目录

# upload-labs 靶场练习实验报告

## 一、实验内容和目的

### 1.1 实验内容

本实验以 upload-labs 靶场为练习环境，该靶场是一款专注于文件上传漏洞学习与实践的平台，包含 20 个不同防护级别与漏洞类型的关卡。实验核心内容为逐一突破靶场的 20 个关卡，通过分析各关卡的文件上传防护机制，利用对应的漏洞利用技术，成功将恶意 PHP 文件（如一句话木马）上传至服务器并实现有效执行，从而全面掌握文件上传漏洞的常见利用方法及绕过技巧。

### 1.2 实验目的

- 深入理解文件上传漏洞的原理、危害及产生原因，明确文件上传功能在 Web 应用中的安全风险点。
- 掌握文件上传漏洞的常见防护机制，包括前端 JS 验证、后端 MIME 类型验证、文件扩展名验证、文件内容验证、文件路径限制等。
- 熟练运用各类文件上传漏洞绕过技巧，如前端验证绕过、MIME 类型绕过、扩展名绕过、文件内容欺骗、%00 截断、条件竞争等，能够针对不同防护场景选择合适的利用方法。
- 提升 Web 安全漏洞分析与利用能力，培养面对实际安全问题时的逻辑思维与问题解决能力。
- 了解文件上传漏洞的防御措施，形成“攻击-防御”双向的安全思维，为后续 Web 应用安全开发与渗透测试工作奠定基础。

## 二、实验要求

- 实验环境需确保 upload-labs 靶场可正常运行，已完成基础环境配置（如 PHP 环境、Apache/Nginx 服务器等），无需重复配置环境步骤。
- 针对每个关卡，需详细分析其防护机制，记录完整的实验步骤，包括漏洞探测、利用思路、操作过程及最终结果，避免仅呈现结果无过程说明的情况。
- 实验过程中需使用合法合规的工具与技术，仅对靶场环境进行操作，严禁

将相关技术用于未授权的网络或系统。

- 及时记录实验中遇到的问题及解决方法，总结各关卡的核心知识点与技巧，形成完整的实验心得。
- 实验报告需结构清晰、逻辑严谨、内容翔实，准确反映实验过程与结果。

# 三、实验步骤和结果

实验前准备：创建基础恶意 PHP 文件，命名为“shell.php”，内容为<?php phpinfo(); @eval($_POST['666']);?>，该文件为一句话木马，可通过 POST 请求的“666”参数执行任意 PHP 代码。同时准备用于测试的正常图片文件“test.jpg”，部分关卡需结合图片文件进行绕过。

## 3.1 第 1 关：前端 JS 验证绕过

### 3.1.1 实验步骤

1. 访问第 1 关页面，观察页面提示“请选择要上传的图片”，点击“选择文件”按钮，尝试直接上传“shell.php”文件。

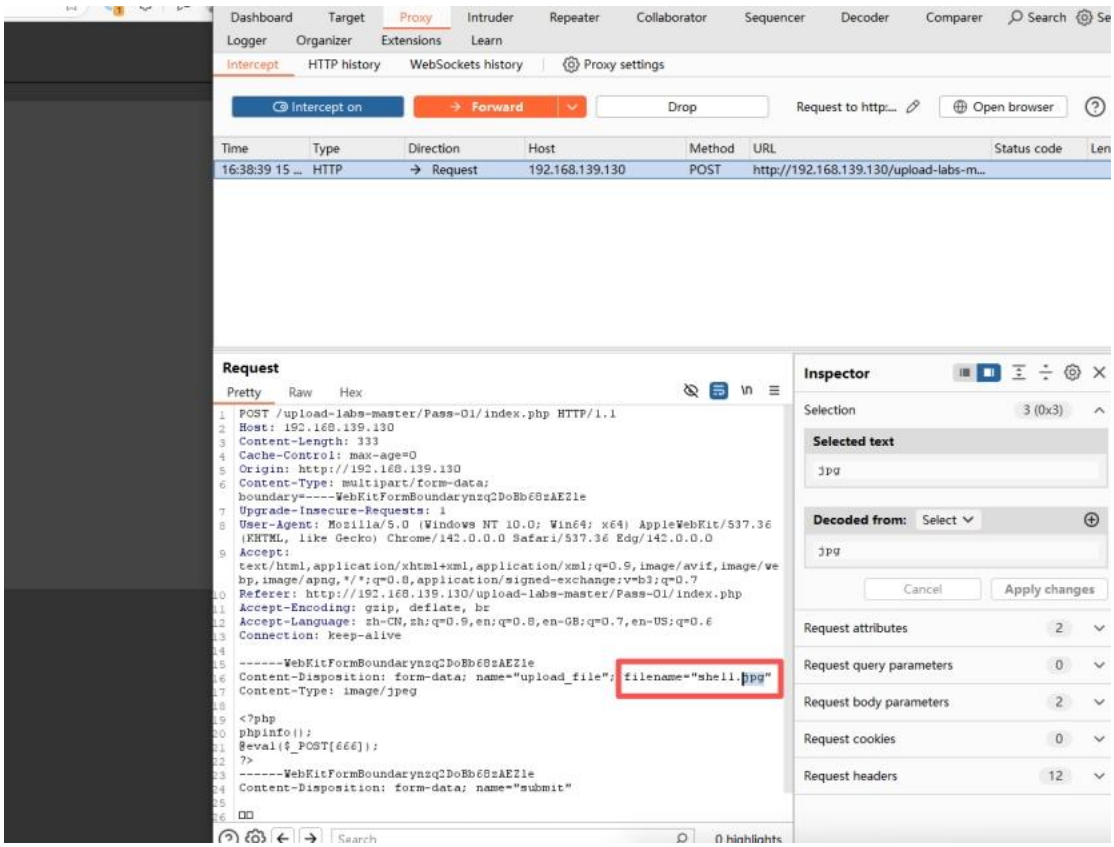
2. 发现页面立即弹出“请选择 jpg/png/gif 类型的文件”提示，未向服务器发送请求，判断该关卡仅通过前端 JavaScript 进行文件类型验证，限制了文件扩展名。

3. 前端验证绕过思路：前端验证仅在客户端生效，可通过修改文件扩展名绕过前端检查，再通过抓包工具将文件名改回原扩展名；或直接禁用页面 JavaScript 功能。本次采用抓包修改方式，具体操作如下：

将“shell.php”文件重命名为“shell.jpg”，此时前端 JS 验证认为其为图片文件，允许上传。

4. 打开 Burp Suite 工具，开启抓包功能，点击页面“上传”按钮，Burp Suite 捕获到文件上传请求包。

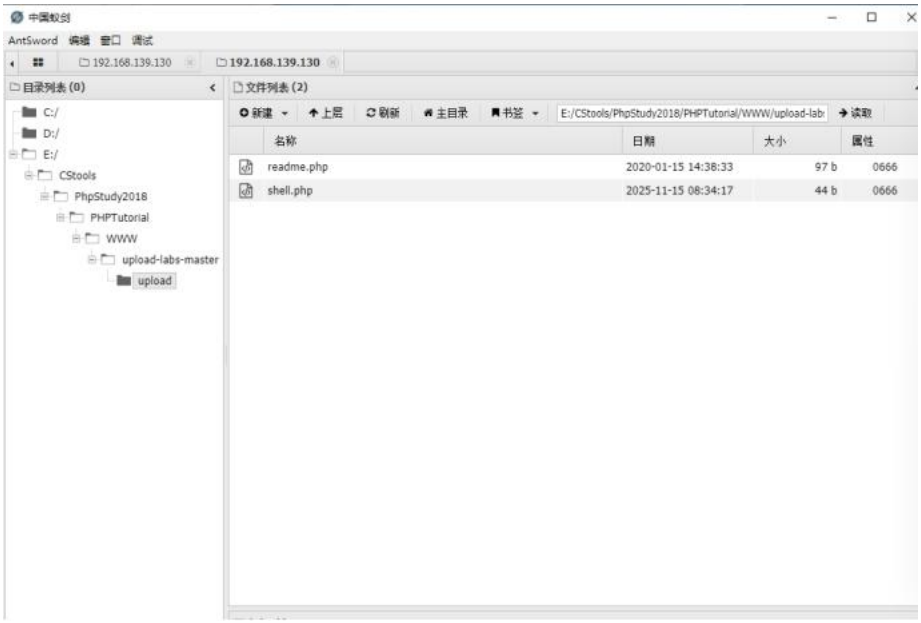
5. 在请求包中找到“Content-Disposition: form-data; name="upload_file"; filename="shell.jpg""这一行，将 filename 后的“shell.jpg”修改为“shell.php”。




6. 保持其他内容不变，点击“Forward”按钮将修改后的请求包发送至服务器。

### 3.1.2 实验结果

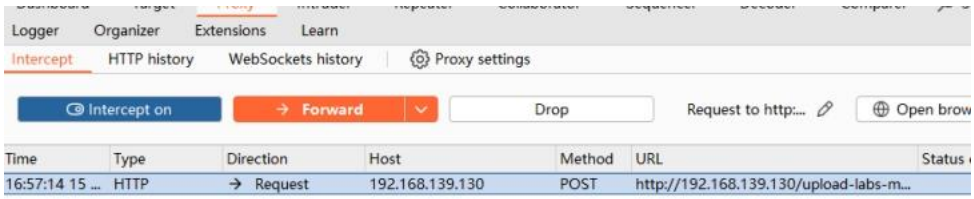
服务器接收请求后，成功将文件以“shell.php”的名义保存，页面提示“上传成功！路径：./upload/shell.php”。通过蚁剑工具，输入文件路径“http://192.168.139.130/upload-labs-master/upload/shell.php”，连接密码“666”，成功连接到服务器，可执行任意命令，证明文件上传及执行成功。

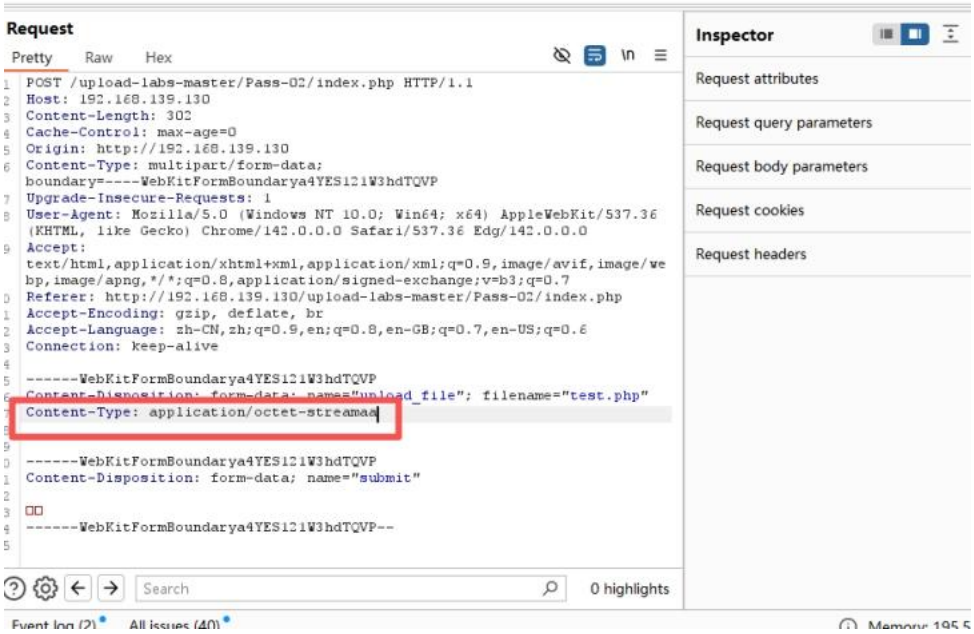



## 3.2 第 2 关：后端 MIME 类型验证绕过

### 3.2.1 实验步骤

1. 访问第 2 关页面，直接上传“shell.php”文件，页面提示“文件类型不正确，请重新上传”。查看 Burp Suite 抓包记录，发现请求头中“Content-Type”字段为“application/octet-streamaa”，判断服务器后端通过验证文件的 MIME 类型进行防护。




