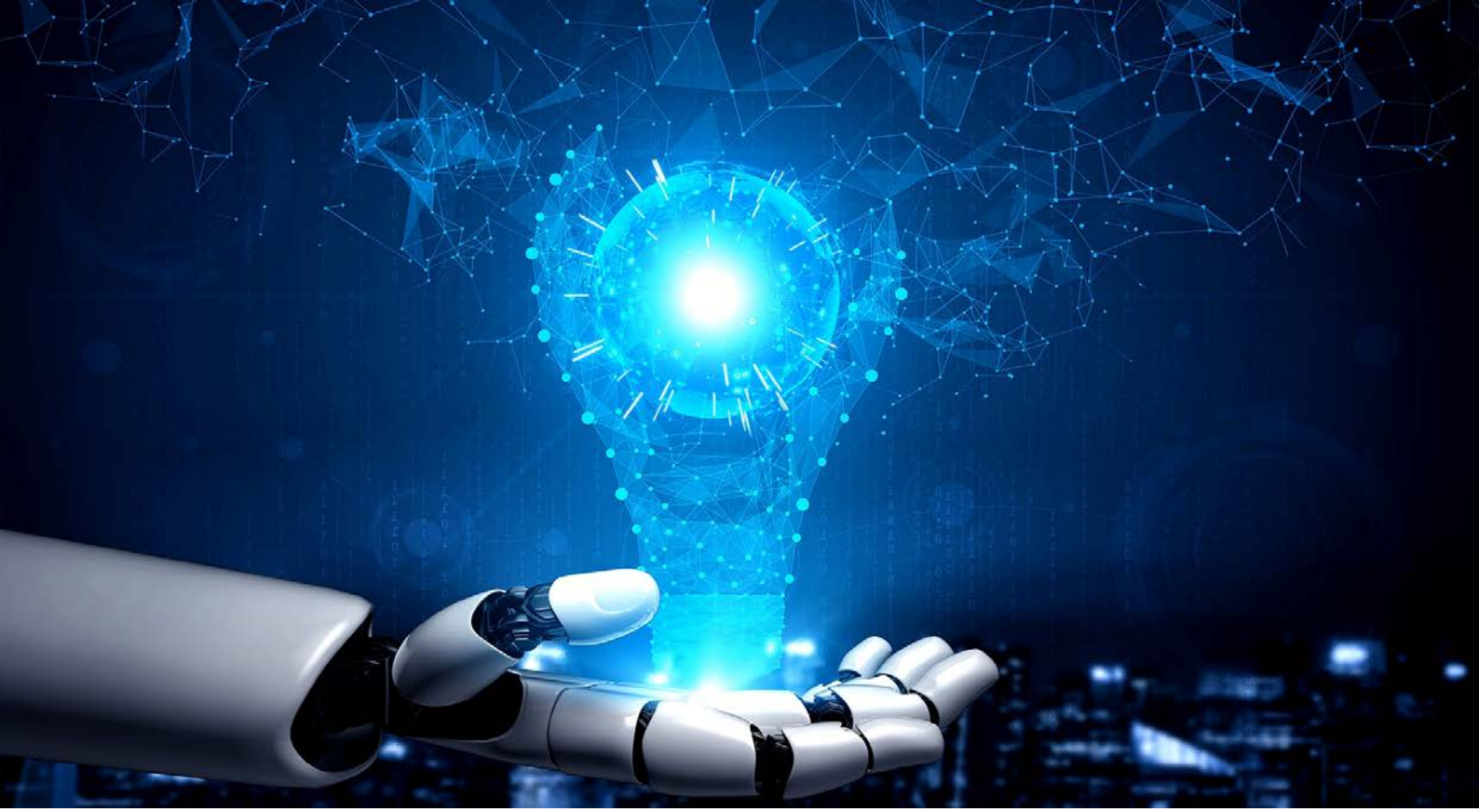

OWASP PROActive CONTROLS

面 向 开 发 者

2024OWASP十大主动控制

OWASP Top Ten Proactive Controls 2024

01.前言

02.正文

03.后记

引言

多年来，开发人员一直饱受在自己所构建的软件中引入相同的安全问题之苦。最常见的问题已经存在几十年，并且OWASP十大安全风险（OWASP Top 10）已经记录了这些问题。最早版本中的许多问题至今仍以某种形式存在。业界需要一种机制来应对这些挑战，这种机制就是主动控制。

