CS2107 Self-Exploration Activity 9: Linux Access Control

Notes:

In this Activity 9 about **access control in Linux/UNIX** and its **controlled invocation**, you will perform the following:

1. To set an executable program as a **set-UID-root** program, and observe the **real UID** and **effective UID** values of the running process of that program;
2. To understand the ***privilege-downgrading*** measure of bash, and how a set-UID-root program can still **bypass** the measure easily.

Note that you can also see view the **accompanying demo video** about Linux permission, which has been uploaded to LumiNUS.

Task 1: Creating a Set-UID-Root Executable Program

Download the `test-uid-1.c` file from LumiNUS, whose content is shown below:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   /* Print UIDs */
   printf("At the beginning of the program:\n");
   printf("Real user ID = %d, Effective user ID = %d\n\n",
           getuid(), geteuid());
   fflush(stdout);

   /* Cat the shadow file */
   system("cat /etc/shadow");

   return 0;
}
```

Do compile your C program using `gcc` as follows:

```
$ gcc -o test-uid-1 test-uid-1.c
```

Check the permission bits of the generated executable file `test-uid-1`.

Now, you want to make the executable program as **a set-UID-root program**. First, change its owner to `root` as follows:

```
$ sudo chown root:root test-uid-1
```

Then, enable its set-UID bit as follows:

```
$ sudo chmod u+s test-uid-1
```

Finally, run your set-UID-root `test-uid-1` program, and observe the **process outputs**, particularly the following:

- The process' two UID values;
- Whether the process can show the content of `/etc/shadow`, which is readable only by `root`.

The outputted two UID values should be clear enough based on how the set-UID bit works in UNIX/Linux. If you are curious on why the process still can't read `/etc/shadow` despite it running with the root's privilege, then you need to understand the **privilege-downgrading measure** of bash as explained below.

Note that `system()` library call uses fork to create a child process that executes the specified shell command using `execl()`. You can check the man page of `system()` at: https://man7.org/linux/man-pages/man3/system.3.html. As mentioned in the man page, if `system()` is called from a set-UID or set-GID program, and that `/bin/sh` is bash version 2, then bash 2 will *drop the elevated privilege* on startup as a security measure. This is why the process, despite having its effective UID of 0, still can't read the content of `/etc/shadow`.

Task 2: Bypassing bash's Privilege-Downgrading Measure

From the Task 1, you should notice that the **effective UID** of the process is 0 (`root`). Yet, the `cat` shell command still can't read the content of `/etc/shadow` due to the security measure applied by bash. The question is: Is it then possible to bypass the measure should the `root` user, who writes the executable, want it?

The answer to the questions is yes. As also mentioned in your tutorial, the bash measure can be easily defeated simply by calling `setuid(0)` prior to calling `system()`. As described at https://linux.die.net/man/3/setuid, the `setuid(`*uid*`)` library call will set the real UID and effective UID of the calling process to the supplied *uid*.

To test that we can bypass the bash's privilege-downgrading measure, download the extended C program named `test-uid-2.c` from LumiNUS, whose content is shown below:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   /* Print UIDs */
   printf("At the beginning of the program:\n");
   printf("Real user ID = %d, Effective user ID = %d\n\n",
           getuid(), geteuid());
   fflush(stdout);

   /* Change the real UID to 0, and print UIDs again */
   setuid(0);
   printf("After setting the real UID to 0:\n");
   printf("Real user ID = %d, Effective user ID = %d\n\n",
           getuid(), geteuid());
   fflush(stdout);

   /* Cat the shadow file */
   system("cat /etc/shadow");

   return 0;
}
```

Do compile the C program, and then turn the generated `test-uid-2` executable program into a set-UID-root program. Run the set-UID-root program, and observe whether the process can show the content of `/etc/shadow`.

You should be able to see now that, once the **real UID** of the process becomes 0 (`root`), the privilege-downgrading measure of bash can't stop the `cat` shell command from accessing the `/etc/shadow` file.