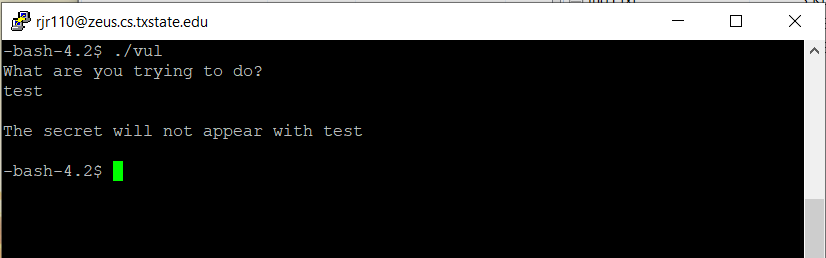
Step 1:

At the beginning I was not able to run “./vul” it was giving a permission denied message to change this I used the command “chmod u+x vul” which grants the me, the owner of that file, execution permission.



After that, I tried "python attack.py | ./vul" and it worked.

The three test strings in attack.py that are used as inputs are:

# test 1

overflow = 'abcd'

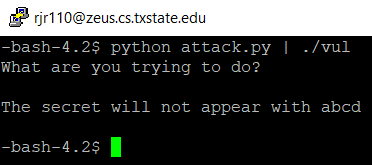
# test 2

overflow = '\x61\x62\x63\x64\x65'

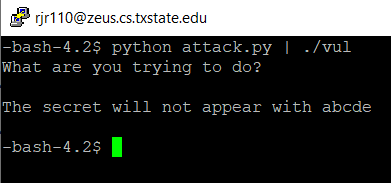
# test 3

overflow = '\x01\x02\x03\x04\x05\x06\x07\x08\x00\x00\x00\x00'

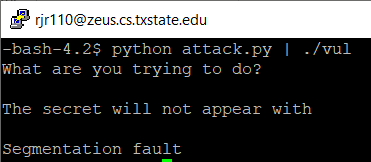
The following picture is for when I ran it with Test 1:



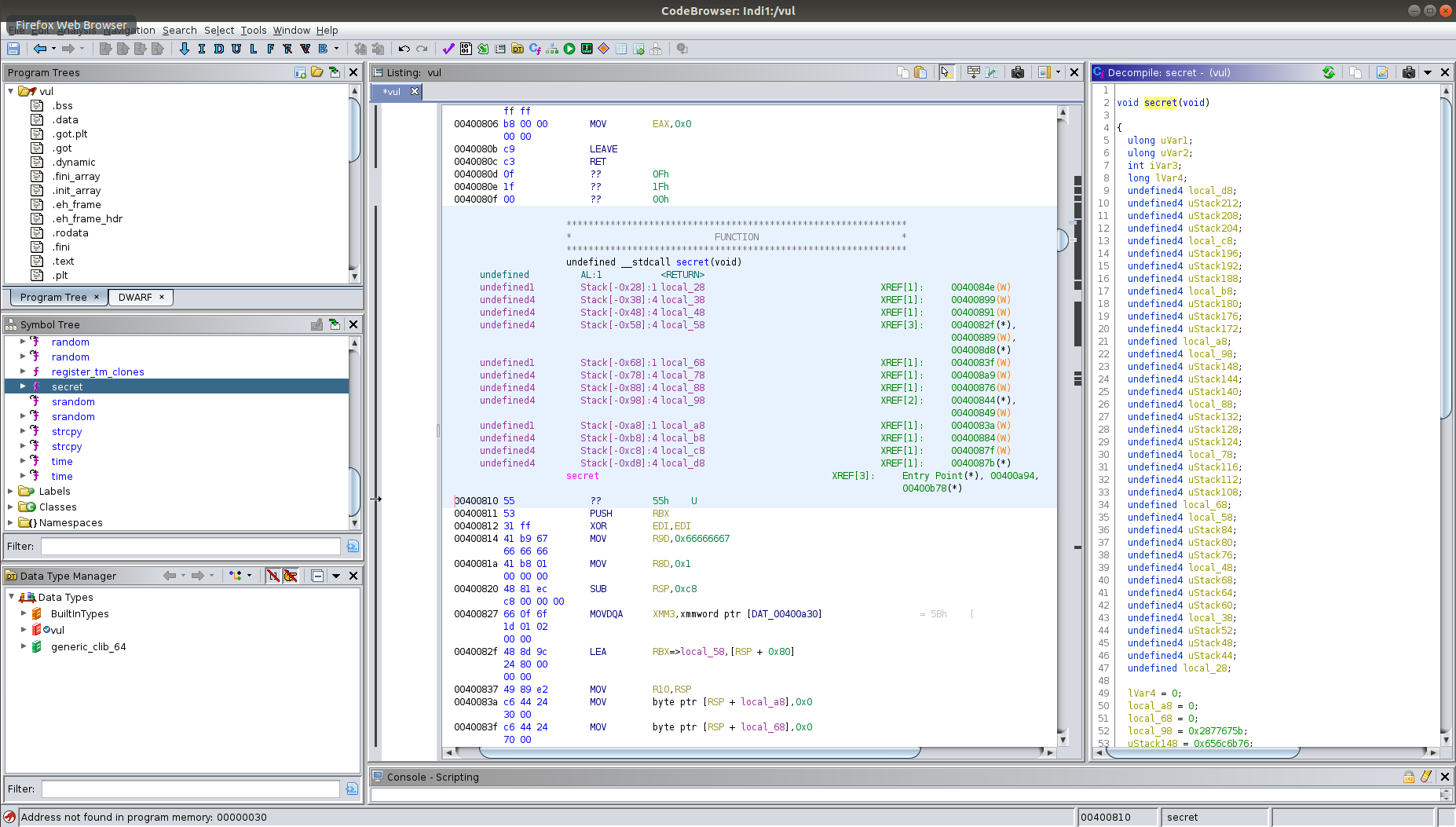
The following picture is for when I ran it with Test 2:

