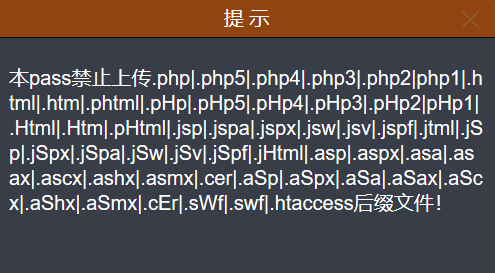
Pass08

打开第八关首先查看提示内容



这里可以看到如同前面几关一样，这个是通过黑名单来防止文件上传漏洞的发生。而且把一般的后缀名都禁止了。那么再来看看源码吧。

$is\_upload = false;$msg = null;if (isset($\_POST['submit'])) {

if (file\_exists(UPLOAD\_PATH)) {

$deny\_ext = array(".php",".php5",".php4",".php3",".php2",".html",".htm",".phtml",".pht",".pHp",".pHp5",".pHp4",".pHp3",".pHp2",".Html",".Htm",".pHtml",".jsp",".jspa",".jspx",".jsw",".jsv",".jspf",".jtml",".jSp",".jSpx",".jSpa",".jSw",".jSv",".jSpf",".jHtml",".asp",".aspx",".asa",".asax",".ascx",".ashx",".asmx",".cer",".aSp",".aSpx",".aSa",".aSax",".aScx",".aShx",".aSmx",".cEr",".sWf",".swf",".htaccess");

$file\_name = trim($\_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

$file\_ext = trim($file\_ext); //首尾去空

if (!in\_array($file\_ext, $deny\_ext)) {

$temp\_file = $\_FILES['upload\_file']['tmp\_name'];

$img\_path = UPLOAD\_PATH.'/'.date("YmdHis").rand(1000,9999).$file\_ext;

if (move\_uploaded\_file($temp\_file, $img\_path)) {

$is\_upload = true;

} else {

$msg = '上传出错！';

}

} else {

$msg = '此文件类型不允许上传！';

}

} else {

$msg = UPLOAD\_PATH . '文件夹不存在,请手工创建！';

