CHAPTER 2
SET-UID PROGRAMS

CS44500 Computer Security

Review: Chapter 1

Users

- In Linux, each user is assigned a unique user ID
- User ID is stored in /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
```

- Find user ID

```
seed@VM:~$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed)

root@VM:~# id
uid=0(root) gid=0(root) groups=0(root)
```

Which Group Does a User Belong To?

```
seed@VM:~$ grep seed /etc/group
adm:x:4:syslog,seed
sudo:x:27:seed
plugdev:x:46:seed
lpadmin:x:120:seed
lxd:x:131:seed
seed:x:1000:
docker:x:136:seed
```

```
seed@VM:~$ groups
seed adm sudo plugdev lpadmin lxd docker
```

```
seed@VM:~$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),27(sudo),
46(plugdev),120(lpadmin),131(lxd),136(docker)
```

File Permissions

permissions

```
- rwx rwx rwx  seed  abc  1802  Feb  6  11:39  xyz
```

owner

group

other

owner

group owner

file name

Default File Permissions

- umask value: decides the default permissions for new files
- Example

```
Initial (0666)        rw- rw- rw-
                      110 110 110
umask   (0022)        000 010 010
---------------------------------
Final permission      110 100 100
                      rw- r-- r--
```

Initial AND NOT_umask

ACL Commands

```
setfacl {-m, -x}  {u, g}:<name>:[r, w, x] <file, directory>
```

```
$ setfacl -m u:alice:r-- example
$ setfacl -m g:faculty:rw- example
$ getfacl example
# file: example
# owner: seed
# group: seed
user::rw-
user:alice:r--
group::rw-
group:faculty:rw-
mask::rw-                    ①
other::r--
```

```
-rw-rw-r--+  1 seed seed  1050 Feb  7 10:57 example
           ↖ indicating that ACLs are defined
```

Running command with privilege

- Three command mechanisms
  - sudo
  - Set-uid programs (covered in a separate chapter)
  - POSIX capabilities

First Command After Login

- The last field of each entry

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
bob:x:1001:1001:Bob,,,:/home/bob:/bin/bash
alice:x:1002:1003:Alice,,,:/home/alice:/bin/bash
```

```
$ sudo su bin
This account is currently not available.
```

The Shadow File

- Store password, why not use /etc/password anymore?
- Structure for each entry

The Purpose of Salt

- Defeat brute-force attacks
  - dictionary attack, rainbow table attack
- These 3 accounts have the same password

```
seed:$6$n8DimvsbIgU0OxbD$YZ0h1EA...(omitted)...wFd0:18590:0:
alice:$6$.1CMCeSFZd8/8QZl$QhfhId...(omitted)...Sga.:18664:0:
bob:$6$NOLhqomO3yNwyFsZ$K.Ql/KnP...(omitted)...b8v.:18664:0:
```

SHA-512

CHAPTER 2:
SET-UID PROGRAMS

Need for Privileged Programs

- Password Dilemma
  - Permissions of /etc/shadow File:

```
-rw-r----- 1 root shadow 1443 May 23 12:33 /etc/shadow
```