2. MIME 类型绕过思路：修改请求头中的“Content-Type”字段，将其改为图片文件对应的 MIME 类型（如 image/jpeg、image/png 等），欺骗服务器后端验证。

3. 具体操作：

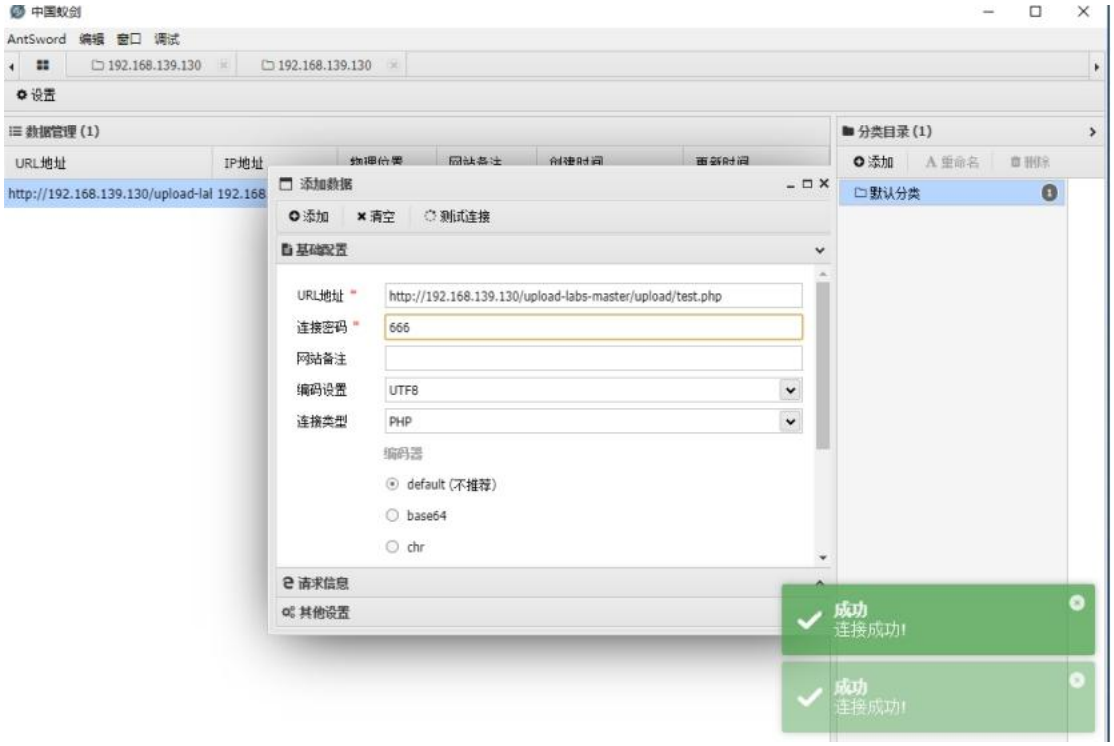
直接上传“shell.php”文件，通过 Burp Suite 捕获请求包。

4. 在请求包中找到“Content-Type: application/x-httpd-php”，将其修改为“Content-Type: image/jpeg”。

5. 发送修改后的请求包至服务器。

### 3.2.2 实验结果

服务器验证通过，页面提示“上传成功！路径：./upload/shell.php”。使用蚁剑工具成功连接该文件，执行命令正常，实验成功。




## 3.3 第 3 关：黑名单扩展名绕过（.php3/.php5 等）

### 3.3.1 实验步骤

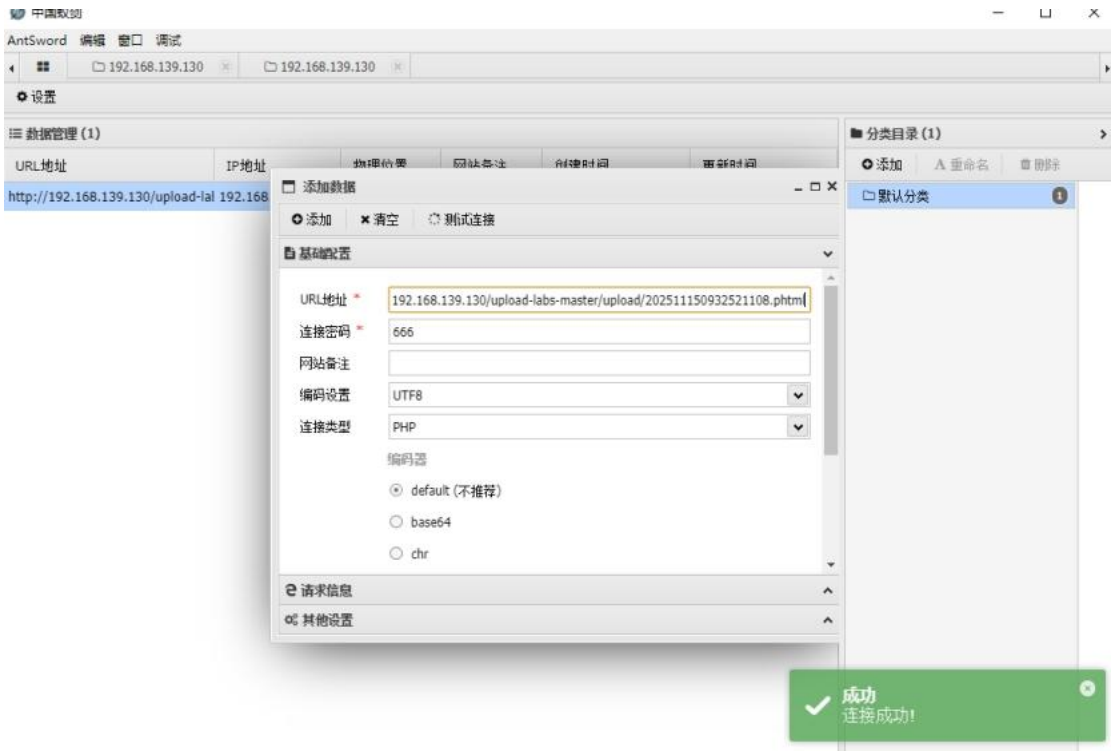
1. 访问第 3 关页面，尝试上传“shell.php”文件，页面提示“不允许上传.asp,.aspx,.php,.jsp 后缀文件！”，判断服务器采用黑名单机制限制文件扩展名。

2. 黑名单绕过思路：部分服务器配置中，除了.php 扩展名外，.php3、.php5、.phtml 等扩展名也会被解析为 PHP 文件，若黑名单未包含这些扩展名，可通过修改文件扩展名实现绕过。(老师提供的工具包中 phpstudy 的 Apache 服务器配置不能解析.php3、.phptml 等扩展名，需自行添加)

3. 具体操作：将“shell.php”文件重命名为“shell.phtml”，直接点击“上传”按钮。

### 3.3.2 实验结果

页面提示“上传成功！路径：./upload/shell.phptml”。在浏览器中访问该文件路径，服务器成功将其解析为 PHP 文件，通过蚁剑工具使用“666”参数连接，可正常执行命令，实验成功。

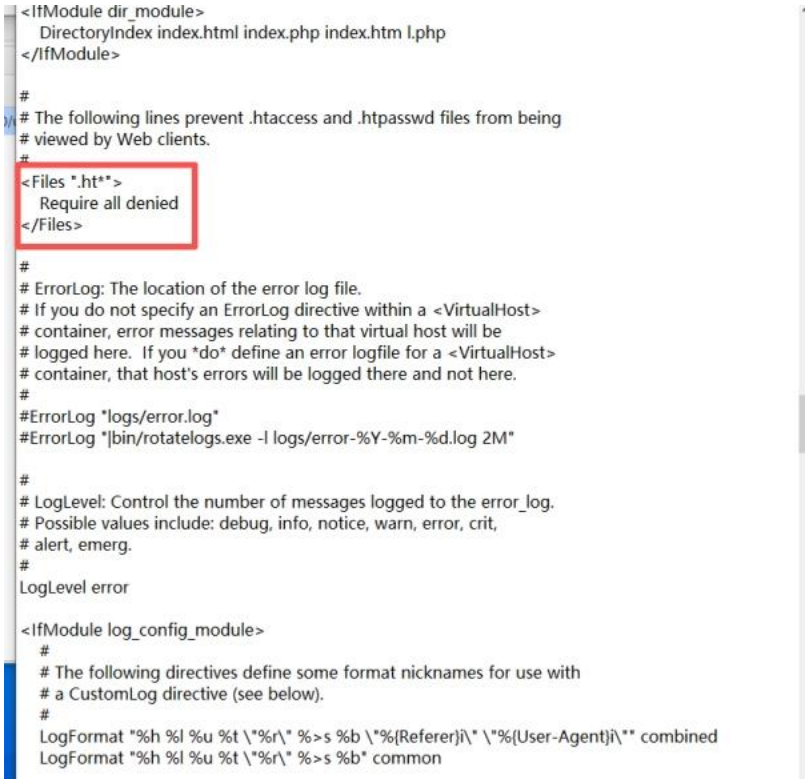



## 3.4 第 4 关：黑名单扩展名绕过（.htaccess）

### 3.4.1 实验步骤

1. 访问第 4 关页面，尝试上传“shell.php”“shell.php3”等文件，均被拒绝，查看源码黑名单包含了常见的 PHP 相关扩展名。

2. htaccess 文件绕过思路：.htaccess 是 Apache 服务器的配置文件，可通过该文件设置文件解析规则。若服务器允许上传.htaccess 文件，可上传自定义的.htaccess 文件，指定特定扩展名的文件被解析为 PHP。




需在 Apache 配置文件中注释掉红框代码，使.htaccess 文件被解析#

3. 具体操作：

创建.htaccess 文件，内容为“AddType application/x-httpd-php .jpg”，表示将所有.jpg 扩展名的文件解析为 PHP。

4. 上传.htaccess 文件，页面提示“上传成功！路径：./upload/.htaccess”。

5. 将“shell.php”文件重命名为“shell.jpg”，上传该文件，页面提示“上传成功！路径：./upload/shell.jpg”。

### 3.4.2 实验结果

由于.htaccess 文件的配置生效，服务器将“shell.jpg”解析为 PHP 文件。通过蚁剑工具连接“http://192.168.139.135/upload/shell.jpg”，密码“cmd”，成功执行命令，实验成功。

