

REVIEW: LECTURE 2 – CHAPTER 2

CS44500 Computer Security

Review: Lecture 2 – Chapter 2

Set-UID Concept

- **Allow user to run a program with the program owner's 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
```

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 \neq 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 Use:

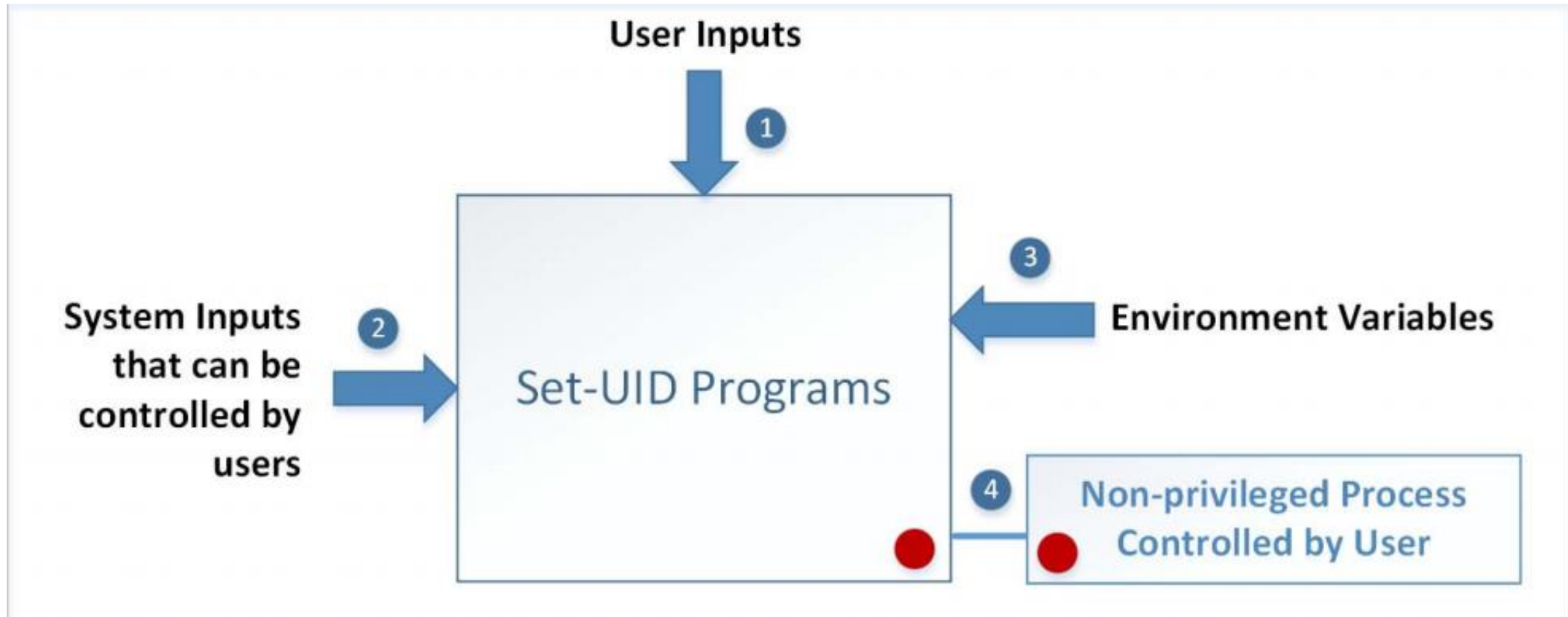
Bob gives you a chance to use his Unix account, and you have your own account on the same system. Can you take over Bob's account in 10 seconds?

- In Bob's account
 - `cp /bin/sh /tmp/mysh`
 - `chmod 4755 /tmp/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
- Note: 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 Surfaces of Set-UID Programs



Attacks via Capability Leaking: An Example

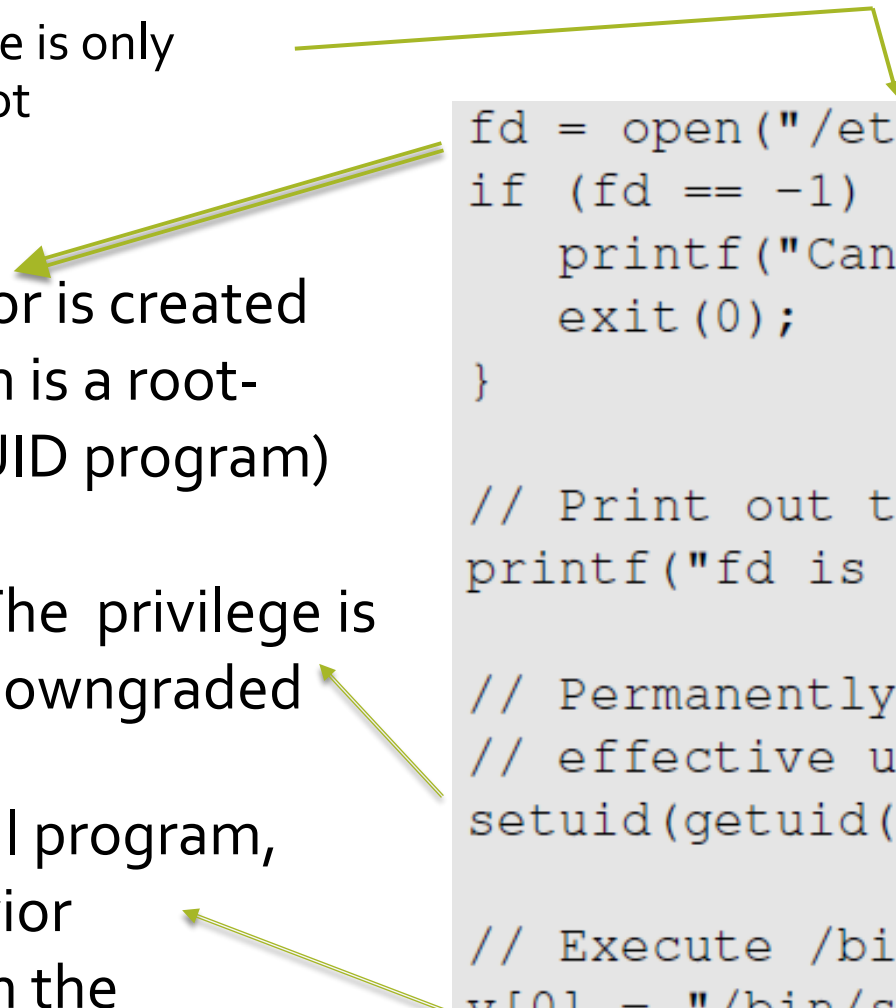
Attacks via Capability Leaking: An Example

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
bbbbbbbbbbbbbbbbbb
$ echo aaaaaaaaaa > /etc/zzz
bash: /etc/zzz: Permission denied ← Cannot write to the file
$ cap_leak
fd is 3
$ echo cccccccccccc >& 3 ← Using the leaked capability
$ exit
$ cat /etc/zzz
bbbbbbbbbbbbbbbbbb
cccccccccccccc ← File modified
```

