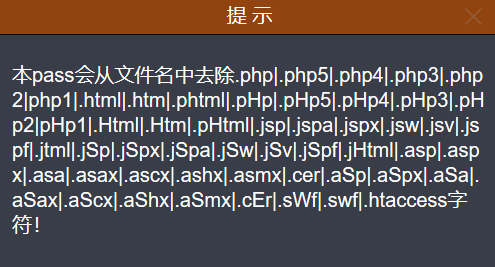
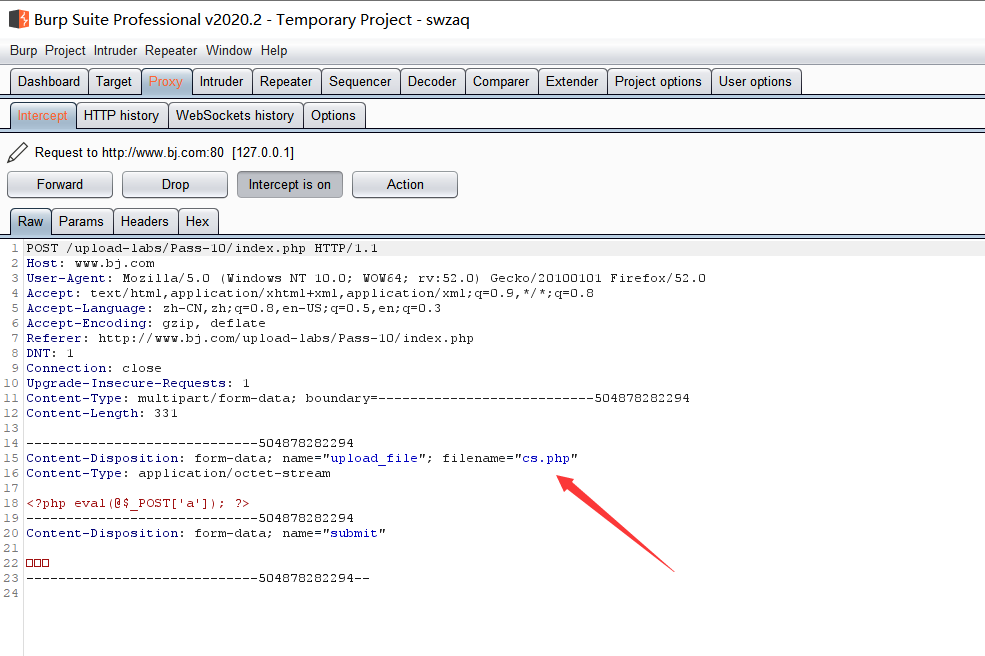
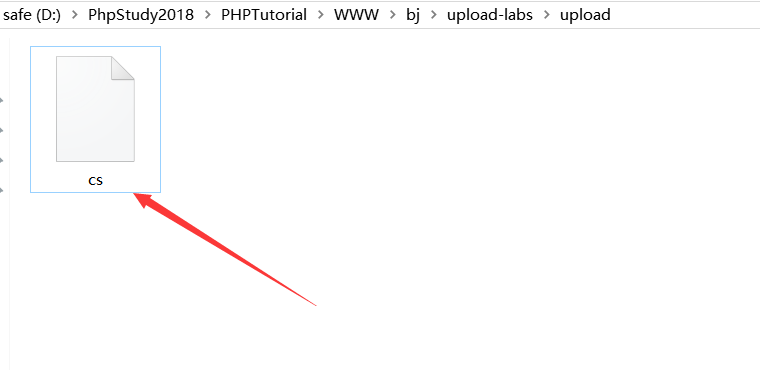
Pass10

打开第十关，查看提示



可以看到跟前面的都不一样，这里提示的是会把这些敏感的后缀从文件名中去除，那么到底是如何去除的呐，我们先用shell测试一下





查看上传至服务器的文件可以看到确实是把敏感的后缀名直接去除了，这样的话即使能上传成功也不是一个可以利用的shell了。

那么接下来从源码入手看看有没有什么漏洞可寻吧。

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

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

$deny\_ext = array("php","php5","php4","php3","php2","html","htm","phtml","pht","jsp","jspa","jspx","jsw","jsv","jspf","jtml","asp","aspx","asa","asax","ascx","ashx","asmx","cer","swf","htaccess");