## 3.5 第 5 关：.user.ini 文件上传绕过

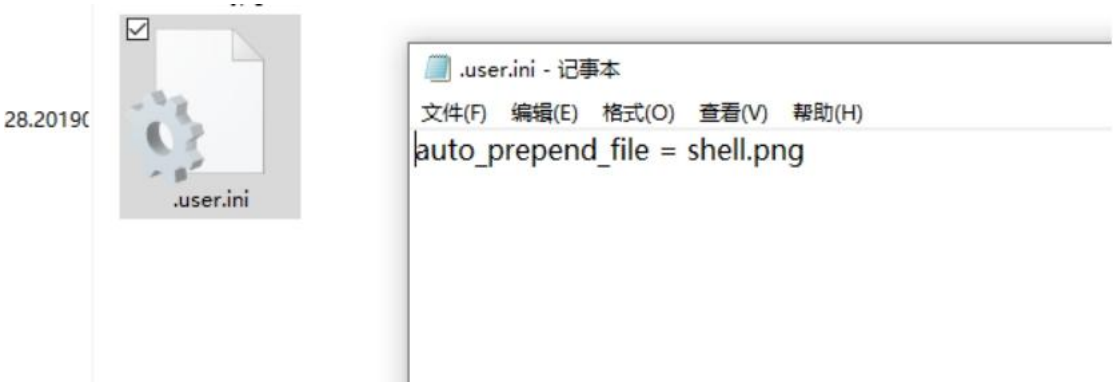
### 3.5.1 实验步骤

1. 访问第 5 关页面，尝试上传 “shell.php”“shell.php3”“shell.phtml” 等 PHP 相关扩展名文件，均被页面提示 “不允许上传该类型文件”，查看源码后端采用

严格的黑名单机制过滤常见危险扩展名。

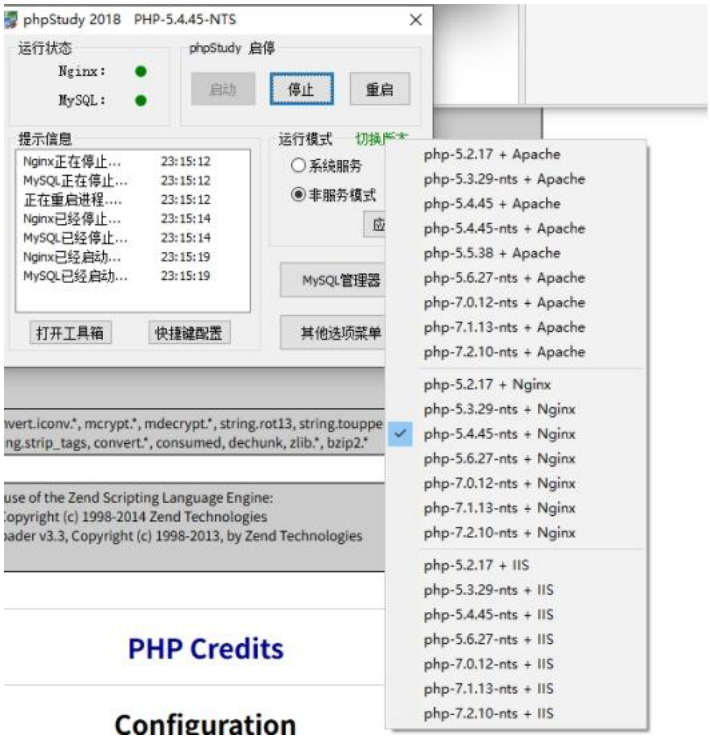
2. user.ini 文件绕过思路：.user.ini 是 PHP 的用户级配置文件，相当于自定义的小型 php.ini，可在不具备服务器 root 权限时修改部分 PHP 配置，若服务器以 CGI/FastCGI 模式运行（phpinfo 中可查看），.user.ini 文件会被自动解析，通过配置 auto_prepend_file 指令，可让服务器访问该目录下任意 PHP 文件时，自动包含指定的恶意文件（即使恶意文件扩展名是白名单允许的类型）。

3. 准备两个文件，一是创建.user.ini 文件，内容为 auto_prepend_file = shell.png，表示访问该目录下 PHP 文件时自动包含 “shell.png” 文件；二是将“shell.php”重命名为 “shell.png”，伪装成图片文件以符合白名单扩展名要求。




4. 先上传.user.ini 文件，页面提示 “上传成功！路径：./upload/.user.ini”，再上传 “shell.png” 文件，页面提示 “上传成功！路径：./upload/shell.png”。

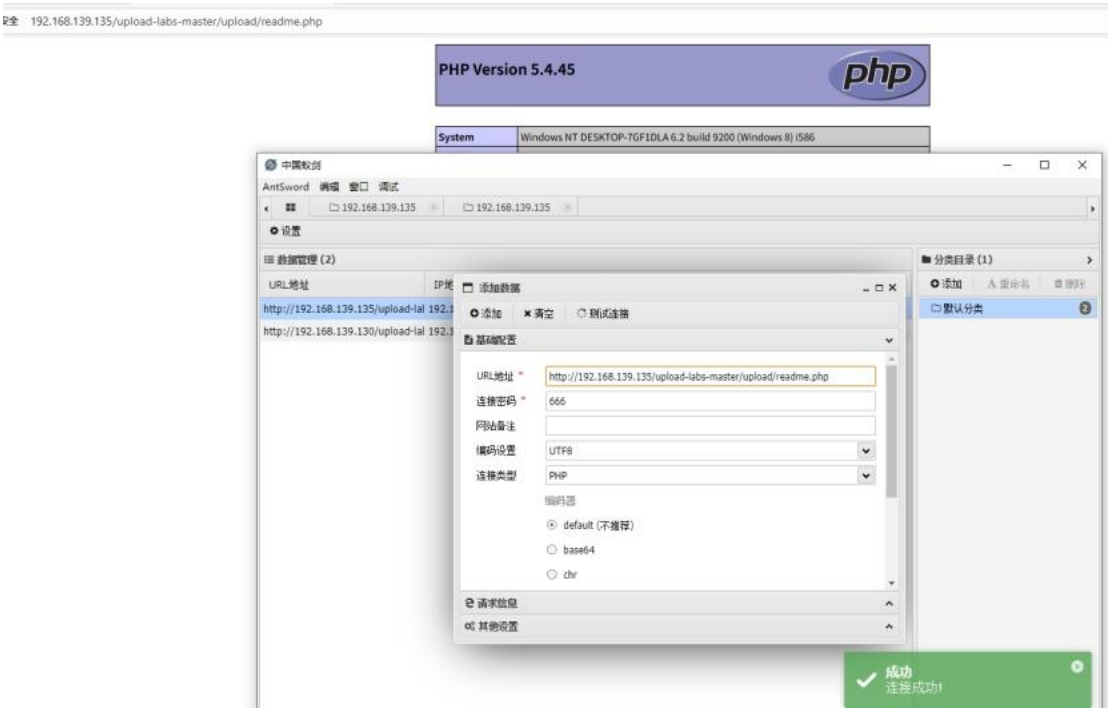
5. 修改 phpstudy 的版本，使服务器以 CGI/FastCGI 模式运行。




6. 确认上传目录下存在靶场预设的默认 PHP 文件“readme.php”，该文件用于触发.user.ini 的自动包含机制。

### 3.5.2 实验结果

由于服务器以 CGI/FastCGI 模式运行，.user.ini 文件的配置成功生效。当访问“http://192.168.139.135/upload-labs-master/upload/readme.php”时，服务器自动包含“shell.png”文件，将其伪装的图片后缀解析为 PHP 代码执行。打开蚁剑工具，输入连接路径“http://192.168.139.135/upload-labs-master/upload/readme.php”，连接密码“666”，点击“测试连接”提示“连接成功”，可正常执行文件浏览、命令执行等操作，实验成功。




## 3.6 第 6 关：黑名单绕过（大小写混淆）

### 3.6.1 实验步骤

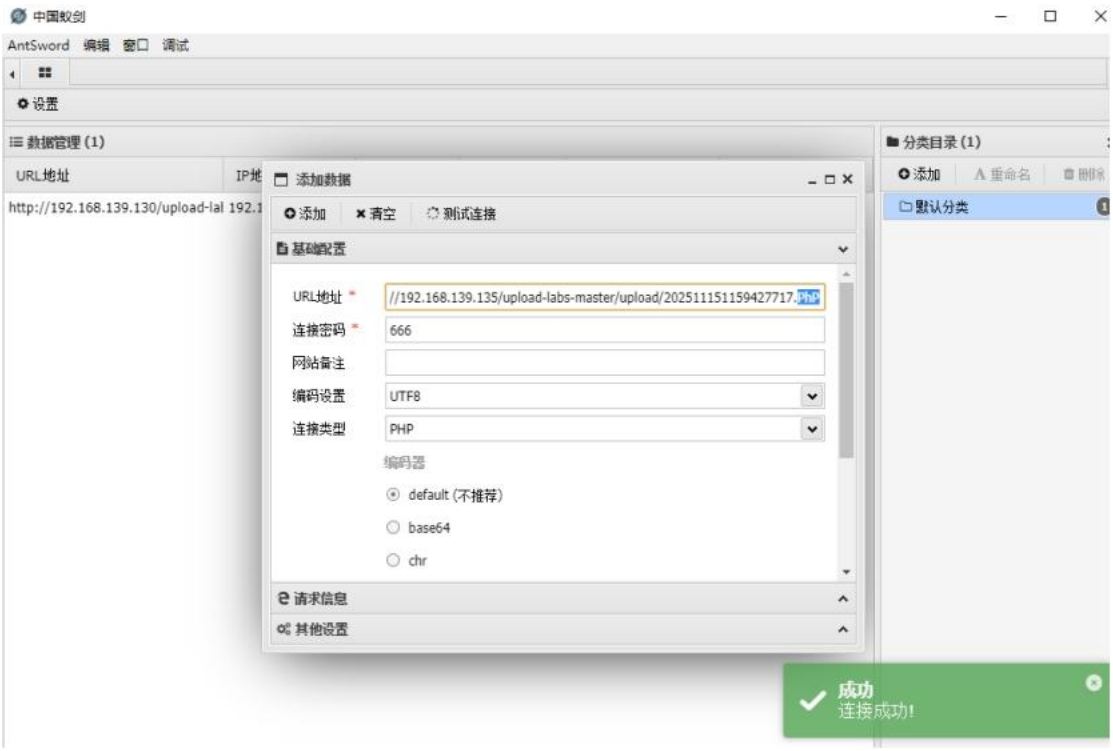
1. 访问第 6 关页面，尝试上传“shell.php”“shell.php5”及.htaccess 文件，均被拒绝，查看源码黑名单规则较为严格，但可能未处理大小写问题。

2. 大小写混淆绕过思路：部分后端验证代码未将文件名统一转换为大写或小写，直接进行匹配，可通过将文件扩展名改为大小写混合形式（如.Php、.PhP）绕过黑名单。

3. 具体操作：将“shell.php”文件重命名为“shell.Php”，点击“上传”按钮。

### 3.6.2 实验结果

服务器未识别出“shell.Php”为被禁止的文件类型，提示“上传成功！路径：./upload/shell.Php”。访问该文件路径，服务器正常解析，蚁剑连接成功，实验成功。




