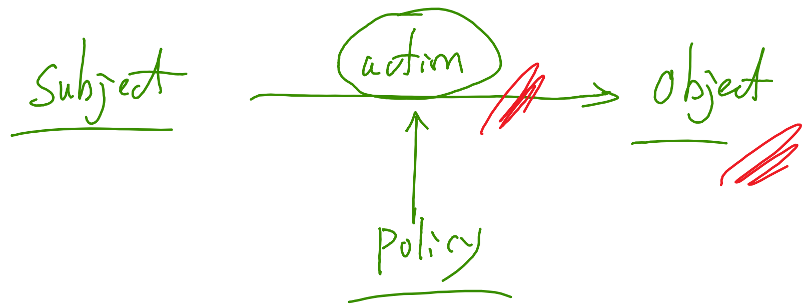
Access Control

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

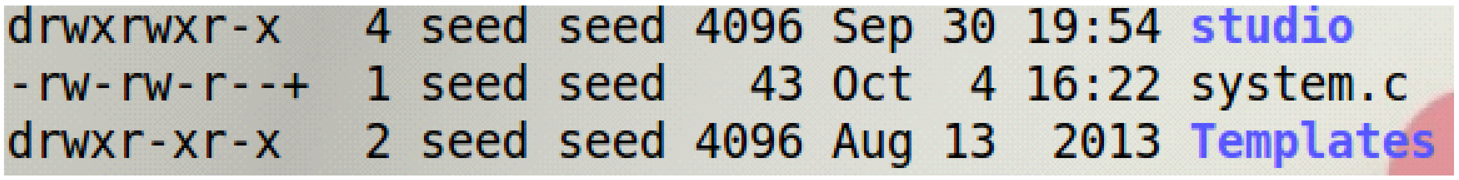
UID-Based Access Control and ACL

Access Control: Introduction

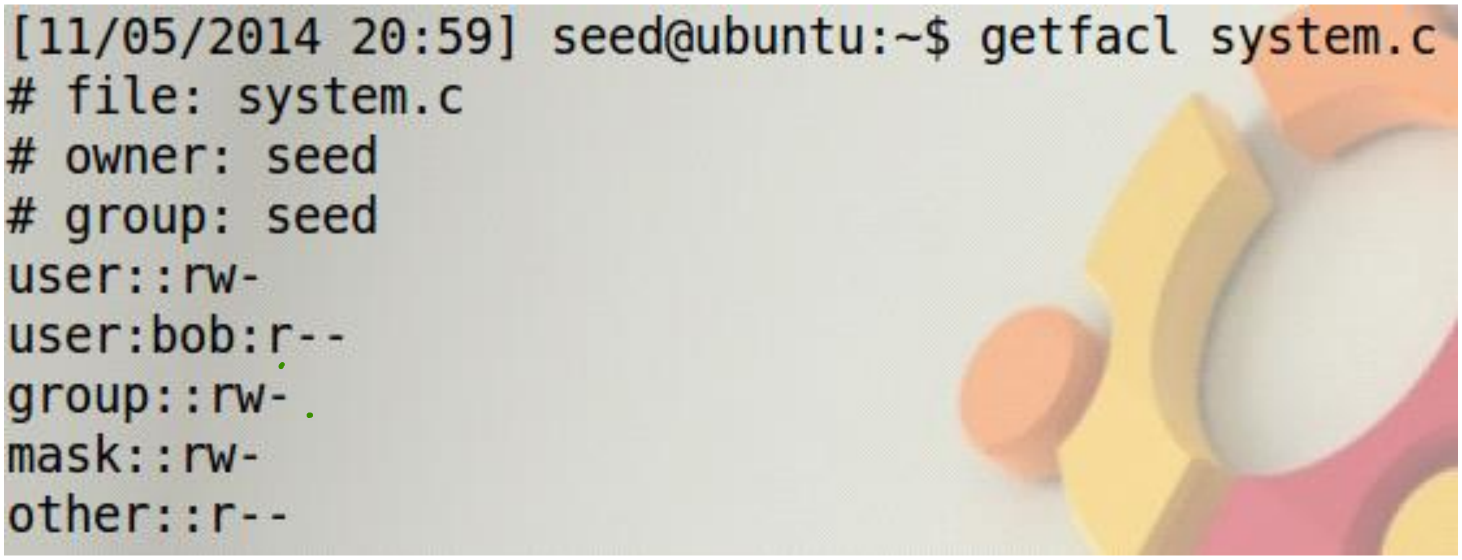


- Access Control List (UID-based)
- Permission-based Access Control. Android
- Capability-based Access Control.

Access Control List (UID-based)



```
drwxrwxr-x   4 seed seed 4096 Sep 30 19:54 studio
-rw-rw-r--+  1 seed seed   43 Oct  4 16:22 system.c
drwxr-xr-x   2 seed seed 4096 Aug 13  2013 Templates
```

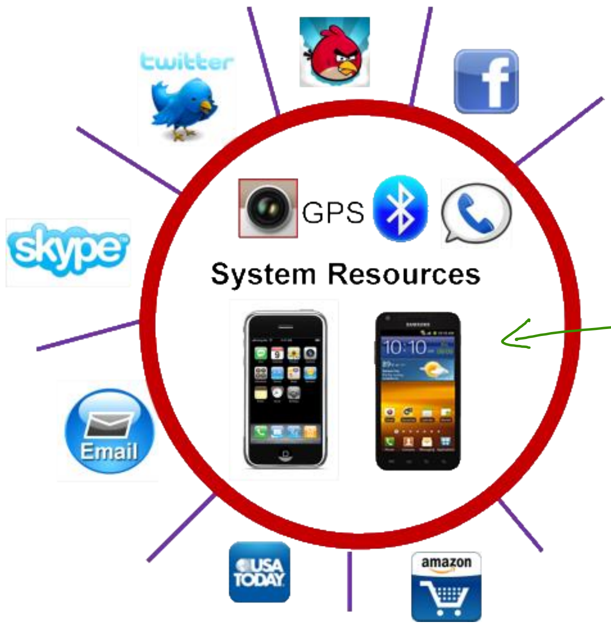


```
[11/05/2014 20:59] seed@ubuntu:~$ getfacl system.c
# file: system.c
# owner: seed
# group: seed
user::rw-
user:bob:r--
group::rw-
mask::rw-
other::r--
```

