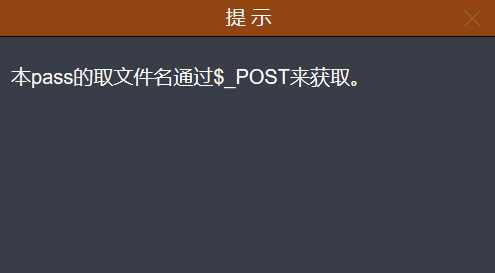
Pass19

打开第十九关，发现上传的地方还可以定义上传的文件名



再看看提示



这里说这个文件名是用的post方式传递的，那么看看源码到底要怎么绕过

$is\_upload = false;$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 = $\_POST['save\_name'];

$file\_ext = pathinfo($file\_name,PATHINFO\_EXTENSION);

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 = '禁止保存为该类型文件！';

}

} else {

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

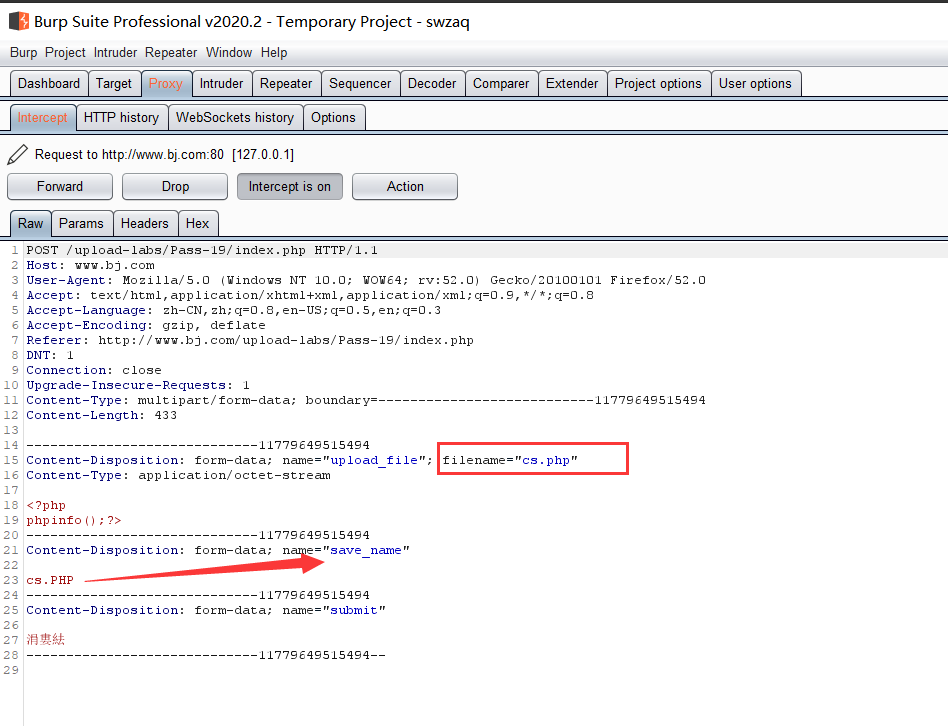
}}

从源码来看的话，服务器端定义了一系列的黑名单，save\_name参数就是我们保存名称的参数了。这里并未对上传的文件类型做什么判断，仅仅是对保存的名称做了黑名单处理。不难看出这些黑名单都是小写的形式，也就是说我们完全可以上传一个一句话木马然后将其命名为PHP后缀即可绕过，真是跟没有防护一样。

