CSCI 530 - Security Systems Lab

Rishit Saiya, Lab - 4

October 11, 2022

1

After making the required changes and recompiling `stack_1.c`, the following execution addresses are found:

```
(gdb) p &buf
$1 = (char (*)[2]) 0xbff7a776

(gdb) p &ebp
$2 = (void *) 0xbff7a778

(gdb) p &esp
$3 = (void *) 0xbff7a770
```

where `$esp` is the stack pointer.

After recompiling, as seen above, the return address after executing `print $ebp` is `0xbff7a778`. Adding 4 as instructed will yield `0xbff7a77c`.

The separation between `$ebp` (break point) and `$esp` now would be as follows:

$$\texttt{0xbff7a77c} - \texttt{0xbff7a770} = \texttt{c} \implies \texttt{12 bytes}$$

The separation between `$ebp` (break point) and `$buf` now would be as follows:

$$\texttt{0xbff7a77c} - \texttt{0xbff7a776} = \texttt{6 bytes}$$

2

a.

The least value where the problem is reached is 24.

b.

In generality, more than 10 is OK up to a point because the stack has more than 10 bytes of space. Also more than 10 characters is OK upto a certain point as beyond that there is overwriting in memory inside the stack. The memory spaces are accessed when they aren't supposed to be accessed. This manifests to segmentation fault whenever there is any attempt to access that value in the memory space.

c.

Despite the error of Segmentation Fault, the function `fn` successfully returned to `main` address. It is so because the **end** is printed on the screen. After inspecting the addresses of `$esp` and `$ebp` of returned function `fn`, which are `0xbfe60ab1` and `0xbfe60b61` respectively. Since the `$esp` place is after the break point `$ebp`, this causes the Segmentation Fault after `main` function return.

Inspecting the `x/12x $esp` at function `fn` after the function `strcpy` call, it clearly states that the value of `$esp + 4` is `0xbfe60b61`. Due to the C language's configuration and build, the least significant `00` is yielded end of the string: `\0` (NULL character). So, the string `12345678901xyz` yields `12345678901xyz\0` in C language. The error is hence caused when the `main` function returns because the corresponding break point `$ebp` is accessing and altered pointing towards an invalid memory address space by that NULL character of the string.

3

Both the stackoverflow and heartbleed vulnerabilities try to manipulate and alter the programs' memory. However, heartbleed, it sends a small input to cause vulnerabilities like `openssl` vulnerability (as performed in lab) and sends back the data in its by obfuscating the input content length. Meanwhile, stackoverflow uses up the large input that is sent to make the program execute the function.

4

The best strategy would be to get as much as data in one response (`64KB`). Besides the previously mentioned strategy, also running the exploit/attack for a while every few seconds (at regular intervals). This statistics can be used to run an algorithm to find common repeated substrings such as `sessionid`, SSN, private keys, password, username, etc.

Due to unavailable of length string data, there always exists an issue of repeated sub-patterns. This gives us the leverage to actually select a group of say 70-80 bytes together and recursively search for such sub-patterns which eventually will lead to meaningful information.