## 3.7 第 7 关：黑名单绕过（空格绕过）

### 3.7.1 实验步骤

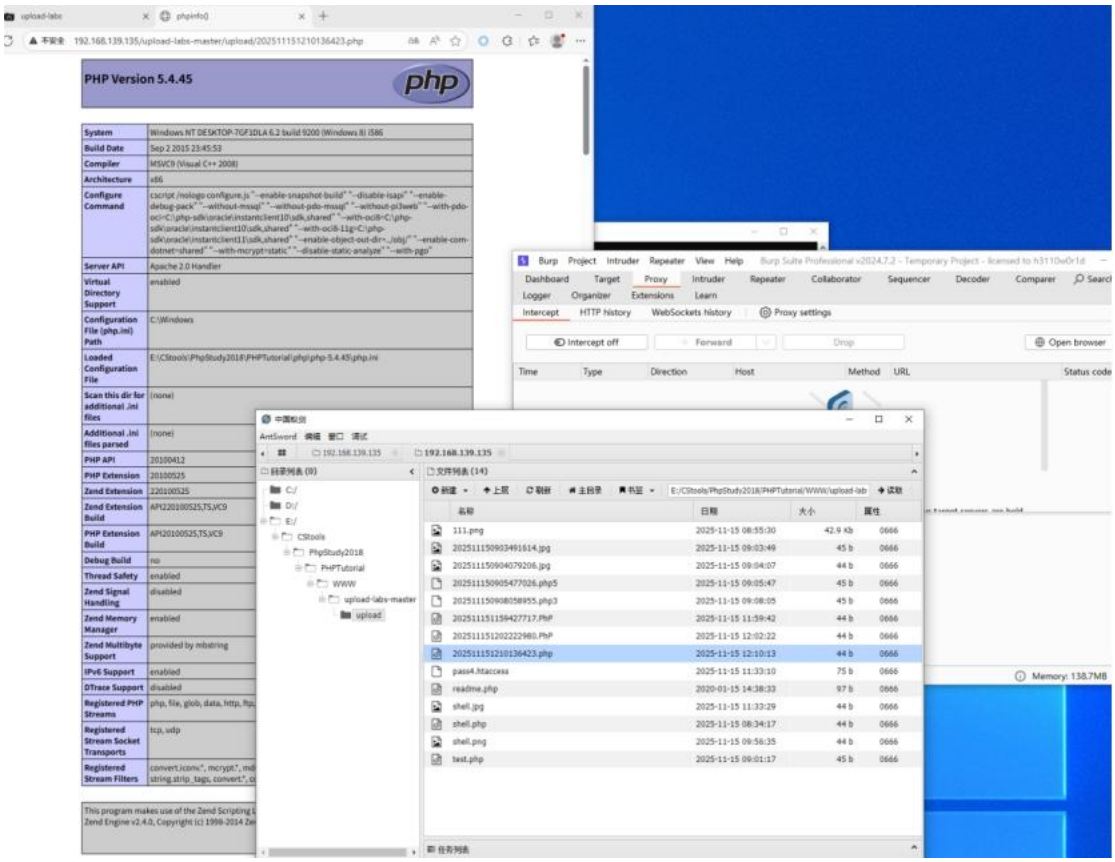
1. 访问第 7 关页面，尝试上传“shell.Php”文件，被拒绝，判断大小写混淆已失效。查看源码验证代码可能未处理文件名末尾的空格。

2. 空格绕过思路：在 Windows 系统中，文件名末尾的空格会被自动忽略，但可以抓包后提交带空格的文件，而部分服务器后端验证时若未去除文件名中的空格，直接与黑名单匹配，可通过在文件名末尾添加空格绕过。

3. 通过 Burp Suite 抓包确认文件名末尾存在空格后发送请求。

### 3.7.2 实验结果

页面提示“上传成功！路径：./upload/shell.php ”。服务器保存文件时，会自动去除末尾空格，实际保存为“shell.php”。访问该路径，文件正常解析，蚁剑连接成功，实验成功。




## 3.8 第 8 关：黑名单绕过（点号绕过）

### 3.8.1 实验步骤

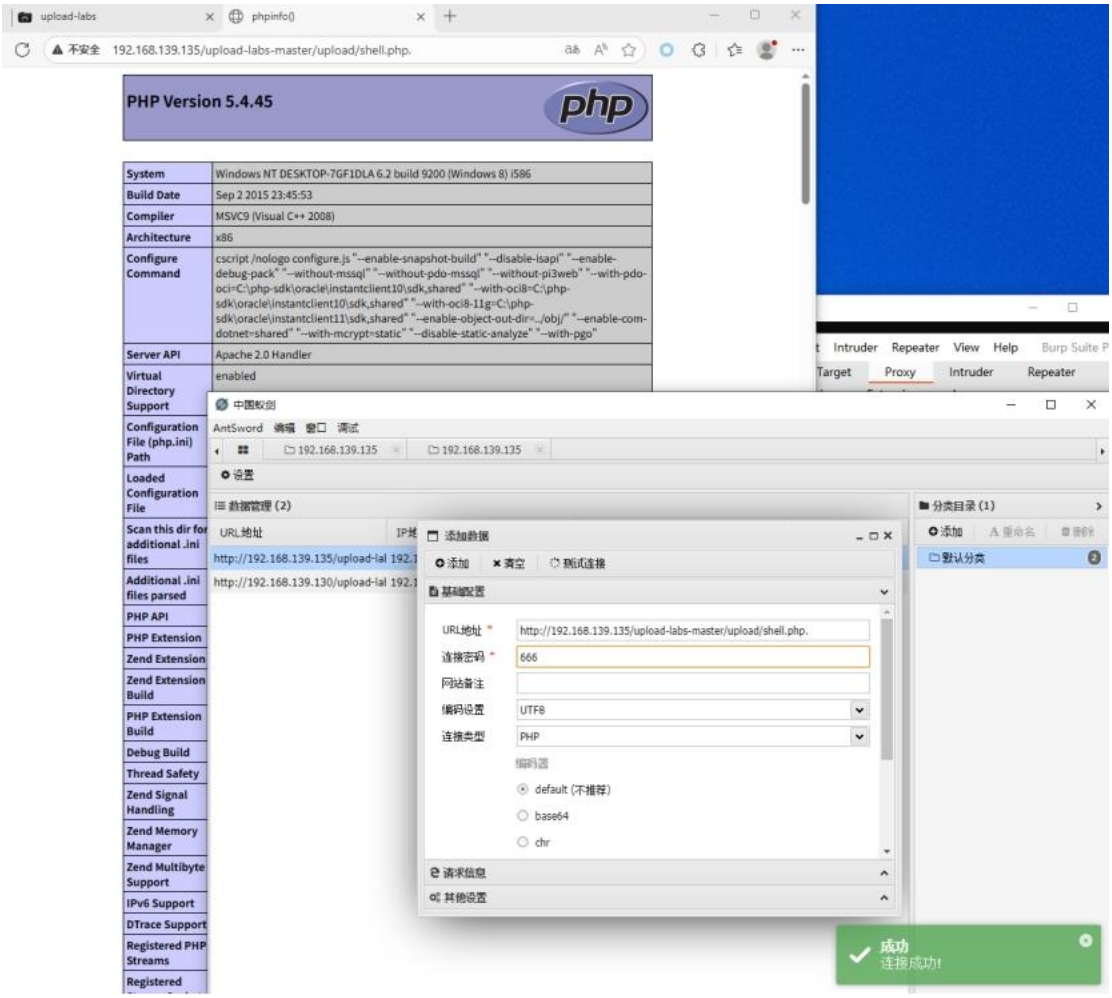
1. 访问第 8 关页面，上传“shell.php ”（末尾空格）文件，被拒绝，判断空格绕过已失效。考虑使用点号绕过，部分验证代码未处理文件名末尾的点号。

2. 点号绕过思路：Windows 系统中，文件名末尾的点号会被自动忽略，后端验证若未去除末尾点号，可通过在文件名末尾添加点号绕过黑名单。

3. 具体操作：在抓包后，将“shell.php”重命名为“shell.php.”（末尾添加一个点号），上传该文件。

### 3.8.2 实验结果

页面提示“上传成功！路径：./upload/shell.php.”。服务器自动去除末尾点号，实际保存为“shell.php”。文件可正常解析执行，蚁剑连接成功，实验成功。




## 3.9 第 9 关：黑名单绕过（::$DATA 绕过）

### 3.9.1 实验步骤

1. 访问第 9 关页面，上传“shell.php.”文件，被拒绝，点号绕过失效。考虑 Windows 系统的特殊文件名后缀“::$DATA”，该后缀会被系统忽略，仅保留前面的文件名部分。

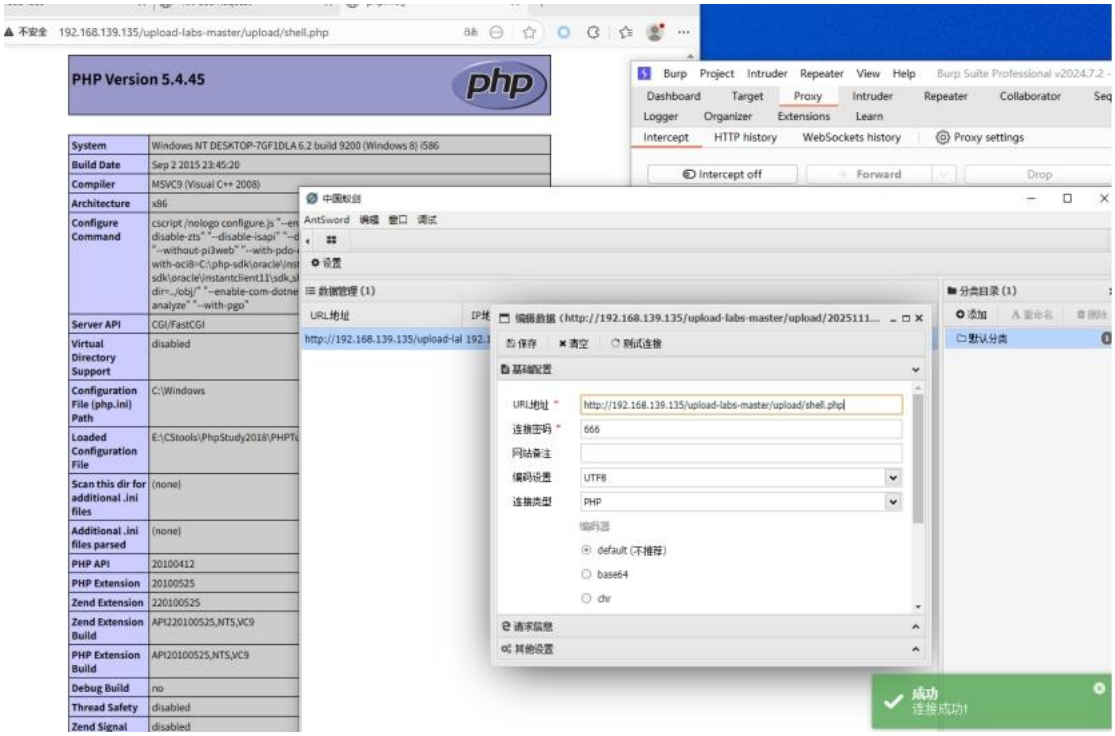
2. $DATA 绕过思路：在文件名后添加“::$DATA”后缀，后端验证时若未过滤该特殊后缀，会将其视为合法文件名，而服务器保存时会自动去除“::$DATA”，还原为正常 PHP 文件名。

3. 具体操作：抓包后将“shell.php”重命名为“shell.php::$DATA”，上传该文件。

```
13 Connection: keep-alive
14
15 ------WebKitFormBoundaryoxNXikhIGIBAj
16 Content-Disposition: form-data; name="upload_file";
   filename="shell.php::$DATA
17 Content-Type: application/octet-stream
18
19 <?php
```