Access Control in Android

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

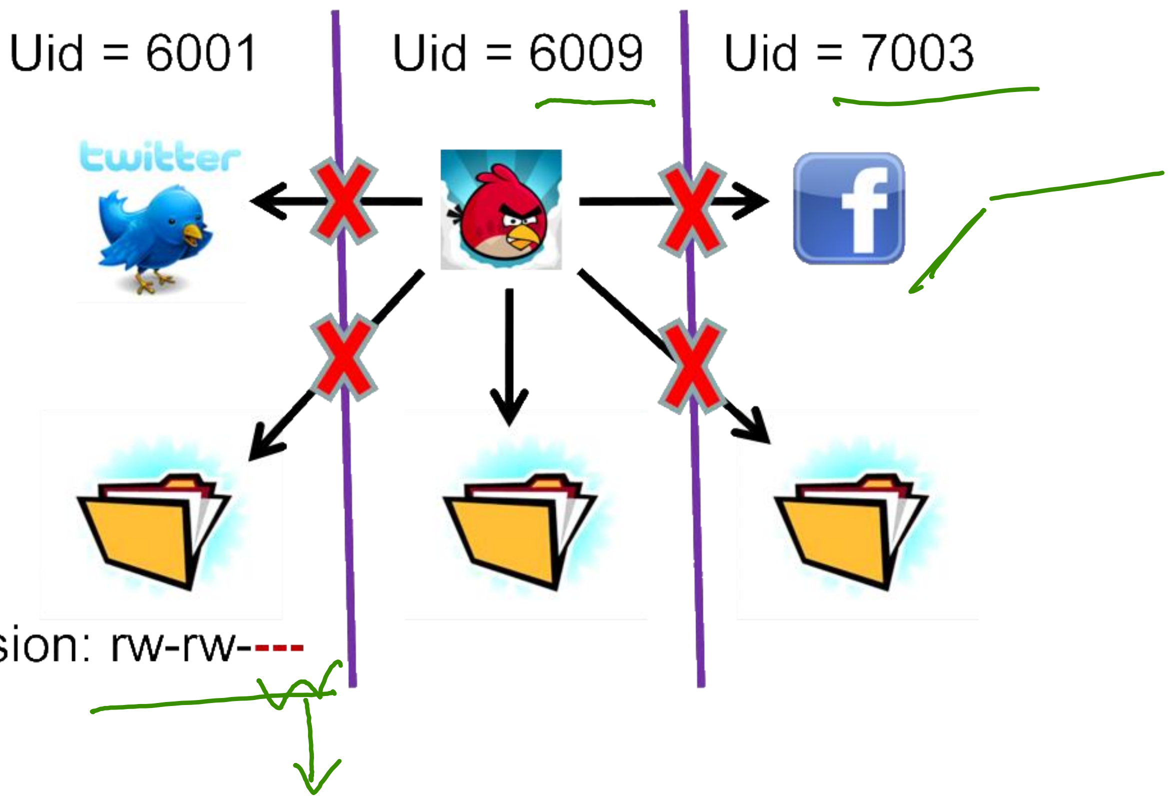
Android's Access Control Model



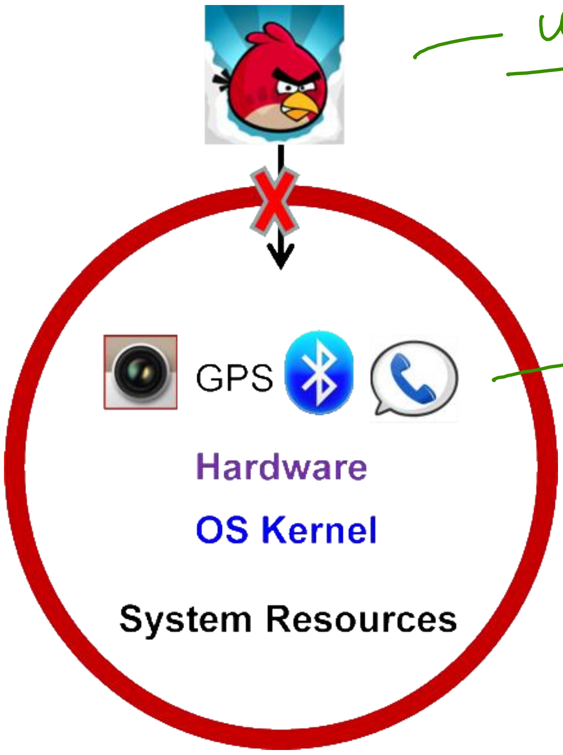
UID-based access control

accessible to privileged users

Isolation Among Apps



Isolation Between App and System



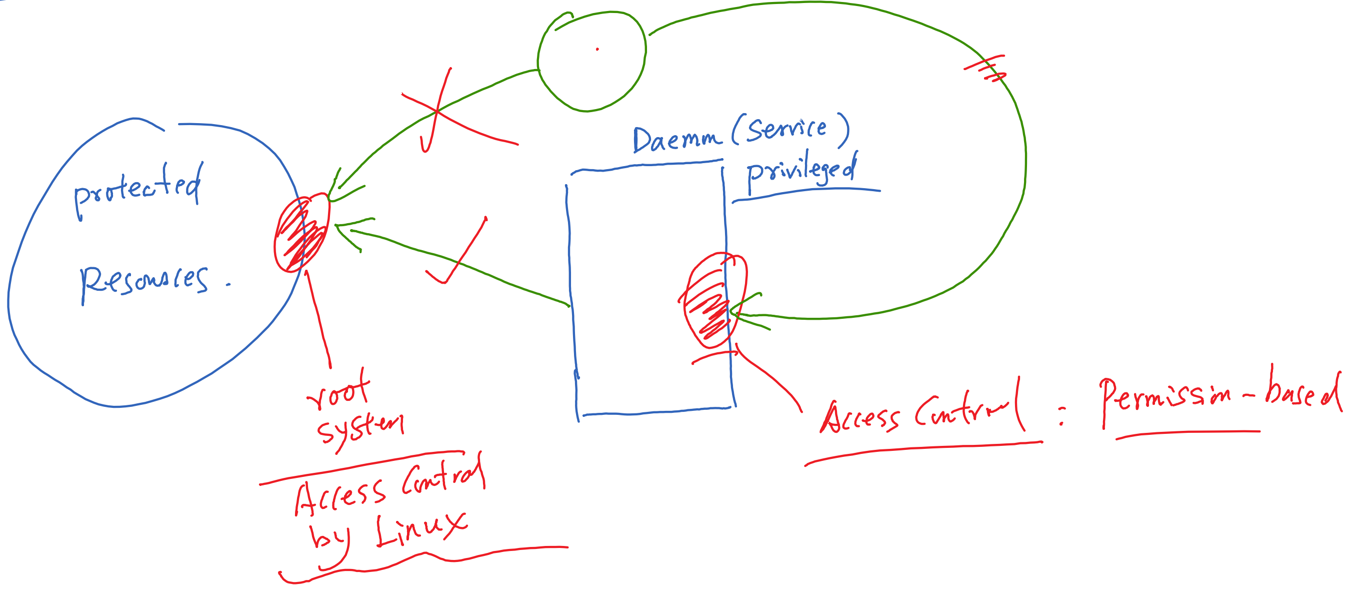
uid : 6009

uid = 0 (root)

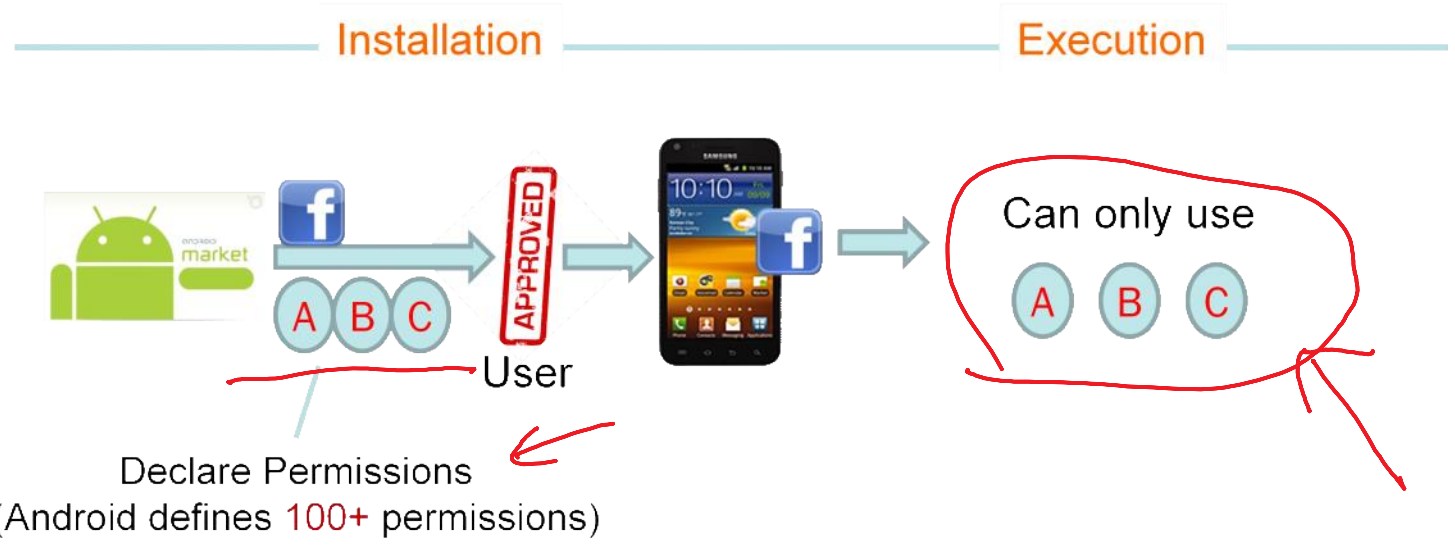
uid = 1000 (system)

Granularity Problem and Proxy Approach

protected resources : root & system



Android Permissions

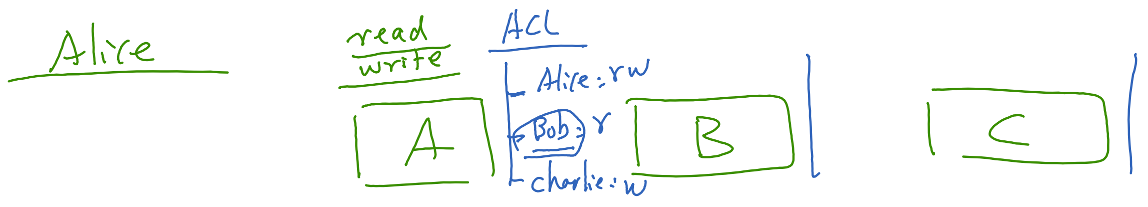


ACCESS_FINE_LOCATION	Access GPS
BLUETOOTH	Connect to Bluetooth device
CALL_PHONE	Directly make phone calls
CAMERA	Use camera
INTERNET	Access to the Internet
READ_CONTACTS	Read user's contacts data
WRITE_CONTACTS	Write contacts data
READ_CALENDAR	Read user's calendar data
READ_SMS	Read SMS messages
SEND_SMS	Send SMS messages

An Example

UID-Based vs. Capability-Based Access Control

An example



UID_based:

subject:

Capability_based

Key

Difference.

Risk Management.

Delegation.

Capability-Based Access Control

Capability for File System

ACL

Normal user

```
char buf[50];
int fd = open("/etc/passwd", O_RDONLY);

read(fd, buf, 10);   ✓
write(fd, buf, 10);  ✗

getchar();  // System pauses
// Now the root changes the permission of the above file to 600.

read(fd , buf, 10);
```

