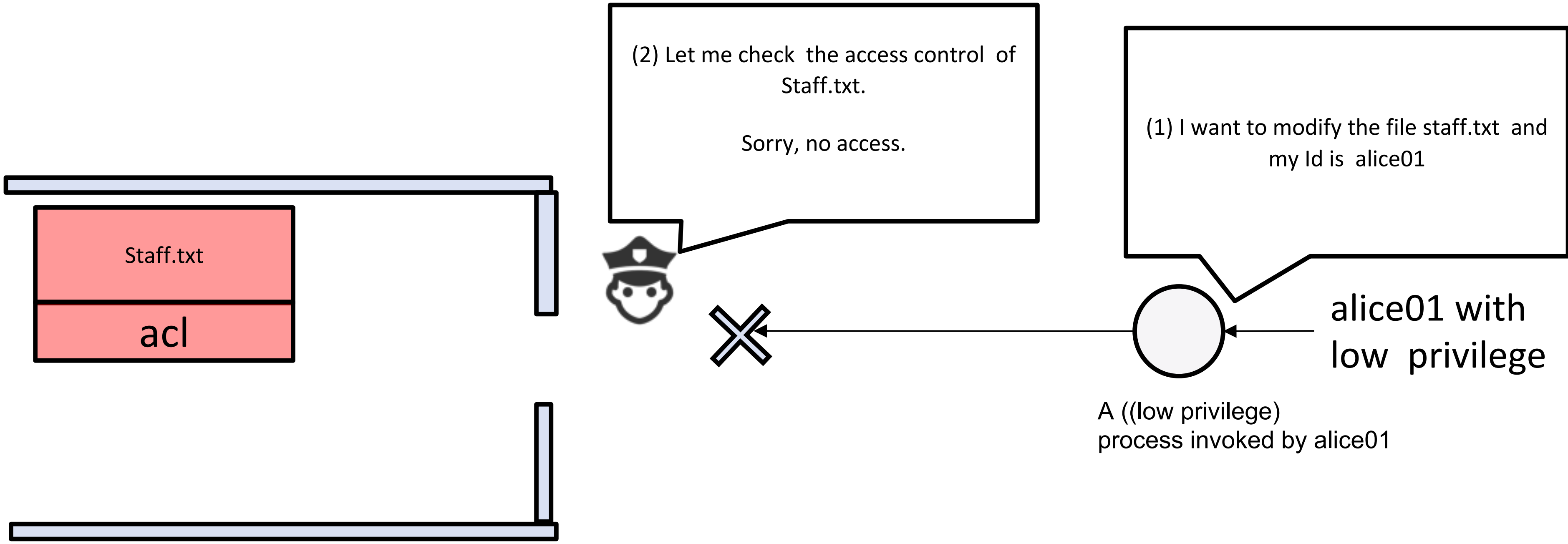
Certain resources in Unix can only be accessed by superuser (for e.g. listen at the trusted port 0-1023, password file). However, sometime a user need those resources for certain operation.

Another example: 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. So, we are stuck!!

Solution: ***controlled invocation***. The system provides a predefined set of applications that have access to **F.** Such applications are very restrictive and can performed limited and controlled operations on **F.** These application is granted "elevated privilege" to access the file. Any user can call these applications to operate on **F.** This elevated application bear the responsibility to make sure that the user stay within the boundary.