第一步：直接上传php文件，保存名称填成PHP

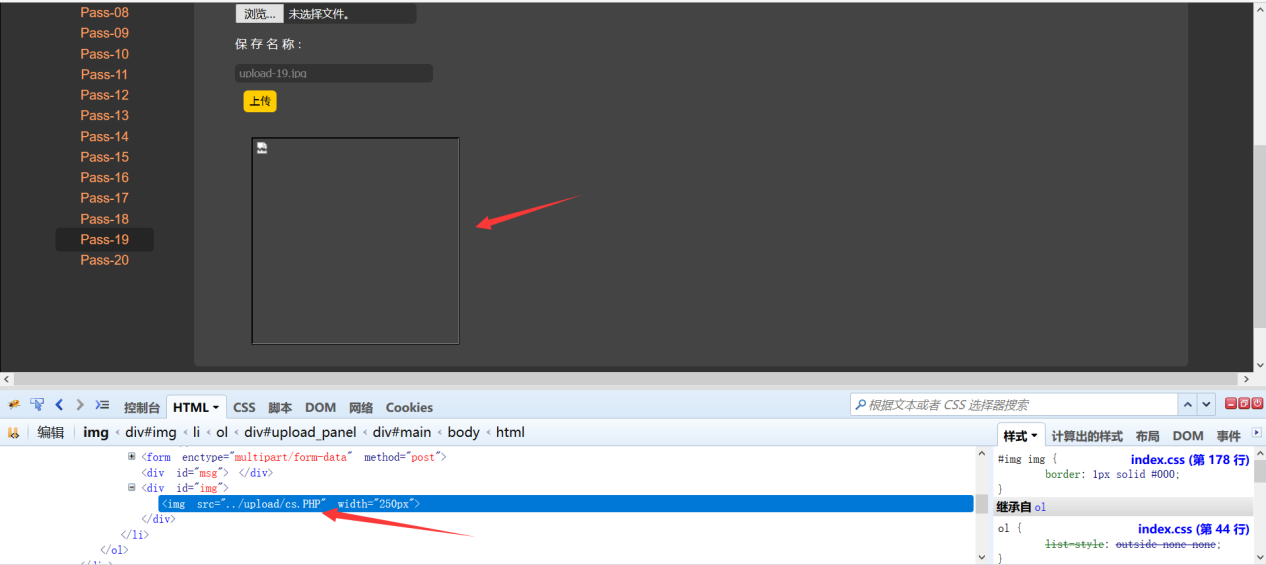


第二步：抓包分析

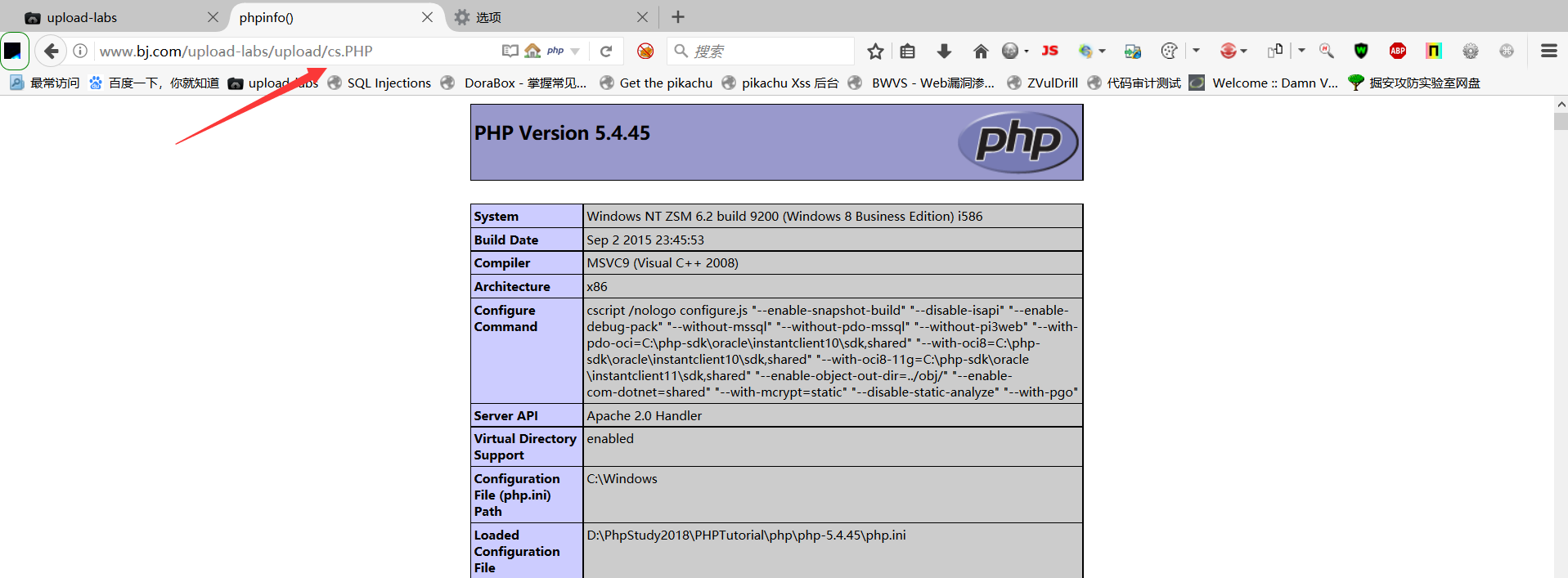


可以看到filename是文件本来的名称，而save\_name是传递的参数，此处改为PHP可以绕过黑名单。

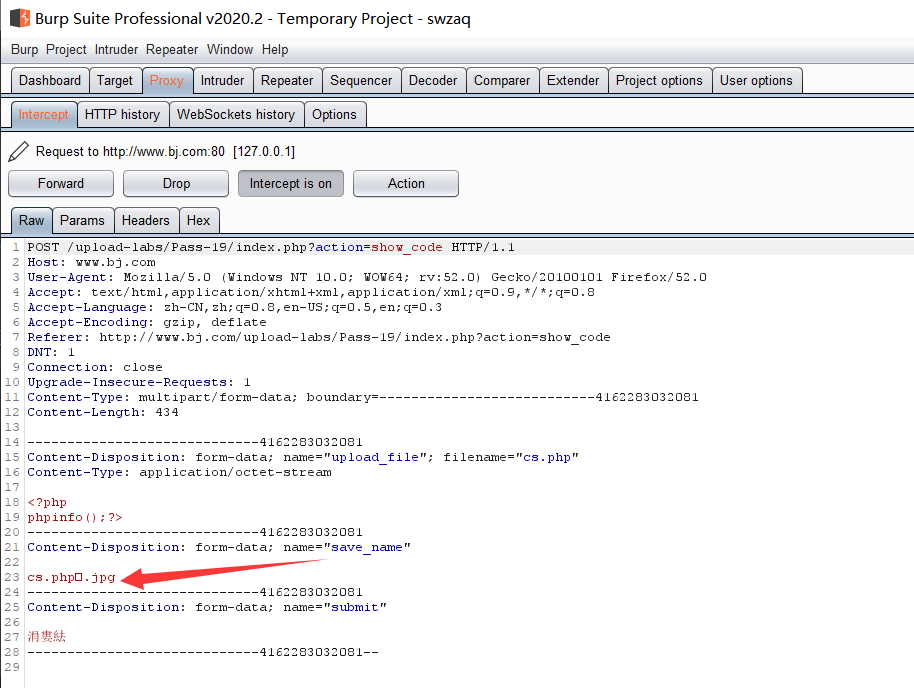
第三步：验证是否上传成功



页面显示是上传成功了，到浏览器访问一下看看



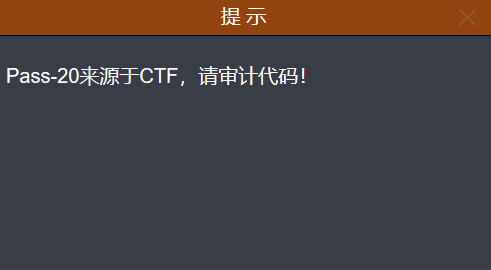
当然这一关绕过方法不止这一种，可以在save\_name参数中文件名称后面加空格绕过，还可以通过save\_name参数来进行00截断（因为参数值带入到了路径拼接中）。当然使用00截断的时候还是要符合环境才行。



希望大家不要被思维所局限，条条大路通shell。

Pass20

打开最后一关，不免有些激动。靶场终于要通关啦。先看看提示



依然是依靠代码审计找出漏洞，那么看看源码吧

$is\_upload = false;$msg = null;if(!empty($\_FILES['upload\_file'])){

//检查MIME

$allow\_type = array('image/jpeg','image/png','image/gif');

if(!in\_array($\_FILES['upload\_file']['type'],$allow\_type)){

$msg = "禁止上传该类型文件!";

}else{

//检查文件名

