Homework 3

Due: Apr. 18, 2022 23:59:59

- Late submission policy.

- During 48 hours after the deadline, you get a half of the available points.
- After 48 hours from the deadline, you get zero.

- Submission rule. We will deduct ***half the point*** if you do not follow the submission rule.

- You should submit a single PDF file. No other file(s) allowed.
- Your PDF should clearly show your name and student ID.
- Submit your PDF to the KLMS.

Problem 1. Shellcoding (10 points)

In this problem, you should write a shellcode that does the following tasks.

1. Open a file "`flag.txt`" located at the current working directory.
2. Read maximum 64 bytes into a buffer on the current stack.
3. Write the content in the buffer (assuming ASCII string) to the standard output.
4. Close the file descriptor.

You should embed your assembly code in the PDF. Do not separately submit your .s file. The submitted assembly code should be properly annotated.

Problem 2. Capture The Flag 1 (quiz) (55 points)

In this problem, you are going to exploit a vulnerable program located at `143.248.38.212`. The port number is 20022, so you can "ssh" into the machine by typing `ssh -p20022 143.248.38.212`.

Your login credential for this problem is:

ID	quiz
Password	quiz

By logging into the machine, you will see two files in your home directory: (1) a binary named `quiz`, and (2) a text file named `flag.txt`. The text file is *not* readable by the user `quiz`. Therefore, you should exploit the program and gain enough privilege to read the text file.

(a) (10 points) You should not be able to write any file to the home directory nor the temporary (`/tmp`) directory. However, you can create your own working directory under the `/tmp` directory. For example, you can create a directory `/tmp/abc` and "`cd`" into the directory. This way, you can put your own script in your own working directory and solve the problems. Note that other students will not be able to list directories under `/tmp`; that is, `ls /tmp` does not work. Therefore, if you create a unique (and hard-to-guess) subdirectory under `/tmp`, you can safely work on your own exploit without worrying about others peeping your solutions.

So in this subproblem, you have to create your own working directory under `/tmp` with a hard-to-guess name. Justify why the name is hard-to-guess. Embed a screenshot, showing your working directory in a terminal, into your PDF.

(b) (2 points) The permission of the binary is somewhat different from regular files. What is the meaning of 's' shown in the group permission?

(c) (3 points) What is the owner group of the text file? How does this affect your exploitation strategy?

(d) (10 points) Reverse-engineer the `answer` function in show a C code. Give proper variable names in your C code. You should embed your code in your PDF file; do not submit a separate C file. We will deduct your points if you don't follow the submission rule.

(e) (5 points) There is a security bug (i.e., vulnerability) in this program. Pinpoint where the vulnerability is, and what kind of vulnerability it is.

(f) (5 points) There is a function call to `tolower` at `0x080491d4`. Why do you think this function is used here in this program?

(g) (5 points) The bug discussed above allows an attacker to hijack the control flow of the program. Notably, the same bug can be used to change the expected behavior of the program without hijacking the control flow. How would you change the expected behavior of the program without hijacking/corrupting the control data? Provide a specific user input to trigger this problem.

(h) (10 points) Exploit the program, and read the flag. Explain in detail how you did it. Embed your code/script in the PDF if necessary.

(i) (5 points) By reading the flag, you will be given an extra problem. In this problem, you are going to crack a leaked password, which is properly hashed with a salt. Try to use a well-known (and free) password cracking tool, such as John the Ripper, to resolve this problem. What is the password? and Why do you think the password is so easy to crack by the tool?

Problem 3. Capture The Flag 2 (echo) (35 points)

This problem is similar to the previous one: you are going to login to the same machine with a different login credential though:

ID	echo
Password	echo

You will see a binary and a flag (text) file as before. Your goal is to exploit the binary and read the flag. You are recommended to use the same working directory as in the previous problem.

(a) (5 points) Disassemble the instruction located at `0x804917b` with both `GDB` and `b2r2`. Note that `b2r2` is already installed on the server. You will see that the two tools will output slightly different disassembly. What's the meaning of the "ds" prefix from `GDB`? Why do you think B2R2 omits the prefix in its output?

(b) (5 points) Reverse-engineer the `main` function, and show the corresponding C function in your PDF. Give an appropriate name for variables.

(c) (5 points) Reverse-engineer the `problematic` function, and show the corresponding C function in your PDF. Give an appropriate name for variables.

(d) (5 points) There is a vulnerability in the `problematic` function. Where is it? and What kind of vulnerability it is?

(e) (10 points) Exploit the program, and obtain the flag stored in the server. Show your code/script in the PDF if necessary.

(f) (5 points) Suppose you are patching this binary to fix the vulnerability. You can overwrite existing bytes, but you cannot prepend/append anything to the binary; that is, the binary size should remain intact. How will you do so? What are the challenges in doing so?