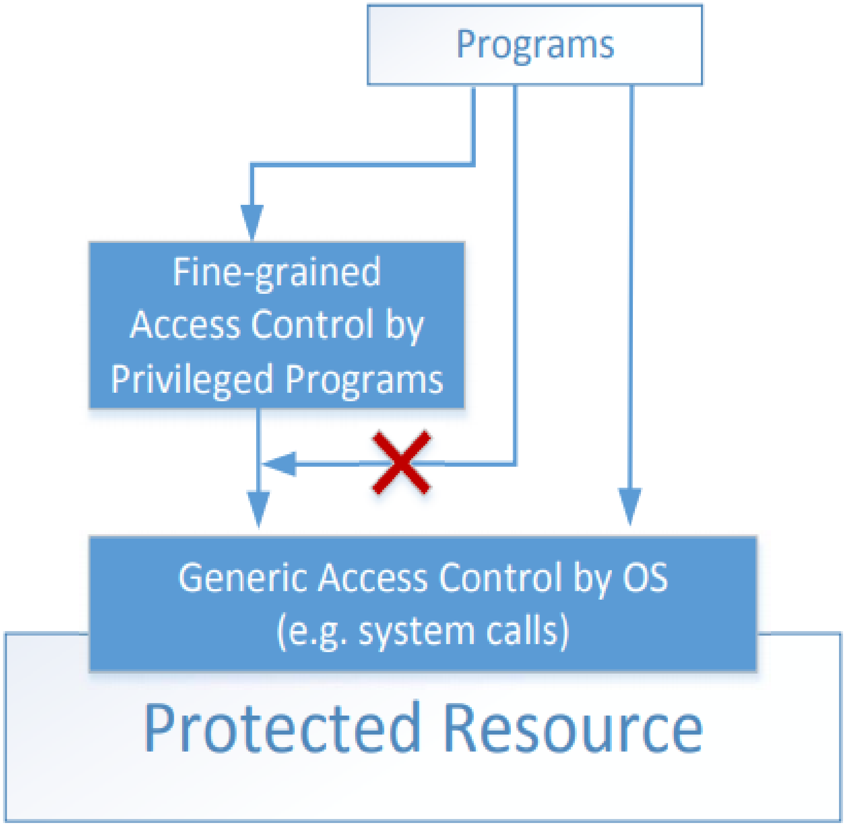
Only writable to the owner

  - How would normal users change their password?

```
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn0R25yqtqrSrFeWfCgybQWWnwR4ks/.rjqyM7Xw
h/pDyc5U1BWOzkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::
man:*:15749:0:99999:7:::
lp:*:15749:0:99999:7:::
```

Two-Tier Approach

- Implementing fine-grained access control in operating systems make OS over complicated.
- OS relies on extensions to enforce fine-grained access control
- Privileged programs are such extensions



Types of Privileged Programs

- Daemons
  - Computer program that runs in the background
  - Needs to run as root or other privileged users

Many operating systems use the daemon approach for privileged operations. In Windows, they are not called daemons; they are called services, which, just like daemons, are computer programs that operate in the background.

- Set-UID Programs
  - Widely used in UNIX systems
  - Program marked with a special bit

Superman Story

- Power Suit
  - Superpeople: Directly give them the power
  - Issues: bad superpeople

- Power Suit 2.0
  - Computer chip
  - Specific task
  - No way to deviate from pre-programmed task

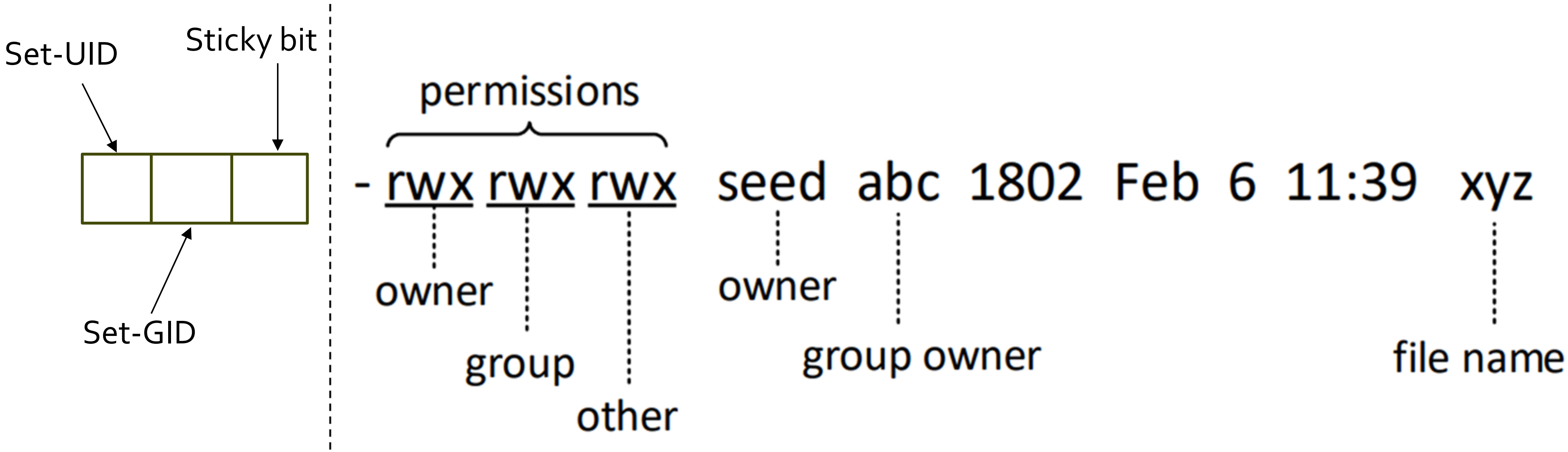
- Set-UID mechanism: A Power Suit mechanism implemented in Linux OS