rw-r--r--

644

600

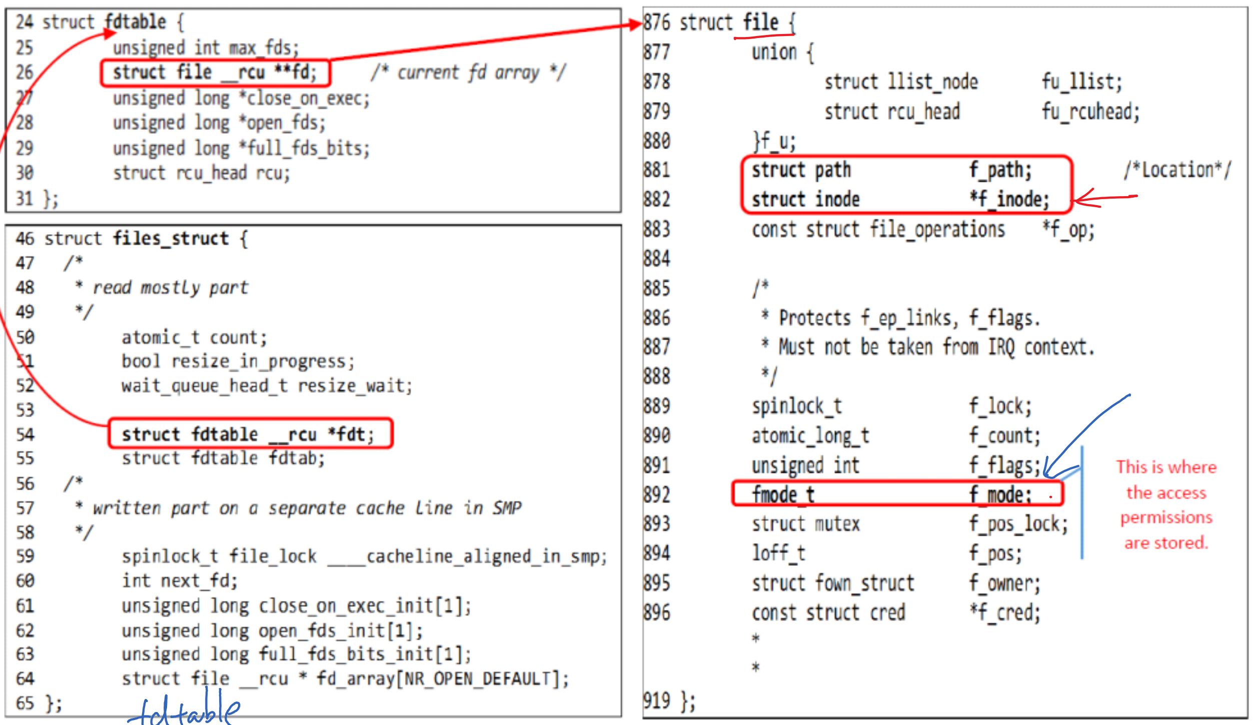
rw-------

ACL

ACL ✗

Capability ✓

Capability for File System: Linux Implementation

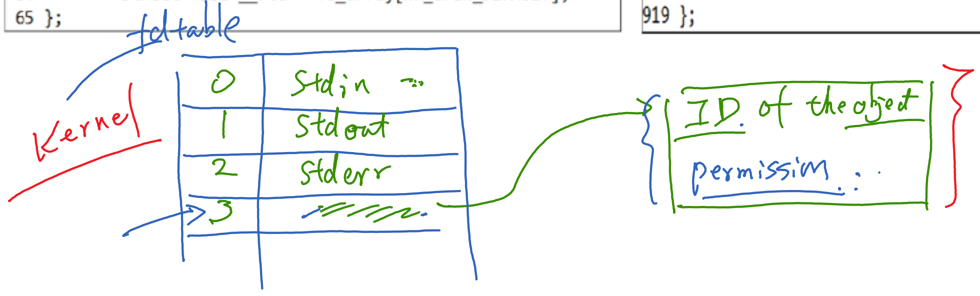


```
struct fdtable {
        unsigned int max_fds;
        struct file __rcu **fd;      /* current fd array */
        unsigned long *close_on_exec;
        unsigned long *open_fds;
        unsigned long *full_fds_bits;
        struct rcu_head rcu;
};
```

```
struct files_struct {
  /*
   * read mostly part
   */
        atomic_t count;
        bool resize_in_progress;
        wait_queue_head_t resize_wait;

        struct fdtable __rcu *fdt;
        struct fdtable fdtab;
  /*
   * written part on a separate cache line in SMP
   */
        spinlock_t file_lock ____cacheline_aligned_in_smp;
        int next_fd;
        unsigned long close_on_exec_init[1];
        unsigned long open_fds_init[1];
        unsigned long full_fds_bits_init[1];
        struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

```
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        }f_u;
        struct path             f_path;         /*Location*/
        struct inode            *f_inode;
        const struct file_operations    *f_op;