$file\_name = trim($\_FILES['upload\_file']['name']);

$file\_name = str\_ireplace($deny\_ext,"", $file\_name);

$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 = UPLOAD\_PATH . '文件夹不存在,请手工创建！';

}}

从源码不难看出，str\_ireplace（）函数就是这一关中至关重要的函数，它的作用主要就是将文件名中含有黑名单字符串的那一部分替换为空（去除）。

该函数的具体用法请参考：

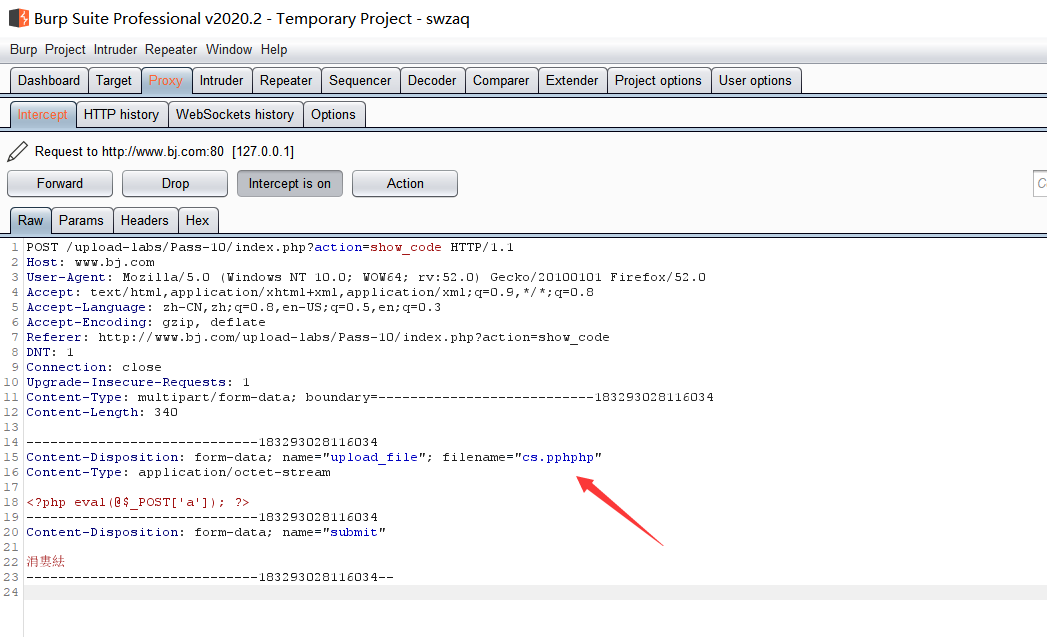
<https://www.w3school.com.cn/php/func_string_str_ireplace.asp>

