Homework 2

Due: Oct. 19, 2024 (23:59 pm)

- This is an individual assignment, not done in group.
- The work must be all your own.
- Please obey the law: do not hack the score server.
- In case you are using the Vagrant VM that we provided in the class, please turn on the Address Space Layout Randomization (ASLR) protection for the VM in order to work on the problems in HW2.
- We will deduct ***half the point*** if you do not follow the submission rule. Submit to KLMS a single PDF file (`your_ID-last_name.pdf`) that includes all the code and explanation. Do not submit a separate script files: embed everything into a single PDF file.
- Use PoE (https://www.nuget.org/packages/PoE.Replayer) to write an exploit (for Problem 2–5). Before installing PoE, make sure you have installed .NET 8 (or higher) SDK on your machine. To install PoE, just run the following command: `dotnet tool install -global PoE.Replayer`
- We will give extra points if you found a valid bug in the PoE interpreter. Report bugs using KLMS (the Q&A board). We will not consider aesthetic bugs (i.e., wrong error messages, typos, etc.) as valid. We also do not consider a stack overflow by creating an enormous expression as a valid bug; our interpreter is written using a simple recursion, and we do not expect to see an expression having thousands of nested expressions.

Problem 1. Warmup: ASLR (10 points)

Read the following C program and answer the following questions.

```
#include <stdio.h>
int findme() {puts("good job!");}
int main()
{
    long addr;
    fread(&addr, sizeof(addr), 1, stdin);
    printf("%lx\n", addr);
    ((void (*)()) addr)();
    return 0;
}
```

Alice compiled the program using the following command on an x86 machine.

```
$ gcc -pie -o prog prog.c
```

She also checked her system setting with the following command:

```
$ cat /proc/sys/kernel/randomize_va_space
2
```

(a) (2 points) Suppose Alice disassembled the resulting binary (`prog`) and obtained the address of `findme` function. However, when she provided the address as input to the program, she failed to get the expected result: she observed the program crashed instead. Why do you think the program crashes when she gave the address obtained from `objdump` as input?

(b) (6 points) This program takes in an address from the standard input (STDIN), and jumps to the address. Your goal is to guess the address of the `findme` function correctly, and get the output string `"Good job!"`. Can you write a wrapper program that repeatedly executes the program (`prog`) in a loop until it finds the answer? You should explain your code with well-written comments how you achieved the goal.

(c) (2 points) Alice wrote her own wrapper program to solve the above issue. She wrote two different versions of the wrapper program: one that tries a single well-chosen address repeatedly until she gets the correct answer, and another that repeatedly tries a random address chosen from a range of possible addresses. In both cases, she knows exactly which address bits are randomized by ASLR. For the first version, she simply picked one address at random within the range of possible addresses. Which version would be more efficient? and why?

Problem 2. RetRet (20 points)

The program `retret` is running on a remote machine. It is running as an `xinetd` service. Download the binary `retret` from the CTF website, and answer the following questions. To run this binary using socket on your own VM, you can either set up the `xinetd` service by yourself, or you can simply use `netcat` (the `nc` command) as follows:

```
nc -l -p 33333 -e ./retret
```

. You can change the port number (33333) to something else if you want. Notice there are two different versions of `netcat` on Debian/Ubuntu. You can install the correct version by typing:

```
apt-get install netcat-traditional
```

. The vagrant VM we provided should already have the correct version of `netcat` installed.

(a) (2 points) This program is running as an xinetd service. Explain how an xinetd service communicates with clients. In particular, you should explain how this program (`retret`) communicates with clients over TCP.

(b) (1 point) Check whether the binary is protected with non-executable stack (NX).

(c) (2 points) What does `cpy` function do? Can you name a libc function that has the same semantics as `cpy`?

(d) (2 points) Do you think ASLR can be applied to the code section of this binary? Why or why not?

(e) (5 points) Pinpoint a vulnerability in this program that allows a control-flow hijack exploit. Explain how you can exploit this vulnerability at a high level.

