Homework 1

Due: Sep. 28, 2024 23:59:59

- **Environment.** We assume that you have already installed the Linux box (vagrant) we provided in the class. All the problems should be solvable in any regular Linux environment, but it would be easier for you if you use the VM that we provided.
- **Late submission policy.** Please refer to the course web page.
- **Submission rule.** We will deduct ***half the point*** if you do not follow the submission rule.
  - You should submit a single PDF file (We recommend using the LaTeX template we provided in the course web page).
  - Write your report using your *mother language*.
  - The name of the PDF file should have the following format: `your_ID-last_name.pdf`. For example, if your name is Gil-dong Hong, and your ID is 20244242, then you should submit a file named "`20244242-Hong.pdf`".
  - If your solution includes some code (assembly, C, etc.), you should embed them in your PDF.

Problem 1. Warm-up: Discovering the CTF Server (5 points)

In this homework, you are going to solve CTF-style problems. However, we are not going to directly give you the IP address of the CTF web server. Instead, you should discover it by yourself using the following hints.

1. Decipher this string: "fihcgimchmcgfgomeee".
2. The encrypted string represents an IP address and a port number.
3. Learn about substitution cipher.
4. In a Linux terminal, type `man ascii`.

Describe how you discovered the CTF server[1]. You should clearly show logical reasoning.

[1] Please note that the server is only accessible inside the campus. You should use VPN to connect to the page from outside.

Problem 2. Shellcoding (20 points)

Shellcode is a small piece of code that runs a command-line shell such as `/bin/bash` on Linux or `cmd.exe` on Windows. Spawning a command-line shell is important in terms of exploitation because it allows an attacker to run any arbitrary code. In this problem we ask you to write your own shellcode.

(a) (1 point) Locate a file in Ubuntu/Debian Linux (you can use the provided VM) that has a complete list of syscall numbers for Linux x86.

(b) (1 point) Locate a file in Ubuntu/Debian Linux (you can use the provided VM) that has a complete list of syscall numbers for Linux x86-64 (amd64).

(c) (1 point) What is an ABI (Application Binary Interface)? Why is this related to syscalls?

(d) (1 point) Describe the syscall calling convention of Linux x86.

(e) (1 point) Describe the syscall calling convention of Linux x86-64.

(f) (15 points) Write a self-contained executable—meaning that the executable binary does not require any external libraries—in x86 (32-bit) assembly. The program should take in a command-line argument (`argv[1]`) as input and output the reversed string. For example, if you give "ABCD", then the program should print out "DCBA". The assembly code you wrote should compile with gcc with the option `-nostdlib` to make sure that you do not link with the standard C library (i.e., libc). ~~Also, your code should be properly commented.~~

Problem 3. CTF 1: ShellEval (30 points)

Assuming that you have figured out the CTF web server from Problem 1, you should now be able to see the CTF problems from the web server. You should be able to get more information about this problem from the CTF web server. In this problem, you can evaluate your shellcode remotely by connecting to the server and send your shellcode (for example, you could use `netcat` to do so). For your reference, you can find the source code of the server application at `https://github.com/sangkilc/shelleval`. Additionally, note that the evaluator processes are respawned every minute: even if you successfully got a shell, the connection will not last for more than a minute. There are two subproblems: one is for ~~x86 shellcoding~~, and the other is for x86-64 shellcoding.

(a) (5 points) Get the flag by spawning a shell (w/ `execve` syscall). The server is running an x86 service. ~~Explain in detail how you created your shellcode with proper comments.~~

(b) (5 points) Get the flag by spawning a shell. The server is running an x86-64 service. ~~Explain in detail how you created your shellcode with proper comments.~~

(c) (5 points) Suppose the service administrator of ShellEval realized that their service can be exploited by a malicious user. So the administrator decided to disable the use of a "shell" (i.e., `/bin/sh`) by closing the connection immediately after a connection is established. This means, user-interaction is prohibited. So the user now can only send a command and see the output once. Can you still read the flag file by modifying your shellcode? Which system call would you use to attack the modified service? Show a code snippet, and explain.

(d) (5 points) Write a shellcode that can extract the flag for the modified service. The shellcode should work on both original and modified service. Explain your shellcode with proper comments.

(e) (10 points) Disassemble the following shellcode written in C string, and explain what it does line by line. Note that the shellcode does not work on Linux, but on another OS. Can you find out which OS will this shellcode spawn a shell?

```
char shellcode[] =
    "\xeb\x0e\x5e\x31\xc9\xb1\x1c\xfe\x04\x0e\xe2\xfb\xfe\x06\x56"
    "\xc3\xe8\xed\xff\xff\xff\xea\x0d\x5d\x30\xbf\x87\x45\x06\x4f"
    "\x53\x55\xaf\x3a\x4f\xcc\x7f\xe7\xec\xfe\xfe\xfe\x2e\x61\x68"
    "\x6d\x2e\x72\x67";
```

Problem 4. CTF 2: SetLev (20 points)

In this CTF problem, you need to gain the local privilege escalation by exploiting a bug in a poorly written program called `setlev`. You need to first connect (via ssh) to the server machine to solve this challenge. You will be given an account (ID = `setlev`), and every student will share the same account. Your goal is to exploit the vulnerable program located in `/home/setlev/setlev` to read the flag stored in `/home/setlev/flag.txt`.

Since the same account is shared by all the students, you may wonder where you should store your script/program for exploitation. To help you have your own private directory, we made the `/tmp` directory to be un-listable, meaning that no one can run `/bin/ls` on the directory. Thus, if you create your own directory under `/tmp` with a random name, then the other people cannot see it easily. Inside the directory, you can list your own files and develop your exploit as usual. Again, the name of the directory should be hard to guess, otherwise, other students will easily access your files.

For your information, this machine has GCC, GDB, VIM editor, Python, Perl, and dotnet (C# and F#) installed. If you need an additional tool installed on the system, ask the TAs.

See the description in the CTF web page and answer the following questions.

(a) (1 point) Who is the owner of the flag located at `/home/setlev/flag.txt`? What kind of permission does the flag have?

(b) (1 point) What is the requirement for your exploit (i.e., shellcode) in order to read the flag? Explain.

(c) (10 points) Reverse engineer the program (`setlev`), and show the corresponding C code. What does this program do? What kind of vulnerability does this program have?

(d) (6 points) Exploit the program (`setlev`) in order to read the flag. Make the best use of your shellcode you wrote in the previous problem. Explain in detail how you exploited the program. If you used a script to exploit it, please include the script in the writeup. If you show your work, you can get partial points even if you cannot get the flag.

(e) (2 points) The server is equipped with PoE. So use PoE to get the flag. If you have already used PoE to solve the previous problem, you can leave this part empty.

Problem 5. CTF 3: BadFormat (25 points)

In this CTF problem, you need to gain local privilege escalation by exploiting a bug in a poorly written program called `badformat`. This problem runs in the same system. so use the `/tmp` directory as you did in Problem 4.

(a) (10 points) Reverse engineer the program (`badformat`), and show the corresponding C code. What does this program do? What kind of vulnerability does this program have?

(b) (15 points) Exploit the program (`badformat`) in order to read the flag. Make the best use of your shellcode you wrote in the previous problem. Explain in detail how you exploited the program. If you used a script to exploit it, please include the script in the writeup. If you show your work, you can get partial points even if you were not able to get the flag. You should use PoE to get the flag this time. Otherwise, you will not get full credit.