}}

可以看到服务器端首先是去除了文件名末尾的点，然后是将文件后缀名转换成小写之后同黑名单进行对比。

将这一关的源码同前面的做了一下对比，发现在过滤过程中少了一种操作，那就是没有对文件名去除::$DATA。

**$file\_ext = str\_ireplace('::$DATA', '', $file\_ext);//去除字符串::$DATA**

这里需要补充一下有关windows系统下的::$DATA的知识。

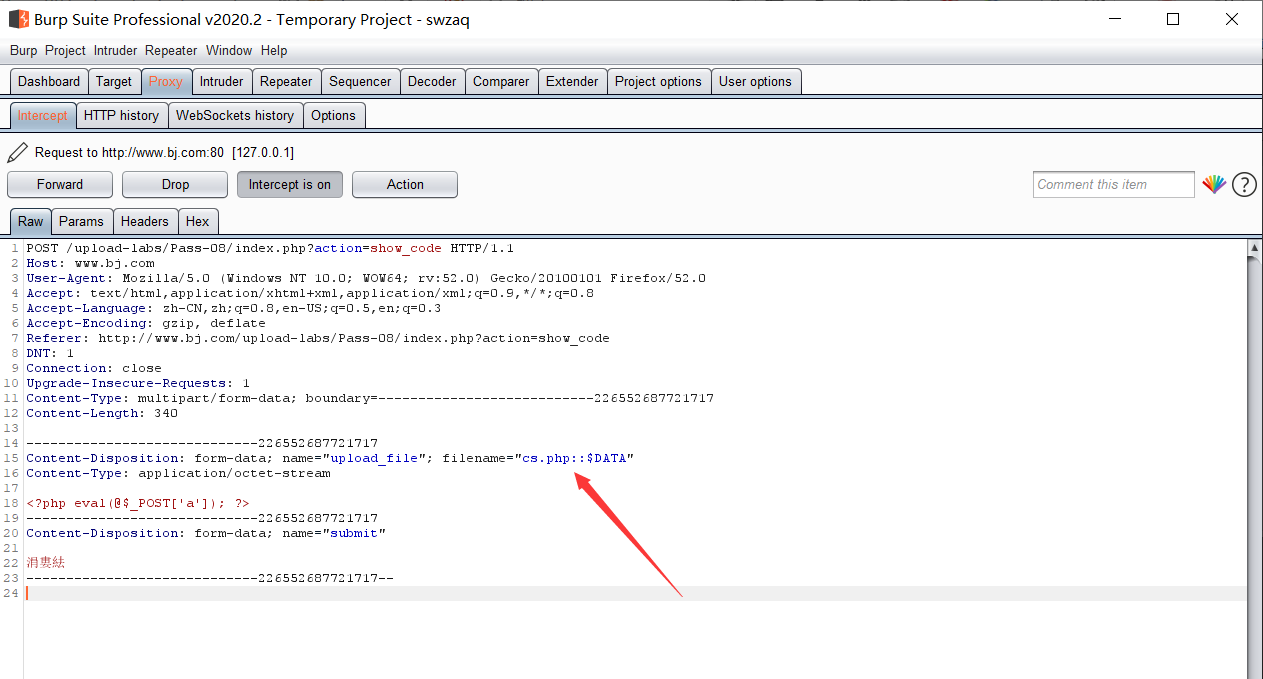
Windows下的NTFS文件系统有一个特性，那就是NTFS文件系统的存储数据流的一个属性DATA。NTFS文件系统包括对备用数据流的支持。备用数据流允许文件包含多个数据流，每个文件至少有一个数据流。在Windows中，这个默认数据流称为：$DATA。当我们访问a.php::$DATA 时，就是请求 a.php 本身的数据，如果a.php 还包含了其他的数据流，比如a.php:lake2.php，请求 a.php:lake2.php::$DATA，则是请求a.php中的流数据lake2.php的流数据内容。