How to fix the program?

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

Invoking Programs : Unsafe vs Safe Approach

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 the 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:::
```

We can get a root shell with this input



```
$ 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), ...
```

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

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

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
```

Principle of Isolation

Principle: **Don't mix code and data.**

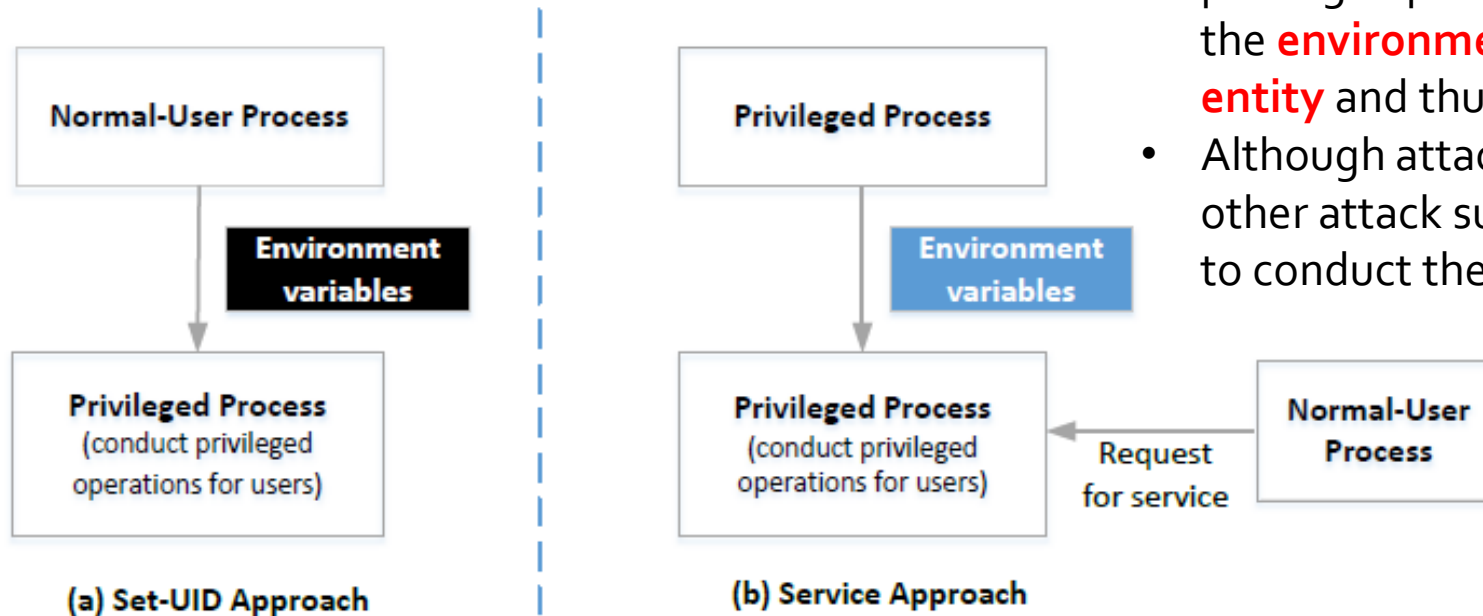
Attacks due to violation of this principle :

