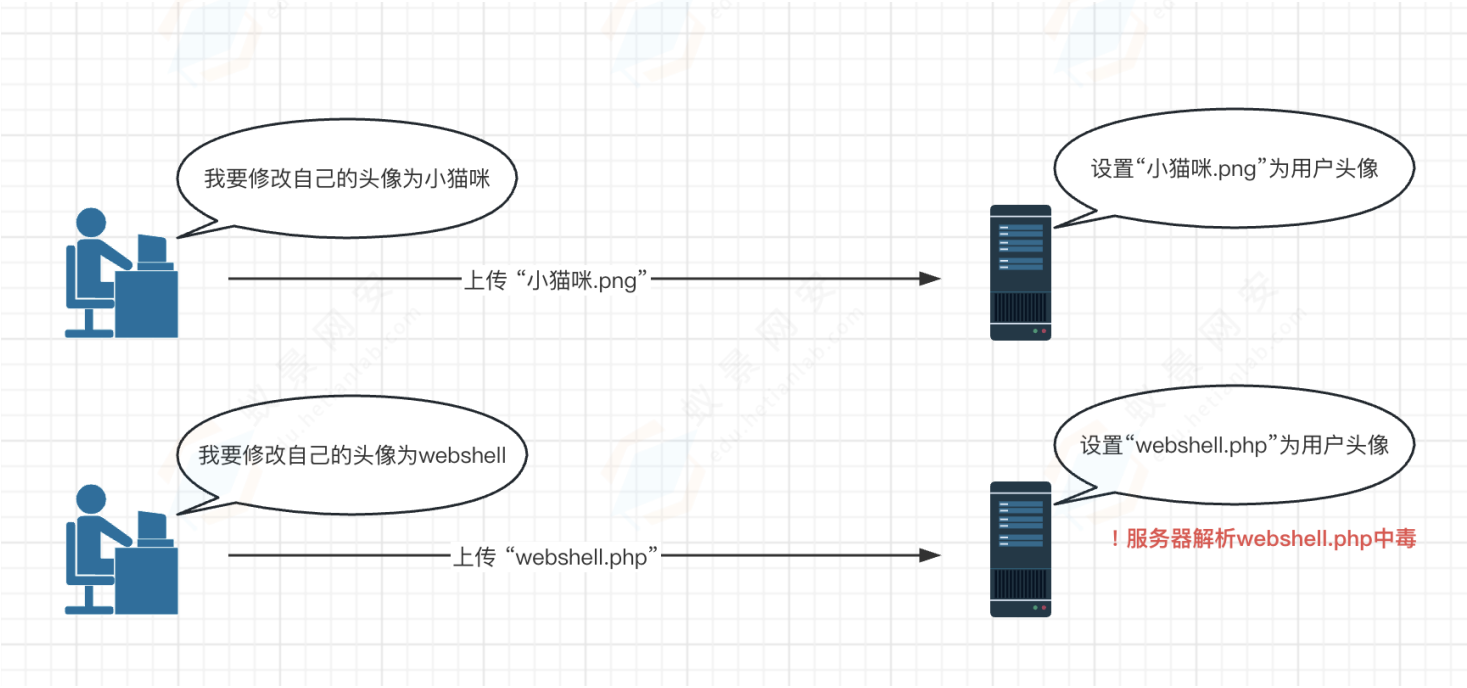
一、文件上传漏洞简介

- 上传文件的时候，如果服务器端后端语言未对上传的文件进行严格的验证和过滤，就容易造成上传任意文件的情况。常见场景是web服务器允许用户上传图片或者普通文本文件保存，而攻击者绕过上传机制上传恶意代码文件并执行从而控制服务器。



1. 文件上传漏洞常见攻击手法

(1) 上传webshell

```
(isset($_GET['cmd']))
{
    system($_GET['cmd'] . ' 2&<1'
}
```