Without controlled escalation

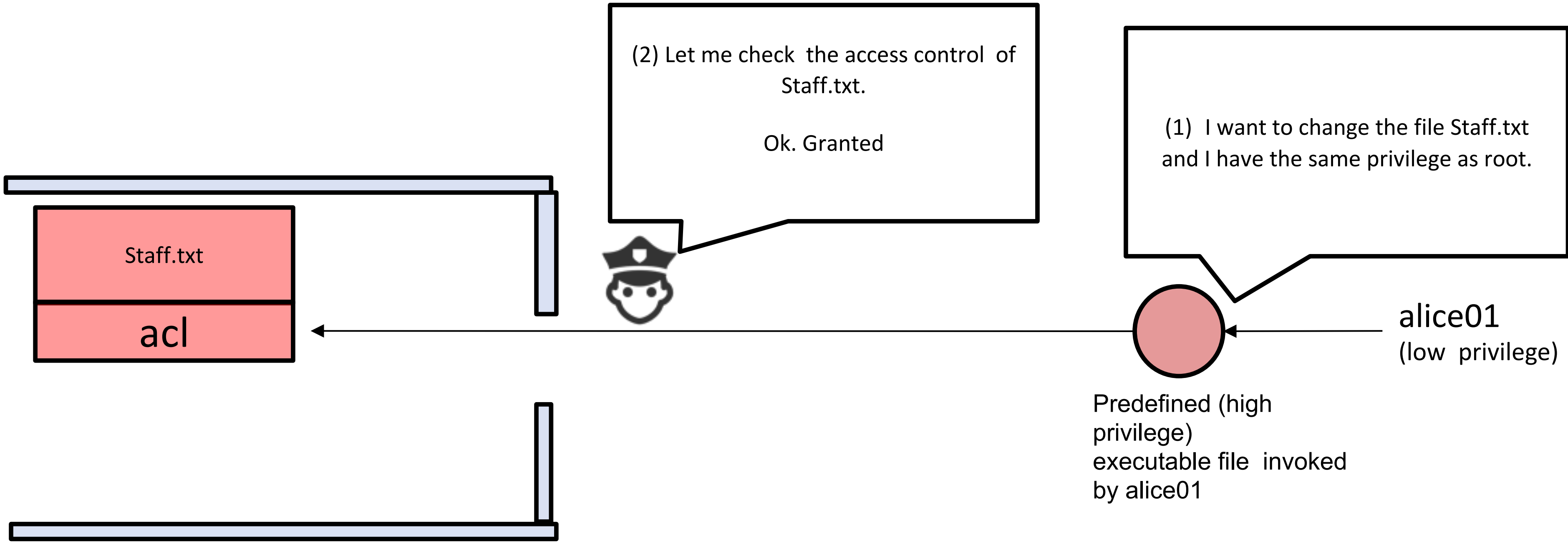
alice01 doesn’t has access right to Staff.txt.
Any processes (subject) invoked by alice01 inherit alice01’s right


(2) Let me check the access control of Staff.txt.
Sorry, no access.
(1) I want to modify the file staff.txt and my Id is alice01
Staff.txt
acl
alice01 with low privilege
A ((low privilege) process invoked by alice01

Staff.txt contains contacts of all staff members.

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.



(2) Let me check the access control of Staff.txt.

Ok. Granted

(1) I want to change the file Staff.txt and I have the same privilege as root.

Staff.txt

acl

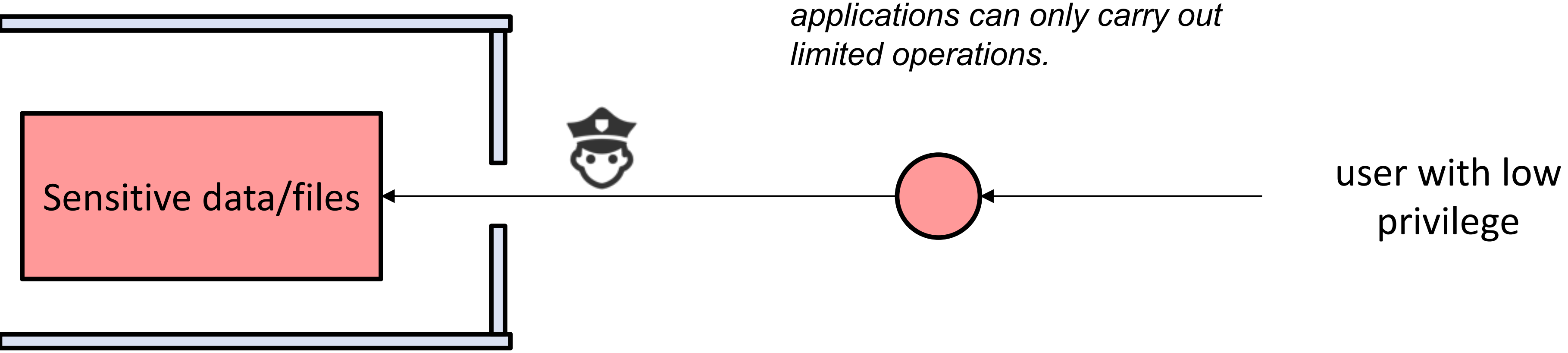
alice01
(low privilege)

Predefined (high privilege) executable file invoked by alice01

Bridges with Elevated Privilege

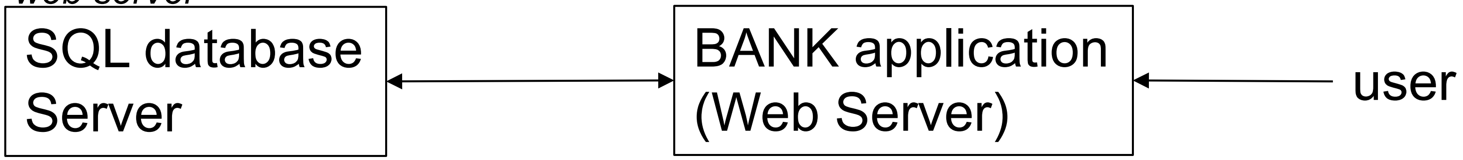
- We can view the predefined application as a predefined “bridges” for the user to access sensitive data. (note that the “bridge” can only be built by the system.)

