01 Set-UID

Information Security: A Hands-on Approach

Teacher: Po-Wen Chi
neokent@gapps.ntnu.edu.tw

Department of Computer Science and Information Engineering,
National Taiwan Normal University

Overview

1. Linux File Permission
2. Set-UID
3. What Goes Wrong?
4. Capability Leaking
5. Countermeasures
6. Conclusions
7. Appendix

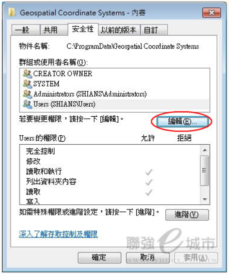
Linux File Permission

Access Control

- **Discretionary Access Control (DAC)**:
  - Restricting access to objects based on the identity of subjects and/or groups to which they belong.
  - The access permission is capable of passing that permission on to any other subject
- **Mandatory Access Control (MAC)**:
  - The policy administrators to implement organization-wide security policies.
  - Users cannot override or modify this policy, either accidentally or intentionally.
- **Role-based access control (RBAC)**:
  - Policy-neutral access-control mechanism defined around roles and privileges.
  - Role, not Identity.

File Permission

- Standard UNIX and Windows systems use DAC for file systems.
  - Users can grant other users access to their files, change their attributes, alter them, or delete them.



Geospatial Coordinate Systems - 內容

物件名稱: C:\ProgramData\Geospatial Coordinate Systems

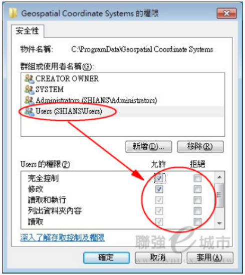
群組或使用者名稱(G):

CREATOR OWNER
SYSTEM
Administrators (SHIANS\Administrators)
Users (SHIANS\Users)

若要變更權限，請按一下 [編輯]。

Users 的權限(P)	允許	拒絕
完全控制		
修改		
讀取和執行	✓	
列出資料夾內容	✓	
讀取	✓	
寫入		

如需特殊權限或進階設定，請按一下 [進階]。



Geospatial Coordinate Systems 的權限

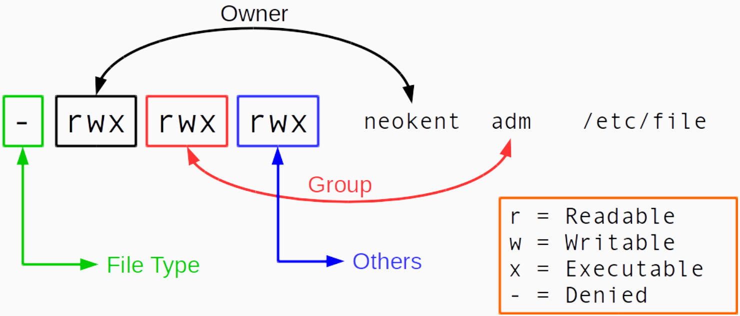
物件名稱: C:\ProgramData\Geospatial Coordinate Systems

群組或使用者名稱(G):

CREATOR OWNER
SYSTEM
Administrators (SHIANS\Administrators)
Users (SHIANS\Users)

Users 的權限(P)	允許	拒絕
完全控制	☑	□
修改	☑	□
讀取和執行	☑	□
列出資料夾內容	☑	□
讀取	☑	□

Linux File Permission



Owner

```
- rwx rwx rwx   neokent  adm   /etc/file
```

Group

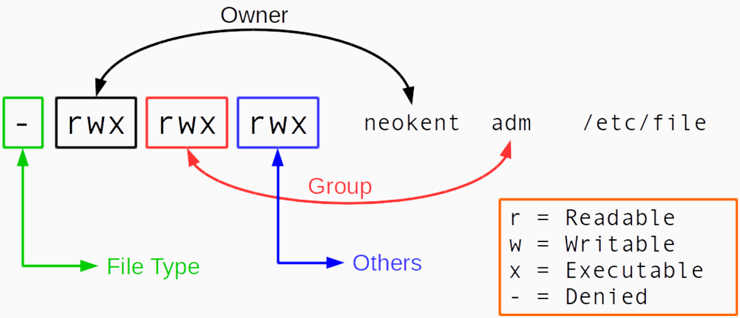
File Type

Others

```
r = Readable
w = Writable
x = Executable
- = Denied
```

Can be presented as a decimal form.

Linux File Permission



Owner

- rwx rwx rwx neokent adm /etc/file

Group

File Type

Others

```
r = Readable
w = Writable
x = Executable
- = Denied
```

Can be presented as a decimal form.