主动控制是一系列最佳实践，开发人员可以在其代码库中采纳和实施这些最佳实践，从而有效地避免许多常见的安全问题。主动控制提供了一种积极而有效的模式来完成安全设计的解决方案。

设想一下，开发人员正在开发基于Web的软件并且发布一个新版本，然后就收到报告称该版本中存在一个黑客可以恶意利用的在野安全漏洞。此时，开发人员必须采取应急措施：分析漏洞、修复漏洞、发布一个新的软件版本并推送给用户更新。尤其是在安全漏洞非常严重的情况下，整个处理过程既繁琐又令人尴尬。

主动控制通过关注开发过程的本身来防止这种情况的出现。主动控制的核心理念是在应用程序开发初期就预防常见的漏洞，从而完全避免后期繁琐且令人尴尬的错误修复工作。众所周知，从长远来看，主动控制既节省资源又节约成本。

需要注意的是，尽管遵循主动控制最佳实践会降低代码中存在安全漏洞的风险，但并不意味着代码中完全没有安全漏洞。这是可以理解的，有得必有失——唯一不引入任何安全漏洞的方法就是根本不写代码。所以开发人员必须接受这个观点，在尽力预防尽可能多的安全问题时，仍然可能会存在某些难以预见的安全漏洞。

项目介绍

《2024OWASP十大主动控制》是一份针对软件架构师和开发人员的重要安全技术文档，旨在为他们提供具体、实用的指导，以帮助软件架构师和技术人员构建更加安全的软件。这份文档强调，软件架构师和开发人员都应该了解和遵循主动控制安全技术，并应在软件开发的早期阶段就主动应用，以确保最大效用。

项目背景

随着全球范围内金融、医疗、国防、能源和其他关键基础设施对软件的依赖加深，不安全的软件正在对这些系统构成严重威胁。随着数字化和全球化基础设施的日益复杂和互联，实现应用程序安全的挑战正呈指数级增长，这使得用户再也无法容忍相对简单的安全问题。

宗旨与目标

OWASP十大主动控制项目（OPC）的核心目标是通过明确描述软件开发过程中必须高度关注的关键领域来实现提升开发人员对应用程序安全性的认识。该项目鼓励开发人员积极采用OWASP主动控制策略，让开发人员在构建软件时能够主动关注应用程序的安全问题。开发人员通过主动关注应用程序的安全问题可以从其他组织的错误中汲取教训，从而更有效地避免类似的安全问题。希望OWASP主动控制项目对开发人员在构建安全软件方面有所帮助。

值得注意的是，尽管本文主要关注Web应用程序的安全问题，但OWASP的其他十大安全风险也同样具有普遍适用性。例如，OWASP API安全十大风险 、OWASP移动应用安全十大风险，都是开发人员在不同领域构建安全软件时需要参考的重要指南。

使用指南

本文旨在为软件开发人员奠定软件安全领域的坚实基础，并提供一个入门级别的培训指南。为了确保软件安全性的提升，主动控制策略应贯穿于整个软件开发过程的始终，且需深入持续执行。

然而，需要明确的是，本文仅作为软件安全学习之旅的起点，而非一套详尽无遗的技术和实践手册。

在构建全面而安全的软件开发流程时，应参考并遵循业界公认的标准，如OWASP的Web应用安全验证标准(Application Security Verification Standard，ASVS)等，这些标准为开发人员提供了确保软件安全性的具体要求和方法。此外，还应借鉴成熟的软件开发模型和框架，如OWASP软件保障成熟度模型(The Software Assurance Maturity Model, SAMM)和软件安全构建成熟度模型(Building Security In Maturity Model, BSIMM)等。这些模型详细描述了在不同开发阶段应如何有效实施安全措施，从而提高整个软件开发生命周期中的安全性。