Set of predefined applications with elevated privilege. These applications can only carry out limited operations.



SQL Server: I will process any query sent from the web server

BANK: I will only issue certain type of queries.



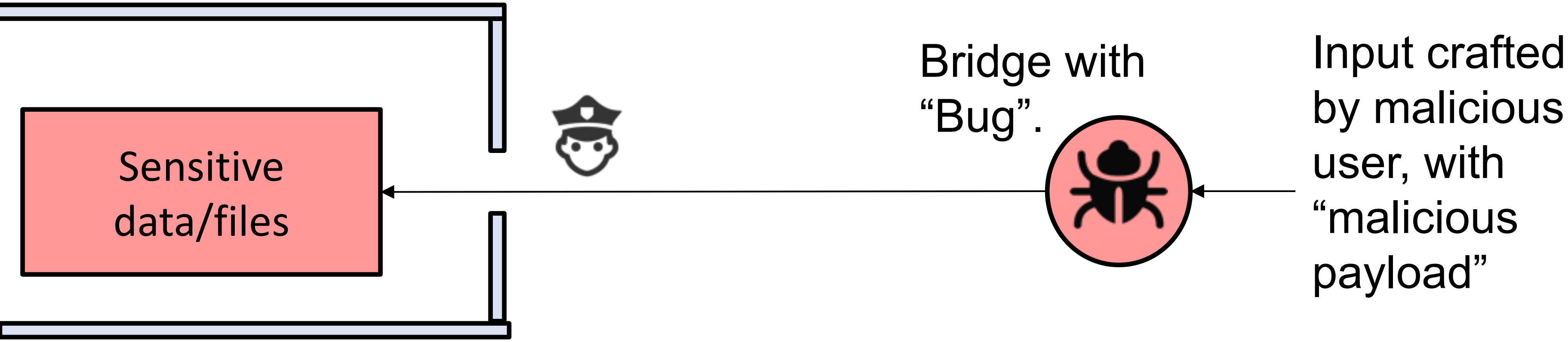
Firewall ensure that user can’t directly send query to the SQL database server. User can only indirectly send query to SQL server via website. *(what if there is a bug in BANK that allows the user to send arbitrary query?)*

- 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

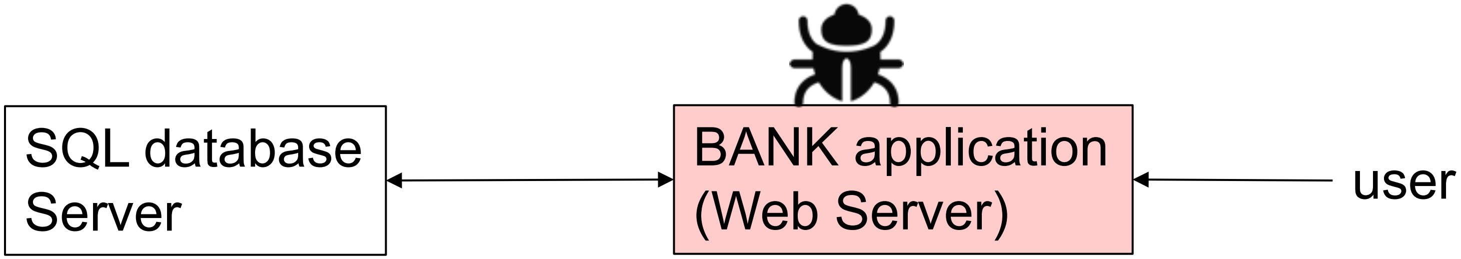
Bugs in bridge leads to Privilege escalation.



Sensitive data/files

Bridge with “Bug”.