$file = empty($\_POST['save\_name']) ? $\_FILES['upload\_file']['name'] : $\_POST['save\_name'];

if (!is\_array($file)) {

$file = explode('.', strtolower($file));

}

$ext = end($file);

$allow\_suffix = array('jpg','png','gif');

if (!in\_array($ext, $allow\_suffix)) {

$msg = "禁止上传该后缀文件!";

}else{

$file\_name = reset($file) . '.' . $file[count($file) - 1];

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

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

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

$msg = "文件上传成功！";

$is\_upload = true;

} else {

$msg = "文件上传失败！";

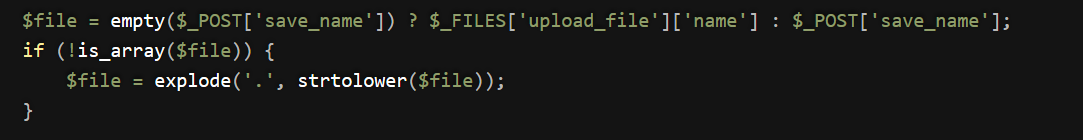
}

}

}}else{

$msg = "请选择要上传的文件！";}

相比较于上一关的源码，此处服务器端先是检查了MIME类型，然后判断save\_name参数是否为空，为空就把文件本来名称赋值给$file,否则就是将save\_name参数的值赋给它。紧接着判断$file是否是数组



