
Set-UID Privileged 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/pDyc5U1BW0zkWh7T9ZGu.: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:::
```

Types of Privileged Programs

- **Daemons**

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

- **Set-UID Programs**

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

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
```

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.

Turn a Program into Set-UID

- **Change the owner of a file to root :**

```
seed@VM:~$ cp /bin/cat ./mycat
seed@VM:~$ sudo chown root mycat
seed@VM:~$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Nov  1 13:09 mycat
seed@VM:~$
```

- **Before Enabling Set-UID bit:**

```
seed@VM:~$ mycat /etc/shadow
mycat: /etc/shadow: Permission denied
seed@VM:~$
```

- **After Enabling the Set-UID bit :**

```
seed@VM:~$ sudo chmod 4755 mycat
seed@VM:~$ mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn
h/pDyc5U1BW0zkWh7T9ZGu.:15933:0:99999:7:::
daemon*:15749:0:99999:7:::
bin*:15749:0:99999:7:::
sys*:15749:0:99999:7:::
```

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) ...
```

Example of Set UID

```
$ 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

How is Set-UID Secure?

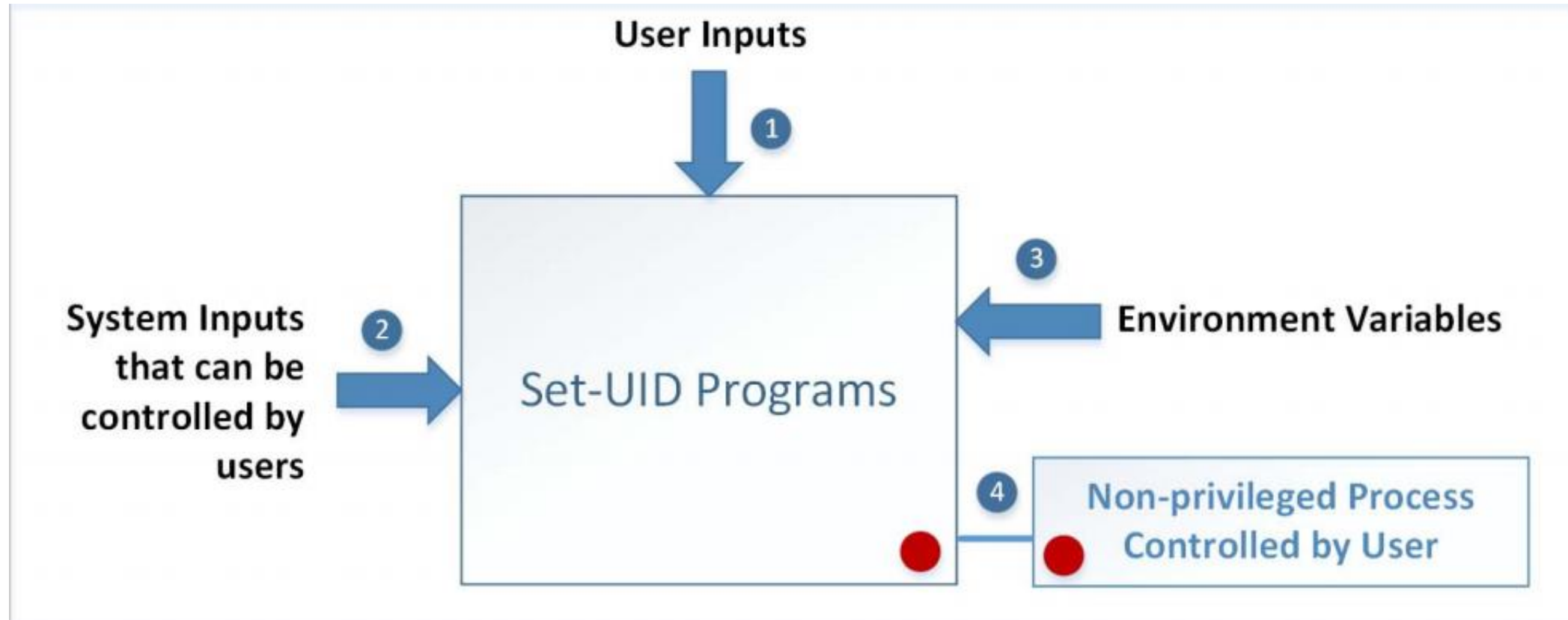
- **Allows normal users to escalate privileges**

- 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

Attack Surfaces of Set-UID Programs



Attacks via User Inputs

User Inputs: Explicit Inputs

- Buffer Overflow – More information next week
 - Overflowing a buffer to run malicious code
- Format String Vulnerability – More information later in the semester
 - 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

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

Attack

Attacks via System Inputs

System Inputs

- Race Condition
 - Symbolic link to privileged file from a 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 :** These can be set by a user before running a program.

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 the lab**

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

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
bbbbbbbbbbbbbbbb
$ 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
bbbbbbbbbbbbbbbb
cccccccccccccc ← File modified
```

How to fix the program?

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

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 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:::

$ 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

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 coming up
- SQL injection - More Information coming up
- Buffer Overflow attacks - More Information next week

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.**

Summary

- **The need for privileged programs**
- **How the Set-UID mechanism works**
- **Security flaws in privileged Set-UID programs**
- **Attack surface**
- **How to improve the security of privileged programs**