目标受众

本文主要面向开发人员，但同样对开发经理、产品负责人、质量保证（Q/A）专业人员、项目经理以及任何参与软件开发流程的人员都极具参考价值。

文档创建过程

本文档最初由当前项目负责人牵头，与数名志愿者共同协作完成。随后，为了更广泛地汲取智慧经验，该项目向全球开放共享，并从开源社区中收到数百条匿名修改建议，提升了其专业性和实用性。

与其他OWASP项目之间的关联性

OWASP是一个由志愿者驱动的组织，这些志愿者创建了很多有价值的文档，为开发人员和安全专家提供了宝贵的资源，以下是与OWASP相关的一些重要文档和项目：

1. OWASP Top 10

   OWASP最著名文档是OWASP十大安全风险，详细描述了最常见的Web应用程序漏洞，同时也是本文的基础。相比之下，本文侧重于防御技术和控制，而不是风险。本文中的每个控制都对应到基于风险的OWASP十大安全风险中的一项或多项，每个控制描述末尾都包含对应的信息。

2. OWASP Application Security Verification Standard (ASVS)

   OWASP ASVS是一个包含可用的安全需求和相关验证标准目录。对于开发团队而言，OWASP ASVS是一个宝贵的安全要求来源，OWASP ASVS根据高层次的安全功能将安全需求划分为不同的类别，例如，认证、访问控制、错误处理与日志记录、以及Web服务等。每个类别下都涵盖了一系列可验证的陈述要求，这些要求代表了该类别中的最佳安全实践。

3. Software Assurance Maturity Model (SAMM)

SAMM是一个开源的框架，旨在协助组织根据自身特定的风险情况，实施一套有效的软件安全成熟策略。SAMM(https://owaspsamm.org/about/)支持软件生命周期的各个阶段，并帮助组织明确在哪些方面需要进一步提升软件安全的成熟度。

4. OWASP Threat Dragon

威胁建模是安全应用开发的重要组成部分，威胁建模有助于识别潜在的安全威胁，推导出安全需求，并定制安全控制措施以防止潜在风险。安全需求的成功使用涉及四个步骤：发现、记录、实现以及验证应用程序中功能的正确实现。威胁建模是推导出安全需求的一种有效方法，但安全需求的来源并不止于此。其他来源还包括行业标准、适用的法律法规、过去漏洞的历史记录。为了更高效地实施威胁建模，开发团队可以利用像OWASP Threat Dragon这样的建模工具。这些工具可以作为安全开发生命周期的一部分，帮助团队创建威胁模型图，从而更加直观地理解和应对潜在的安全威胁。

项目其他信息

问题反馈

可以联系OWASP主动控制项目，将问题、评论及想法通过在GitHub页面上公开添加问题或提交代码更改，也可以私发电子邮件至 jim@owasp.org。

版权许可

项目负责人

- Jim Manico
- Andreas Happe
- Katy Anton

撰稿人

- Chris Romeo
- Jasmin Mair
- Abdessamad Temmar
- Carl Sampson
- Eyal Estrin
- Israel Chorzevski

中文项目团队

- 翻译：程远冲、杜文龙、郭佩刚、李霈文、林科辰、牛承伟
- 审查：王颉、肖文棣

文章架构

项目结构组织

本文档按照安全控制措施的重要性进行排列，其中列表项编号1代表最重要的控制措施：

- C1:实施访问控制
- C2:正确应用加密技术
- C3:验证所有输入并处理异常
- C4:使用安全架构模式
- C5:默认安全配置
- C6:维护组件安全
- C7:实施数字身份
- C8:协助浏览器守护用户安全
- C9:实施安全日志和监控
- C10:阻止服务器端请求伪造（SSRF）

