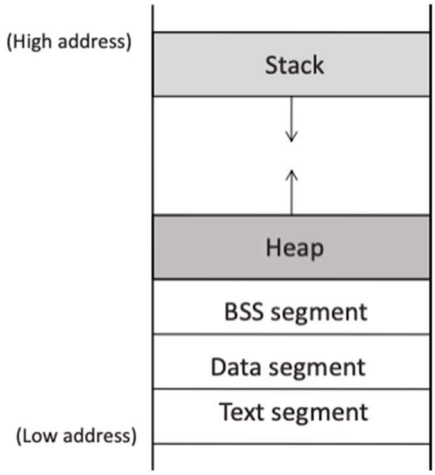
Buffer Overflow (CS 915) Post-Lab Assignment Report

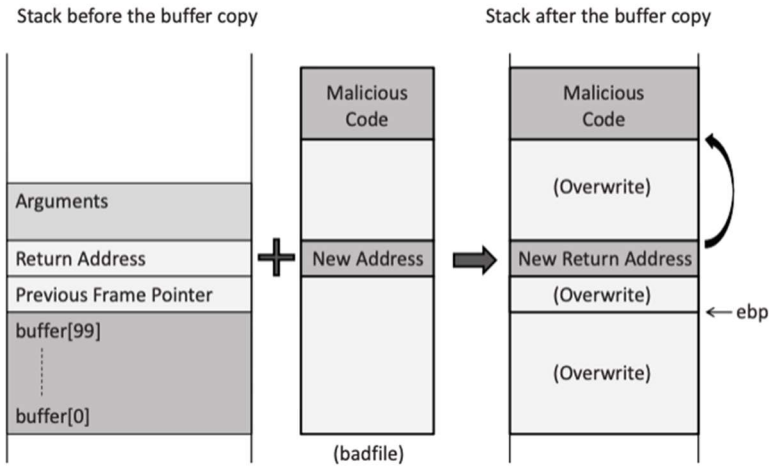
Hanzhi Zhang - 5525549

Explain what buffer overflow is, how it works with the help of memory layout diagrams, the countermeasures, and how these countermeasures may address this problem.

Buffer overflow is a type of software vulnerability when a program writes data beyond the regions of memory it is allocated for. This can be exploited by attackers to overwrite memory holding executable code to compromise the security of a system.



When it comes to memory allocation, code and data are stored separately, executable code of a program, for example, is stored in Text Segment. But apart from location there is no fundamental difference in how code and data are represented, it depends solely on how the computer interprets. So, attacker may simply put malicious code in the stack, and somehow tricks the machine to jump over there and execute it.



For example, a buffer overflow attack in the picture overwrites the stack, places malicious code in it and changes the return address of a function that is being called, as this function returns, program will jump to the malicious code and execute it.

Countermeasures include using safer functions like strncpy(), strncat(), or dynamic link libraries that check the length of the data before copying, instead of standard strcpy() which allows input length longer than allocated, and is much more easily exploited.

On operating system level, ASLR (Address Space Layout Randomization) is used, so that every time the code is loaded in the memory, the stack address changes, making it

more difficult for attackers to guess the address of the malicious code (they may still be able to put malicious code in the stack, but would have trouble deciding the new return address to overwrite - they know the offset but not the buffer's address).

Stackguard is a compiler approach, in which a random secret value is generated, stored and assigned to a local variable of the function currently in the stack. Because attackers can only overwrite the entire stack and cannot bypass this local variable, and they also do not know the value, if they attempt to write longer data than they are allowed to, the variable will be changed, and the attack will be detected by the program.

Non-executable stack is a CPU feature that separates code from data by marking certain areas of the memory as non-executable, for example, the stack. Then if the attacker tries to insert malicious code into the stack it will not be allowed to be executed.

Level 1 attack:

1) Explain the results of this attack with screenshots.

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd4c8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd458
server-1-10.9.0.5 | (^_^) SUCCESS SUCCESS (^_^)
```

The message is displayed correctly in the container's console, attack was successful.

```
[11/13/23]seed@VM:~/.../Lab 2$ nc -lnv 7070
Listening on 0.0.0.0 7070
Connection received on 10.9.0.5 54622
root@dae1c16639c2:/bof#
```

The root access to the remote target machine is gained, attack was successful.

2) Explain how you set the ret and offset values, and why.
Try echo hello to see the value of the frame pointer and the address of the buffer

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd4c8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd458
server-1-10.9.0.5 | ==== Returned Properly ====
```

Frame pointer (ebp) = 0xffffd4c8, buffer's address = 0xffffd458
ret >= return address + 4 = ebp + 8 = 0xffffd4c8 + 8 = 0xffffd4d0
offset = return address – base of the buffer = 0xffffd4c8 + 4 – 0xffffd458 = 116

Level 2 attack:

2) Explain the results of this attack with screenshots.

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd408
server-2-10.9.0.6 | (^_^) SUCCESS SUCCESS (^_^)
```

The message is displayed correctly in the container's console.

```
[11/13/23]seed@VM:~/.../Lab 2$ nc -lnv 7070
Listening on 0.0.0.0 7070
Connection received on 10.9.0.6 43168
root@b119bb1d2a0b:/bof#
```

The root access to the remote target machine is gained.

2) Explain how you set the ret and S values, and why.

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 6
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd408
server-2-10.9.0.6 | ==== Returned Properly ====
```

Buffer's address = 0xffffd408.
Consider max 300 bytes for the buffer, ret = buffer address + 300 (size of the buffer) + 4 (size of the previous frame pointer) + 4 (size of the return address) = 0xffffd53c
Since the range of the buffer size (in bytes) is [100, 300], the offset should fall in the same range, and should be divisible by 4. For i in range(S) we calculate i * 4 to represent the offset, so range(S) should cover [25, 75], and thus we have S = 76.