Input crafted by malicious user, with “malicious payload”



SQL database Server

BANK application (Web Server)

user

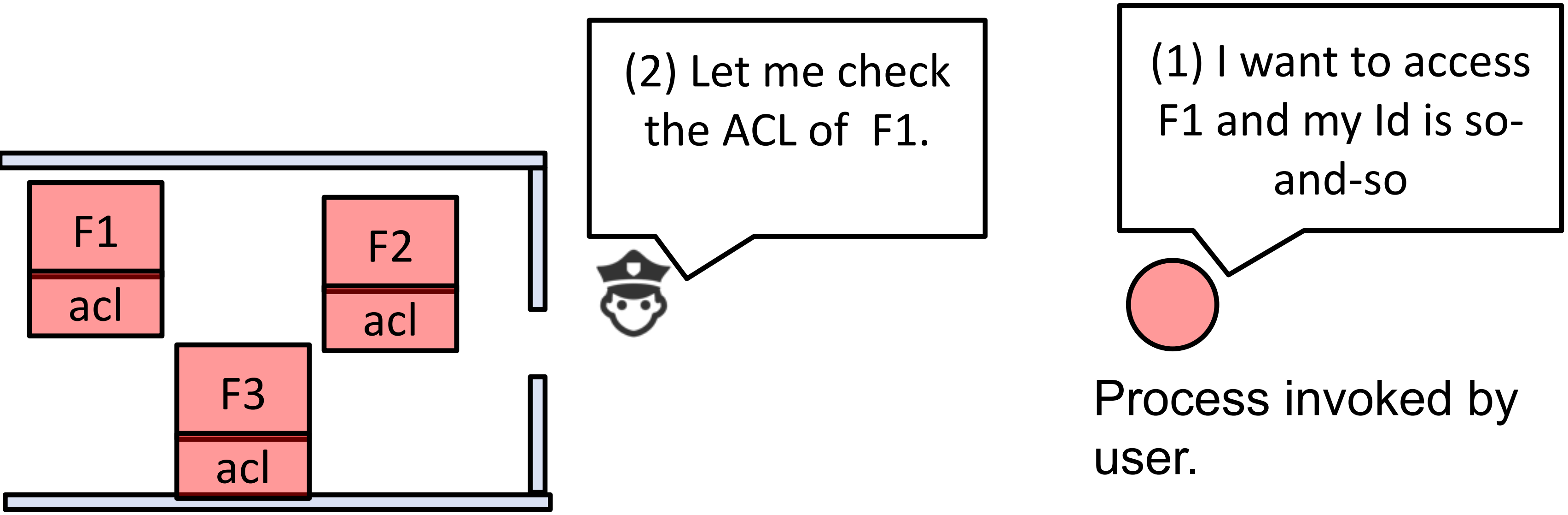
A bug in BANK that allows the user to send in arbitrary query. (SQL injection)

6.6 Controlled Invocation in UNIX

Real UID, Effective UID, privilege escalation

The definitions, steps are very complicated with many exceptions. Complexity is bad for security. Users/programmers would get confused, which lead to buggy implementation.

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



(2) Let me check the ACL of F1.

(1) I want to access F1 and my Id is so-and-so

Process invoked by user.

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)

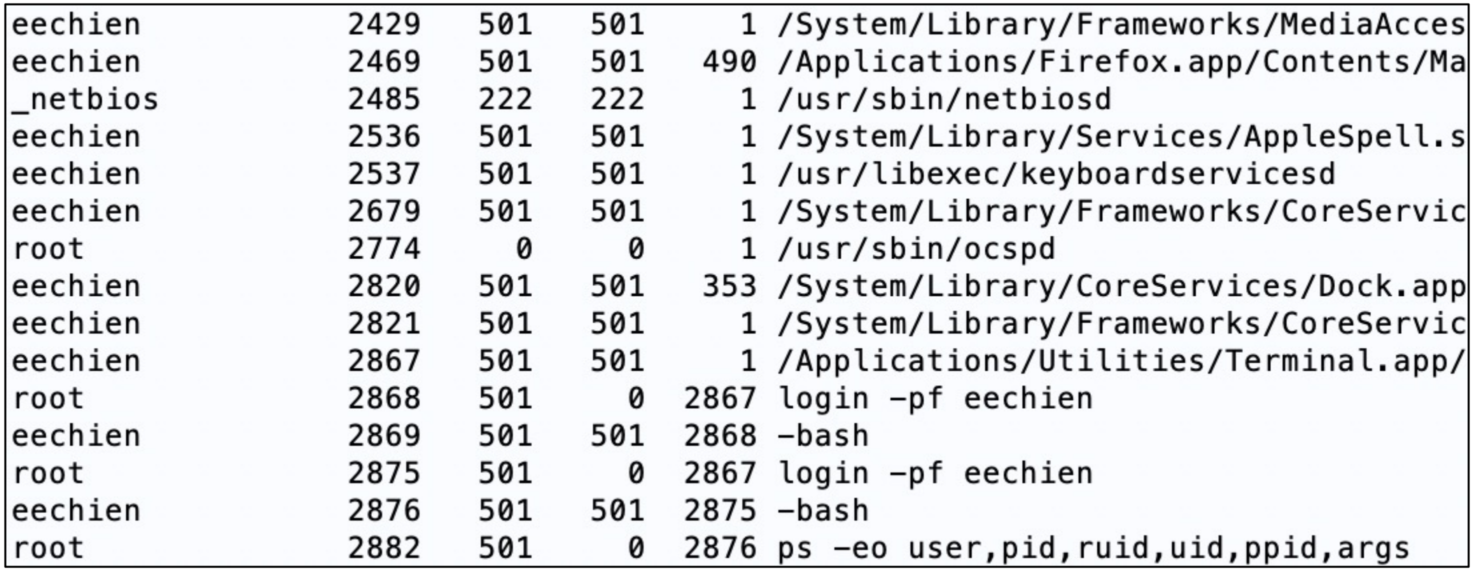
- A process is a subject.
- A process has an identification (PID). New process can be created by executing a file or by "forking" an existing process.
- A process is associated with a ***Real UID*** and an ***Effective UID****.*
- The ***real UID*** is inherited from the user who invokes the process.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