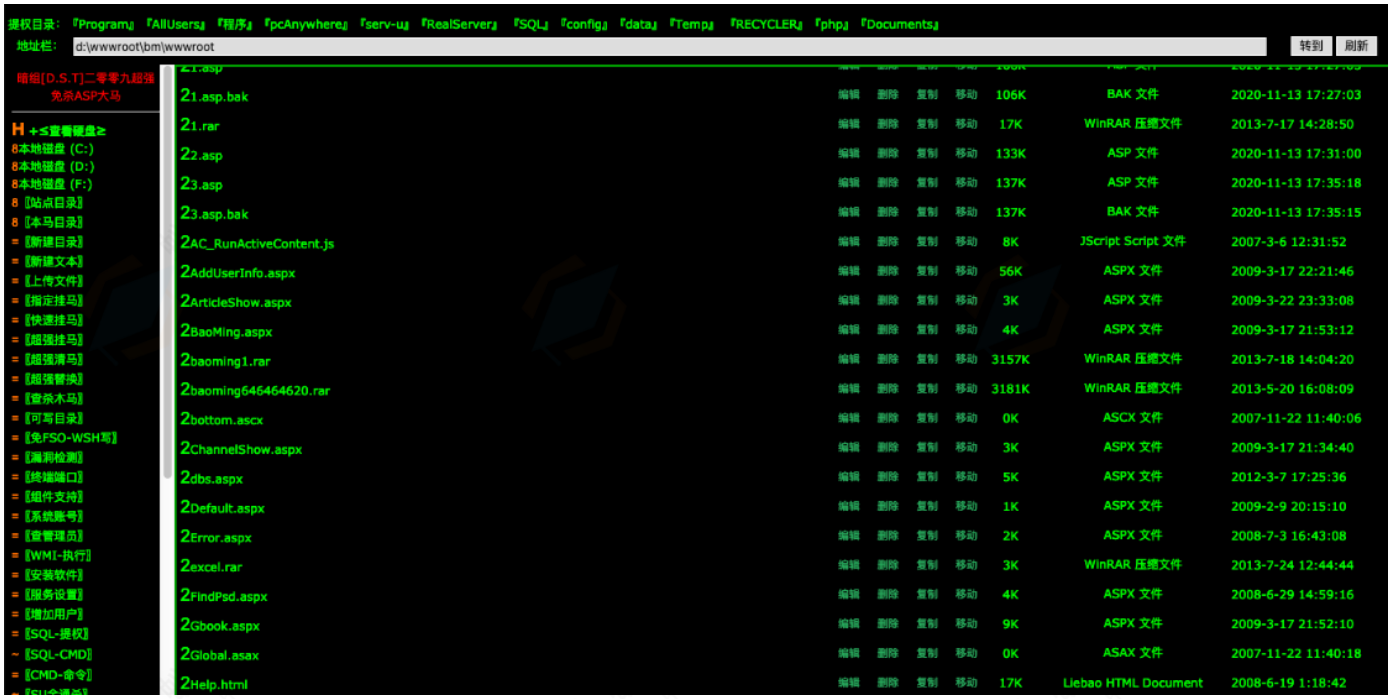
(2) 上传其他可能造成威胁的文件

- 上传HTML，用作钓鱼
- 上传大容量文件，消耗磁盘与流量

2. webshell

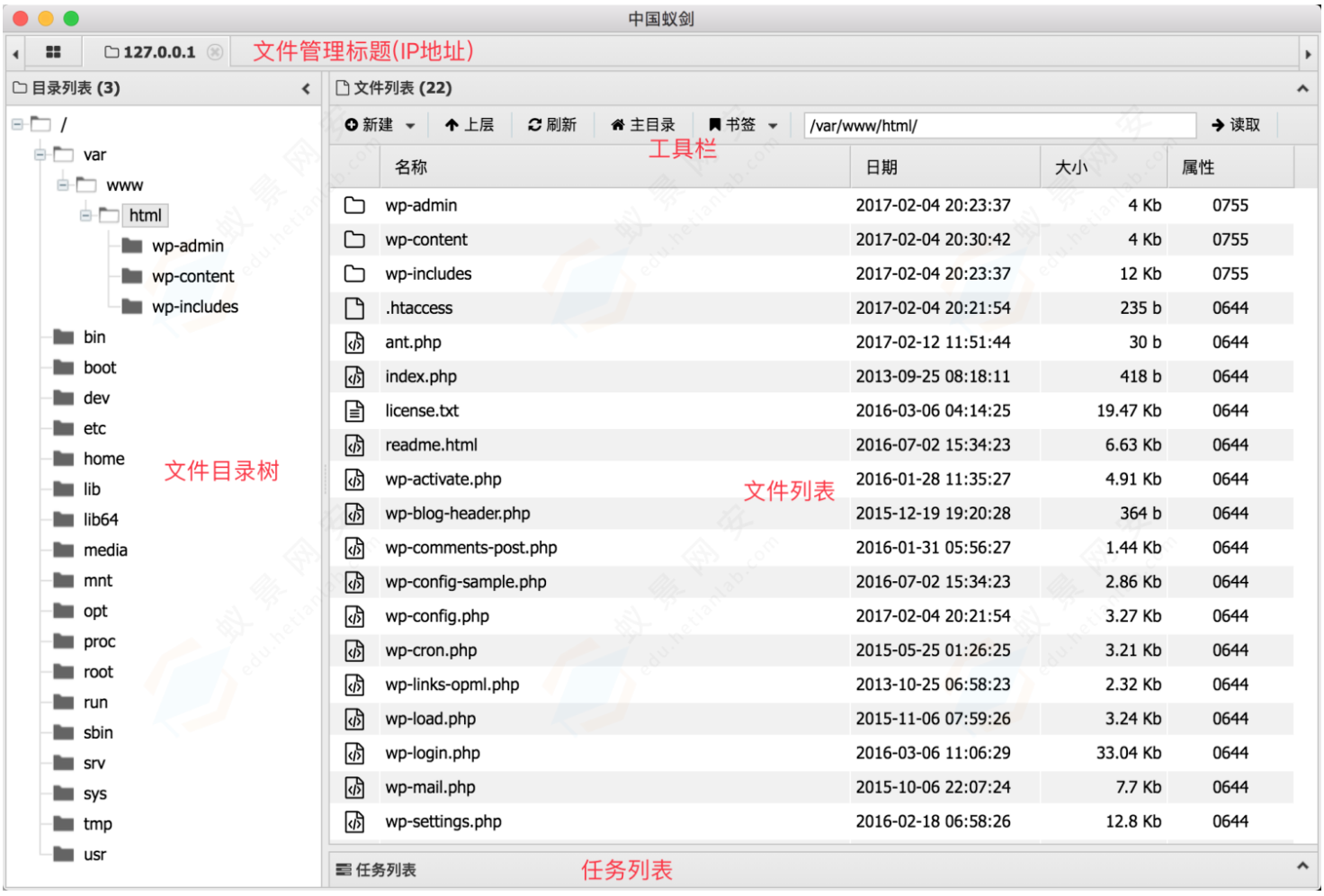
(1) 什么是webshell

- 网站的后门，可以通过webshell控制网站服务器
- webshell
  - 大马
  - 故名思议，占用空间大，代码量多



- 小马、一句话木马
- 只有一行或几行代码构成，需要借助webshell管理器使用

```
<?php @eval($_POST['CMD']?>
```



- webshell管理器
  - 蚁剑
  - https://github.com/AntSwordProject/antSword
  - 冰蝎
  - https://github.com/rebeyond/Behinder

- 哥斯拉
- https://github.com/BeichenDream/Godzilla

> 后面有课程专门分析webshell与常见管理器流量，做到研判

3. 文件上传漏洞防御

(1) 代码角度

- 黑名单
  - 提取上传文件后缀名，不允许像 php、jsp、asp、htaccess后缀的文件上传

```
// 获取上传文件的后缀名
$filename = $_FILES['file']['name'];
$ext = pathinfo($filename, PATHINFO_EXTENSION);

// 检查是否允许该类型的文件上传
 (!in_array($ext, $allowed) || preg_match("/(php|jsp|asp|htaccess)$/i", $filename)) {
