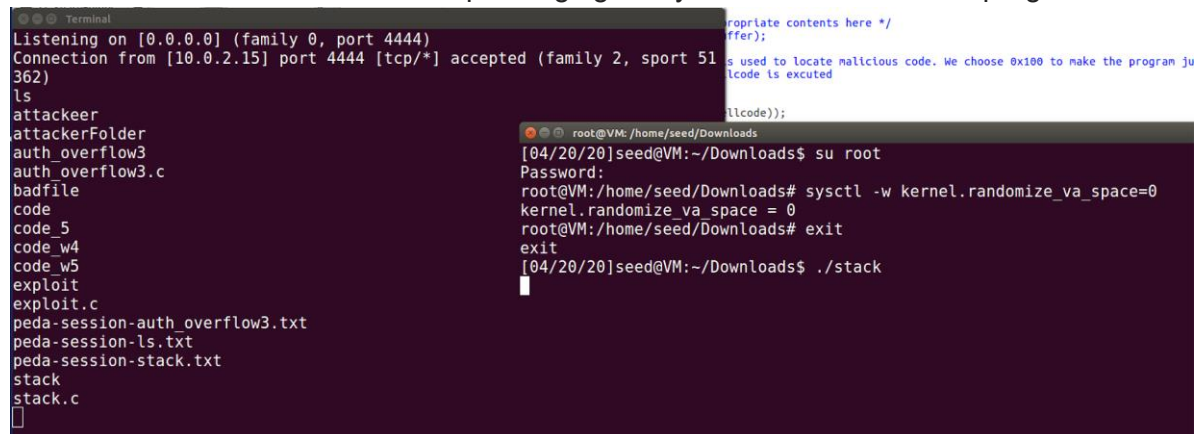


```
memcpy(buffer + XXX, shellcode, sizeof(shellcode))
```

#### 4. Why do we need to turn off the stack memory randomisation?

Disable memory randomisation is important because we hardcode the return address value with a new one. Otherwise, the addresses where the buffer in the bof function and that functions' return address value keep changing every time we run the stack program.



```
Terminal
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [10.0.2.15] port 4444 [tcp/*] accepted (family 2, sport 51362)
ls
attacker
attackerFolder
auth_overflow3
auth_overflow3.c
badfile
code
code_5
code_w4
code_w5
exploit
exploit.c
peda-session-auth_overflow3.txt
peda-session-ls.txt
peda-session-stack.txt
stack
stack.c

root@VM: /home/seed/Downloads
[04/20/20]seed@VM:~/Downloads$ su root
Password:
root@VM:/home/seed/Downloads# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@VM:/home/seed/Downloads# exit
exit
[04/20/20]seed@VM:~/Downloads$ ./stack
```

#### 5. Do I need to inject more NOP instructions into the badfile generated by the exploit.c program?

In the exploit.c, there is a line of code that initialises the badfile with 517 NOP instructions already. You do not need to inject any more NOP instructions in the badfile's content again.

Let's say the return address value is stored where it starts after 10 bytes from the beginning of the buffer, then you only need to overwrite exactly this location with a new return address value you want.

#### 6. Where should we inject the shellcode into the badfile?

We should inject the shellcode somewhere at the middle-> end of the bad file.

We should not inject the shellcode at the beginning of the badfile. The reason is that it will overwrite the return address of the bof function when the badfile's content is loaded into the buffer variable declared in that function. As a result, the program will jump to some unexpected memory address and cause a crash.

#### 7. What should be my approach to doing this assignment?

First, you can run the exploit.c code as it is, don't change anything in the code (for shellcode just initialize it with an empty string) and run it. It should create a badfile full of NOPs. Run stack.c now in the debugger, inspect the stack starting from the *buffer* (see our buffer overflow lab recording for help). You should be able to see NOPs in the stack. Now go back to exploit.c and change the first few bytes (currently, it's all 0x90, you need to change it using memcpy/memset. Now again run stack.c and find those new bytes. If everything is going well, it's time to generate the shellcode using msfvenom and copy it into the badfile, which will overwrite the return address and run your shellcode. We suggest arranging the badfile as follows:

**Few NOPs + Return Address \* 16 + A lot of NOPs + Shellcode**

Notice we are repeating the return address sixteen times (you don't have to do this but it's a good practice as in real exploitation you don't exactly know the location of the return address). The return address will be pointing to somewhere in the "A lot of NOPs", and eventually run shellcode.

**8. My exploit only works in gdb and not without it, is it acceptable?**

Yes. If your exploit works inside gdb and you provide evidence (screenshots), you will get full marks. For the remaining part (stack guard and address randomization sections), you don't have to actually get the shell access, you can try your exploit and theoretically explain what's happening, i.e., what these countermeasures are doing, and how your exploit is supposed to work or what will be a way to overcome these measures.

Please see this, it's a good explanation:

<https://stackoverflow.com/questions/17775186/buffer-overflow-works-in-gdb-but-not-without-it>

**9. What is a badfile?**

Badfile generated from exploit.c contains your *malicious payload* that includes the combination of NOPs, shellcode (generated from msfvenom or taken from internet), repeating return address. The stack (vulnerable) program will take this badfile as an input that will lead to successful buffer overflow (if drafted properly). Look at Week 3 lab sheet for more details.

**10. I am facing issues in creating a badfile from exploit.c. What can I do? Is there any other way?**

Yes. You can use the *perl commands* similar to the ones we used in Week 3 lab inside *gdb*. You can use the same method and format to exploit buffer overflow. If you choose this way, you will still get *some* marks.

---

In this section, we list some questions that either lack important details or evidence of sufficient efforts made for this assignment. We expect you to provide concrete contexts, screenshots, and descriptions for the teaching team.

**1. I am getting a segmentation fault. What should I do?**

Segmentation fault can happen for many reasons, your return address might be wrong, location of the return address might be wrong. Maybe your badfile is just full of NOPs (your memset function was not working properly).

We suggest, please attach screenshots of the stack and a description of what's in the badfile.

**2. Please find attached my code. Can you please tell me what's wrong?**

Sorry, we can't. We cannot help you with your code. We can have a look at your stack in gdb and suggest what you should do, but we can't debug your code.

**3. I know I have to overwrite the return address, but I don't understand how to do it.**

This is taught and demonstrated in the lecture and lab. Please review the videos and let us know if there are further questions after your careful investigation.

**4. Shellcode generated via msfvenom not working**

The possible reason is the shellcode includes bad characters. You forgot to exclude them. Please use the "-b" option followed by a list of bad characters such as "\x00" in the msfvenom command. "\x00" value when it is part of a string it marks the NULL character thus the C program does not load whatever is after NULL

**5. Do not do your tasks in the shared folder.**

Some students may do their tasks/labs in the shared folder directly. Sometimes, this may cause some unexpected errors. So please copy the files in the shared folder to the home folder (or any other location) of your VM. Then please perform your tasks in that folder.

**6. You need to listen on a port using Netcat before running the stack.**

Some students may successfully overwrite the return address, while they still get the segmentation fault (or error: Cannot access memory at address 0x90909090). Then they may need to open another terminal and listen on a port using Netcat before running stack (type command “nc -lvp 4444”).

**7. I am getting a message “Illegal Instruction” when running buffer overflow code.**

There is something wrong with your malicious payload, not the shellcode. It probably has to do with where you place the shellcode in the malicious payload and how you invoke it.