安全控制

每个控制描述的结构都是相同的，由控制编号和唯一的控制名称组成。前面是控制编号：Cx:，后面是控制名称，例如，C1：实施访问控制。

每个控制都有相同的部分：

- 描述: 对控制的详细描述，包括一些需要考虑的最佳实践。
- 威胁: 该控制可以抵御或应对的威胁。
- 实施:最佳实践和示例，阐明如何实施该控制。
- 漏洞预防: 漏洞预防或风险列表 (OWASP TOP 10、 CWE TOP 25)等。
- 参考资料: 为进一步研究提供的参考资料列表（OWASP Cheat Sheets、安全加固指南Security Hardening Guidelines ）等。
- 工具: 一组工具或项目，以便轻松将安全控制引入或集成到软件中。

C1:实施访问控制

描述

访问控制(或授权)是允许或拒绝来自于用户、程序或进程的特定请求。在每次访问控制决策中，给定的主体都会请求访问给定的对象。访问控制是一个过程，该过程考虑已定义的策略，并确定给定的主体是否被允许访问给定的对象。当然访问控制还涉及特权的授予、撤销等行为。访问控制通常应用于多个级别，例如，授权应用程序访问后端数据库，访问控制既适用于业务逻辑级别，也适用于数据库行级别。此外，应用程序可以提供多种执行操作的方式（例如，API或网站）。所有这些不同的级别与访问路径必须保持一致，即，使用相同的访问控制检查，以防范安全漏洞。请注意，授权（核实对特定功能或资源的访问）与身份验证（核实身份）并不等同。

威胁

- 攻击方能够利用组织松散的访问控制配置策略来访问组织不对外的数据。
- 攻击方能够发现在应用程序中多个访问控制组件，并利用最薄弱的组件进行攻击。
- 如果管理员忘记停用旧帐户，攻击方可能发现并利用该旧帐户来访问数据。
- 因缺失默认拒绝的策略，攻击方能够跳到允许访问最后一步的数据。

实施

以下是在应用程序开发的初始阶段应该考虑最小的访问控制设计要求。

1、提前进行彻底的访问控制设计

组织一旦选择特定的访问控制设计模式，在应用程序中重新使用新的模式来设计访问控制，这个过程通常既困难又耗时。访问控制也是应用程序安全设计的主要领域之一，必须提前进行彻底的设计，尤其在满足多租户和水平（数据相关）访问控制等要求时。

组织应该考虑两种核心类型的访问控制设计：

- 基于角色的访问控制（Role-based Access Control，RBAC)是用于控制对资源访问的模型。在RBAC模型中，对资源的允许操作通过角色来识别，而不是个人主体标识。
- 基于属性的访问控制（Attribute-Based Access Control，ABAC）将根据用户的任意属性和对象的任意属性，以及可以全局识别的和与当前策略相关的环境条件来授予或者拒绝用户访问。访问控制设计可能起初很简单，但往往会变成复杂并且功能繁琐的安全控制。在评估软件框架的访问控制能力时，请确保组织的访问控制功能能够根据特定访问控制功能需求进行定制。

2、强制每个访问请求都经过访问控制检查

确保所有访问请求都必须经过访问控制验证层。Java过滤器或其他自动请求处理机制等技术是理想的编程组件，这些技术能够确保所有请求都经过访问控制检查。这在 RFC 2904里面称为策略执行点PEP。

3、整合访问控制检查

使用单一的访问控制程序或例程（Routine），防止出现多个访问控制实现的情况。虽然多数访问控制是正确，但有一些访问控制是存在缺陷的。通过使用集中方法，组织可以将安全资源集中在审查和修复执行访问控制检查的中央库或函数上，然后在整个代码库和组织中重复使用该中央库或函数。