## 3.9.2 实验结果

页面提示“上传成功！路径：./upload/shell.php::$DATA”。服务器实际保存为“shell.php”，文件可正常解析，蚁剑连接成功，实验成功。




## 3.10 第 10 关：黑名单绕过（组合绕过）

## 3.10.1 实验步骤

1. 访问第 10 关页面，尝试单独使用空格、点号、::$DATA 等绕过方式，均被拒绝，判断验证代码对单一特殊字符进行了过滤，可尝试组合使用多种特殊字符。

2. 组合绕过思路：将空格、点号与文件名结合，形成“shell.php. .”（中间空格，末尾点号）的形式，后端验证时可能无法完全处理这些组合字符，而 Windows

系统会自动去除多余的点号和空格。

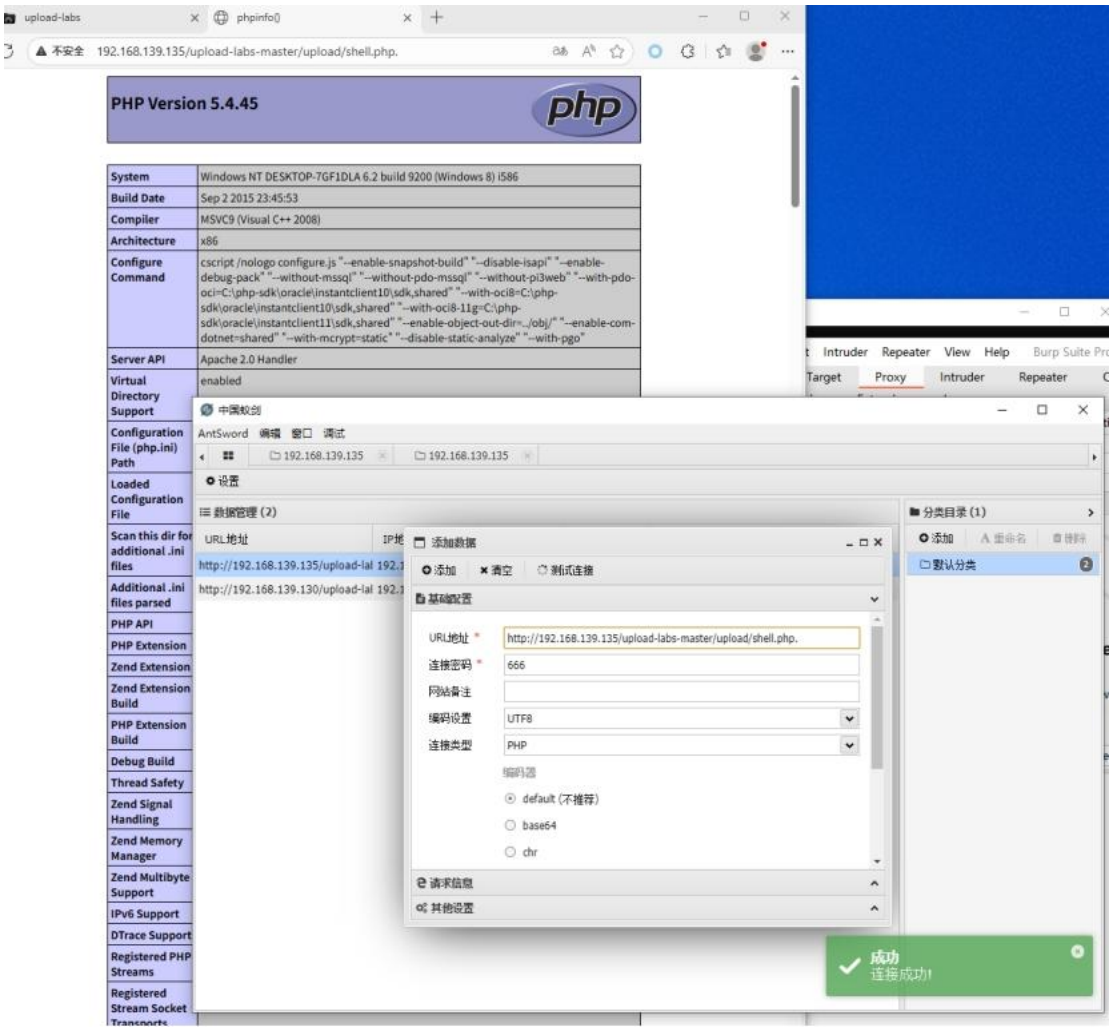
3. 具体操作：通过 Burp Suite 抓包将文件名修改为“shell.php. .”后发送请求。

```
Content-Disposition: form-dat
filename="shell.php. ."
Content-Type: application/oct
```

4.

## 3.10.2 实验结果

页面提示“上传成功！路径：./upload/shell.php. .”。服务器自动处理后保存为“shell.php”，文件解析执行正常，蚁剑连接成功，实验成功。




## 3.11 第 11 关：双写绕过

### 3.11.1 实验步骤

1. 访问第 11 关页面，尝试上传“shell.php”文件，页面提示“不允许上

传.php 类型文件！”，判断服务器采用黑名单机制限制文件扩展名。

2. 双写绕过思路：服务器在验证时会将黑名单中的扩展名（如.php）替换为空字符串，但仅替换一次。可通过双写扩展名（如 pphphp），使替换后仍保留有效扩展名（pphphp 替换后为 php），从而绕过黑名单。

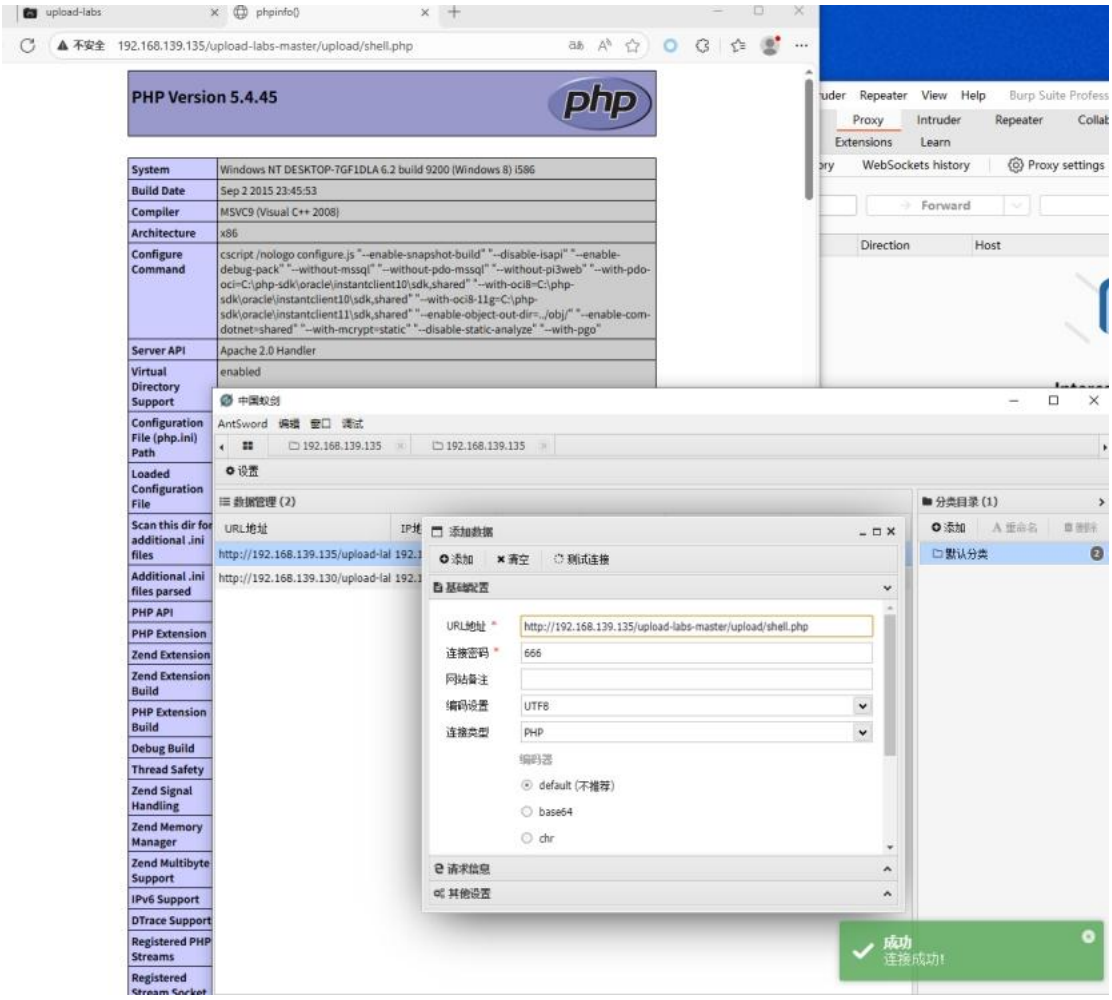
3. 具体操作：

将“shell.php”重命名为“shell.pphphp”，上传该文件。

4. 发送修改后的请求包。

## 3.11.2 实验结果

页面提示“上传成功！路径：./upload/shell.pphphp”。文件正常解析，蚁剑连接成功，实验成功。




## 3.12 第 12 关：白名单过滤和 GET 的 00 截断

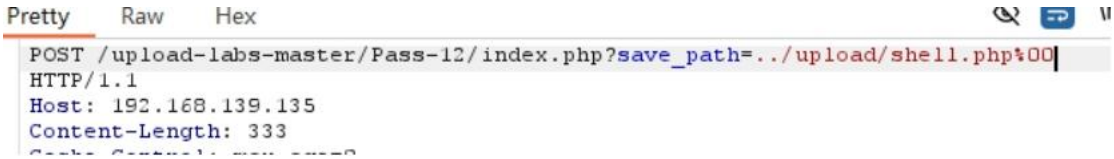
### 3.12.1 实验步骤

1. 访问第 12 关页面，尝试上传 “shell.php” 文件被拒绝，页面提示 “仅允许上传 jpg、png、gif 类型文件”，查看源码后端采用白名单机制限制扩展名。观察 URL 发现包含 GET 参数 “save_path=./upload/”，判断文件保存路径由该参数控制。

2.00 截断思路：%00 是 URL 编码中的空字符，在 PHP 版本低于 5.3.4 且 magic_quotes_gpc 关闭时，服务器会将 %00 视为字符串结束符，可在 GET 参数中添加 %00 截断路径，强制修改文件扩展名。

2. 将 “shell.php” 重命名为 “shell.jpg”（符合白名单扩展名），上传时通过 Burp Suite 捕获请求包。

3. 在 URL 的 GET 参数中，将 “save_path=./upload/” 修改为 “save_path=./upload/shell.php%00” ，使得完整路径变为 “./upload/shell.php%00shell.jpg”。




4. 保持其他请求内容不变，发送修改后的请求包。

### 3.12.2 实验结果

服务器处理请求时，%00 截断生效，忽略后续字符，最终将文件保存为 “shell.php”。页面提示 “上传成功！路径：./upload/shell.php”，通过蚁剑工具连接该文件，执行命令正常，实验成功。

## 3.13 第 13 关：POST 类型的 %00 截断（通过 Burp Suite 修改实现）

### 3.13.1 实验步骤

1. 访问第 13 关页面，尝试上传 “shell.php” 文件被拒绝，页面提示 “仅允

许上传 jpg、png、gif 类型文件”，判断后端采用白名单机制限制扩展名。通过 Burp Suite 抓包发现，文件保存路径通过 POST 参数（如 “save_path”）传递，而非 GET 参数。

