CS2107 Tutorial 7 (Privilege Escalation)

School of Computing, NUS

April 15, 2021

Complete this tutorial on Ubuntu or your favourite Unix-based os (e.g. macos) where you have root access. The description here makes a distinction between "program" and "process". For eg., "program `more`" refers to a file (which is an object in access control), and "process `more`" refers to the process executing the command (which is a subject in access control).

1. Try out these commands.

   - `man ps`
   - `man chown`
   - `man chmod`
   - `ps -a -o user -o ruid -o uid -o pid -o ppid -o command`
   - `which more`

   `more` is a command that display a file. E.g. `more a.txt` will display the file `a.txt`. By using `which`, we can get the directory that stores the program `more`. If you don't like `more`, use `less`.

   (a) What is the file permission of the program `more`?

   (b) What about the file permission of the program `ps`?

2. The setuid of the program `more` is different from the program `ps`. Give a reason on why they have to be different.

   Solution

   - The `ps` command displays the process status. Some information on processes are considered sensitive, for e.g. swap address, total blocks written (this could be viewed as side-channel information that might be useful in some attacks), and thus required root privilege to access. So, the setuid is on.
   - The `more` command reads and displays a file specified in the argument. Certainly, the user can only reads the file the user that has read access. So, not necessary to have setuid on. In fact, it doesn't make sense to have setuid on for `more`, as this will lead to confidentiality breach as demonstrated in the next few questions.

3. Create a file `a.txt`. The can be easily done by, for e.g., `echo Hello > a.txt` or `cat > a.txt`. What is the file permission and "owner" of the file?

4. Open a new shell (window). In this shell, issue the command `more a.txt`. Now, back to the original shell and issue the long `ps` command in the previous question.

(a) Note that there is a process executing the command `more a.txt`. What is its Process ID, ID of the Parent Process, Real UID and Effective UID?

Solution

Real and Effective UID is Alice(i.e the user).

(b) What is the Real UID and Effective UID of the process `ps`?

Solution

Real is Alice, Effective UID is root.

(c) The following lists all processes.

- `ps -ax -o user -o ruid -o uid -o pid -o ppid -o command`

Do you see a tree-like structure among all the PID's and PPID's?

Solution

note a process generated by a command named "bash" (depending on setting. In macos, you may see the "zsh" (known as z-shell)). This is the "shell". Commands entered in this shell are its child processes.

5. Copy the program `more` to your directory. This can be done by the command

   - `cp` ⟨path⟩`\more mymore`

   where ⟨path⟩ is the directory that stores `more`. Now you can view a file `a.txt` by using the command

   - `mymore a.txt`

6. Switch to the `root` user. (By issuing the command `sudo su`). Create a text file `b.txt` that stores your phone number and your favourite food. Change the file permission of `b.txt` so that it is not group readable, and not world (other) readable. Switch back to normal user (by issuing `exit`) and issue the command `more b.txt`. What is the outcome?

   Solution

   permission denied. The owner of `b.txt` is root and the file is not world readable.

7. **Backdoor Program.** Bob was an unhappy system administrator, and had root access on a system (i.e. he knew the root password). He was going to quit his job soon. Bob knew that his root password would be changed immediately after he left and that his home directory would be erased. In addition, the new system administrator would

also compare the system-level directories, such as `/usr/bin`, to make sure that no malicious changes had been made.

Let us assume that Bob would either still be given a local user, or knew the password of an existing local user, or alternatively had created a local user that would go unnoticed.

Bob wanted to maliciously plant a backdoor into the target system. He wanted to make a very small change to the system so that later, when he logged in as a normal user, he could read any file with root privilege. Describe how Bob could create the backdoor.
(*Hint*: by creating a program like `mymore` with an elevated privilege, and "hiding" that program somewhere in the system.)

Solution

using `mymore`. (1) switch to root. (2) change owner of the file mymore to root. (3) make the file world executable. (4) change the setuid of the file to on. (5) move the file to a "hidden directory", for e.g. `/tmp` and change the filename to some obscure name, e.g. `temp1399822`).

8. **(Optional. This was in previous year tutorial. Included here FYI).**
(Setuid in program) The real and effective user IDs of a process can be modified by the process. Compile and run the C program, eg.

   - `gcc test.c -o test`
   - `./test`

Run the program under root. Also run the program under a normal user. Why the outcome is different? What should you do if you want to get the same outcome when a normal user runs the program? (Note that the root's id is 0).

Solution

When ran by normal user, the effective uid is not changed to "0" (which is the root). This is because the process (that runs the program) has effective uid of the normal user. See the man page of `setuid`: "the setuid() function is permitted if the effective user ID is that of the super user, or if the specified user ID is the same as the effective user ID. If not, but the specified user ID is the same as the real user ID, setuid() will set the effective user ID to the real user ID." So, to get the same outcome, change the owner to root, and set the permission to "s".)

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

int main()
{
  printf ("Real user id = %d, Effective user id = %d\n", getuid(), geteuid());
  seteuid (1001);    /* set effective uid*/
  printf ("Real user id = %d, Effective user id = %d\n", getuid(), geteuid());
  seteuid (0);
  printf ("Real user id = %d, Effective user id = %d\n", getuid(), geteuid());
  seteuid(1001);    /* set only the effective uid */
```

```
  printf ("Real user id = %d, Effective user id = %d\n", getuid(), geteuid());
  return(0);
}
```

9. Terminology: *Insider threat*.

— End of Tutorial —