Quiz:

```
$ ls -l /bin/ls
-rwxr-xr-x 1 root root 126584  3月  3  2017 /bin/ls
```

User Scenarios

- Can a user change its own password?

User Scenarios

- Can a user change its own password?
- Can the system user password file, which contains all users' passwords, be accessed by every user?
  - If YES, it implies that you can access others' passwords.
  - If NO, it implies that you cannot change your password.

User Scenarios

- Can a user change its own password?
- Can the system user password file, which contains all users' passwords, be accessed by every user?
  - If YES, it implies that you can access others' passwords.
  - If NO, it implies that you cannot change your password.
- How will you set the file permission of the system password file?

By the Way

Actually, the user's password is not stored in **/etc/passwd**. Instead, the user's password is stored in **/etc/shadow** in its hash form.

Two Solutions

1. Daemon.
    - A daemon is a computer program runs with a privileged user as a **background** process.
    - When you want to change your password, send your request to the program.
2. Set-UID program.

Two Solutions

1. Daemon.
   - A daemon is a computer program runs with a privileged user as a **background** process.
   - When you want to change your password, send your request to the program.
2. Set-UID program.

Of course, you can give the user root's password directly... But you would not, right?

Set-UID

Set-UID Concept

- Allow a user to run a program with the program owner's privilege.
    - This is called **escalate privileges**.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 54256  3月 27  2019 /usr/bin/passwd
```

- s is the setuid flag.

User ID

In Unix, a process has three user IDs:

1. *Real User ID (RUID)*: The user who owns this process, not program.
2. *Effective User ID (EUID)*: The privilege that the process has.
3. *Saved User ID (SUID)*: A temporary space for switching effective user ID.

When a program is executed:

- For normal programs, EUID=RUID, which is equal to the ID of the user who runs the program.
- For set-UID programs, EUID≠RUID. RUID is equal to the ID of the user who runs the program, but EUID is equal to the ID of the user who owns the program.

If a Set-UID program is owned by root, the Set-UID program runs with the **root** privilege.

showid.c

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <pwd.h>

int main()
{
    uid_t ruid, euid, suid;
    struct passwd *pwd = NULL;

    getresuid( &ruid, &euid, &suid );

    pwd = getpwuid( ruid );
    printf( "Real User ID: %d (%s)\n", ruid, pwd -> pw_name );
    pwd = getpwuid( euid );
    printf( "Effective User ID: %d (%s)\n", euid, pwd -> pw_name );
    pwd = getpwuid( suid );
    printf( "Saved User ID: %d (%s)\n", suid, pwd -> pw_name );

    return 0;
}
```

showid

```
$ gcc showid.c -o showid
$ sudo chown root showid
$ ./showid
Real User ID: 1000 (neokent)
Effective User ID: 1000 (neokent)
Saved User ID: 1000 (neokent)
$ sudo chmod 4755 showid
$ ./showid
Real User ID: 1000 (neokent)
Effective User ID: 0 (root)
Saved User ID: 0 (root)
```

You can see the effective UID is changed to **root**.

An Example of Set-UID Program

```
$ cp /bin/cat mycat
$ sudo chown root mycat
$ ls -l mycat
-rwxr-xr-x 1 root neokent 52080  6月 28 15:38 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: 拒絕不符權限的操作
$ sudo chmod +s mycat
$ ./mycat /etc/shadow
root:!:17589:0:99999:7:::
...
$ sudo chown neokent mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: 拒絕不符權限的操作
```

Set-UID is Secure

- In principle, the Set-UID mechanism is secure.
- Though the Set-UID program allows the user to escalate its privilege, the program behavior is restricted by the software developer.

How about the following programs?

- vi
- /bin/bash

What Goes Wrong?

So Set-UID Program is Secure ... ?

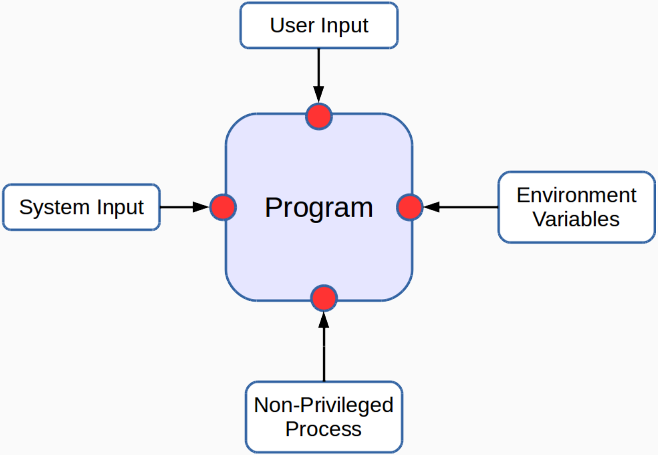
- The program is developed by human.
- **To err is human; ~~to forgive, divine~~.**
- There are many Code Flaws.

