Chapter 1: Set-UID Privileged Programs and Attacks on Them

Problems

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 what we covered in Chapter 1.4.4. Can this pose any danger?