CSE 467 Homework #2: Software Security

Due: Oct. 24, 11:59 PM

- **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. It is recommended to turn off ASLR on your VM via following commends:

  ```
  $ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
  ```

- **Late submission policy.** Please refer to the course web page.
- **Submission guidelines.**
  - You should submit both your report and flags.
    - Solve the problems and submit the flags to our homework webpage: `http://10.20.12.187:4000`
    - You should upload a single PDF file on BlackBored. Your report must describe the answer to each question in this homework.
  - 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 20231234, then you should submit a file named "`20231234-Hong.pdf`".
  - If you solution includes some code (assembly, C, etc.), you should embed them in your PDF.
- **Capture The Flag (CTF) guidelines.**
  - You can find each problem on our homework webpage: `http://10.20.12.187:4000`
    - This server is accessible only from the UNIST internal network. If you want to connect from outside, use a VPN.
    - Please note that we have sent an email to everyone regarding your login credentials to the web server. Please contact course staffs if you cannot log in to the web site.
    - You can change your password, but we recommend entering a random password that you do not normally use. Note that our webpage does not provide a secure connection.
  - If you solve each challenge, you will be able to obtain a flag. Submit the found flag to the website. Each flag is in the following format: `flag{some_string}` (e.g., `flag{C0N9R@7u1aT1on!}`).
  - Your score in the CTF scoreboard is nothing to do with the actual score for your homework. The CTF score is just for fun.
  - Do not attack the CTF environments, including web services!
  - If you think the services are not working properly or have any questions, please publicly upload your question on the BlackBored.

Problem 1. CTF Warm-up: Basic (10 points)

In this challenge, you should donwload and analyze the provided binary.

(a) (3 points) Reverse engineer the main function of the provided binary (`basic`), and show the corresponding C code. What does this program do?

(b) (3 points) There is a function that is not affected by control flow. What is the name of that function?

(c) (4 points) Reverse enginner aforementioned function, show the corresponding C code, and get the flag.

Problem 2. RetFunc (10 points)

(a) (3 points) Reverse engineer the main function of the provided binary (`retfunc`), and show the diagram of the stack at the execution of `0x80491fb`.

(b) (3 points) There is a function that is not affected by control flow of the program. Reverse enginner this function, show the corresponding C code.

(c) (4 points) What is the requirement for your exploit in order to read the flag? 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.

Problem 3. ShellEval (10 points)

Shellcode is a small piece of code that runs a command-line shell such as `/bin/sh` 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 the provided VM that has a complete list of syscall numbers for Linux x86.

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

(c) (1 point) Describe the syscal calling convention of Linux x86.

(d) (4 points) Write your own shellcode in 32-bit x86 assembly (i.e., .s file) that spawns a shell. Make sure that your shellcode meets the following conditions: (1) assembly code for your shellcode must include detailed comments; (2) when compiled, the binary representation of your shellcode should have size no larger than 100 bytes; (3) the binary representation of your shellcode should not contain any zero (NULL) and `\x0a` (new line) byte; and (4) your shellcode should be semantically equivalent to the following C code snippet: it invokes setuid with 0 as the first argument, invokes setgid with 0 as the argument, and then finally execute the command `/bin/sh` with execve.

```
\\ Shellcode written in C
# include <stdio.h>
# include <unistd.h>

void shellcode() {
    char* shell[] = {"/bin/sh", NULL};
    setreuid(geteuid(), geteuid());
    execve(shell[0], shell, 0);
}
```

(e) (2 points) Describe the meaning of the `setreuid(geteuid(), geteuid())`. If an attacker were to successfully inject this shell into a vulnerable program and run it, what privileges would they gain?

(f) (1 point) Inject your shellcode to the program `shelleval` and get the flag located in `/home/shelleval/flag.txt`.

Problem 4. RetShell (20 points)

In the forth CTF problem, you need to gain the local privilege escalation by exploiting a bug in a poorly written program called `retshell`.

To work on this problem, we recommend you to create your own directory at `/tmp`. Create a directory with a random name so that people cannot guess it easily. This is going to be your working directory. For example, you can write a script in the directory while exploiting the program, or you can put your GDB script inside the directory. Remember every student will use the same user ID, and therefore, they can see your files if they know the path to your working directory. This is the reason why you want to make your directory with a random string that cannot be guessed easily. For your information, this machine has GDB, VIM editor, Python, and Perl installed.

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/retshell/flag.txt`? What kinds of permission does the flag have?

(b) (1 point) Is it enough to use your exploit (i.e., shellcode) in order to read the flag? Explain.

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

(d) (8 points) Exploit the program (`retshell`) 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.

Problem 5. RetFunc2 (15 points)

In this CTF problem, you need to gain local privilege escalation by exploiting a bug in a poorly written program called `retfunc2`. Use `/tmp` directory as you did in Problem 4.

(a) (5 points) Analyze the source code of `retfunc2`, and pinpoint the vulnerable code spots. What kind of vulnerability does this program have?

(b) (10 points) Exploit the program (`retfunc2`) 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.