read UID

effective UID



```
eechien           2429   501   501     1 /System/Library/Frameworks/MediaAcces
eechien           2469   501   501   490 /Applications/Firefox.app/Contents/Ma
_netbios          2485   222   222     1 /usr/sbin/netbiosd
eechien           2536   501   501     1 /System/Library/Services/AppleSpell.s
eechien           2537   501   501     1 /usr/libexec/keyboardservicesd
eechien           2679   501   501     1 /System/Library/Frameworks/CoreServic
root              2774     0     0     1 /usr/sbin/ocspd
eechien           2820   501   501   353 /System/Library/CoreServices/Dock.app
eechien           2821   501   501     1 /System/Library/Frameworks/CoreServic
eechien           2867   501   501     1 /Applications/Utilities/Terminal.app/
root              2868   501     0  2867 login -pf eechien
eechien           2869   501   501  2868 -bash
root              2875   501     0  2867 login -pf eechien
eechien           2876   501   501  2875 -bash
root              2882   501     0  2876 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 Real UID is `alice`

Effective UID is `alice`

e.g. If the process is invoked by executing the following file.

```
-r-sr-xr-x   1 root  staff     6 Mar 18 08:00 check1
```

then Real UID is `alice`

Effective UID is `root`

This indicates that SUID is enabled.

When a process (subject) wants to read 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 read the file.

Use Case Scenario of “s” (SUID)

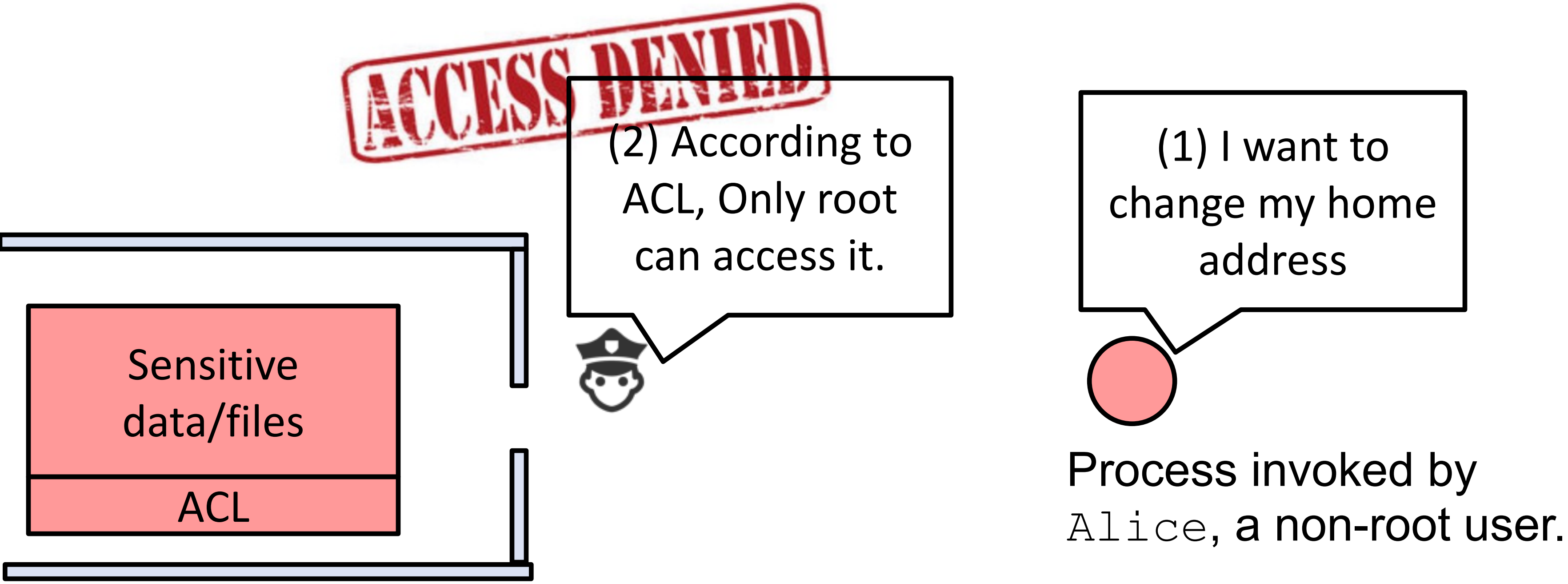
- Consider a scenario where the file `employee.txt` contains personal information of the users.

- 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

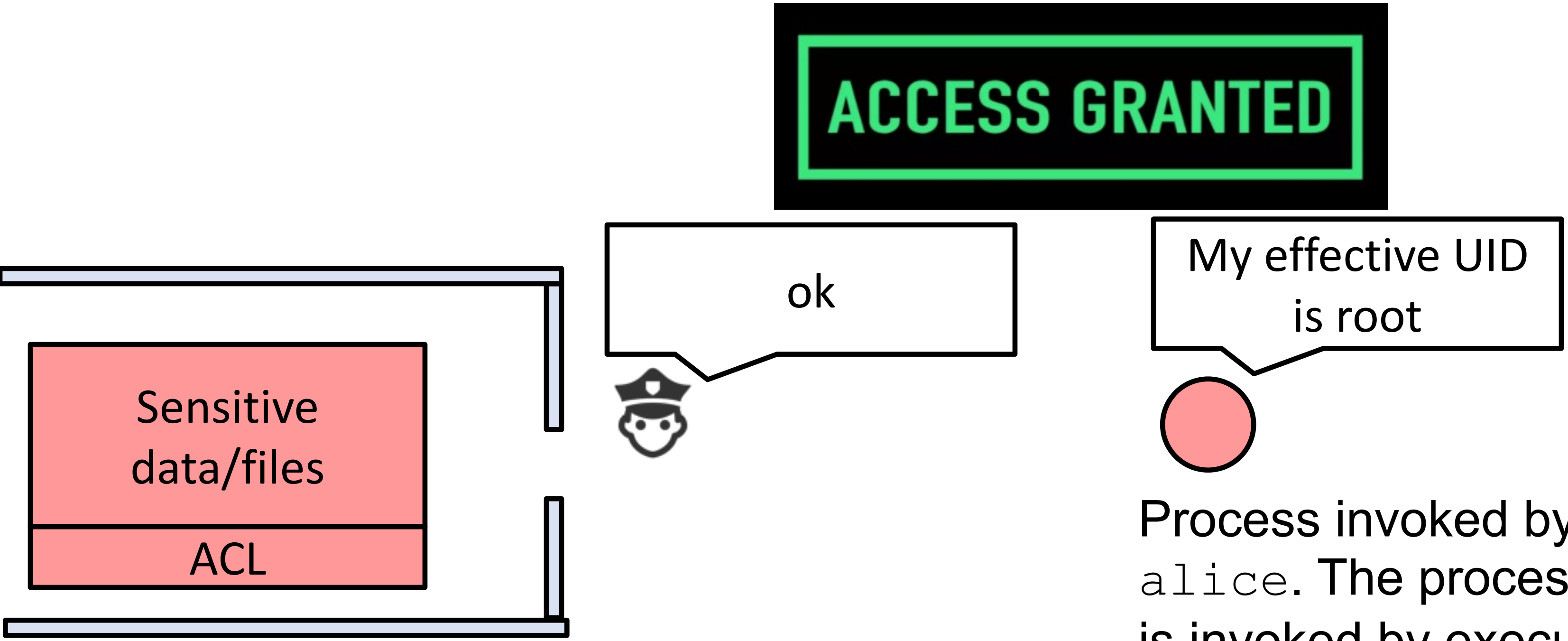
Process invoked by `Alice`, a non-root user.

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”
- This is an example of a *setuid-root* file/executable
- Now, if `alice` executes the file, the process’ real UID is `alice`, but its effective UID is `root`:
  this process can now read/write the file `employee.txt`



ok

My effective UID is root

Sensitive data/files

ACL

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`)

ACCESS DENIED