当你对该函数足够了解了之后不难发现，它是拿匹配字符串去挨个的对等待匹配字符串进行对比，当匹配上了就替换并返回。

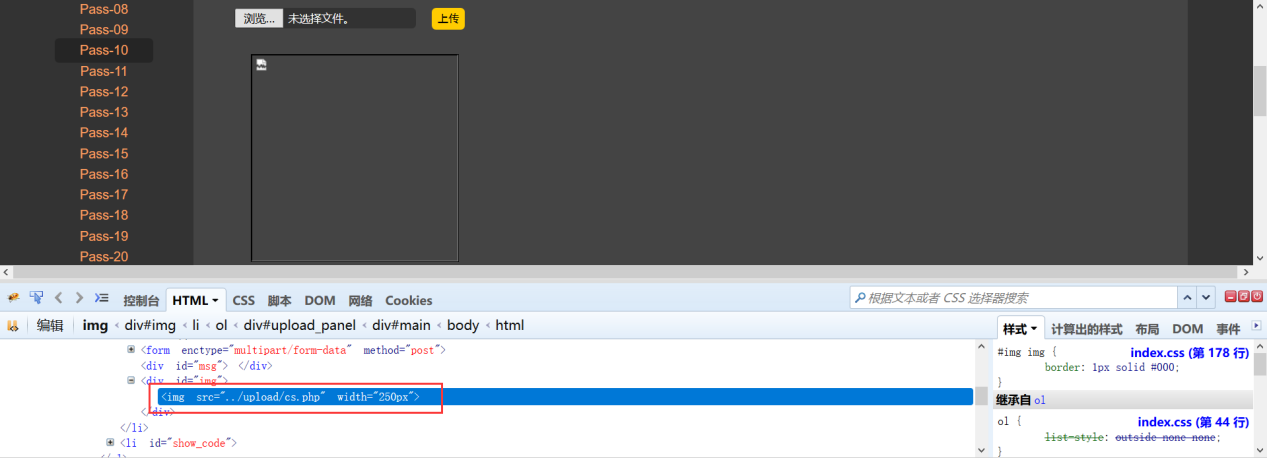
所以在这里如果我们构造文件为cs.pphphp呐？当匹配到第一个p的时候肯定是不满足的，但是匹配到第二个就能匹配到php并且被去除了。这里在匹配完成之后就不会对剩下的继续进行匹配了，因此剩下的p和hp自然就组合为了php。完美绕过str\_ireplace（）函数的过滤并且进入了后续的拼接。

理论很nice，接下来实践才能知道是否符合我们的预期

第一步：上传shell，抓包修改文件名



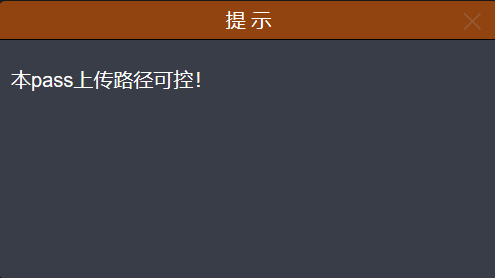
第二步：查看页面返回信息，判断是否成功上传（能被解析）



通过返回的信息来看，文件名确实是被去除了一个php，但是剩下的组合起来依然是正常的，所以成功绕过了。值得一提的是，大家都把这种方法叫做文件名双写绕过。

Pass11

打开第十一关，查看提示



貌似跟之前的都不太一样呐，估计是个新姿势。上传路径可控？难道是跟Apache服务器的解析漏洞有关吗？

还是看看源码再说吧

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

$ext\_arr = array('jpg','png','gif');

$file\_ext = substr($\_FILES['upload\_file']['name'],strrpos($\_FILES['upload\_file']['name'],".")+1);

if(in\_array($file\_ext,$ext\_arr)){

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

$img\_path = $\_GET['save\_path']."/".rand(10, 99).date("YmdHis").".".$file\_ext;

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

$is\_upload = true;

} else {

$msg = '上传出错！';

}

} else{

$msg = "只允许上传.jpg|.png|.gif类型文件！";

}}

从源码不难看出，服务器是用白名单来进行验证文件上传的。只允许了jpg、png、gif三种格式。不过与之前不同的是，在提示中说上传路径是可控的，也就是

$img\_path=$\_GET['save\_path']."/".rand(10,99).date("YmdHis").".".$file\_ext;

可以看到save\_path这个参数是客户端可控的。但是那又能怎么样呐？这里就不得不提到%00截断了。

很多初学者可能会一脸懵逼，不过不要紧，我详细说明一下。

首先理解下截断的原理，在C/PHP等语言中，截断的核心，就是chr(0)。chr()是一个函数，这个函数是用来返回参数所对应的字符的，也就是说，参数是一个ASCII码，返回的值是一个字符，类型为string。