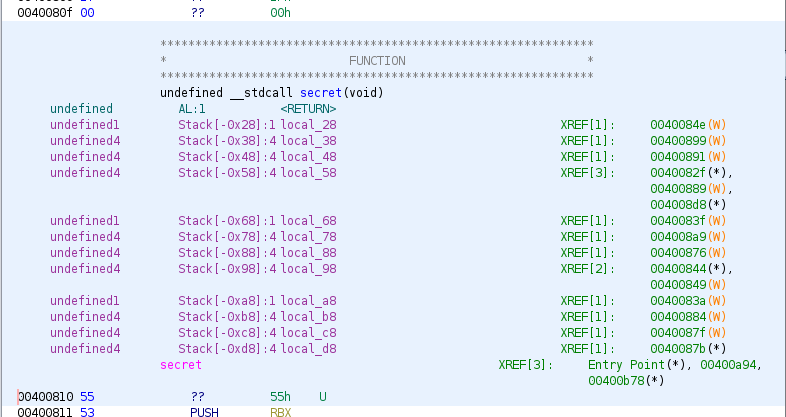


The following picture is for when I ran it with Test 3:

a

Step 2:





I used Ghidra to analyze vul. Here I found the address of the secret function which is 0x00400810.

Step 3: A close up of a street

Description automatically generated

As shown in the picture I was able to run GDB from the department server. I ran the program vul instruction by instruction using the “step” command followed by the “x/32x $sp” command to find exactly where the address of the buf[8] variable and the good return is at. The “step” command executes the next instruction on the program. “x/32x $sp” prints the top 32 elements of the stack. I did this after playing around a lot with Ghidra. “Info frame” listed general information about the frame such as where exactly memory starts.

The address of buff[8] is 0x7fffffffe1a0. I was able to tell since I saw every single step before it was saved into the one that matters.  
The address of the good return is 0x00400806, which is the return of the main function is at. One can see that on Ghidra. The distance between those is 16 bits. This is located at 0x7fffffffe1b0

Step 4:

A correct overflow for the string in attack.py will then be anything that is equal to the distance between the good return address and the buff[8] address, which is 16 bytes, + the address of the secret function where the message is located at (little endian style). Which should be 00040810 but fore some reason it only works if it is the following one. In my case I decided to use:

# test 4

overflow = '\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x11\x08\x40\x00'

But again, the first 16 bytes do not matter. This could have also worked: ‘abcdefghijklmnop\x11\x08\x40\x00’

Step 5:

The following picture shows the flag that I got after running the it with my updated attack.py file that used the new overflow string:

