Programming 3

Instructor: Guanhua Yan
Due date: October 30 (Sunday).

It's really a cool thing that we try to "hack" some code once. In the last project, you will learn how to use what you've learned so far to exploit an actual buffer overflow. The program you will be exploiting is a vulnerable ELF executable written in C. Your task is to craft an exploit that will call a function (that is never meant to be called) to print something.

Step 1:
Use the vulnerable source code at the end of the assignment. Compile it with:
gcc ./vuln_program.c -fno-stack-protector -z execstack -static -o vuln_program

Step 2: Disable protections on your VM (use a 32 bit VM)

To make your job easier, you should disable address space layout randomization (ASLR). On Ubuntu-based systems, run the following command:

sudo sysctl -w kernel.randomize_va_space=0

If you are working on a RedHat-based system (e.g., RHE or Fedora), you will also need to disable Execshield, as follows:

sudo sysctl -w kernel.exec-shield=0

Step 3: Crash the program.

Run the program vuln_program and provide a long input to cause it to crash. How long an input do you need? You can use the source code to analyze why that makes sense. You may also want to use "objdump -D ./vuln_program" or gdb to figure it out.

This article may be of help: Examining a Buffer Overflow in C and assembly with gdb. You may also find this gdb cheat sheet helpful.

Step 4: Exploit the overflow

Now that you understand what's going on, write an exploit that, instead of causing a crash, causes the program to print out "Haha! You got pwned!". There's an existing function target in the executable that prints this. So all you need to do is use the buffer overflow to overwrite return address on the stack to point to target's address.

In your program, you will need to write a separate program to generate the attack string into a file, say, "attack.input". You then run the program as "./vuln-program < ./attack.input" to see whether you can exploit the code to print "Haha! You got pwned!" without crashing the program.

You can choose any programming language for this assignment, as long as it generates the attack string correctly. You will need to submit a separate readme file that describes the details about why that specific attack string is chosen. We will hand out submission guidelines later.

In your submission, you will need to provide the program that generates the attack string. Given that the address of the target function is X, your program runs as "./attack-string X > ./attack.input", which generates the attack string to file attack.input. If your hack string is correct, the message "Haha! You got pwned!" will be printed without crashing the program.

Grading: You will need to come to my office to demonstrate the attack within five minutes, using the code you have submitted at Blackboard. Each extra minute will cost you five points. You will have to finish the attack within 15 minutes.

========== vuln_program.c ==========

```
#include <stdio.h>
#include <stdlib.h>

static int x = 8;

void prompt(){
        char buf[100];

        gets(buf);
        printf("You entered: %s\n", buf);

}

int main(){
        prompt();

        return 0;
}

void target(){
        printf("Haha! You got pwned!\n");
        exit(0);
}
```

==========