        /*
         * Protects f_ep_links, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t              f_lock;
        atomic_long_t           f_count;
        unsigned int            f_flags;
        fmode_t                 f_mode;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        *
        *
};
```

This is where the access permissions are stored.



- open : create a key
- read (fd, . . .)
  write (fd, . . .)
- close (fd) : destroy key

Capability Concepts

A capability is a token, a ticket, or key that gives the possessor permission to access an entity or object in a computer system.

object + Permissions

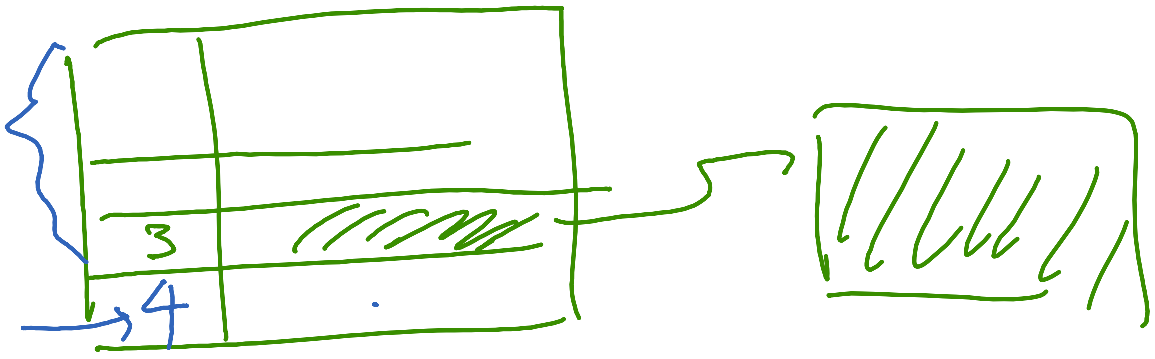
ticket

obj: movie

perm:

Discussion Questions

Question 1: Can you forge a capability? Why or why not?

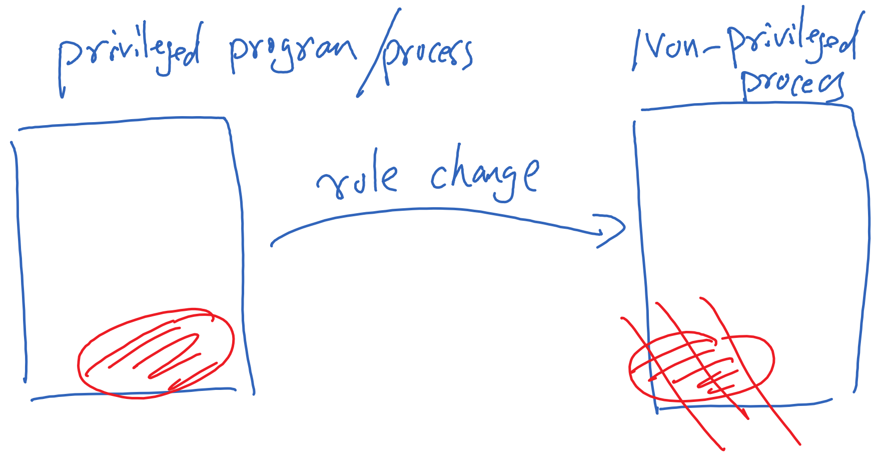


Question 2: Where should we store capabilities? Why?

- User space
- Kernel space (correct place)

Capability Leaking

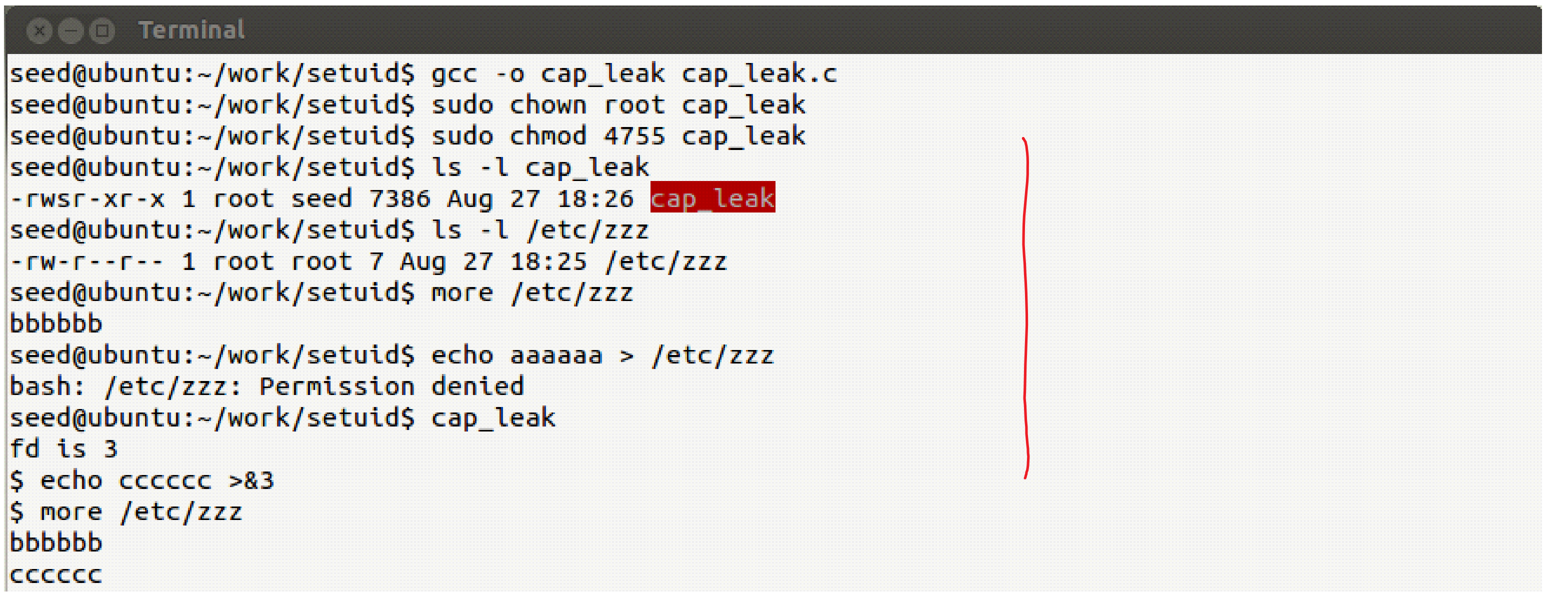
Case Study: Capability Leaking



Capability Leaking Example

Example: **cap_leak.c**

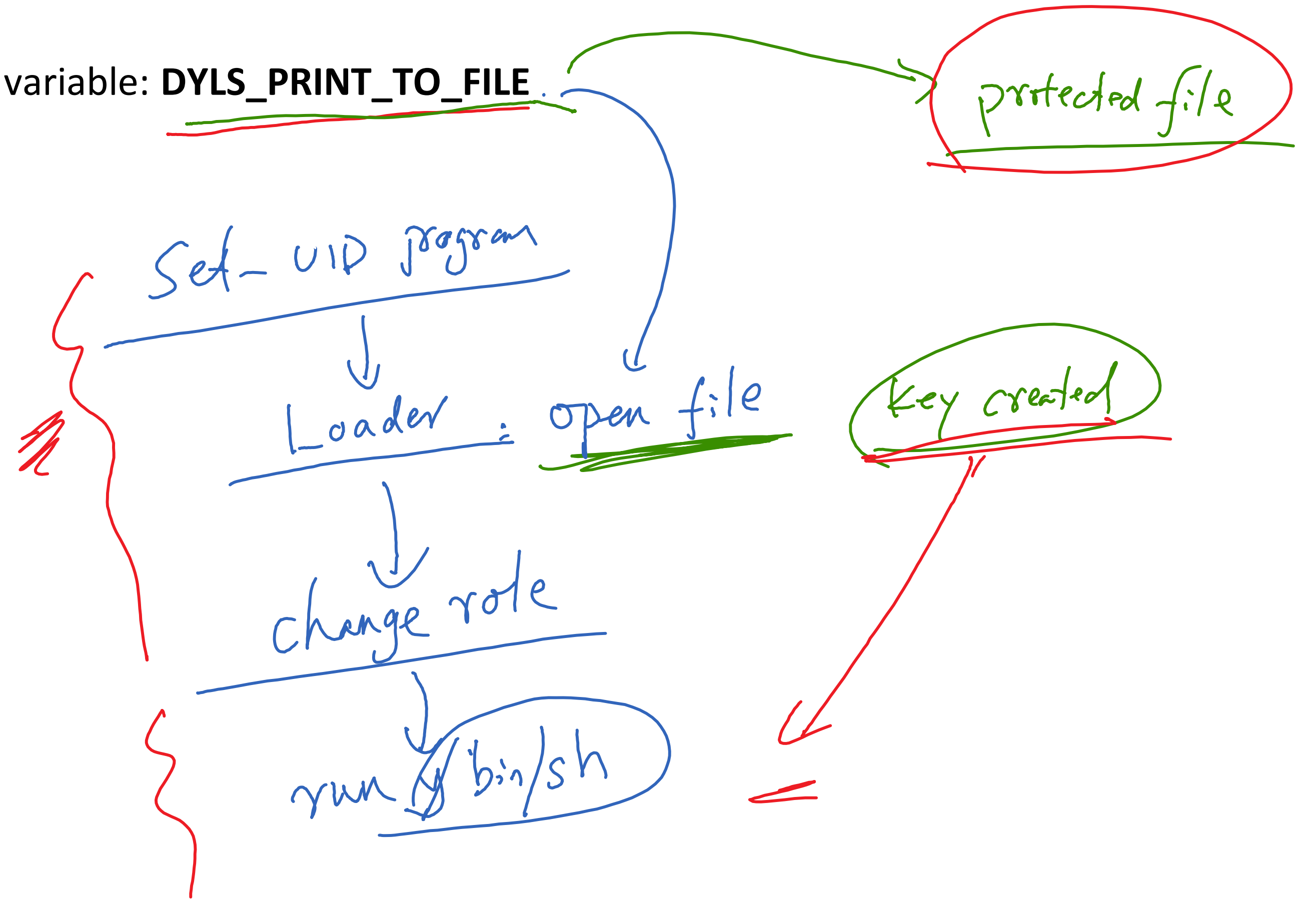
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

void main()
{
  int fd;