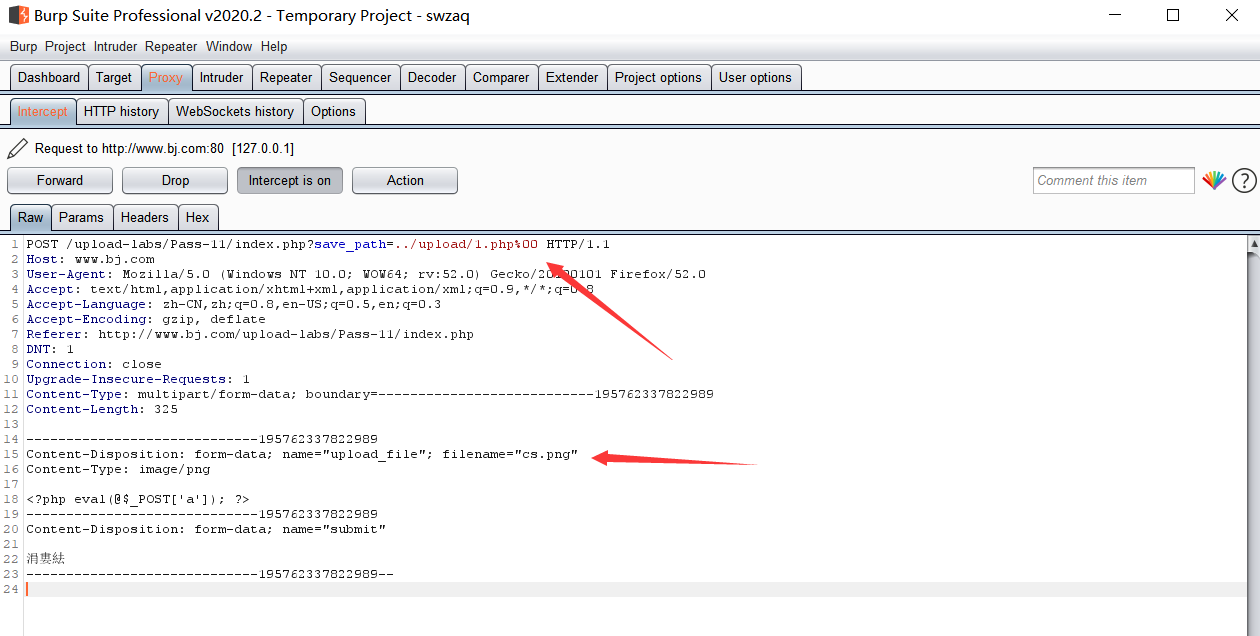
那么chr(0)就很好理解了，对照ASCII码表可以知道，ASCII码为0-127的数字，每个数字对应一个字符，而0对应的就是NUT字符（NULL），也就是空字符，而截断的关键就是于这个空字符。当一个字符串中存在空字符的时候，在被解析的时候会导致空字符后面的字符被丢弃。换句话说就是会误把它当成结束符，后面的数据直接忽略，这就导致漏洞产生。

可能有时候我们会看到%00截断和0x00截断等说法。那么 %00截断和0x00截断有啥区别呢？其实他们的截断原理都一样，%00只不过是对ascii码中的0对应的字符编码后的结果。0x开头表示16进制，0在十六进制中是00, 0x00则是%00解码成的16进制。

值得注意的是：当url中的参数是通过GET方式获取时，%00会被自动解码。而当参数是通过POST方式获取时，是不会自动解码的，也就是说%00只会原样被当成字符串来输出。所以通过POST方式请求此类参数的时候，我们需要手动将它的十六进制改写为0x00。