因此可以借助这个特性绕过黑名单，不过这个利用方法仅仅限于windows系统。

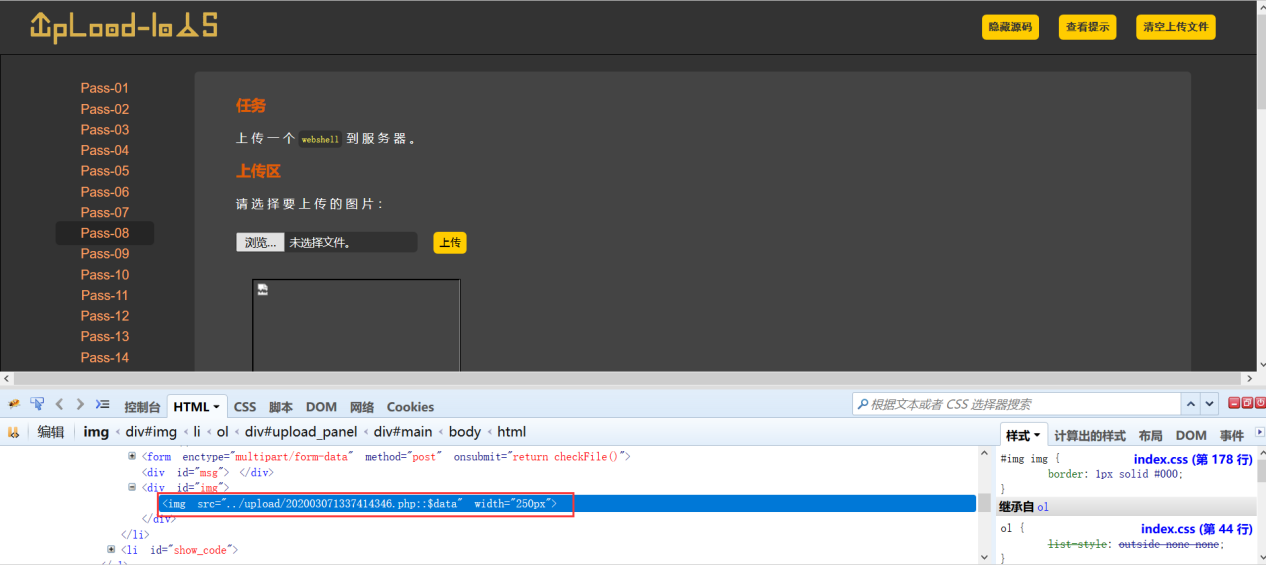
参考链接：

<https://owasp.org/www-community/attacks/Windows_::DATA_alternate_data_stream>

第一步：上传shell，抓包修改后缀名



修改完成之后放行数据包可以看到成功上传了



不过服务器端会对文件进行重命名。

第二步：查看服务器端的文件名，直接蚁剑连接测试

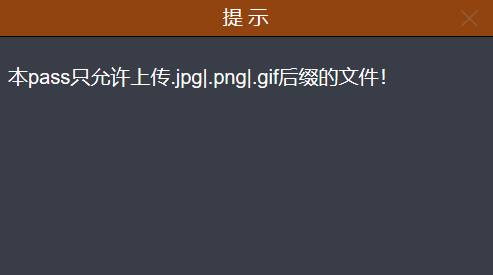
如果上传至服务器端的文件名称后面带有::$data的话，我们还怎么访问shell呐，所以看看该文件上传到服务器里面的样子



可以看到服务器上的文件是正常的，那么就可以对shell进行正常操作了。

Pass09

打开第九关首先查看提示



难道这一关又重复到了以前吗？看看源码再说

$is\_upload = false;$msg = null;if (isset($\_POST['submit'])) {

if (file\_exists(UPLOAD\_PATH)) {

$deny\_ext = array(".php",".php5",".php4",".php3",".php2",".html",".htm",".phtml",".pht",".pHp",".pHp5",".pHp4",".pHp3",".pHp2",".Html",".Htm",".pHtml",".jsp",".jspa",".jspx",".jsw",".jsv",".jspf",".jtml",".jSp",".jSpx",".jSpa",".jSw",".jSv",".jSpf",".jHtml",".asp",".aspx",".asa",".asax",".ascx",".ashx",".asmx",".cer",".aSp",".aSpx",".aSa",".aSax",".aScx",".aShx",".aSmx",".cEr",".sWf",".swf",".htaccess");

$file\_name = trim($\_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

if (!in\_array($file\_ext, $deny\_ext)) {

$temp\_file = $\_FILES['upload\_file']['tmp\_name'];

$img\_path = UPLOAD\_PATH.'/'.$file\_name;

if (move\_uploaded\_file($temp\_file, $img\_path)) {

$is\_upload = true;

} else {

$msg = '上传出错！';

}

} else {

$msg = '此文件类型不允许上传！';

}

} else {

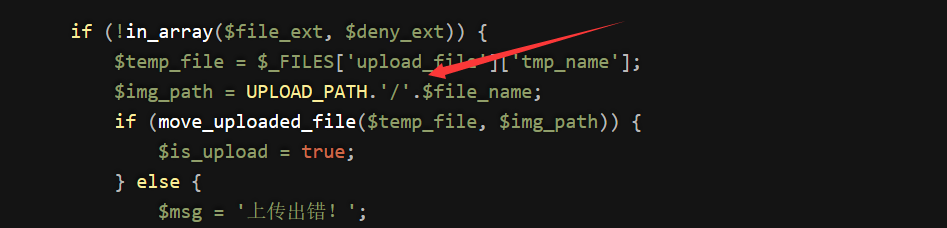
$msg = UPLOAD\_PATH . '文件夹不存在,请手工创建！';

}}

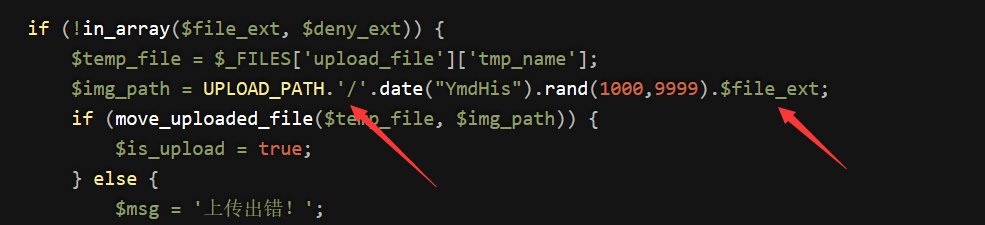
从源码来看，貌似之前遇到的所有的绕过手段都被考虑到了，还能怎么绕过呐，在没有思路之前先好好看看源码是一个不错的选择。