4、默认拒绝

确保默认情况下，安全控制拒绝所有请求，除非组织明确允许，当然也包括访问缺少访问控制的API（REST或者网络钩子）。默认拒绝规则在应用程序代码中有多种表现方式，例如：

① 应用程序代码在处理访问控制请求时可能抛出错误或异常。在这种情况下，应用程序代码也应该始终默认拒绝访问控制。

② 当管理员创建新用户或用户注册新帐户时，在配置新账户访问权限之前，新帐户默认只具有最小的访问权限或无访问权限。

③ 当应用程序添加新功能时，应用程序应该拒绝所有用户访问该新功能，直到正确配置该新功能为止。

5、最小特权、即时（Just in Time，JIT）和适切访问（Just Enough Access，JEA）原则

实现该原则的例子是为每个需要高特权活动的组织职能创建专用的特权账号及角色，并避免使用全时全特权的“管理员”角色或帐户。

为进一步提高安全性，组织可以实现安全控制的及时（JIT）或者适切访问(JEA)原则：确保所有用户、程序或进程只获得足够完成当前任务所需的访问权限。该访问权限应该只在主体发出请求时即时提供，并应当是短期的。同时组织要警惕不提供细粒度访问控制配置功能的系统。

6、避免对角色进行硬编码

许多应用程序框架默认使用基于角色的访问控制。同时类似这样的检查在应用程序代码中很常见：

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {deleteAccount();}
```

在代码中要谨慎使用这类基于角色的编程方式，因为这类方式有以下限制或风险：

- 这类基于角色的编程非常脆弱，容易在代码中创建不正确角色检查或缺失角色检查。
- 硬编码角色不允许多租户。为允许基于角色的系统对不同用户有不同规则，需要采取部分极端措施，比如代码分支或为每个用户都添加检查。
- 具有许多访问控制检查的大型代码库可能会使审核或验证整个应用程序访问控制策略变得困难。
- 在审计期间，硬编码角色也可以视为后门。

7、基于属性的访问控制策略执行点示例

请使用如下编程方法，考虑访问控制执行点：