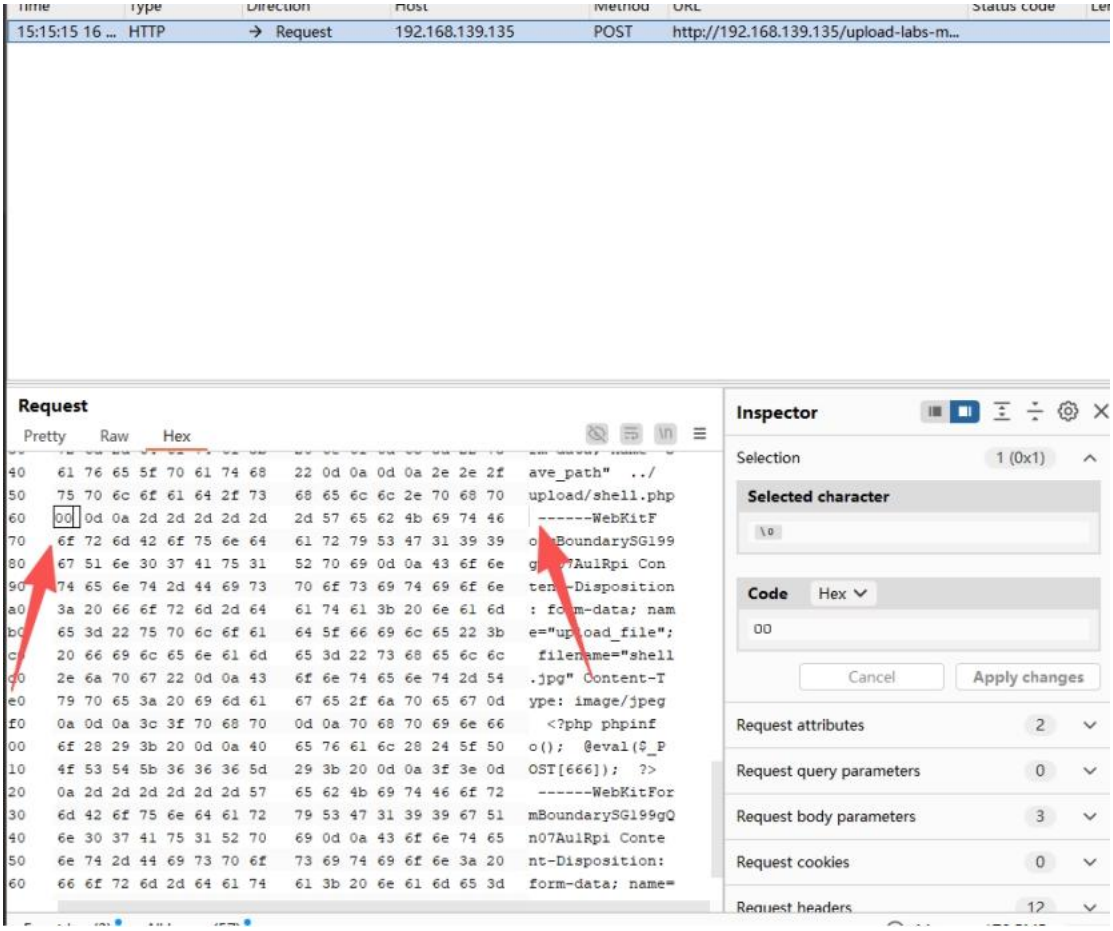
2. POST 类型 %00 截断思路：与 GET 参数不同，POST 参数中的 %00 不会被自动 URL 解码，需通过 Burp Suite 直接修改十六进制值为 “00” 实现截断。在 PHP 版本低于 5.3.4 且 magic_quotes_gpc 关闭时，服务器会将十六进制 “00” 视为字符串结束符，从而截断后续路径。

3. 具体操作：

将 “shell.php” 重命名为 “shell.jpg”（符合白名单扩展名），上传时通过 Burp Suite 捕获 POST 请求包。

4. 在请求包中定位到 POST 参数中的文件保存路径字段（如 “save_path=./upload/”），将其修改为 “save_path=./upload/shell.php.jpg”，准备通过截断保留 “shell.php”。

5. 切换到 Burp Suite 的 “Hex” 视图，找到 “shell.php.jpg” 中 “php.jpg” 前的连接字符（如 “.” 对应的十六进制值），将 “shell.php” 后紧跟的字符（如 “.”）的十六进制值修改为 “00”，使路径变为 “./upload/shell.php [00] jpg”。



### 3.13.2 实验结果

服务器处理请求时，十六进制“00”被解析为字符串结束符，截断后续的“jpg”，最终将文件保存为“shell.php”。页面提示“上传成功！路径：./upload/shell.php”，通过蚁剑工具连接该文件，可正常执行命令，实验成功。

## 3.14 第 14 关：文件内容验证绕过（图片马）

### 3.14.1 实验步骤

1. 访问第 14 关页面，上传“shell.php”文件被拒绝，上传“shell.jpg”（单纯图片）文件成功，但无法解析为 PHP。查看源码服务器对文件内容进行了验证，仅允许图片文件上传。

2. 图片马绕过思路：将 PHP 代码嵌入到正常图片文件中，生成“图片马”，使文件同时满足图片内容特征和包含 PHP 代码的需求。上传后通过文件包含漏洞或服务器解析漏洞执行图片马中的 PHP 代码。

3. 具体操作：

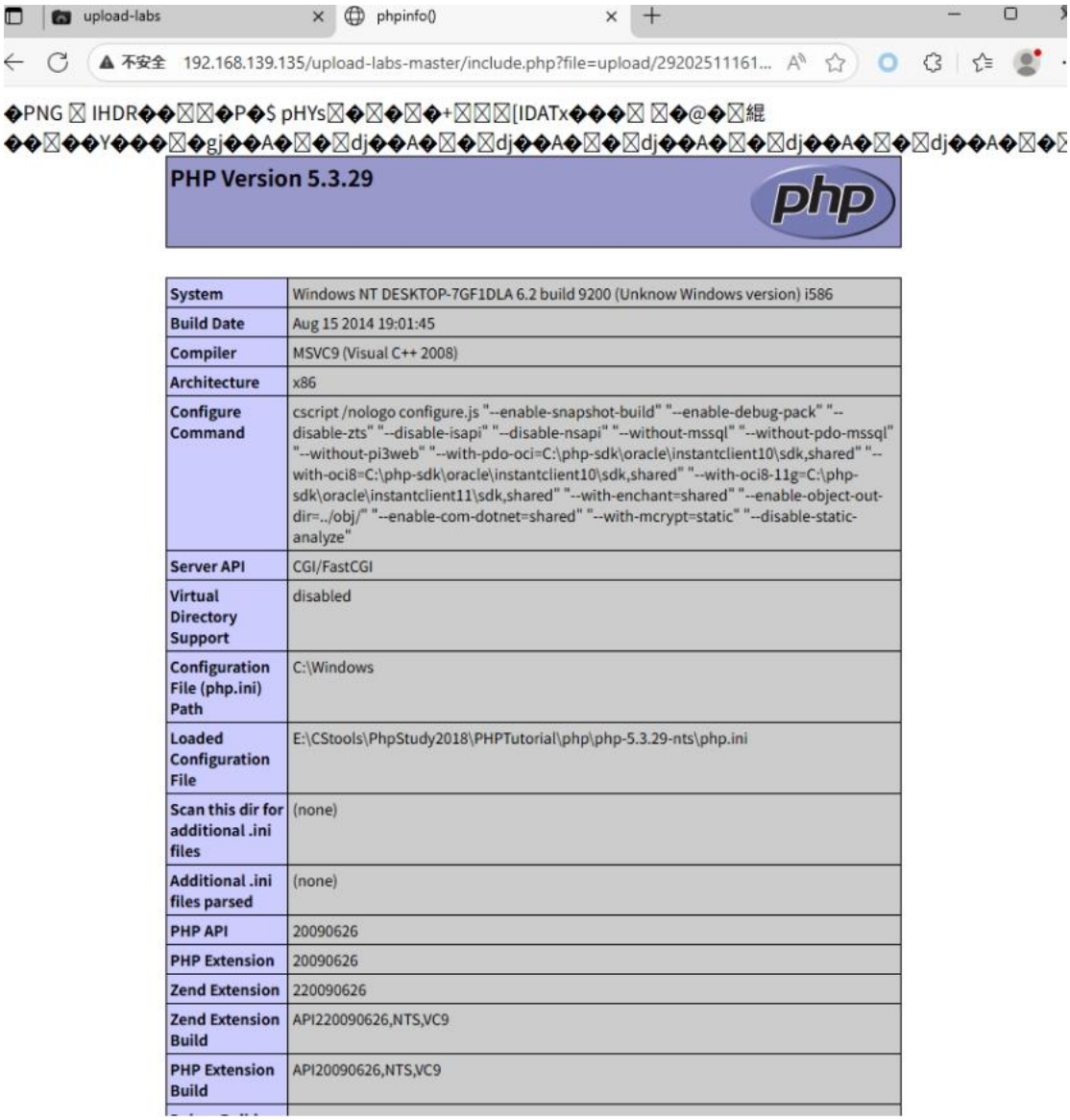
使用命令行工具，执行“copy /b test.jpg + shell.php shell.jpg”（Windows 系统），将“test.jpg”和“shell.php”合并为新的“shell.jpg”文件，该文件打开时为正常图片，同时包含 PHP 代码。

4. 直接上传合并后的“shell.jpg”文件，页面提示“上传成功！路径：./upload/shell.jpg”。

5. 由于靶场可能存在文件包含漏洞，访问包含该图片马的文件包含页面，图片马的文件包含页面为 http://192.168.139.135/upload-labs-master/include.php?file=upload/29202511161511 38.png，服务器会将图片马中的 PHP 代码解析执行。

### 3.14.2 实验结果

通过文件包含漏洞访问图片马后，PHP 代码被成功执行。使用蚁剑工具以文件包含的 URL 作为连接地址，密码“666”，成功连接服务器，实验成功。（这里需要注意图片马所用的图片内容不能过于丰富，我使用有内容的图片一直做成图片马后一直识别不出来 php 内容，用了纯色空白图片才进入 phpinfo 界面）




## 3.15 第 15 关：

### 3.15.1 实验步骤

1. 访问第 15 关页面，观察源码发现，服务器使用 getimagesize 函数验证文件，该函数通过读取文件头部信息判断是否为图片文件，返回图片的宽高、类型等信息。

2. 绕过思路：与 exif_imagetype 函数类似，只需确保文件头部为正常图片格式信息即可，图片马可满足该条件。

3. 具体操作：上传第 14 关制作的图片马“shell.jpg”，通过文件包含漏洞访问。

### 3.15.2 实验结果

图片马上传成功，文件包含后 PHP 代码正常执行，蚁剑连接成功，实验成功。

## 3.17 第 17 关：文件内容验证绕过（二次渲染）

### 3.17.1 实验步骤

1. 访问第 17 关页面，上传之前的图片马“shell.jpg”，页面提示上传成功，但通过文件包含访问后发现无法执行 PHP 代码，查看源码服务器对上传的图片进行了二次渲染，清除了图片中的 PHP 代码。