Set-UID Concept

- **Allow user to run a program with the program <u>owner's</u> privilege.**
- Allow users to run programs with temporary elevated privileges
- Example: the `passwd` program

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 41284 Sep 12  2012 /usr/bin/passwd
```

Turn a Program into a Set-UID Program



setuid: a bit that makes an executable run with the privileges of the owner of the file

setgid: a bit that makes an executable run with the privileges of the group owner of the file

sticky bit: a bit set on directories that allows only the owner or root to delete files and subdirectories

How it Works

A Set-UID program is just like any other program, except that it has a special marking, which a single bit called Set-UID bit

```
$ cp /bin/id ./myid
$ sudo chown root myid
$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed), ...
```

```
$ sudo chmod 4755 myid
$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```

Set-UID Concept

- Every process has two User IDs.
- **Real UID (RUID)**: Identifies real owner of process
- **Effective UID (EUID)**: Identifies privilege of a process
  - Access control is based on EUID
- When a normal program is executed, RUID = EUID, they both equal to the ID of the user who runs the program
- When a Set-UID is executed, RUID ≠ EUID. RUID still equal to the user's ID, but EUID equals to the program **owner**'s ID.
  - If the program is owned by root, the program runs with the root privilege.

Example of Set UID

It should be noted that in the experiment, we have to run chmod again to enable the Set-UID bit, because the chown command automatically turns off the Set-UID bit

```
$ cp /bin/cat ./mycat
$ sudo chown root mycat
$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Feb 22 10:04 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

Not a privileged program

```
$ sudo chmod 4755 mycat
$ ./mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8c...
daemon:*:15749:0:99999:7:::
...
```

Become a privileged program

```
$ sudo chown seed mycat
$ chmod 4755 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

It is still a privileged program, but not the root privilege

Example Use:

Bob gives you a chance to use his Unix account, and you have your own account on the same system. Can you take over Bob's account in 10 seconds?

Example Use:

Bob gives you a chance to use his Unix account, and you have your own account on the same system. Can you take over Bob's account in 10 seconds?

- In Bob’s account
  - cp /bin/sh /tmp/mysh
  - chmod 4755 /temp/mysh
- In my account
  - /tmp/mysh (ruid is going to be “me” but the euid is “Bob”)

How is Set-UID Secure?

- Allows normal users to escalate privileges
  - This is different from directly giving the privilege (sudo command)
  - Restricted behavior – similar to superman designed computer chips