- 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 its 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



- 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].

Worksheet Questions

1. What is Set-UID bit?

Worksheet Questions

1. What is Set-UID bit?

The owner of the program allows other users to run the program with the owner's privilege.

2. What are RUID and EUID? When are they the same, and when are they different?

Worksheet Questions

1. What is Set-UID bit?

The owner of the program allows other users to run the program with the owner's privilege.

2. What are RUID and EUID? When are they the same, and when are they different?

Real UID (RUID): identifies the user who launched the process.

Effective UID (EUID): identifies privilege of a process.

Access control is based on EUID.

When a normal program is executed, $RUID = EUID$.

When a Set-UID is executed, $RUID \neq EUID$. RUID equals to the user's ID, but EUID equals to the program owner's ID.

Worksheet Questions

3. How to change a program to a Set-UID program owned by Bob?

Worksheet Questions

3. How to change a program to a Set-UID program **already** owned by Bob?

~~sudo chown Bob myProg~~ // not needed assuming already owned by bob

sudo chmod 4755 myProg

Note: cannot change the order! When changing the owner, the Set-UID bit is removed.

4. Alice run a Set-UID program that is owned by Bob. The program tries to read from /tmp/x, which is readable to Alice, but not to anybody else. Can this program successfully read from the file?

Worksheet Questions

3. How to change a program to a Set-UID program **already** owned by Bob?

~~sudo chown Bob myProg~~ // not needed assuming already owned by bob

sudo chmod 4755 myProg

Note: cannot change the order! When changing the owner, the Set-UID bit is removed.

4. Alice run a Set-UID program that is owned by Bob. The program tries to read from /tmp/x, which is readable to Alice, but not to anybody else. Can this program successfully read from the file?

No! EUID = Bob, which is what is used by the program to check the permission. The read access will be denied!

Worksheet Questions

5. What are attack surfaces of Set-UID programs?

Worksheet Questions

5. What are attack surfaces of Set-UID programs?

User inputs / system inputs / environment variables / non-privileged process controlled by user

6. How to steal root account / privilege within 10 seconds?

Worksheet Questions

5. What are attack surfaces of Set-UID programs?

User inputs / system inputs / environment variables / non-privileged process controlled by user

6. How to steal root account / privilege within 10 seconds?

```
cp /bin/sh /tmp/mysh  
chmod 4755 /tmp/mysh
```

Note: no need to use sudo as you already logged into the root account and have root privilege

Worksheet Questions

7. What are two ways to invoke external commands or programs? Which way is safer? Why?

Worksheet Questions

7. What are two ways to invoke external commands or programs? Which way is safer? Why?

Two ways: `system()` and `execve()`.

`Execve()` is safer. Code and data are clearly separated; there is no way for the user data to become code!

8. What are two principles in Chapter 1?

Worksheet Questions

7. What are two ways to invoke external commands or programs? Which way is safer? Why?

Two ways: `system()` and `execve()`.

`Execve()` is safer. Code and data are clearly separated; there is no way for the user data to become code!

8. What are two principles in Chapter 1?

(1) Separation of code and data (or “don’t mix code and data”).

(2) Least privilege.