    echo '上传文件类型不被允许！';
} else {
    // 文件上传逻辑
}
```

- 白名单
  - 提取上传文件后缀名，根据业务需求，例如只允许png、jpg、jpeg、gif后缀的文件上传

```
$allowed = array('png', 'jpg', 'jpeg', 'gif'

// 获取上传文件的后缀名
$filename = $_FILES['file']['name'];
$ext = pathinfo($filename, PATHINFO_EXTENSION);
$ext = strtolower($ext

// 检查是否为允许上传的文件类型
 (!in_array($ext, $allowed)) {
    echo '只允许上传 png、jpg、jpeg、gif 格式的文件';
} else {
    // 文件上传逻辑
}
```

- 内容检测

> 黑名单、白名单的检测机制可能会被黑客利用中间件漏洞、其他web漏洞绕过限制，所以内容检测是一种更精准的黑名单防御方法

- 如果是php网站，不论黑客如何变换php木马，由于php必须要用规定的标签包裹，所以检测用户传入的文件内容中是否含有php标签，即可做到文件上传防御

```
    $content = file_get_contents($_FILES['file']['tmp_name'
    // 匹配PHP开始标记 <?php、<?、<%、<?=、<script
     (preg_match('/(<\?(\s|\w)|<%|<\?=|<script)/i', $content)) {
        echo '上传文件存在潜在威胁！';
    } else {
      // 文件上传逻辑
    }
```