```
-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`

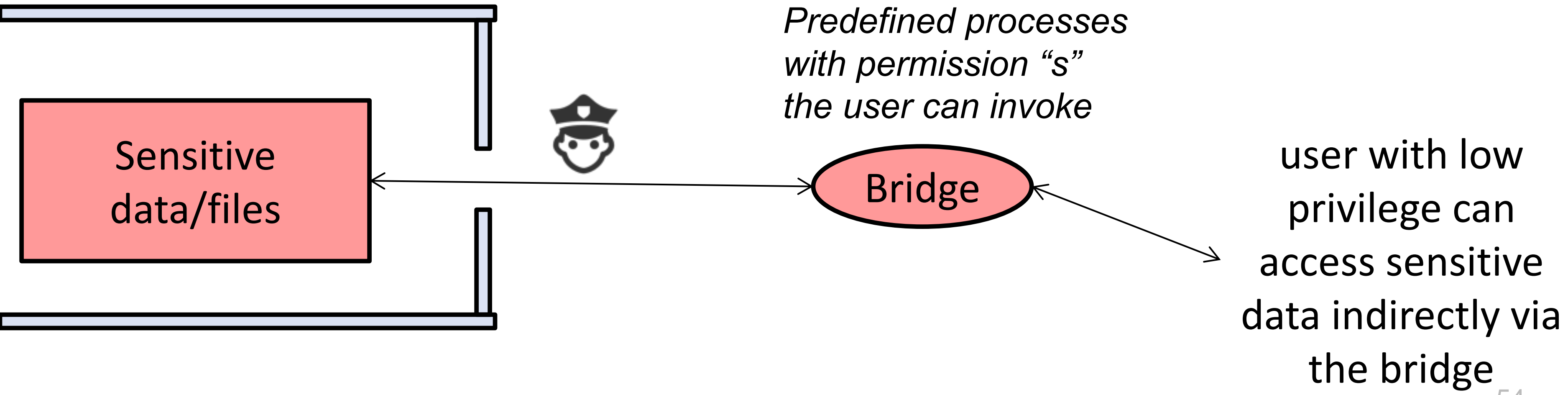
Process info: name (`editprofile`) real ID (`alice`) effective ID (`root`)

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

Elevated Privilege

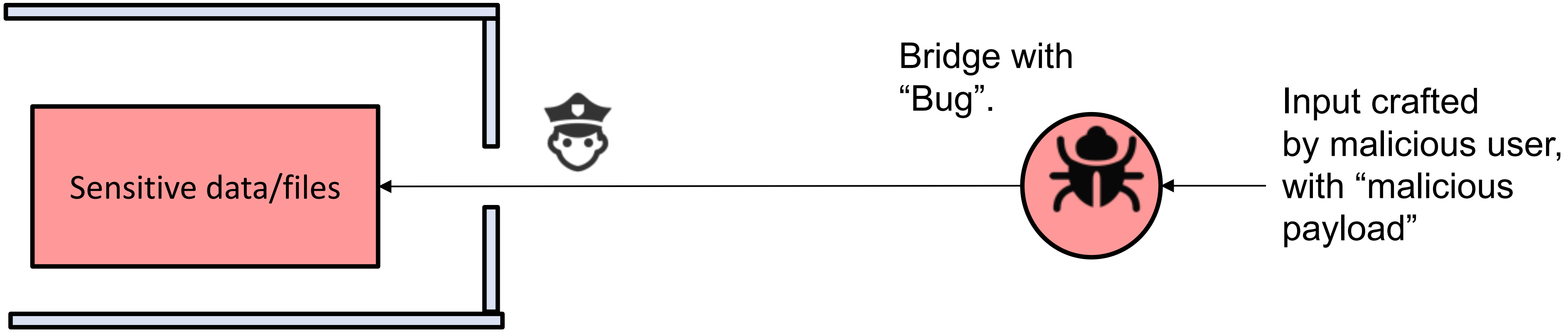
- In this example, the process `editprofile` is temporary ***elevated*** to superuser (i.e. root), so that it can access sensitive data. We can view the elevated process as the interfaces where a user can access the "sensitive" information.
  - They are the predefined "bridges" for the user to access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


Sensitive data/files
Predefined processes with permission "s" the user can invoke
Bridge
user with low privilege can access sensitive data indirectly via the bridge

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

- This leads us to another topic: secure programming and software security.