- Unsafe to turn all programs into Set-UID
  - Example: /bin/sh
  - Example: vi

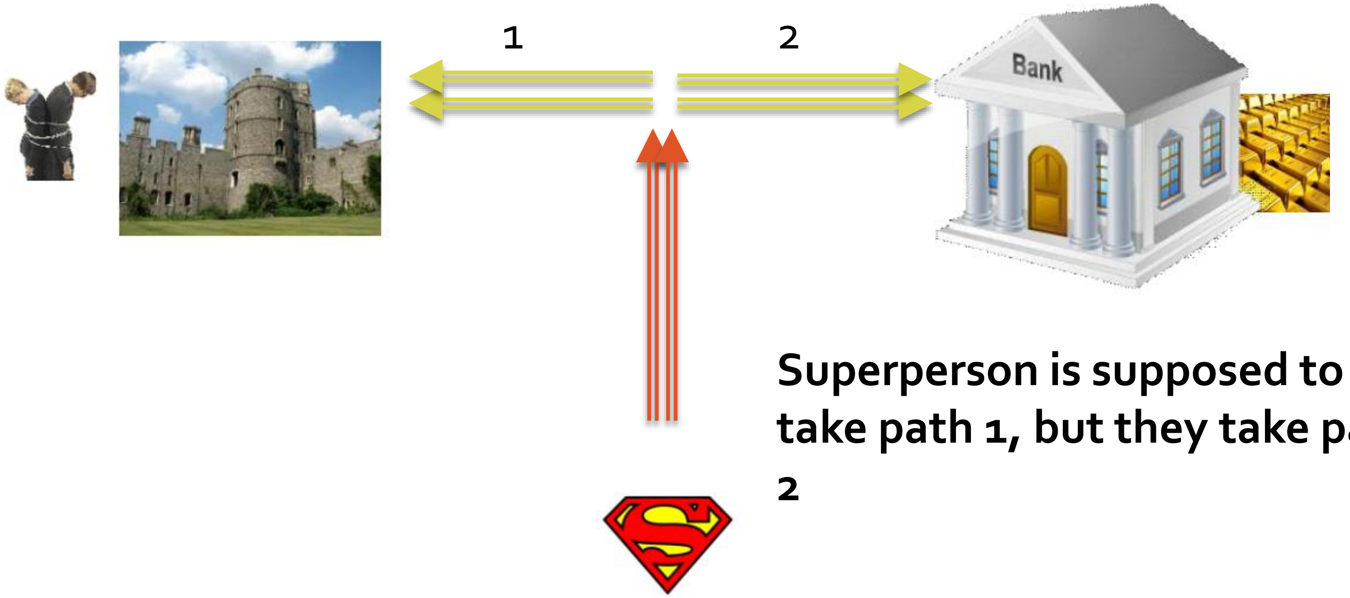
- <u>Note:</u> The Set-UID mechanism can also be applied to groups, instead of users. This is called Set-GID. Namely, a process has effective group ID and real group ID, and the effective group ID is used for access control.

Attack on Superman

- Cannot assume that user can only do whatever is coded
  - Coding flaws by developers

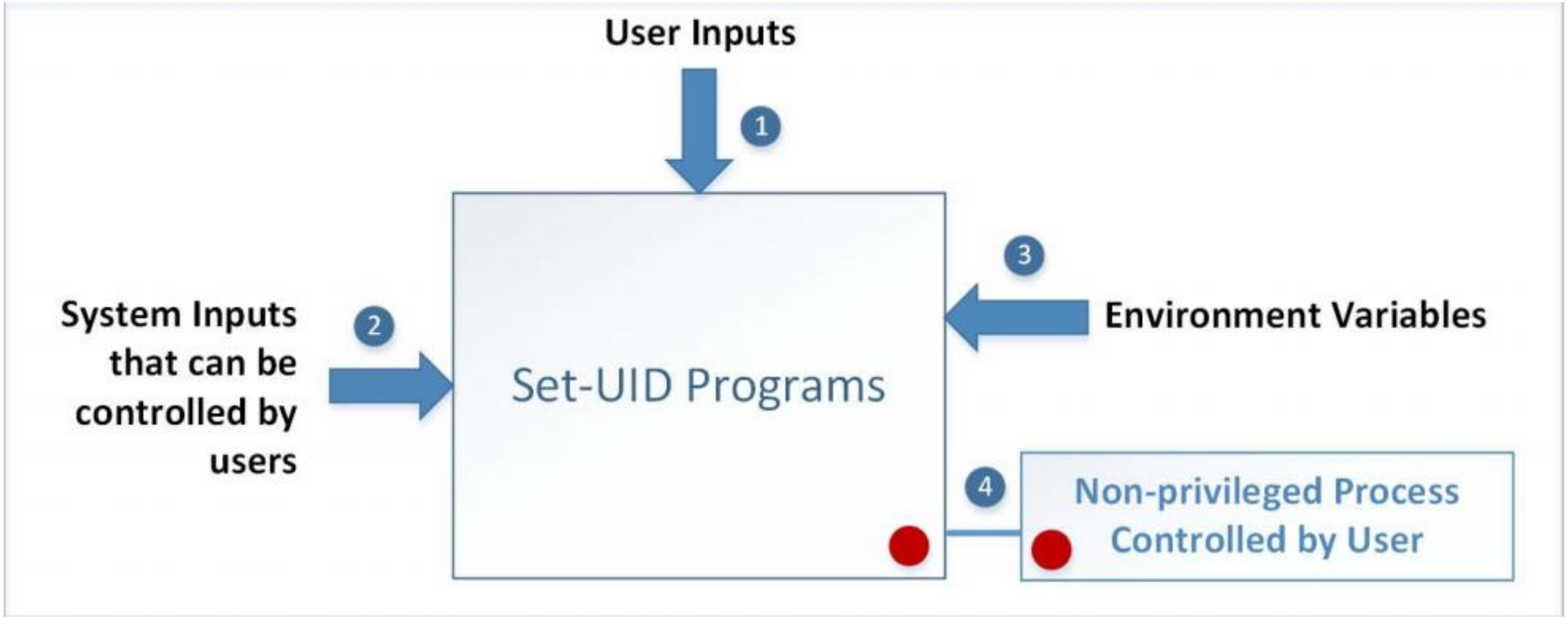
- Superperson Mallroy
  - Fly north then turn left
  - How to exploit this code?