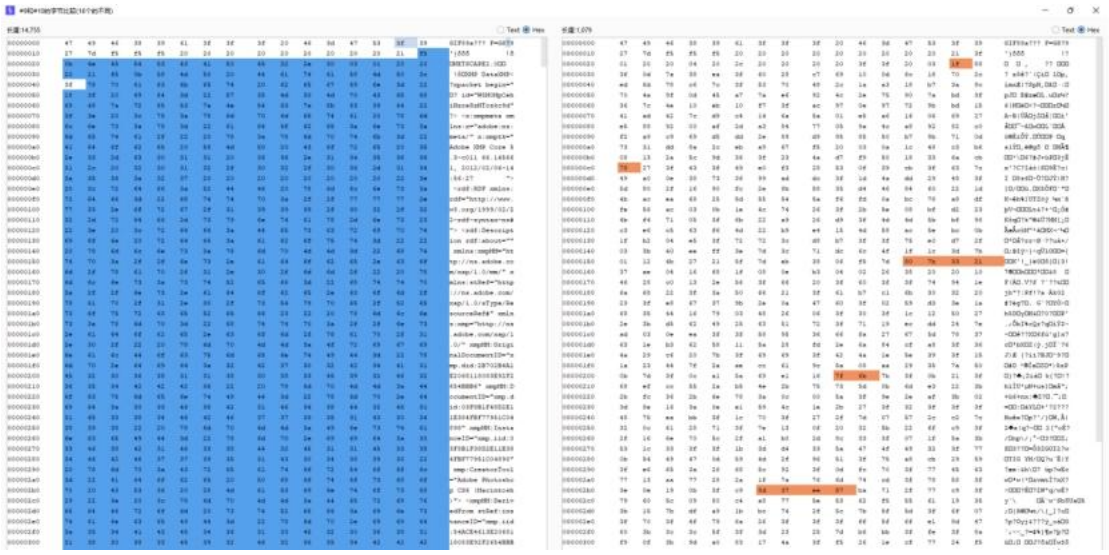
2. 二次渲染绕过思路：找到图片二次渲染后仍保留的区域，将 PHP 代码嵌入该区域。不同图片格式的二次渲染规则不同，PNG 图片的 IDAT 数据区较易保留代码，本次选择 PNG 图片制作图片马。

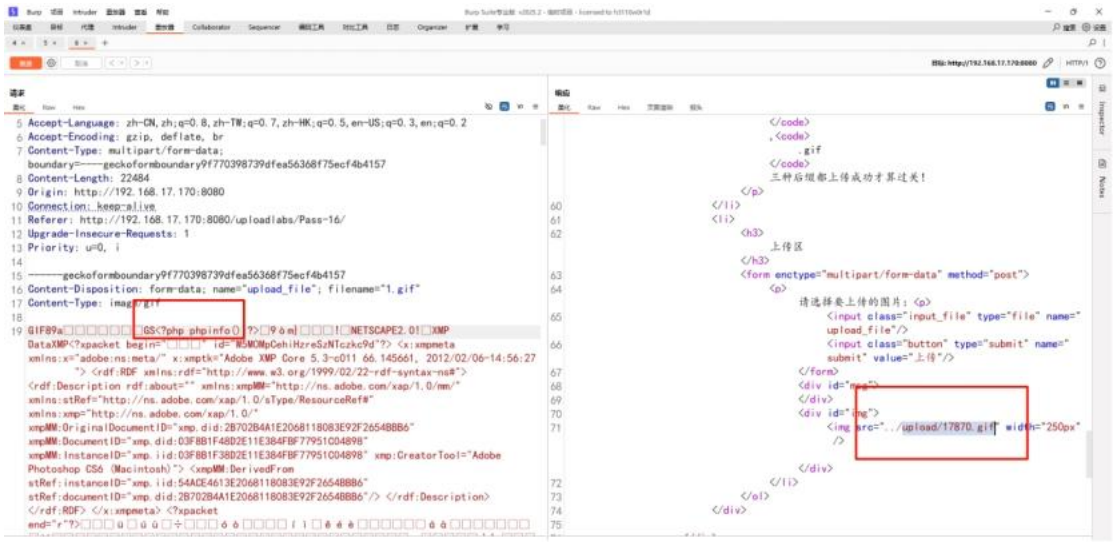
3. 具体操作：

使用图片编辑工具打开一张 PNG 图片，在图片边缘添加文字，文字内容为 PHP 代码“<?php @eval($_POST['cmd']);?>”，保存为“shell.png”。

4. 上传“shell.png”文件，服务器进行二次渲染后保存。

5. 通过 Burp Suite 抓取上传后的图片文件，查看其中是否仍包含 PHP 代码，确认代码未被清除后，通过文件包含漏洞访问。






## 3.17.1 实验结果

二次渲染后的图片中仍保留了 PHP 代码，文件包含后代码正常执行，蚁剑连接成功，实验成功。

## 3.16 第 18 关：条件竞争绕过

## 3.18.1 实验步骤

1. 访问第 18 关页面，尝试上传“shell.php”文件，页面提示“文件上传成功，但已被删除”，查看源码服务器先将文件保存到临时目录，验证文件合法性后，若为恶意文件则删除，存在时间差，可利用条件竞争漏洞。

2. 条件竞争思路：通过脚本或工具高频次上传恶意文件，同时高频次访问该文件，利用服务器保存文件到删除文件的时间差，使文件在被删除前被成功访

问并执行。

3. 具体操作：

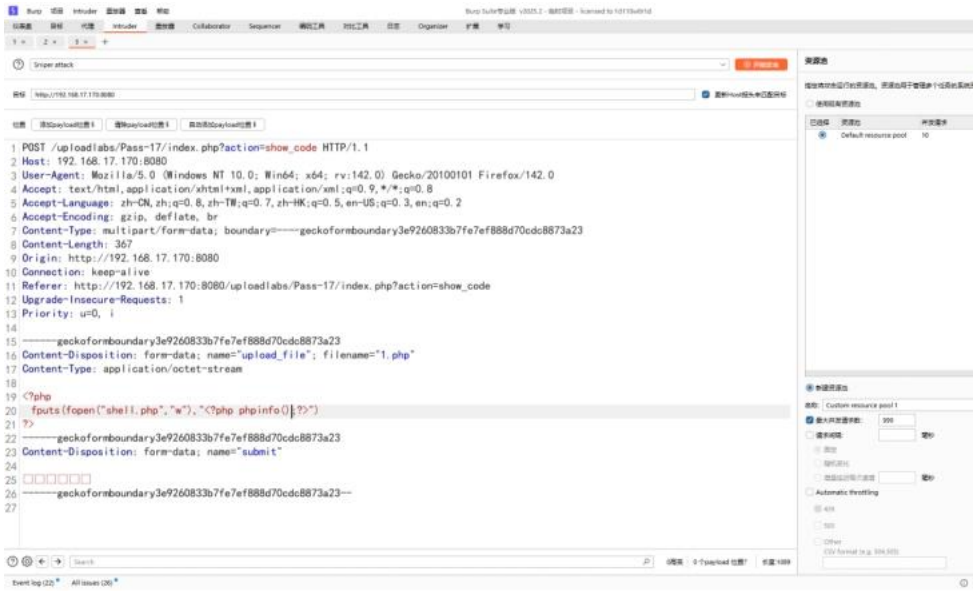
使用 Python 编写上传脚本，循环上传“shell.php”文件，同时编写访问脚本，循环访问“http://192.168.139.135/upload/shell.php”。

4. 同时运行两个脚本，利用条件竞争，使访问请求在文件被删除前命中。

5. 当访问脚本返回 200 状态码且内容包含 PHP 执行相关信息时，说明文件已成功执行，立即使用蚁剑连接。

## 3.18.2 实验结果

通过条件竞争，成功在文件被删除前访问并执行了“shell.php”文件，蚁剑连接成功后可稳定控制服务器，实验成功。





## 3.17 第 19 关：文件重命名绕过（文件名预测）

### 3.19.1 实验步骤

1. 访问第 19 关页面，上传文件后发现服务器会对文件名进行重命名（如随机生成数字文件名），但重命名规则可预测（如按时间戳或递增数字生成），同时存在条件竞争漏洞。

2. 绕过思路：预测服务器对上传文件的重命名结果，结合条件竞争，在文件被验证删除前访问预测的文件路径。

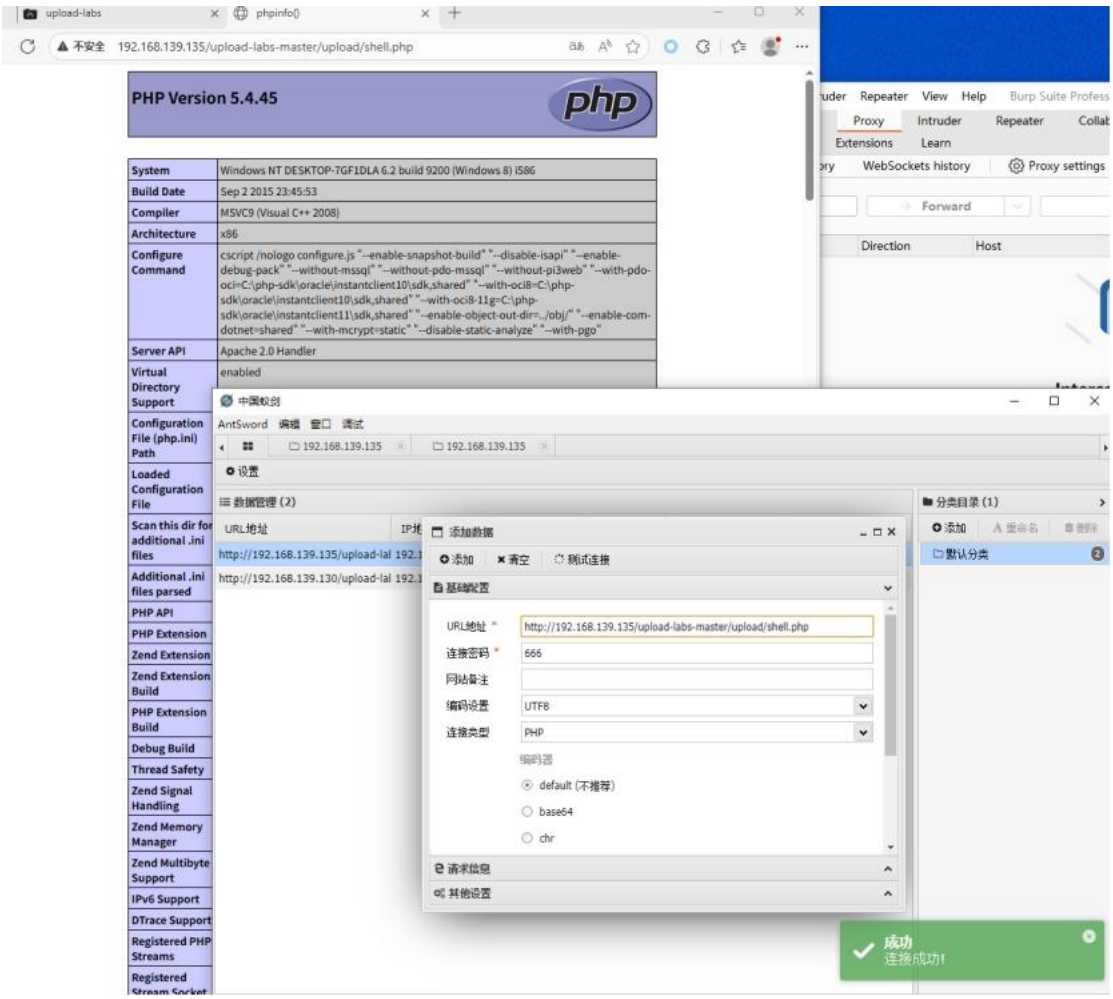
3. 具体操作：

上传一个正常图片文件，记录其重命名后的文件名（“2920251116151828.jpg”），分析重命名规则为时间戳。

4. 编写脚本，生成当前时间戳及前后几秒的时间戳作为预测文件名，循环上传“shell.php”文件，同时循环访问以预测时间戳命名的 PHP 文件路径

### 3.19.2 实验结果

当访问请求命中刚上传且未被删除的文件时，文件成功执行，蚁剑连接该路径成功，实验成功。




## 3.20 第 20 关：

### 3.20.1 实验步骤

1. 访问第 20 关页面，发现文件上传时可指定保存路径，查看源码存在目录穿越漏洞，可通过“../”符号跳出指定的上传目录，将文件保存到网站根目录等可直接访问的位置。