  char *v[2];

  /* Assume that /etc/zzz is an important system file,
   * and it is owned by root with permission 0644.
   * Before running this program, you should creat
   * the file /etc/zzz first. */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
     printf("Cannot open /etc/zzz\n");
     exit(0);
  }

  // Print out the file descriptor value
  printf("fd is %d\n", fd);                    close(fd)

  // Permanently disable the privilege by making the
  // effective uid the same as the real uid
  setuid(getuid());

  // Execute /bin/sh
  v[0] = "/bin/sh"; v[1] = 0;
  execve(v[0], v, 0);
}
```



```
seed@ubuntu:~/work/setuid$ gcc -o cap_leak cap_leak.c
seed@ubuntu:~/work/setuid$ sudo chown root cap_leak
seed@ubuntu:~/work/setuid$ sudo chmod 4755 cap_leak
seed@ubuntu:~/work/setuid$ ls -l cap_leak
-rwsr-xr-x 1 root seed 7386 Aug 27 18:26 cap_leak
seed@ubuntu:~/work/setuid$ ls -l /etc/zzz
-rw-r--r-- 1 root root 7 Aug 27 18:25 /etc/zzz
seed@ubuntu:~/work/setuid$ more /etc/zzz
bbbbbb
seed@ubuntu:~/work/setuid$ echo aaaaaa > /etc/zzz
bash: /etc/zzz: Permission denied
seed@ubuntu:~/work/setuid$ cap_leak
fd is 3
$ echo cccccc >&3
$ more /etc/zzz
bbbbbb
cccccc
```

