CSCI 530 – Security Systems
Lab 4 – StackOverflow; Submission Due: 21st October 2024
Name: Anne Sai Venkata Naga Saketh
USC Email: annes@usc.edu
USC ID: 3725520208

Question 1:

In stack_1.c the separation in the stack between the beginning of the argument buf and the beginning of the stored return address is 28 bytes. See the screenshot in the slide entitled "Stack separation between argument and return address," also the article's page 7. In the screenshot, the argument starts at 0xbfffd530. And the return address (always at ebp plus 4 bytes, and ebp holds 0xbfffd548) is at 0xbfffd54c. The separation between their starting points is therefore

0xbfffd54c - 0xbfffd530 = 0x0000001c
or twenty-eight bytes.

Change the code in stack_1.c to provide a buffer length of 2 instead of 10. That is, change line 2 from "char buf[10];" to "char buf[2];". If you are not comfortable with the vi editor, this stream editor (sed) command can be used:

```
sed  -i  's/10/2/'  stack_1.c
```

Recompile and investigate (break/interrupt just after the stack buffer has been given data). Determine where exactly the argument starts by executing "print &buf". Determine where the return address lies by executing "print $ebp" and adding 4. What is the separation now?

Answer:

After making the changes, recompiling, and reinvestigating, the values are as follows –

print &buf = 0xbffd9586
print $ebp+4 = 0xbffd958c

The separation is as follows – 0xbffd958c – 0xbffd9586= 0x06

Hence, the separation is now 6 bytes.

Question 2:

An easier, more casual way to observe the buffer overflow is to run stack_1 from the command line instead of the debugger, supplying the variable's content as a command-line argument (note the program is written with formal parameters to receive it). While you can get away with passing a few more bytes than the buffer's designed for, without error, at a certain point and beyond you'll experience "Segmentation fault" error messages. Segmentation being a memory management technique, and you having screwed up memory pointers, there's no surprise in that terminology. Recompile stack_1.c with its original 10-byte buf instead of 2. Run stack_1.c and cheat by a byte, by passing it 11 characters (using "12345678901..." lets you visually keep track). Pass it increasing numbers of characters until you get the "Segmentation fault". Note the lowest/first value at which you reach that problem.

a. What is that value?
b. in broad generality, why is more than 10 OK up to a point?
c extra credit: I expected the lowest encounter with the problem at 1 byte more than it actually was. Why was I off by 1 byte and what's the reason for the problem appearing when it does? What is getting clobbered at that particular point?

Answer:

a. The lowest value is when argv[1] is given a length of **24** characters.
b. Optimized machine code is what compilers try to produce. A segmentation fault might only occur after you first exceed the buffer's capacity by a few extra characters because the rewritten memory addresses might not be necessary for the program to execute. But, if you go above a certain point, you risk corrupting memory locations needed for the program to run correctly, which could result in a segmentation fault. As a result, it is considered reasonable to permit some leeway in writing beyond the buffer size.
c. The reason for this is that the strcpy() function we're using transfers the source string to the destination string if the destination string has a larger buffer than the source string with a null character. The source string in our example is longer than the destination string; therefore, characters are copied without a null character at the end of the string. Because of this, we are receiving a segmentation error at the 24th rather than the 25th byte.

Question 3:

There are both similarities and differences between the stack buffer and tls heartbeat flaws you exploited. Name one of each. That is, identify something they have in common that makes them the same, plus something else that distinguishes them as different. Any intelligent (and accurate) similarity and difference will be fine.

Answer:

Stack overflow attacks and the Heartbleed exploit are similar in that they both rely on missing bounds checks, allowing attackers to change memory beyond what is intended. When a stack overflow occurs, memory corruption results from the program's failure to impose buffer constraints. Similarly, Heartbleed occurs when the TLS heartbeat extension is used without sufficient message size validation, which results in accidental memory exposure.

The way the missing bounds are used makes all the difference. Heartbleed allows an attacker to read up to 64k of memory from the target without changing the target system's behavior. On the other hand, a stack overflow enables the attacker to alter memory, including return addresses, which permits arbitrary code execution and gives them control over the target system.

Question 4:

How would an attacker who wished to optimize heartbleed's value to him do it? Unlike transferring an entire database or shadow file that can net useful sensitive data from a system in great volume, this exploit is so random and modest that it can't really be controlled to advantage. Running it gets a mere few thousand bytes, and most is uninteresting garbage. What would you do to refine it and extract some metal from this raw ore?

Answer:

I would initially select a web server that runs an impacted version of OpenSSL to maximize the heartbleed attack. Then, I could extract tiny memory dumps and search the data for private information like passwords, encryption keys, and usernames. This process can be repeated to identify the patterns in the data to do a cryptanalytic attack. Should login credentials be discovered, I then concentrate on user accounts for additional attacks. These attacks may result in deeper access to the company's systems or private information. Constant memory leaks might expose session tokens, allowing me to take over running sessions. More memory fragments could eventually be found, and those could contain secret keys or other important information.