Assignment 2 (Marks - 100)

Deadline: September 23, 2023. Time 23:55 IST.

Setup:

To fetch the assignment, open the terminal in the VM and run the following command:

```
$ wget http://172.29.233.235:8000/Assignment_2.zip
```

This will fetch the zip file in your current directory. Then you can run the following to extract the files:

```
$ unzip Assignment_2.zip
```

Now you should have all the relevant files to complete the assignments. However, before you start it is important to create a backup of the file `/bin/sh` as in the following assignments you will have to temporarily modify this file. Therefore, make a copy using the command:

```
$ sudo cp /bin/sh /bin/sh.orig
```

At the end of the assignment, you can restore the `/bin/sh` file.

Assignment 2a (20 marks):

After extraction, in the `Assignment_2a` directory you will find two programs that contain use of multiple programs and functions that might contain buffer overflow, integer overflow and format string vulnerability.

Task: Your task is to identify each of these problems, cite the vulnerable line(s) in the code and give precise argument about how you can exploit this vulnerability and give countermeasures to rectify such problems.

Assignment 2b (4x20 marks):

In the Assignment_2b directory, you are given directories named `Task_1`, `Task_2`, and `Task_3`. These directories contain essential files to complete the assignment. In each of these folders, you will be given a program with a buffer-overflow vulnerability named `vulnerable.c`; your task is to develop a scheme to exploit the vulnerability and finally gain the root privilege. In addition to the attack, students need to create a report on whether their schemes work or not and explain why.

Environment Setup

Modern operating systems have implemented several security mechanisms to make the buffer-overflow attack difficult. To simplify our attacks, we need to disable them first. Later, we will enable them to see if our attack can be successful.

Address Space Randomization: Ubuntu and several other Linux-based systems use address space randomization to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult; guessing addresses is one of the critical steps of buffer-overflow attacks. This feature can be disabled using the following command:

```
$ sudo sysctl -w kernel.randomize_va_space=0
```

Configuring `/bin/sh`: In the recent versions of OS, the `/bin/sh` symbolic link points to the `/bin/dash` shell. The dash program, as well as bash, has implemented a security countermeasure that prevents itself from being executed in a Set-UID process. Since our attack relies on running `/bin/sh`, the countermeasure in `/bin/dash` makes our attack more difficult. Therefore, we will link `/bin/sh` to another shell that does not have such a countermeasure. The following command can be used to link `/bin/sh` to `zsh`:

```
$ sudo ln -sf /bin/zsh /bin/sh
```

Remember that `zsh` will already be installed on the system.

StackGuard and Non-Executable Stack: These are two additional countermeasures implemented in the system. They can be turned off during the compilation by using the flags:

```
-z execstack -fno-stack-protector
```

You are already provided the file `setup.sh` in each of the directories `Task_1`, `Task_2`, and `Task_3`. This will take care of the first two cases if you run:

```
$ bash setup.sh
```

But remember to create a backup of `/bin/sh` as it was mentioned in **Setup** section before you run `setup.sh`.

Also, the `Makefile` (in each `Task_X` directory for `X=1,2,3`) contains methods to compile the code with the required flags to turn off the third countermeasure. All you need to do is to run:

```
$ make
```

to compile the code. After compilation it will create the files `vul-LX`, `vul-LX-dbg` in the directory `Task_X`.

Problems to Solve

You have already been provided with the code `vulnerable.c`, which has to be compiled with the provided `Makefile` by simply calling make in the terminal. Three Set-UID programs will be created, namely `vul-L1`, `vul-L2`, and `vul-L3`. Also, the files `vul-L1-dbg`, `vul-L2-dbg`, and `vul-L3-dbg` will also be created to be used for debugging the program in gdb. You also must fill in the data in `exploit.py` that writes the binary data in the `badfile`, which is read by the vulnerable program. To check the exploit, you can run:

```
$ python3 exploit.py && ./vul-L1
```

Or you can make the `exploit.py` file executable and run `test.sh`. You must complete the following four tasks:

Task 1: The program `vulnerable.c` is compiled as 32-bit program which is the executable file `vul-L1`. Your first task is to exploit the buffer overflow vulnerability here.

Task 2: Similarly, the program `vulnerable.c` is compiled as 64-bit program which is the executable file `vul-L2`. You must exploit the buffer overflow vulnerability here.

Task 3: In the previous task, the `BUFF_SIZE` was significantly large but for this task, the program `vul-L3` is compiled with small `BUFF_SIZE` of value 10 (as a 64-bit program). You must exploit the buffer overflow vulnerability here.

Task 4: For this last task, we will see what happens if we use bash instead of zsh. Copy the directory Task_1 and create a directory Task_4. In the setup.sh file delete the line-3 and uncomment the line-7. Run the following command to setup:

```
$ bash setup.sh
```

This will link `/bin/bash` to `/bin/sh`. Now if you try to run the above exploits you will see that instead of a root shell, you will get a normal shell i.e., it drops the root privileges. Why does this happen? What can you change in your attack method that can bypass this countermeasure? Complete the attack by bypassing this countermeasure.

You will have to provide documentation with proper screenshots, explaining every step of how you exploit the vulnerability for all the tasks above. Also, you would have to provide a working code for each of the tasks that give a root shell by exploiting buffer overflow vulnerability. (You must complete/answer the tasks/questions highlighted in grey). Each of the tasks Task_1, Task_2, Task_3 and Task_4 carries 20 marks each.

NOTE: *The* `shellcode.c` *file in the directory* `shellcode`*,contains the binary shell code that executes* `/bin/sh` *which gives you the shell. Executing from a Set-UID program makes it open a shell with root privilege. Both 32-bit shell code and 64-bit shell codes are provided. You can try compile, run the code, and see if the provided shell code is able to open a shell or not.*

Deliverables:

1. For the Assignment_2a, you will create a report as it is mentioned in the Assignment_2a section.
2. For the Assignment_2b, again you will have to create a report as mentioned in the question.
3. Keep the reports of two sections in one single document. Make sure to use only **Calibri font** to write the document. Use **12-pt** font to write your document, **16-pt** for heading, and **14-pt** for any subsection heading. Make sure that any screenshot you attach here is not blurry or illegible.
4. The codes that you have modified/created for Task_1, Task_2, Task_3, Task_4 all must be zipped along with your report. Zipped file must be named in the following format:

   `<roll number>_<firstname>_<lastname>_assignment2.zip`

You can upload the zip file in the same manner as you did it for assignment 1.