Capability Leaking in OS X 10.10 (2015)

Environment variable: **DYLS_PRINT_TO_FILE**



Basic Functionalities

Basic Functionalities of Capabilities

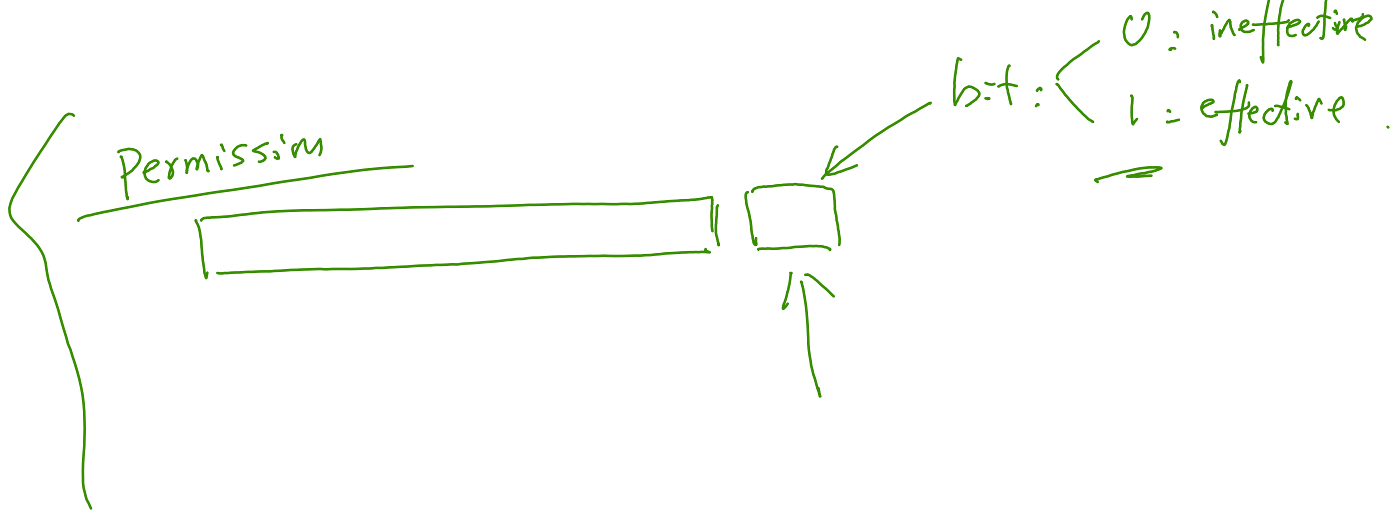
- Create
- Destroy

- Delegation
- Revocation
- Disable / Enable

$\left.\begin{array}{l}\text{Delegation}\\ \text{Revocation}\\ \text{Disable / Enable}\end{array}\right\}$ Manage risk