如果不是数组则将其拆成数组，然后数组最后一个的值(end函数就是取数组最后一个的值)同白名单做比较，符合jpg、png、gif中的一种就允许上传了。

在允许上传之后还要把数组的值拼接在一起对文件进行重命名。

所以我们可以构造save\_name[0]=cs.php/ save\_name[1]置为空 save\_name[2]=jpg(一个白名单的合法后缀)。

这样的话，reset($file)取的是数组的第一个元素即cs.php/，然后接了一个’.’符号，之后又将数组最后一个元素内容拼接到一起。

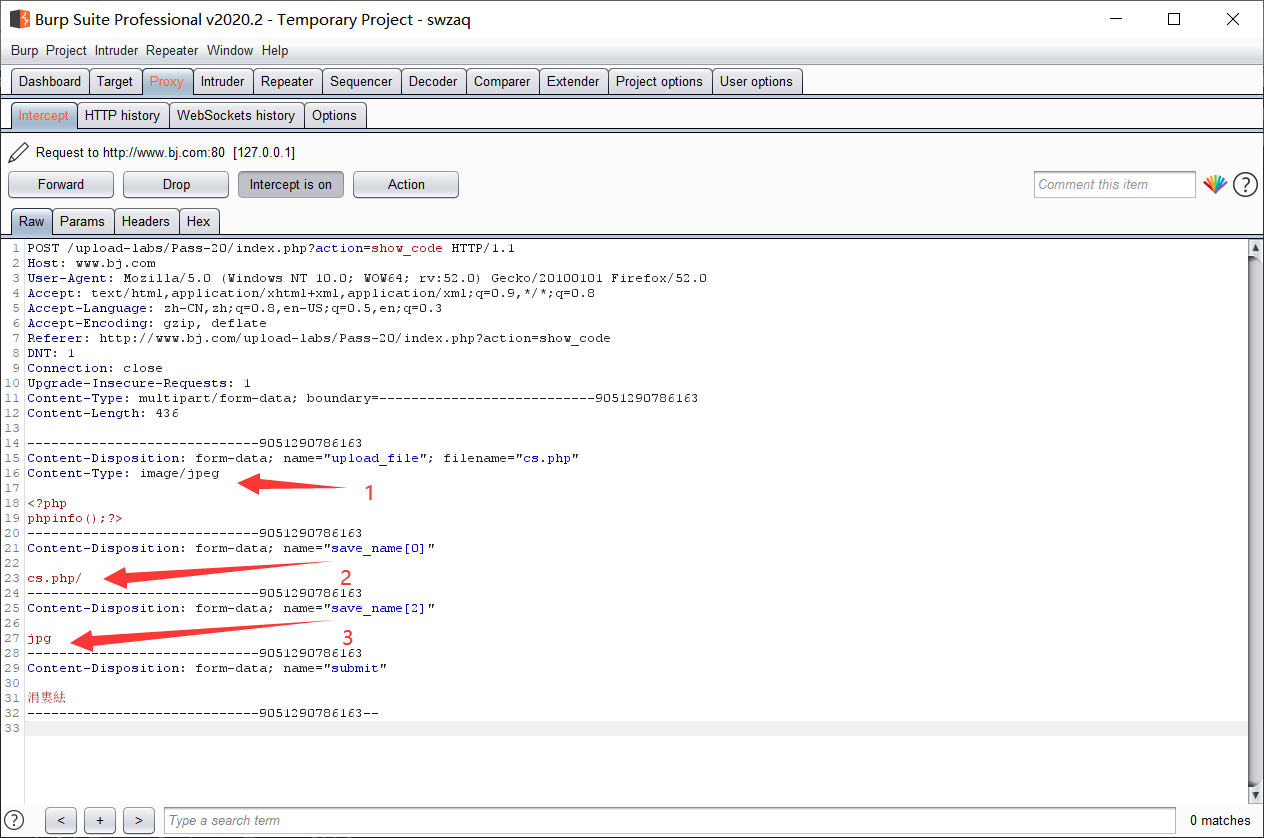
可能有的人会疑问数组最后一个值不是jpg吗？其实当我们只设置了两个数组元素的时候，数组的元素个数就只有两个了。