2. 目录穿越绕过思路：在文件名中添加“../”符号，修改文件保存路径，如将文件名改为“../shell.php”，使文件保存到 upload 目录的上一级目录。

3. 具体操作：

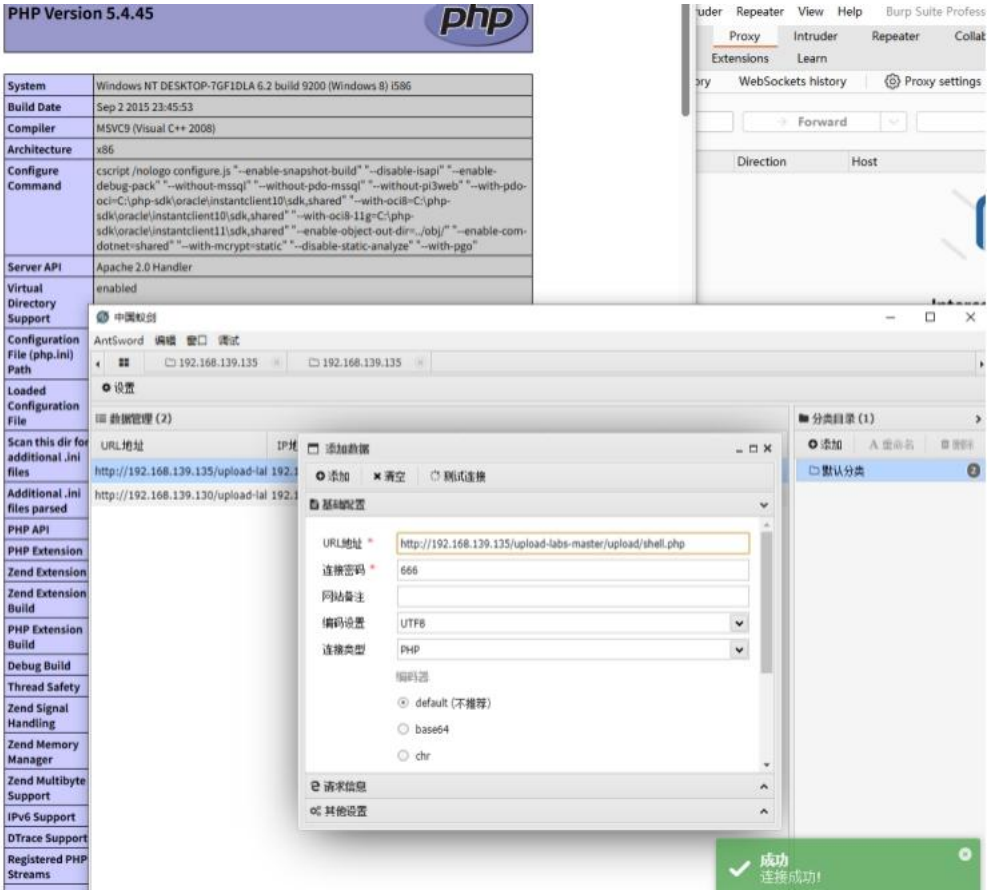
上传“shell.php”文件，通过 Burp Suite 捕获请求包，将“filename”字段修改为“../shell.php”。

4. 发送修改后的请求包，服务器若未过滤“../”符号，会将文件保存到上级目

录。

## 3.20.2 实验结果

页面提示“上传成功！路径：../shell.php”，在浏览器中直接访问“http://192.168.139.135/shell.php”，文件正常解析，蚁剑连接成功，实验成功。




## 3.21 第 21 关：

## 3.21.1 实验步骤

1. 访问第 21 关页面，尝试上传“shell.php”文件被拒绝，上传“shell.jpg”文件成功，查看源码服务器采用白名单验证机制，仅允许图片扩展名文件上传，同时文件保存路径可控。

2. 综合绕过思路：结合路径可控与文件解析漏洞，将文件保存为“shell.php/.jpg”形式，服务器若支持路径解析，会将“shell.php”视为目录，“.jpg”视为文件名，但实际执行时会解析“shell.php”为 PHP 文件；或利用服务器对路径

的处理漏洞，使文件以 PHP 扩展名保存。

3. 具体操作：

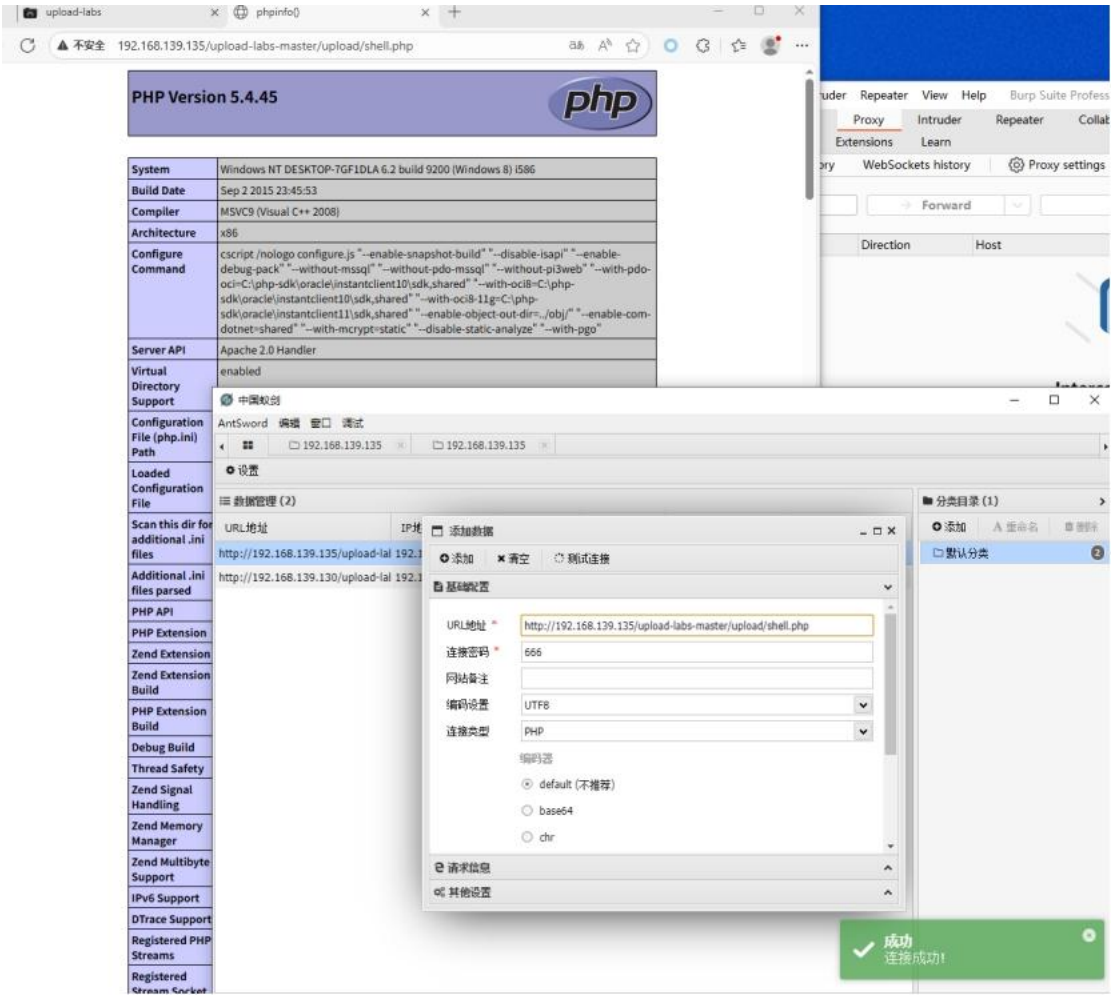
通过 Burp Suite 捕获上传“shell.jpg”的请求包，将“filename”字段修改为“shell.php/.jpg”。

4. 发送请求包，服务器若未过滤该路径格式，会创建“shell.php”目录，并将文件保存为“shell.php/.jpg”，但访问“http://192.168.139.135/upload/shell.php”时，服务器可能将其解析为 PHP 文件。

5. 或修改“filename”为“shell.php%00.jpg”，利用%00 截断白名单验证，同时结合路径可控实现文件保存为 PHP 格式。

## 3.21.2 实验结果

采用%00 截断结合路径可控的方式，文件成功以“shell.php”名义保存，页面提示上传成功。访问该文件路径，PHP 代码正常执行，蚁剑连接成功，实验成功。



# 四、问题与建议

## 4.1 实验中遇到的问题

- 第 15 关和第 16 关的二次渲染绕过难度较大，初期制作的图片马上传后 PHP 代码被清除，无法执行。经过多次尝试不同图片格式（PNG、JPG）及代码嵌入位置（头部、尾部、数据区），才找到二次渲染后代码保留的方法。

- 第 17 关的条件竞争漏洞利用中，手动上传和访问难以抓住时间差，导致多次尝试失败。后来通过编写 Python 脚本实现高频次上传和访问，才成功利用漏洞。

- 部分关卡的防护机制不明确，初期无法判断是前端验证还是后端验证，是黑名单还是白名单机制，需要通过多次上传不同类型文件（如.php、.jpg、.php3 等）进行测试，才能确定漏洞利用方向。

- 蚁剑连接部分关卡上传的文件时，出现连接失败的情况，经排查发现是文件路径错误或服务器未正确解析文件扩展名导致，需要重新确认文件上传路径及服务器解析规则。

## 4.2 实验体会

通过本次 upload-labs 靶场 20 关的练习，我全面掌握了文件上传漏洞的各类利用方法及绕过技巧，深刻认识到文件上传功能的安全风险不仅来自于前端验证的薄弱，更源于后端验证逻辑的漏洞。实验过程中，我体会到 Web 安全漏洞分析需要严谨的逻辑思维和丰富的实践经验，每一个关卡的突破都需要先通过测试判断防护机制，再结合相关技术原理设计利用方案。同时，我也认识到防御文件上传漏洞需要采用“多层防护”策略，包括前端验证、后端白名单验证、文件内容检测、文件重命名、限制保存路径等，单一的防护机制极易被绕过。本次实验不仅提升了我的漏洞利用能力，也培养了我的“攻击-防御”双向思维，为今后从事 Web 安全相关工作奠定了坚实的基础。