Discussion

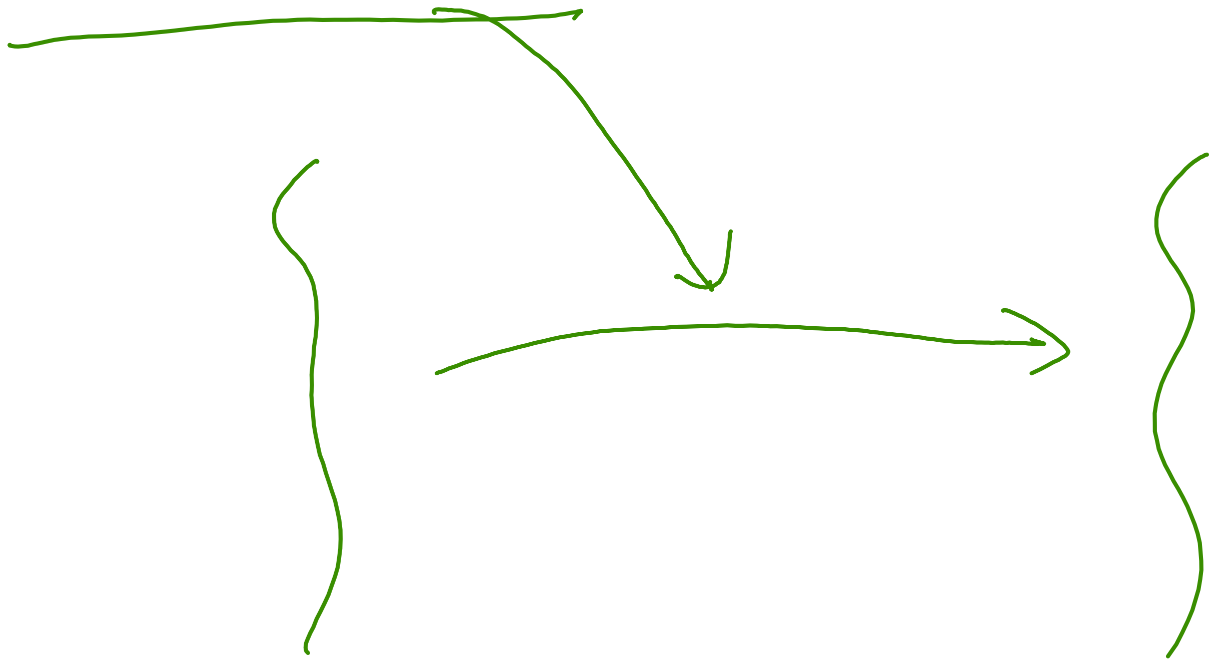
Question: Describe how you can add the "**disable**" and "**enable**" functionalities to the file-descriptor's capability mechanism. Namely, if a process disables a file-descriptor capability, the process will not be able to use the capability to access the file until the process specially enables the capability again.



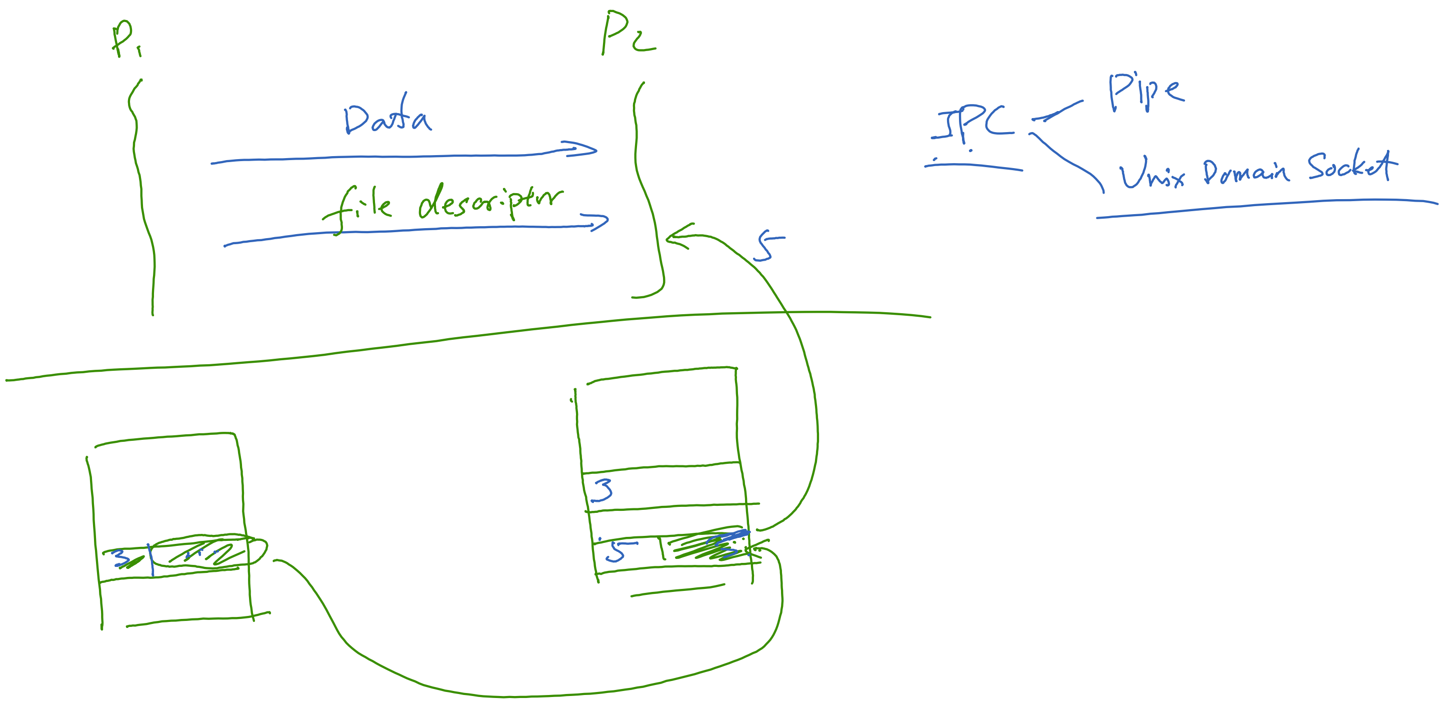
Delegation

Delegation

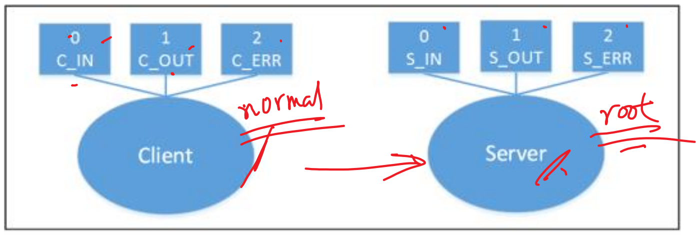
- ❖ Sending a file descriptor from one process to another:
  - ○ Through inheritance: **fork()**
  - ○ Using **Unix Domain Socket**