从源码来看先删除了文件末尾的”.”。然后转换了大小写还把::$DATA也给删掉了。最后还进行了首尾去空。这tm还怎么玩？

等等，好像有问题。这里只进行了一次删除”.”的操作。然后就直接拼接到了上传目录的后面



然而前面几关都是用做完所有的防绕过处理之后得到的文件名进行拼接的。



那么我们只需要构造后缀名，使它在经过所有的防绕过操作之后不在黑名单中，并且经过去除末尾”.”号之后拼接还能解析即可。

所以我们是不是可以构造文件后缀名为”. .”呐？比如”cs.php. .”

注意：这里两个”.”之间是含有一个空格的。

在第一步去除”.”之后就是“cs.php. ”

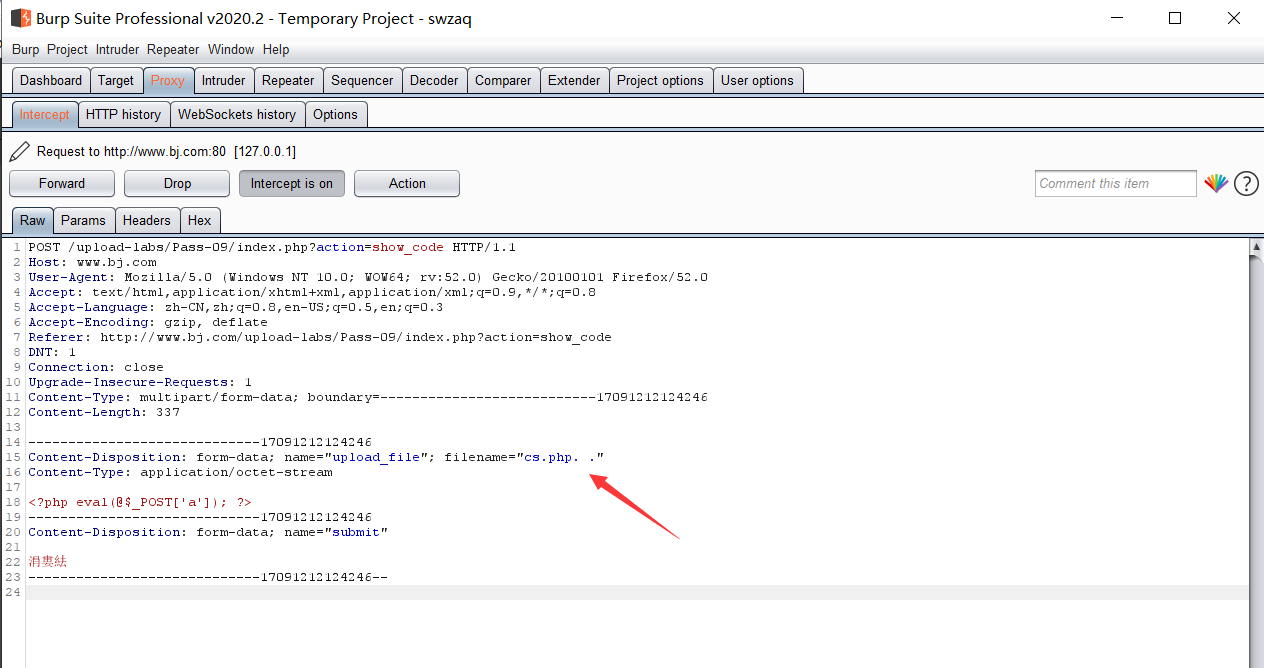
然后截取第一个”.”之后的后缀名就是”.php. ”