(2) 服务器或配置角度

- 慎用高权限系统用户
  - Windows Server 不要以管理员身份执行 apache 、nginx等服务
  - Linux 用户权限

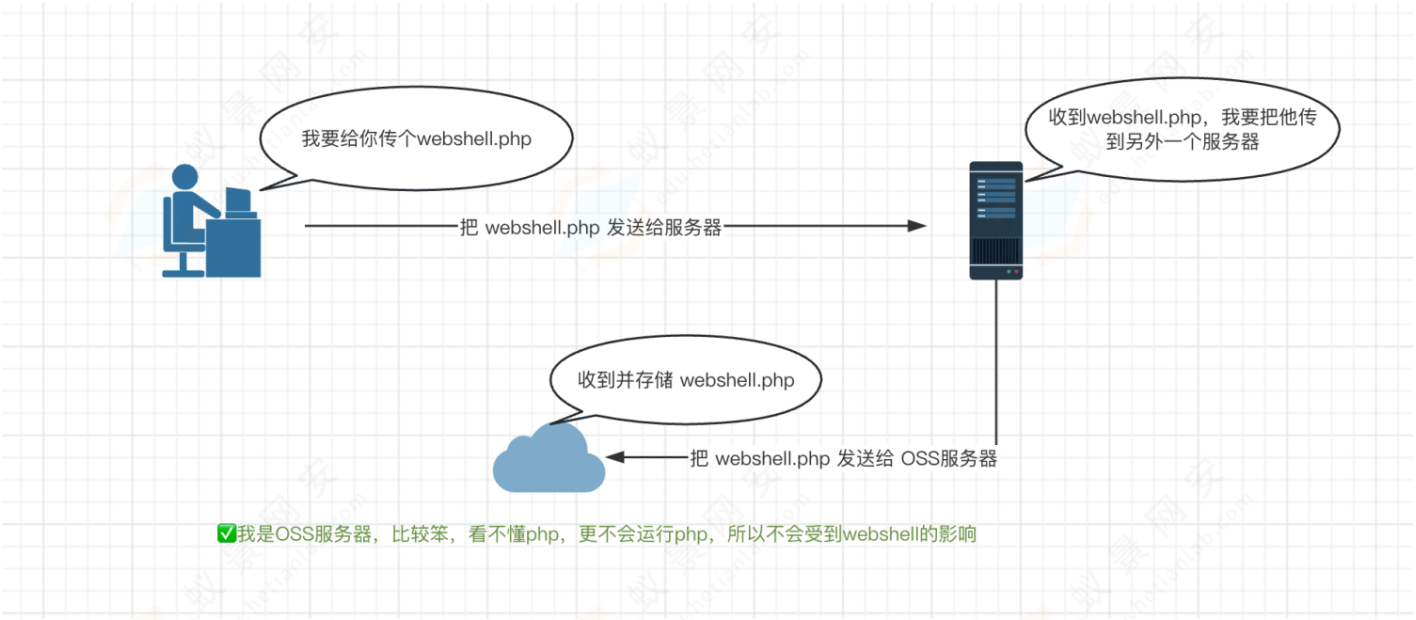
1. 使用以下命令获取 Nginx 运行进程的用户和组：ps -aux | grep nginx

2. 查找具有最高优先级的nginx进程，通常是主/父进程：

```
root      5703  0.0  0.7 107652  7664 ?        Ss   Sep01   0:00 nginx: master process /usr/sbin/nginx -g daemon on; mas
ter_process on;
www-data  6175  0.0  0.6 107900  6320 ?        S    Sep01   0:00 nginx: worker process
www-data  6176  0.0  0.6 107900  6320 ?        S    Sep01   0:00 nginx: worker process
```

在上述输出中，www-data 用户属于最底部的两个 Nginx 子进程，如果出现其他用户，则正在使用该用户运行Nginx Web服务器，而不是www-data

- 静态资源分离网站服务器
  - OSS 对象存储服务
  - https://www.alibabacloud.com/zh/product/object-storage-service
  - https://aws.amazon.com/cn/what-is/object-storage/
  - Nginx静态资源服务器
  - https://tsejx.github.io/devops-guidebook/server/nginx/static-resource-server



4. WAF是怎么做的

> - **https://github.com/loveshell/ngx_lua_waf**

沿用上节课的开源WAF，浏览post.rule可以看出，waf做的也是黑名单形式的内容检测，检测在post请求体中是否存在(?:define|eval|file_get_contents|include|require|require_once|shell_exec|phpinfo|system|passthru|preg_\w+|execute|echo|print|print_r|var_dump|(fp)open|alert|showmodaldialog)(xwork.MethodAccessor 这些字符。
这些字符包含了常见的php代码执行与命令执行函数，在一定程度上可以防御黑客上传webshell，但是可能不全面，且存在绕过的风险。