So Set-UID Program is Secure … ?

- The program is developed by human.
- **To err is human; ~~to forgive, divine~~.**
- There are many Code Flaws.

Wait!! I only release my software without source codes. How can an attacker affect my software?

Program Interfaces



Attacks via User Input

- Buffer Overflow.
- Format String Vulnerability.
- chsh.
  - Change your login shell.
  - The user's login shell is in **/etc/passwd**.
  - chsh is a Set-UID program.
  - Issues:
    - Failing to sanitize user inputs that the user may input two lines.
    - Attackers could **create a new account**, even **root**.

Attacks via System Input

Programs may get inputs from the underlying system.

- A privileged program may access a file which is stored in **/tmp**.
- /tmp is **world-writable**.
- So the attacker can control the file that the program accesses.

Attacks via Environment Variables

Environment Variable

An environment variable is a dynamic-named value that can affect the way running processes will behave on a computer.

Please try the command **env, export**.

Attacks via Environment Variables

```
system( "ls" );
```

- It seems that the program is secure since the command is hard-coded in the program and no one can change the command.
- The system() library function executes the shell (/bin/sh) command specified in command.
    - /bin/sh uses the **PATH** environment variable to find the program *ls*.
    - Let's check **PATH**.

Attacks via Environment Variables

```
system( "ls" );
```

- It seems that the program is secure since the command is hard-coded in the program and no one can change the command.
- The system() library function executes the shell (/bin/sh) command specified in command.
  - /bin/sh uses the **PATH** environment variable to find the program *ls*.
  - Let's check **PATH**.
  - Can you change **PATH**?

Attacks via Environment Variables

```
system( "ls" );
```

- It seems that the program is secure since the command is hard-coded in the program and no one can change the command.
- The system() library function executes the shell (/bin/sh) command specified in command.
  - /bin/sh uses the **PATH** environment variable to find the program *ls*.
  - Let's check **PATH**.
  - Can you change **PATH**?
  - export PATH=/some/new/path:$PATH

Attacks via Environment Variables

```
system( "ls" );
```

- It seems that the program is secure since the command is hard-coded in the program and no one can change the command.
- The system() library function executes the shell (/bin/sh) command specified in command.
  - /bin/sh uses the **PATH** environment variable to find the program *ls*.
  - Let's check **PATH**.
  - Can you change **PATH**?
  - export PATH=/some/new/path:$PATH
- Any other ways to attack?

Invoking Other Programs

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

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

    char *pCmd = malloc( strlen( pCatStr ) + strlen( argv[1] ) + 2 );
    sprintf( pCmd, "%s %s", pCatStr, argv[1] );
    system( pCmd );

    return 0;
}
```

What is the problem about the above code?

Invoking Other Programs

```
$ gcc catall.c -o catall
$ sudo chown root catall
$ sudo chmod 4755 catall
$ ./catall /etc/shadow
$ ./catall "aa;/bin/sh"
# whoami
root
```

Invoking Other Programs

```
$ gcc catall.c -o catall
$ sudo chown root catall
$ sudo chmod 4755 catall
$ ./catall /etc/shadow
$ ./catall "aa;/bin/sh"
# whoami
root
```

If this does not work, try the following command.

sudo ln -sf /bin/zsh /bin/sh

Shell's Countermeasure

Some shells do not allow that they are executed in a Set-UID process.

Invoking Other Programs: PHP

```php
<?php
    print( "Please specify the path of the directiry" );
    print( "<p>" );
    $dir=$_GET['dir'];
    print( "Directory path: " . $dir . "<p>" );
    system( "/bin/ls $dir" );
?>
```

php -S localhost:8000 list.php

Now you can use your web browser to browse the following url.