- Superperson Malorie
  - Fly North and turn West
  - How to exploit this code?



Superperson is supposed to take path 1, but they take path 2

Attack Surfaces of Set-UID Programs



Attacks via User Inputs

User Inputs: Explicit Inputs

- Buffer Overflow – More information in Chapter 4
  - Overflowing a buffer to run malicious code
- Format String Vulnerability – More information in Chapter 6
  - Changing program behavior using user inputs as format strings

Attacks via User Inputs

chsh – Change Shell

- Set-UID program with ability to change default shell programs
- Shell programs are stored in /etc/passwd file

Issues:

- Failing to sanitize user inputs
- Attackers could create a new root account

Attack (it allows addition of a new line to passwd file):

```
bob:$6$jUODEFsfwfi3:1000:1000:Bob Smith,,,:/home/bob:/bin/bash
```

Attacks via System Inputs

System Inputs

- Race Condition – More information in Chapter 7
  - Symbolic link to a privileged file from an unprivileged file
  - Influence programs
  - Writing inside world writable folder

Attacks via Environment Variables

- Behavior can be influenced by inputs that are not visible inside a program.

- Environment Variables: A user can set these before running a program.

- Detailed discussions on environment variables will be in Chapter 3.

Attacks via Environment Variables

- `PATH` Environment Variable
  - Used by shell programs to locate a command if the user does not provide the full path for the command
  - system(): call /bin/sh first
  - system("ls")
    - /bin/sh uses the PATH environment variable to locate "ls"
    - Attacker can manipulate the PATH variable and control how the "ls" command is found
- More examples on this type of attacks can be found in Chapter 3

Capability Leaking

- In some cases, Privileged programs downgrade themselves during execution
- Example: The `su` program
  - This is a privileged Set-UID program
  - Allows one user to switch to another user (say user1 to user2)
  - Program starts with EUID as root and RUID as user1
  - After password verification, both EUID and RUID become user2's (via privilege downgrading)
- Such programs may lead to capability leaking
  - Programs may not clean up privileged capabilities before downgrading

Example cap_leak.c

```c
#include <unistd.h>
#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 create
   * 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);

  // 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);
}
```

Attacks via Capability Leaking: An Example

The /etc/zzz file is only writable by root

File descriptor is created (the program is a root-owned Set-UID program)

The privilege is downgraded

Invoke a shell program, so the behavior restriction on the program is lifted

```
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);

// 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);
```

Attacks via Capability Leaking (Continued)

The program forgets to close the file, so the file descriptor is still valid.