(f) (3 points) Enumerate ROP gadgets in the program that you are going to use, and explain in detail why you chose such gadgets. (Hint: you should find a way to invoke syscalls)

(g) (5 points) Show your final exploit script, and explain in detail how you obtained the flag. Embed PoE in your PDF.

Problem 3. NetCalc (25 points)

The program `netcalc` is running on a remote machine. It is running as an `xinetd` service. Download the binary `netcalc` from the CTF website, and answer the following questions. To run this program with socket, you can use `netcat` as explained in the previous problem. With `netcat`, you can also attach a debugger to the process while testing your exploit. For example, you can create a GDB wrapper script, named `gdb.sh` that looks as follows.

```
#!/bin/bash
gdb -ex "run" -ex "quit" ./netcalc
```

. You then modify the `-ex` option to run some more GDB commands if necessary. With the script, you can run `netcat` with the following command:

```
nc -l -p 33333 -e ./gdb.sh
```

.

(a) (5 points) Reverse engineer the binary and explain what this program does.

(b) (4 points) Find a vulnerability in the program. Explain how you found the vulnerability. Show an input that crashes the program.

(c) (1 point) Check whether the program (`netcalc`) is protected with non-executable stack. How did you figure this out?

(d) (1 point) Check whether the program is protected with stack canary or not. How did you figure this out?

(e) (4 points) This program is protected with ASLR. Explain in detail how would you bypass ASLR using the vulnerability that you found in (b). To successfully exploit this program using a ret-to-stack attack, how can you find the exact address of the user-controlled buffer?

(f) (5 points) Even though you found the correct buffer address in a loop, the address should be adjusted in the next loop. Explain why this program creates a new stack frame for each loop. How would you account this problem when exploiting the program.

(g) (5 points) Write a program (or a script) in PoE that exploits the program. Get a shell and read the flag. You must explain in detail how you exploited the program. Embed your PoE code in the PDF and explain with proper comments.

Problem 4. Canary (15 points)

Download the binary (`canary`) from the CTF server, and answer the following questions. You can use the provided VM for running the binary and for debugging your exploit. Remember to turn the ASLR on. To debug a child process after `fork`, you can change the `follow-fork-mode` of GDB. If you don't know what it is, Google it.

(a) (1 point) Check whether the program (`canary`) is protected by non-executable stack (NX). How did you figure this out?

(b) (1 point) Check whether the program (`canary`) is protected by stack canaries. How did you figure this out?

(c) (4 points) Reverse engineer the binary and find a vulnerability in the program. Find an input that crashes the program.

(d) (1 point) In this binary, there is a hidden function that can be used to read the flag. Find the function and explain how you can utilize this function in your exploit.

(e) (2 points) Discuss how you would bypass any defense mechanism discussed both in (a) and (b).

(f) (6 points) Exploit the program running on the server. Your goal is to read the flag located at `/home/canary/flag.txt`. Embed your PoE code in the PDF with proper comments. Please note that the `canary` service is re-launched every 10 minute. This means, the canary values will change every 10 minute, even though you are attacking the same IP and port. In PoE, you should call an "act" within a loop.

Problem 5. Checker (30 points)

In this problem, you should bypass canary, ASLR, as well as DEP (NX). Download the binary code of the program (`checker`) and answer the following questions.

(a) (6 points) Analyze the source code, and find three vulnerable points in this program.

(b) (8 points) Using one of the bugs you found in (a), disclose memory contents of the program and find out useful pointers on the stack. One such example is a return address

to LIBC, or a pointer to `stdout`. Analyze the binary and explain why such pointers are located on the stack.

(c) (3 points) Obtain the LIBC file that is used by the programs in HW2. From the LIBC file, figure out the relative offset from the obtained pointer in (b) to `system` function. You can get a function address by calling the `libcFuncAddr` function of PoE.

(d) (5 points) Build an exploit payload that performs the following three steps: (1) leak memory contents to figure out the canary value and the pointers to `stdin` and `stdout`; (2) bypass a length check; and (3) initiate a stack overflow with the leaked canary value in order to jump to an arbitrary address. Explain in detail how you can achieve this.

(e) (8 points) Exploit the service. Embed your PoE exploit in the PDF with proper comments.