最后首尾去空之后就是”.php.”，这个没有在黑名单中，所有通过了黑名单。

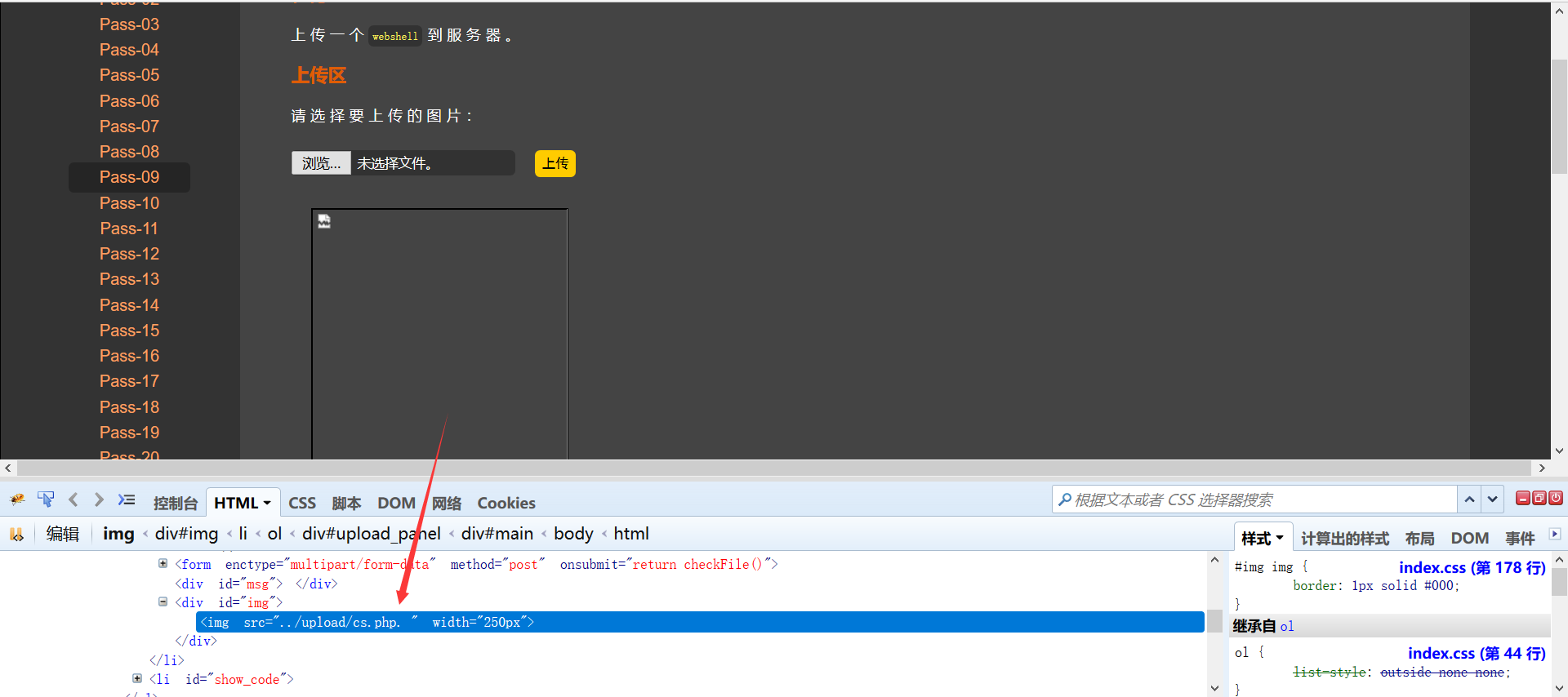
当在第一步去除”.”之后的”cs.php. ”拼接到上传目录之后，靠着windows的特性就完全可以解析了。

具体操作如下：

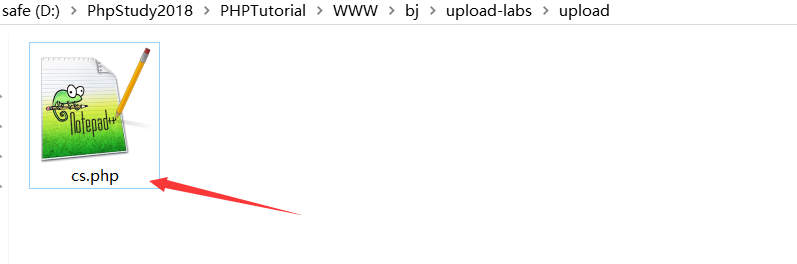
第一步：上传shell，抓包修改文件名



第二步：成功上传，对比页面显示和服务器文件名



可以看到在页面上显示的是文件名后面还有”.”，那么看看在服务器上是如何的吧



可见文件正常，那么就可以对shell进行常规操作了。