Capability Leak

```
$ gcc -o cap_leak cap_leak.c
$ sudo chown root cap_leak
[sudo] password for seed:
$ sudo chmod 4755 cap_leak
$ ls -l cap_leak
-rwsr-xr-x 1 root seed 7386 Feb 23 09:24 cap_leak
$ cat /etc/zzz
bbbbbbbbbbbbbb
$ echo aaaaaaaaaa > /etc/zzz
bash: /etc/zzz: Permission denied   ← Cannot write to the file
$ cap_leak
fd is 3                  Inside shell invoked by cap_leak
$ echo cccccccccccc >& 3            ← Using the leaked capability
$ exit
$ cat /etc/zzz
bbbbbbbbbbbbbb
cccccccccccc                        ← File modified
```

How to fix the program?

Destroy the file descriptor before downgrading the privilege (close the file)

close (fd)

Capability Leaking in OS X – Case Study

- OS X Yosemite found vulnerable to privilege escalation attack related to capability leaking in July 2015 (OS X 10.10)
- Added features to dynamic linker `dyld`
  - DYLD_PRINT_TO_FILE environment variable
- The dynamic linker can open any file, so for root-owned Set-UID programs, it runs with root privileges. The dynamic linker `dyld`, does not close the file. There is a capability leaking.
- Scenario 1 (safe): Set-UID finishes its job, and the process dies. Everything is cleaned up and it is safe.
- **Scenario 2 (unsafe):** Similar to the "`su`" program, the privileged program downgrades its privilege and lifts the restriction.

Invoking Programs

- Invoking external commands from inside a program
- External command is chosen by the Set-UID program
  - Users are not supposed to provide the command (or it is not secure)
- Attack:
  - Users are often asked to provide input data to the command.
  - If the command is not invoked properly, user's input data may be turned into command name. This is dangerous.

Invoking Programs : Unsafe Approach

```
int main(int argc, char *argv[])
{
  char *cat="/bin/cat";

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  char *command = malloc(strlen(cat) + strlen(argv[1]) + 2);
  sprintf(command, "%s %s", cat, argv[1]);
  system(command);
  return 0 ;
}
```

- The easiest way to invoke an external command is the system() function.
- This program is supposed to run the `/bin/cat` program.
- It is a root-owned Set-UID program, so the program can view all files, but it can't write to any file.

Question: Can you use this program to run other command, with root privilege?

Invoking Programs : Unsafe Approach (Continued)

```
$ gcc -o catall catall.c
$ sudo chown root catall
$ sudo chmod 4755 catall
$ ls -l catall
-rwsr-xr-x 1 root seed 7275 Feb 23 09:41 catall
$ catall /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWb....
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::

$ catall "aa;/bin/sh"
/bin/cat: aa: No such file or directory
#         ← Got the root shell!
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root), ...
```

We can get a root shell with this input

Problem: Some part of the data becomes code (command name)

A Note

- In Ubuntu 16.04, /bin/sh points to /bin/dash, which has a countermeasure
  - It drops privilege when it is executed inside a set-uid process
- Therefore, we will only get a normal shell in the attack on the previous slide
- Do the following to remove the countermeasure

```
Before experiment: link /bin/sh to /bin/zsh
$ sudo ln -sf /bin/zsh /bin/sh

After experiment: remember to change it back
$ sudo ln -sf /bin/dash /bin/sh
```

Invoking Programs Safely: using **execve()**

```
int main(int argc, char *argv[])
{
  char *v[3];

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;
  execve(v[0], v, 0);

  return 0 ;
}
```

`execve(v[0], v, 0)`

Command name is provided here (by the program)

Input data are provided here (can be by user)

Why is it safe?

Code (command name) and data are clearly separated; there is no way for the user data to become code

Invoking Programs Safely (Continued)

```
$ gcc -o safecatall safecatall.c
$ sudo chown root safecatall
$ sudo chmod 4755 safecatall
$ safecatall /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWb....
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::

$ safecatall "aa;/bin/sh"
/bin/cat: aa;/bin/sh: No such file or directory    ← Attack failed!
```

The data are still treated as data, not code

Additional Consideration

- Some functions in the exec() family behave similarly to execve(), but may not be safe
  - execlp(), execvp() and execvpe() duplicate the actions of the shell. These functions can be attacked using the PATH Environment Variable

Principle of Isolation

Principle: Don’t mix code and data.