明白了截断的原理，再结合此处的上传路径可控，我们就可以进行绕过了。

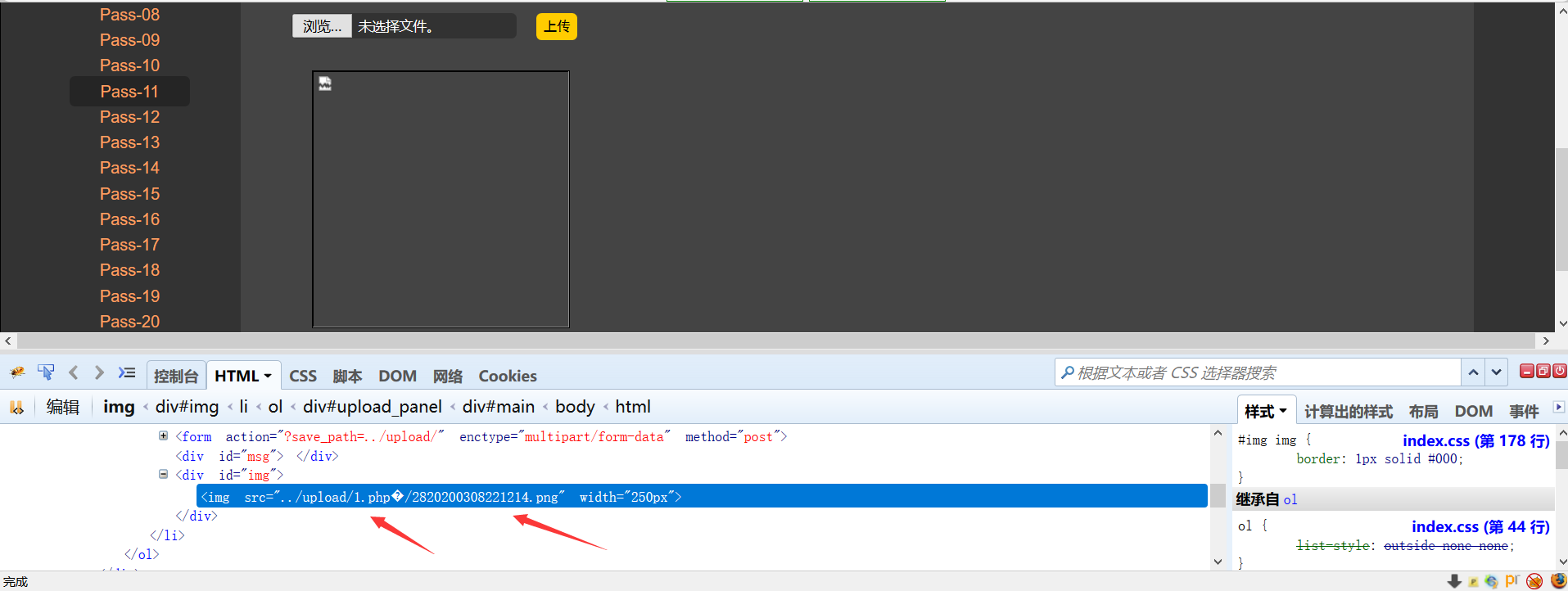
第一步：上传shell，抓包修改上传路径参数



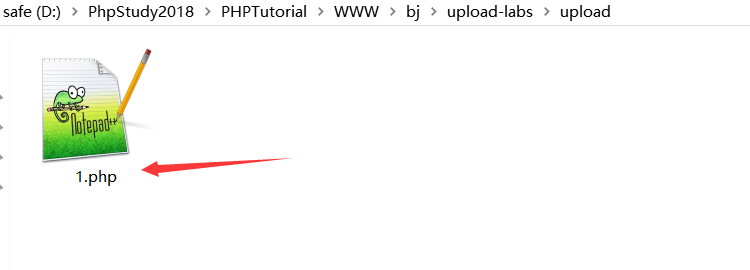
可以看到这里的文件名是符合白名单的，并且当修改后的路径参数被服务器获取和文件名进行拼接之后。系统会将临时文件移动到拼接的路径中。

此时当系统读取到1.php的时候就会以为结束了并停止继续读取，同时在目录下创建一个1.php文件，然后就会将cs.png的内容附加到此文件中。

第二步：对比返回页面内容和服务器文件实际情况



可以看到返回页面中显示文件位置是拼接过后的形式，不过像这样的形式必然是无法访问也没法用菜刀等连接的。下面看看服务器上是怎么样的。



可以看到系统在向服务器创建文件时确实被截断了，顺利的创建了一个shell。用蚁剑连接测试成功。

