Set-UID Programs

1.1. Alice runs a `Set-UID` program that is owned by Bob. The program tries to read from `/tmp/x`, which is readable to Alice, but not to anybody else. Can this program successfully read from the file?

1.2. A process tries to open a file for read. The process's effective user ID is 1000, and real user ID is 2000. The file is readable to user ID 2000, but not to user ID 1000. Can this process successfully open the file?

1.3. A root-owned `Set-UID` program allows a normal user to gain the root privilege while executing the program. What prevents the user from doing bad things using the privilege?

1.4. We are trying to turn a program `prog` owned by the `seed` user into a `Set-UID` program that is owned by root. Can running the following commands achieve the goal?

```
$ sudo chmod 4755 prog
$ sudo chown root prog
```

1.5. The `chown` command automatically disables the `Set-UID` bit, when it changes the owner of a `Set-UID` program. Please explain why it does that.

1.6. When we debug a program, we can change the program's internal variables during the execution. This can change a program's behavior. Can we use this technique to debug a `Set-UID` program and change its behavior? For example, if the program is supposed to open the `/tmp/xyz` file, we can modify the filename string, so the `Set-UID` program ends up opening `/etc/passwd`.

1.7. Both `system()` and `execve()` can be used to execute external programs. Why is `system()` unsafe while `execve()` is safe?

1.8. When a program takes an input from users, we can redirect the input device, so the program can take the input from a file. For example, we can use `prog < myfile` to provide the data from `myfile` as input to the `prog` program. Now, if `prog` is a root-owned `Set-UID` program, can we use the following method to to get this privileged program to read from the `/etc/shadow` file?

```
$ prog < /etc/shadow
```

1.9. When a parent `Set-UID` process (effective user ID is root, and the real user ID is `bob`) creates a child process using `fork()`, the standard input, output, and error devices of the parent will be inherited by the child. If the child process drops its root privilege, it still retains the access right to these devices. This seems to be a capability leaking, similar to the capability-leaking case covered in this chapter. Can this pose any danger?

1.10. ★
The superuser wants to give Alice a permission to view all the files in the system using the `more` command. He does the following:

```
$ cp /bin/more /tmp/mymore
$ sudo chown root /tmp/mymore
$ sudo chmod 4700 /tmp/mymore
```

Basically, the above commands turns `/tmp/mymore` into a `Set-UID` program. Right now, because the permission is set to `4700`, other users cannot execute the program. The superuser uses another command (now shown) to grant the execution permission only to Alice. We are not assuming that Alice is completely trusted. It is OK if Alice can only read other people's files, but it is not OK if Alice can gain any privilege beyond that, such as writing to other people's files. Please read the manual of the `more` program and find out what Alice can do to gain more privilege.

1.11. Assume that you have a file that you would allow other users to read, only if a user's ID is smaller than 1000. Please describe how you can actually achieve this.

1.12. Sam found a very useful web page, which contains links to many interesting papers. He wants to download those papers. Instead of clicking on each of the links, he wrote a program that parses a HTML web page, get the papers URLs from the web page, and then use a program called wget to fetch each identified URL. The following is the code snippet:

```
char command[100];
char* line, url;
line = getNextLine(file);// Read in one line from the HTML file.
while (line != NULL) {
  // Parse the line to get a URL string.
  url = parseURL (line);
  if (url != NULL){
    // construct a command, and execute it
    sprintf(command, "%s %s", "wget", url);
    system(command);
  }
  line = GetNextLine(file);
}
```

The function `sprintf()` is quite similar to `printf()`, except that `sprintf()` puts the output in a buffer pointed by the first argument, while `printf()` sends the output to the display. Please be noted that the functions `getNextLine()` and `parseURL()` are also implemented by Sam (their code is not displayed here). The program `wget` is a command-line program in Unix that can be used to download web files from a given URL.

The owner of the web page knows what Sam is doing with his page; he wants to attack Sams program. He knows the code above, but he does not know how Sam implements `GetNextLine()` or `ParseURL()`, but he suspects that Sam may make some mistakes there. (1) If you are the attacker, please describe how you plan to attack. (2) How do you fix the problem?