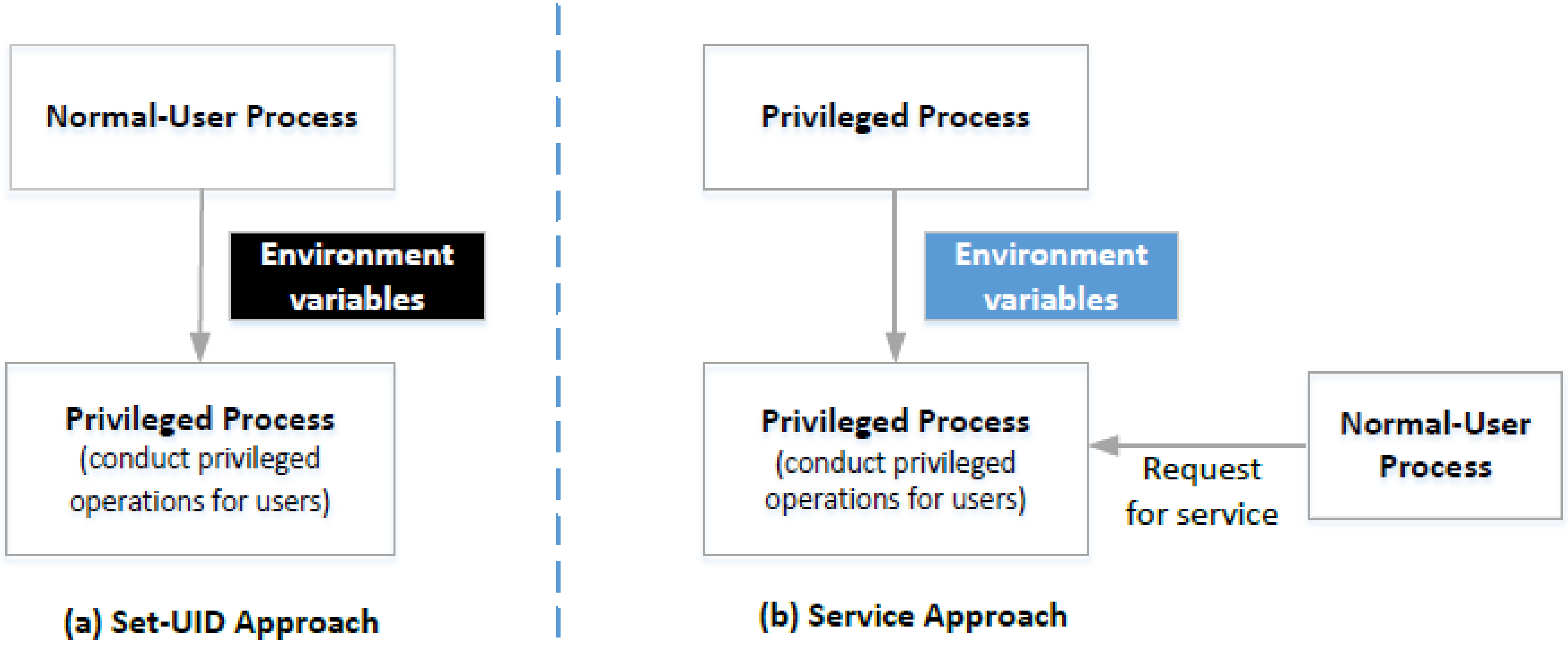
Attacks due to violation of this principle :

- system() code execution
- Cross Site Scripting – More Information in Chapter 10
- SQL injection - More Information in Chapter 11
- Buffer Overflow attacks - More Information in Chapter 4