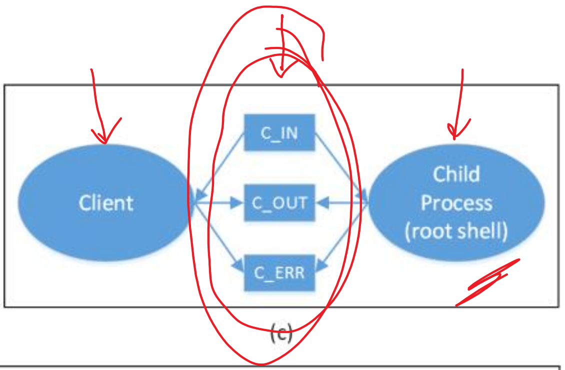
Unix Domain Socket



Case Studies

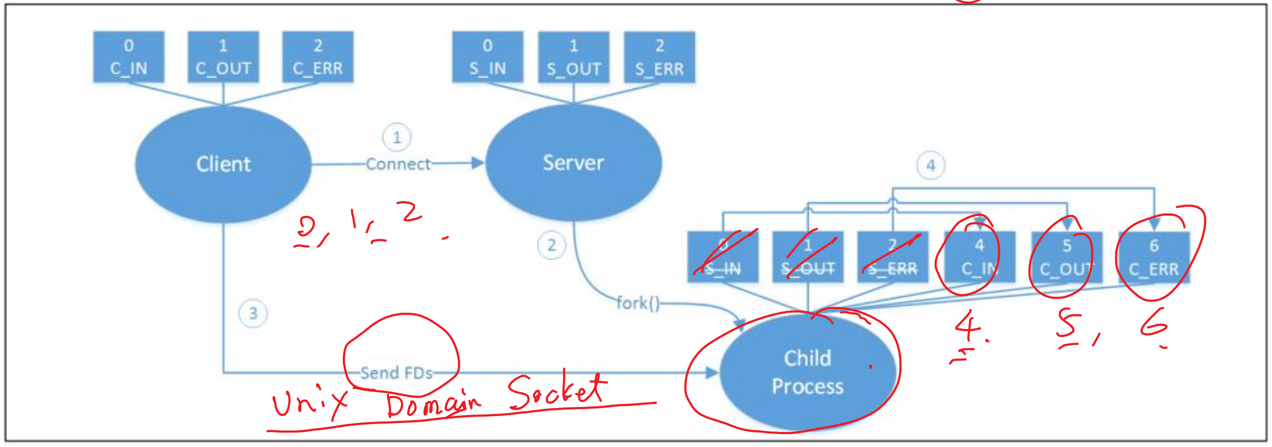


(a)



(c)

SimpleSU.



(b)

❖ How server changes its file descriptors

```
int client_in = recv_fd(socket);
int client_out = recv_fd(socket);
int client_err = recv_fd(socket);

dup2(client_in, STDIN_FILENO);        //STDIN_FILENO = 0
dup2(client_out, STDOUT_FILENO);      //STDOUT_FILENO = 1
dup2(client_err, STDERR_FILENO);      //STDERR_FILENO = 2
```

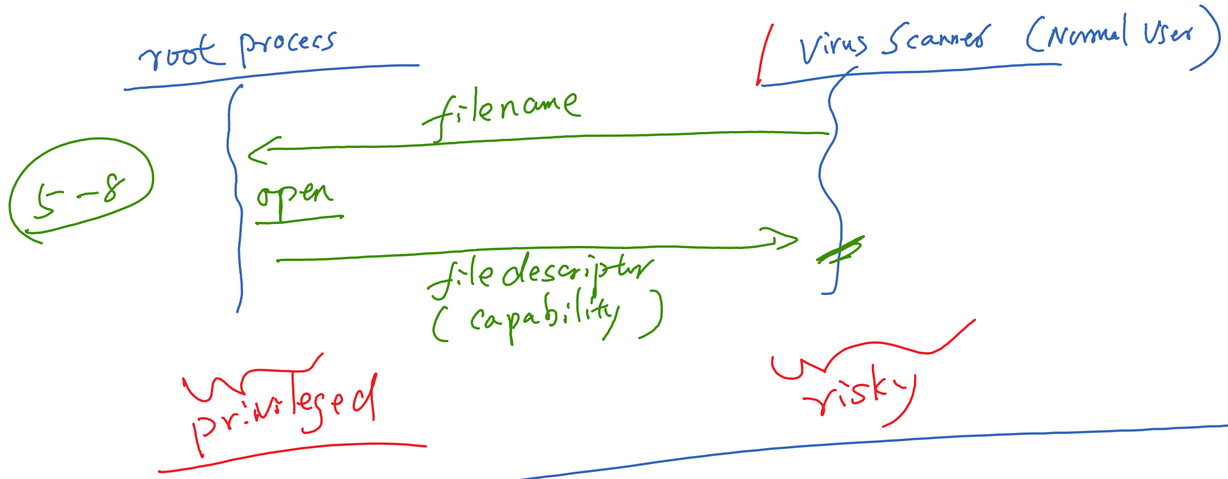
Applications

Capability Applications

Virus Scanner

- scan all files
- can't have root

} reduce risk



ACL

group: scanner

Add group to ACL of all files

New requirement

: 5:00pm – 8:00 am, /xyz

Review Question

Question: Which access control mechanism, ACL or capability, is better regarding privilege management (enabling, disabling, discarding)? Why?

risk

Capability

Managed by Subject

ACL

Managed by Object

Summary

- ❖ UID-based access control and ACL
- ❖ Access control in Android
- ❖ Capability-based access control
- ❖ Applications