Footnote

More Complications (Saved UID, Real UID, Effective UID)

Optional

- The OS actually maintains three IDs for a process:
  real UID, effective UID, and saved UID
- Saved UID is like a “temp” container: when a setUID program is invoked, a copy of the original real UID is stored in the saved UID
- Saved UID is useful for a program running with elevated privileges to drop privilege *temporarily*
- The process removes its privileged user ID from its effective UID, but stores it in its saved UID. Later, the process may restore privilege by restoring the saved privileged UID into its effective UID.
  (See https://en.wikipedia.org/wiki/User_identifier#Saved_user_ID)
- The details may easily confuse many programmers
  (Read http://stackoverflow.com/questions/8499296/realuid-saved-uid-effective-uid-whats-going-on)
- Different UNIX versions may have different behaviors!
  (Optional: Chen et al., “Setuid Demystified”, USENIX Security, 2002)

Side remark on Unix's Searchpath

When a user types in the command to execute a program, say "su" without specifying the full path, which program would be executed?

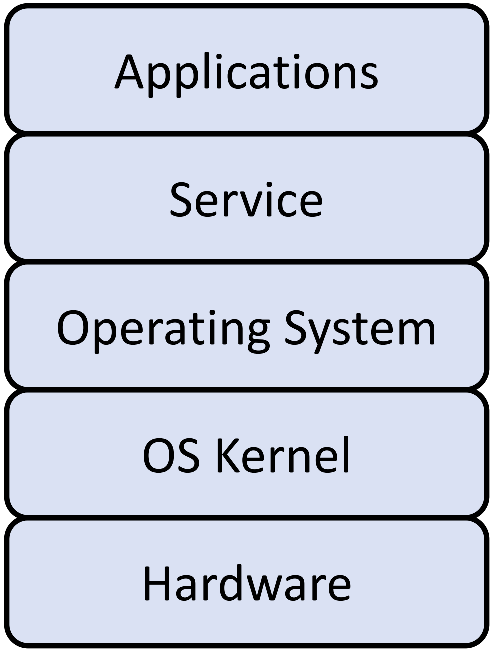
/usr/bin/su or ./su ?

The program to be executed is searched through the directories specified in the *searchpath,* starting from the front of the list. The first program found will be executed.

Now, it is possible that an attacker somehow stored a malicious program in the directory that appears in the beginning of the search path, and the malicious program has a common name, say "su". Now, when a user executes "su", the malicious program will be invoked instead.

To prevent such attack, specify the full path.

Computer System Layers



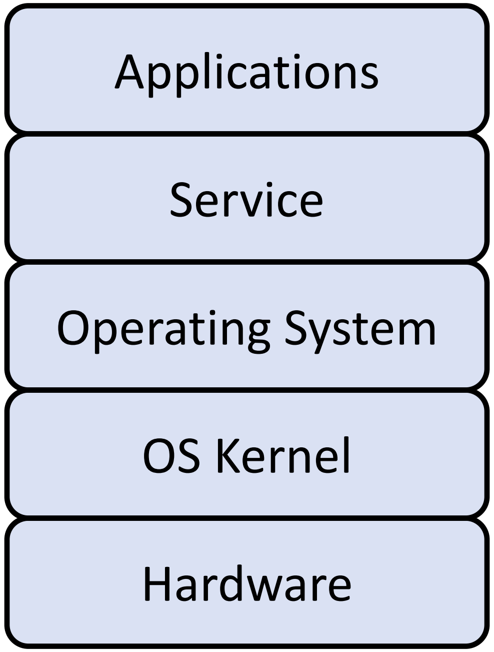
Remark:

1. These layers are used as a guideline. Actual systems typically don’t have distinct layers (for example, the windowing system spans across multiple “layers”.

Computer System Layers

- Same as networking, to manage the complexity, design of computer system is modular with layers.

- To access an object, users in the upper layer use “library call”, “services” or “API” provided by the lower layers. Each layer could use different representation and abstraction of the objects.



(e.g. Browser, mail reader, QR code reader apps)

(e.g. Database management system, Java Virtual Machine, camera API with various features, e.g. live feedback, record etc)

(e.g. iOS, macos, UNIX, Window XP, library call to directly access camera, etc)

(including system calls to handle the memory, manages virtual memory, etc)

CPU, Memory, Storage, I/O devices

- If there is bug or malware residue at a lower layer, it is very difficult for the upper layers to detect and sidestep it. Hence, malwares and vulnerabilities in the kernel (e.g. the “ps” command) are much more critical compare to vulnerabilities in the application layer (e.g. camera apps).