10

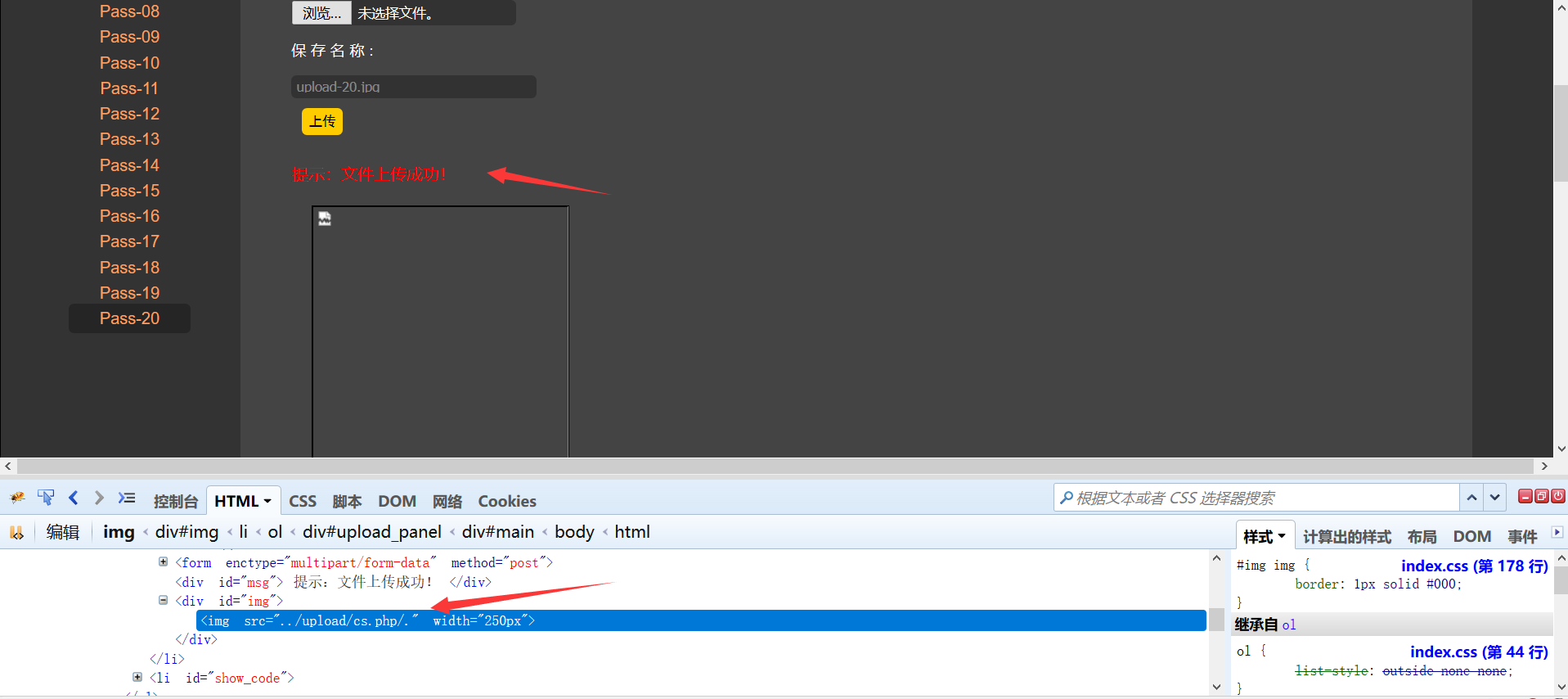
既然一共只有两个元素，这里就是$file[2-1]也就是$file[1]。因此拼接的就是空的，最终得到的文件名就是cs.php/.。

对于像cs.php/.这样的文件路径，move\_uploaded\_file()函数会忽略掉文件末尾的/.。如此一来我们上传到服务器的文件还是被重命名为了php后缀。

第一步：上传文件，抓包修改参数



第二步：放行数据包，查看是否成功上传



根据网页的回显是成功上传到服务器了。

总结：对于upload-labs靶场的文章到此就结束了，但是对于文件上传漏洞的学习还未终止。靶场只是给我们提供了一些不错的绕过思路，但是在实际的应用环境中，肯定会遇到更加全面的防御。在渗透测试过程中就要善于思考，结合其他漏洞一起突破。万万不能被局限了创新。如果觉得本次系列文章还不错，请点点赞，多多评论哟。