Set-UID Program Vulnerability Graduate Lab Session

Kailiang Ying

Task 1: Understanding “passwd”, “chsh”, “su” and “sudo”

1. What do these commands do?

2. Why these commands need to be Set-UID program?

3. What if they are not Set-UID program?
    1> Where are these files?
    2> copy to your own folder
    3> run it and show me the result

Task 2: Run Set-UID programs

1. Where are these files?
2. Login as root and copy /bin/zsh to /tmp/
3. Make /bin/zsh set-root-uid program
4. Run it and show result in your report

Question: How do you know whether you get the root privilege or not?

5. Repeat the steps for /bin/bash and compare the difference with /bin/zsh.

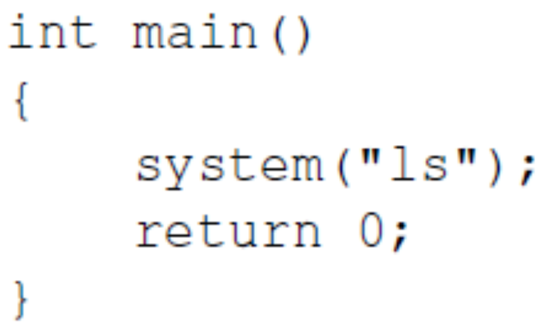
Task 3: Preparation for the following tasks

Use zsh as shell program and follow the commands in the lab description.

```
$ su
  Password: (enter root password)
# cd /bin
# rm sh
# ln -s zsh sh
```

Task 4: The PATH environment variable

Vulnerable Program:



```
int main()
{
    system("ls");
    return 0;
}
```

1> Set-root-Uid 4755
2> Relative path for invoking ‘ls’

Question: Which bit is set-uid bit?

Task 4: The PATH environment variable

(a) /bin/sh points to /bin/zsh

1> Change the PATH variable so that the set-uid program will run your own implementation of “ls”.

2> gcc –o ls ls.c

In your report, try to show the different from origin ls command.

(b) /bin/sh points to /bin/bash

repeat step 1 & 2 in (a)

Task 5: system() and execve()

Vulnerable Program:

```
v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;

/* Set q = 0 for Question a, and q = 1 for Question b */
int q = 0;
if (q == 0){
  char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);
  system(command);
}
else execve(v[0], v, 0);
```

1> Set-Root-Uid

2> User Input as part of the command

3> system() / execve()

Task 5: system() and execve()

/bin/sh points to /bin/zsh

(a) q=0

1. Compile the program and make it Set-Root-UID
2. Input the file name you want to see. What kind of attack can you do? (Can you run two commands at the same time?)

(b) q=1

repeat (a)

Task 5: system() and execve()

Attention:

Copy code directly from pdf to gedit may cause format issue.

Task 6: LD PRELOAD environment variable

1. mylib.c

```
#include <stdio.h>
void sleep (int s)
{
  printf("I am not sleeping!\n");
}
```

2. Build dynamic link library

Attention: -Wl, not one. Slash means concatenation

```
% gcc -fPIC -g -c mylib.c
% gcc -shared -Wl,-soname,libmylib.so.1 \
      -o libmylib.so.1.0.1 mylib.o -lc
```

Task 6: LD PRELOAD environment variable

3. pre-load the library

```
% export LD_PRELOAD=./libmylib.so.1.0.1
```

4. myprog.c

```
/* myprog.c */
int main()
{
  sleep(1);
  return 0;
}
```

Task 6: LD PRELOAD environment variable

1. Follow the instructions from (a) to (d)
2. Try the following cases:

    a>myprog a regular program, and run it as a normal user

    b>myprog a Set-UID root program, and run it as a normal user.

    c>myprog a Set-UID root program, and run it in the root account

    d>myprog a Set-UID user1 program, and run it as a different user.

Task 7: Relinquishing privileges and cleanup

Vulnerable Program:

```
fd = open("/etc/zzz", O_RDWR | O_APPEND);
if (fd == -1) {
   printf("Cannot open /etc/zzz\n");
   exit(0);
}

/* Simulate the tasks conducted by the program */
sleep(1);

/* After the task, the root privileges are no longer needed,
   it's time to relinquish the root privileges permanently. */
setuid(getuid());  /* getuid() returns the real uid */

if (fork()) { /* In the parent process */
  close (fd);
  exit(0);
} else { /* in the child process */
  /* Now, assume that the child process is compromised, malicious
     attackers have injected the following statements
     into this process */

  write (fd, "Malicious Data\n", 15);
  close (fd);
}
```

Task 7: Relinquishing privileges and cleanup

1. Compile and program and make it set-root-uid
2. run it
3. show me the result and **explain it**

Task 7: Useful Knowledge

- `seteuid(uid)`: It sets the effective user ID for the calling process.
  - If the effective user ID of the calling process is super-user, the uid argument can be anything. This is often used by the super-user to temporarily relinquish/gain its privileges. However, the process's super-user privilege is not lost, the process can gain it back.
  - If the effective user ID of the calling process is not super-user, the uid argument can only be the effective user ID, the real user ID, and the saved user ID. This is often used by a privileged program to regain its privileges (the original privileged effective user ID is saved in the saved user ID).

Task 7: Useful Knowledge

- `setuid(uid)`: It sets the effective user ID of the current process. If the effective user ID of the caller is root, the real and saved user IDs are also set.
  * If the effective user ID of the process calling setuid() is the super-user, all the real, effective, and saved user IDs are set to the uid argument. After that, it is impossible for the program to gain the root privilege back (assume uid is not root). This is used to permanently relinquish access to high privileges.
  * A setuid-root program wishing to temporarily drop root privileges, assume the identity of a non-root user, and then regain root privileges afterwards cannot use setuid(). You can accomplish this with the call seteuid().
  * If the effective user ID of the calling process is not the super-user, but uid is either the real user ID or the saved user ID of the calling process, the effective user ID is set to uid. This is similar to seteuid().

Task 7: Useful Knowledge

- Examples (in Fedora Linux): A process is running with effective user ID=0, and real user ID=500, what are the effective and real user IDs after running
    - `setuid(500); setuid(0);` Answer: 500/500 (the first call generates 500/500, and the second call fails).
    - `seteuid(500); setuid(0);` Answer: 0/500 (the first call generates 500/500, and the second call generates 0/500).
    - `seteuid(600); setuid(500);` Answer: 500/500 (the first call generates 600/500, and the second call generates 500/500).
    - `seteuid(600); setuid(500); setuid(0);` Answer: 0/500 (the first call generates 600/500, the second generates 500/500, and the third generates 0/500).

Criteria:

Task1:	10%
Task2:	10%
Task3:	0%
Task4:	25%
Task5:	20%
Task6:	20%
Task7:	15%