```
http://127.0.0.1:8000/list.php?dir=.;date
```

Capability Leaking

Capability Leaking

Please see **cap_leak.c** and explain its function.

Let’s See What Happens

```
$ gcc cap_leak.c -o cap_leak
$ sudo chown root cap_leak
$ sudo chmod +s cap_leak
$ cat /etc/zzz
aaa
$ echo "bbb" > /etc/zzz
-bash: /etc/zzz: Permission denied
$ ./cap_leak
fd is 3
$ echo "bbb" >> /etc/zzz
/bin/sh: 1: cannot create /etc/zzz: Permission denied
$ echo bbb >& 3
$ cat /etc/zzz
aaabbb
$ exit
```

Wait! I think I have already disabled the privilege?

Capability Leaking

Always destroy the capability before downgrading the privilege.

For the above example, you should **close** the file descriptor before downgrading.

OS X: Case Study

- Version: OS X 10.10.
- **DYLD_PRINT_TO_FILE**
  - A new environment variable.
  - This is a path to a (writable) file. Normally, the dynamic linker writes all logging output to file descriptor 2 (stderr). But this setting causes the dynamic linker to write logging output to the specified file.
  - It allows to open or create arbitrary files owned by the root user (Set-UID programs) anywhere in the file system.
- Issue: the opened log file is never closed.
- Reference: `https://www.sektioneins.de/blog/15-07-07-dyld_print_to_file_lpe.html`

Countermeasures

Important Principles

Principle of Isolation

Data should be clearly isolated from code.

Principle of Lest Privilege

Every program and every privileged user of the system should operate using the least amount of privileges necessary to complte the job.

Principle of Isolation

You should use **execve()** instead of system().

```
#include <unistd.h>

int execve( const char *pathname,
            char *const argv[],
            char *const envp[]);
```

You can see the **code** and **data** are separated.

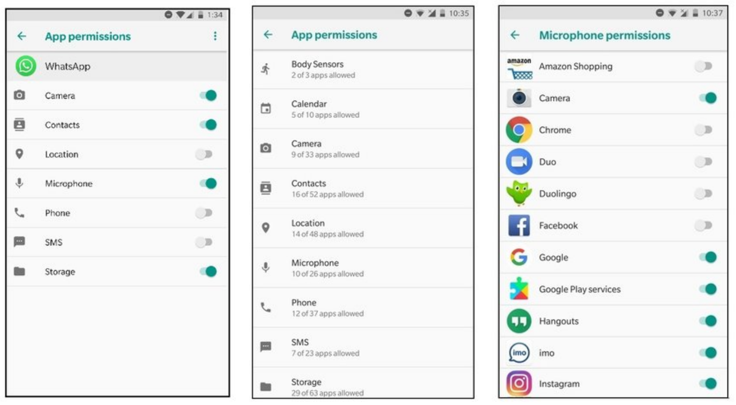
We will revisit this principle many many times in this class.

Principle of Privilege

You should use **setuid()** and **seteuid()** to disable the privilege when not necessary.

What is the difference?

Principle of Privilege: Android Example



1:34

← App permissions ⋮

WhatsApp

Camera	●
Contacts	●
Location	○
Microphone	●
Phone	○
SMS	○
Storage	●

10:35

← App permissions

Body Sensors	2 of 3 apps allowed
Calendar	5 of 10 apps allowed
Camera	9 of 33 apps allowed
Contacts	16 of 52 apps allowed
Location	14 of 48 apps allowed
Microphone	10 of 26 apps allowed
Phone	12 of 37 apps allowed
SMS	7 of 23 apps allowed
Storage	29 of 63 apps allowed

10:37

← Microphone permissions

Amazon Shopping	○
Camera	●
Chrome	○
Duo	○
Duolingo	○
Facebook	○
Google	●
Google Play services	●
Hangouts	●
imo	●
Instagram	●

Mehrnezhad, Maryam & Toreini, Ehsan. (2019). What Is This Sensor and Does This App Need Access to It?. Informatics. 6. 7. 10.3390/informatics6010007.

Conclusions

Conclusions

- Set-UID is a mechanism that can escalate the user's privilege in some restricted behavior temporarily.
- If the Set-UID program has flaws, the attacker can launch its attack through several interfaces with the root's privilege.
- When an attacker wants to launch attacks, generally it will focus on those Set-UID programs.

Appendix

How to Find Set-UID Programs?

```
$ find /bin -user root -perm -4000 -exec ls -ldb {} \; > ./tmp
$ cat tmp
-rwsr-xr-x 1 root root 40152  1月 27  2020 /bin/mount
-rwsr-xr-x 1 root root 44680  5月  8  2014 /bin/ping6
-rwsr-xr-x 1 root root 40128  3月 27  2019 /bin/su
-rwsr-xr-x 1 root root 30800  7月 12  2016 /bin/fusermount
-rwsr-xr-x 1 root root 44168  5月  8  2014 /bin/ping
-rwsr-xr-x 1 root root 27608  1月 27  2020 /bin/umount
```