**Kaleb Alstott**

**Lab 1**

**TASK 2**

A screenshot of a computer

Description automatically generated

In the screenshot above, after we downloaded the files needed and change directory to the correct place, we first do a gcc command in which is a compiler system that compiles code into an executable file in which we named p\_capNewFile. After we had to change ownership of this file as well as the permissions. The ownership was changed to Root and the permissions to a 4755. This is the commands used to make this a SET-UID program. This is displayed with the ls -l command.

A screenshot of a computer

Description automatically generated

We run the program as a normal user, and we cannot open and or touch the /etc/zzz due to permission being denied as of a result of the file being permissioned to root. We run the “sudo touch” super user command and created a new empty file named /etc/zzz. We then run the cat command and see that the file is empty. We can now sudo in with our text editor and put in the text of this new empty file as a normal user.

A screenshot of a computer screen

Description automatically generated

Since /etc/zzz is owned by root, we need to use sudo to open and modify the file.

A screenshot of a computer screen

Description automatically generated

After the program drops the privileges and spawns a shell, we can attempt to write to /etc/zzz using the file descriptor (fd) that was opened while the program still had the root privileges.

The file descriptor number printed out by the program in which our case is 3 when it opened /etc/zzz. The file descriptor now remains open and writable even after the program's privileges have been dropped.

A screenshot of a computer

Description automatically generated

Due to the file remaining open here we have been able to write to the file using the echo command and produce our attack which is “NEW TEXT ATTACK!!!!!”.

**Questions:** Can you exploit the capability leaking vulnerability in this program? The goal is  
to write to the /etc/zzz file as a normal user.

**Response -** Yes, we can exploit the capability leaking vulnerability in this program due to the file descriptor remaining open and writable even after the program's privileges have been dropped. Although the program downgrades its privileges by using SET-UID, the file descriptor remains open and allows the non-privileged process to read from and write to the file. Due to this we can see above that the new text in comparison to the original text of “hello world”. We were indeed able to read and write into the open file and deploy and see our attack as a normal user.

**TASK 1**

A screenshot of a computer

Description automatically generated

In the screenshot above, after we downloaded the files needed and change directory to the correct place, we first do a gcc command in which is a compiler system that compiles code into an executable file in which we named prog-catall. After we had to change ownership of this file as well as the permissions. The ownership was changed to Root and the permissions to a 7455. This is the commands used to make this a SET-UID program. This is displayed with the ls -l command. The following command “sudo ln -sf /bin/zsh /bin/sh” is for removing the countermeasure (make sure to do this at the end as well). As Bob we are able to executes the program with the specific file, we chose /etc/shadow.

A screenshot of a computer

Description automatically generated

In this command we are showing how Bob could potentially remove files or perform other unauthorized actions if the command string is not proper.

A screenshot of a computer

Description automatically generated

Command for showing the current user and groups we currently belong to

A screenshot of a computer

Description automatically generated

Showing that we are the root user currently logged in.

A screenshot of a computer

Description automatically generated

Showing the change in file when we commented out system(command); for the execve().

A screenshot of a computer

Description automatically generated

We recompile the new code and set the ownership and permission to the according specifications in the document. The execve() does not invoke a shell instead it will directly execute the specified program with the given arguments. In which we can see we are denied due to the specific requirements not being met.

A screenshot of a computer

Description automatically generated

Final command to reinstate the countermeasure.

**Question -** Step 1: Compile the above program, make it a root-owned Set-UID program. The program will use system() to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?

**Answer -** Yes, Bob was able to compromise the integrity of the system using the system() function. Since system() invokes a shell Bob could pass his input to the program, allowing him to execute the commands needed. Even though the program is designed to only display files, Bob was able to still exploit the root privileges granted by the Set-UID program

**Question -** Step 2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it a root-owned Set-UID. Do your attacks in Step 1 still work? Please describe and explain your observations.

**Answer -** The attacks in step 1 didn’t work due to the new line of code that we uncommented which was the execve() statement. Previously we learned that system() invokes a shell for Bob while in comparison execve() does not invoke a shell. With execve() Bob is restricted to only viewing files as intended thus being without the ability to execute specific commands or escalating privileges.