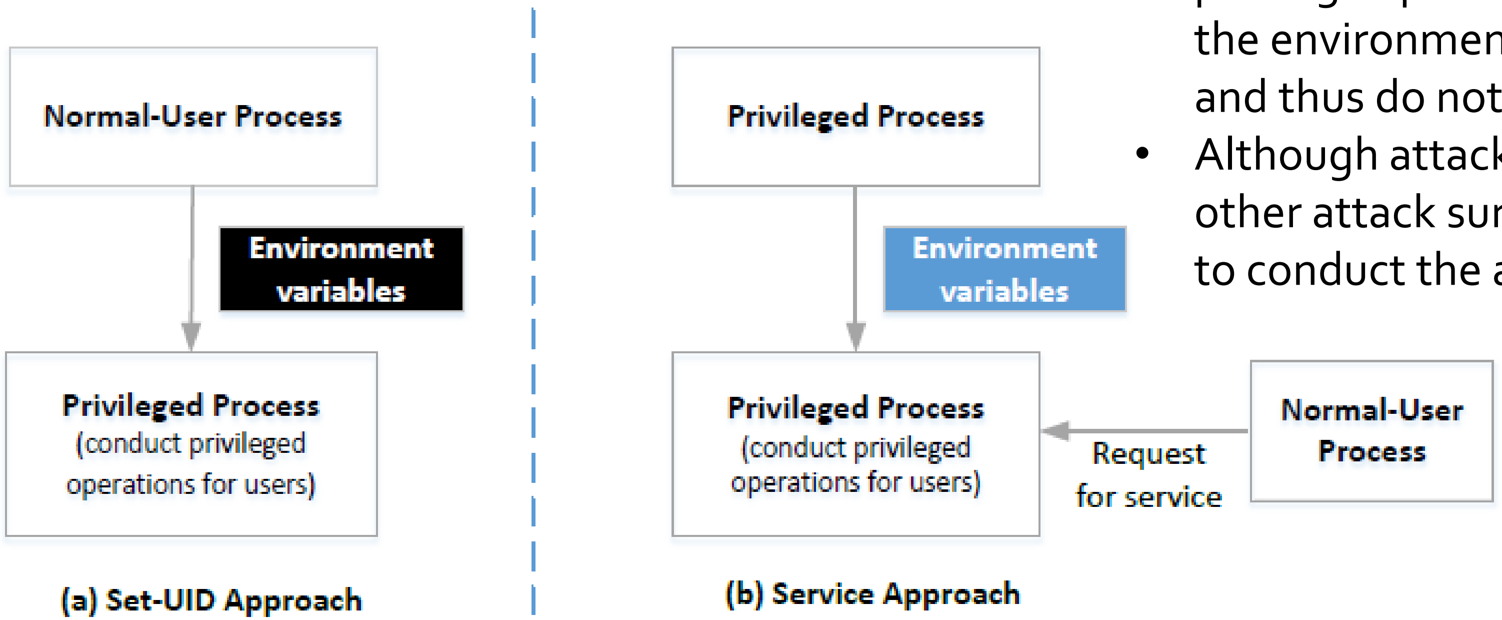
Principle of Least Privilege

- A privileged program should be given the power which is required to perform it’s tasks.
- Disable the privileges (temporarily or permanently) when a privileged program doesn’t need those.
- In Linux, seteuid() and setuid() can be used to disable/discard privileges.
- Different OSes have different ways to do that.

Set-UID versus Service Approach



Set-UID versus Service Approach



- In the service approach, the service is started by a privileged parent process or the operating system, so the environment variables come from a trusted entity, and thus do not increase the attack surface.
- Although attackers can still attack the service using other attack surfaces, there is no way for a normal user to conduct the attack via the environment variables.

Due to this reason, the Android operating system, which is built on top of the Linux kernel, completely removed the Set-UID and Set-GID mechanisms [Android.com, 2012].

Lab 1: Part 2

- **Use** the seed Ubuntu Lab image

Source code lab1.c:

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

int main(int argc, char * argv[], char * envp[])
{
	const char * env_var_name = "SHELL";
	char * shell;
	int i = 0;

	while (envp[i] != NULL)	{
		printf("%s\n", envp[i++]);
	}

	shell = (char *)getenv(env_var_name);

	if (shell) {
		printf("%s's value:   %s\n", env_var_name, shell);
		printf("%s's address: %p\n", env_var_name, shell);
	}
	else {
		printf("Value is not found for %s\n", env_var_name);
	}
}
```

```
$ gcc -o lab1 lab1.c -Wall
$ ./lab1
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1876,unix/VM:/tmp/.ICE-unix/1876
QT_ACCESSIBILITY=1
COLORTERM=truecolor
......
SHELL's value:   /bin/bash
SHELL's address: 0x7fff1c2a6460
```