```
if (user.hasPermission("DELETE_ACCOUNT")) {deleteAccount();}
```

基于属性或功能访问控制检查是构建设计良好且功能丰富的访问控制系统的起点。随着时间推移，这种类型的编程还允许更多的访问控制定制功能。

C1:实施访问控制

漏洞预防

- OWASP Top 10 2021-A01_2021-Broken Access Control
- CWE Top 25 - 11:CWE-862 Missing Authorization
- CWE Top 25 - 24:CWE-863 Incorrect Authorization

参考资料

- OWASP Cheat Sheet: Authorization Cheat Sheet
- OWASP Cheat Sheet: Logging Cheat Sheet
- OWASP ASVS V4 Access Control
- OWASP Testing Guide: Authorization Testing
- OAuth2.0 protocol for authorization
- Draft OAuth2.1
- Policy Enforcement in RFC 2904

工具

- ZAP with the optional Access Control Testing add-on
- Open Policy Agent

C2:正确应用加密技术

描述

敏感数据，如密码、信用卡号、健康记录、个人信息和商业秘密等需要额外的保护，特别是，如果该数据属于隐私法（如GDPR）、金融数据保护规则（如PCI-DSS）或其他法律法规管辖范围内。

攻击方可以通过多种方式从web和web服务应用程序窃取数据。例如，如果敏感信息在没有通信安全的情况下通过互联网发送，那么攻击方可以通过无线共享连接捕获并窃取不同用户的数据。此外，攻击方可以使用SQL注入从应用程的序数据库中窃取密码及其他凭证，并将窃取的信息公开。

隐私是确保关于某个实体的某些信息的机密性以及访问权限得到保护的保证。当用户使用开发人员构建的应用程序时，用户期望自己的信息得到妥善保护，而不发生泄露。

组织需要保护敏感数据，例如，密码、信用卡号、健康记录、个人信息和商业秘密。

现有的法律法规强制组织保护用户的个人信息。欧盟重视个人隐私，因此制定了GDPR。还有如PCI-DSS等金融数据保护规则，旨在保护持卡人隐私。

密码学是研究如何将明文信息变得不可理解（即加密），以及将加密信息恢复为可理解形式的原理、手段和方法的艺术或科学。个人用户数据需要加密，以确保在存储时得到妥善保管。

对应用程序的数据类型进行分类

对系统中的数据进行分类并确定数据的敏感级别是至关重要的，然后每个数据类别都可以映射到对应敏感类别所需的保护规则。

组织对系统中发送、处理和存储的数据进行分类，并确定数据的敏感级别。组织对数据进行分类以定义每种类型的特定保护规则。规则创建要满足数据最小化，并尽可能不存储敏感数据的要求。

例如，不敏感的公共营销信息可以归类为公共数据并放在公共网站上，而信用卡号码需要归类为用户私人数据，在存储、处理或传输过程中需要加密。

数据分类也可能通过立法强制执行，例如，在欧盟内为用户提供服务时，GDPR就是强制要求的。

实施

说到加密，有几个简单的规则：

- 永远不要传输明文数据。现有的技术能力很容易加密任何A和B点之间发送的数据。采用加密技术来保护所有静态数据和传输中的数据。
- 不要创建自己的加密协议。创建加密协议是个棘手的问题。当NIST创建AES时，举办了一场公开的竞赛，由世界上最好的密码学家提交提案，然后寻找所有提案的缺陷。与其将开发周期用于创建新的加密协议，不如使用现有的、经过实战检验的标准，然后将创新集中在提升产品或功能上。
- 不要自行实现加密例程，而使用已实现加密例程的现有库。

静态数据保护

敏感数据管理的第一条规则是尽可能避免存储敏感数据。如果组织必须存储敏感数据，请确保以特定方式对敏感数据进行加密保护，避免未经授权的泄密和修改。密码学（或加密）是信息安全中比较高级的主题之一，理解密码学需要更多的教育和经验。实现加密技术的困难之处在于有很多加密方法，不同的加密方法都有优缺点，需要web解决方案架构师和开发人员对这些加密方法有深入且全面的理解。此外，严肃的密码学研究通常基于高等数学和数论，为入门设置了很高的门槛。设计或构建加密算法非常容易出错（参见信道侧攻击）。因此强烈建议使用经过同行评审和开放共享的解决方案，例如，Google Tink、Libsodium等项目，以及内置在许多软件框架和云服务中的安全存储功能，而不是从头开始构建加密功能，

【安全保存密码】

大多数web应用程序都面临着存储用户密码以设置身份验证服务的挑战。为了确保攻击方无法轻易获取密码，应用程序必须安全地存储这些密码，绝对不要以明文的方式将密码存储在数据库的任何位置，相反，必须使用哈希运算（Hashing）存储密码。此外，为了增强安全性还需要为每个密码的哈希运算添加随机盐（Random Salt），以增加哈希运算的随机性。

【特殊案例：应用程序密钥管理（Application Secrets Management）】

应用程序包含许多安全操作所需的“密钥”，包括证书、SQL连接密码、第三方服务帐户凭证、密码、SSH密钥、加密密钥等等。未经授权的披露或修改这些密钥可能破坏整个系统。因此，在管理应用程序密钥时，应考虑以下因素：

不要将密钥存储在代码、配置文件或通过环境变量传递。使用GitRob或TruffleHog等工具来扫描查找代码库中的密钥。编写代码的方式应确保即使代码泄露（例如，由于配置github库存在缺陷），正在运行的应用程序仍然安全。将密钥和其他应用程序的密钥保存在密钥库中，例如，KeyWhiz或Hashicorp的vault项目、Amazon KMS或AWS密钥管理器，以便在运行时提供与应用程序级别相同的密钥安全存储和访问。许多web框架（例如，Ruby on Rails），提供了处理密钥和凭证的集成方式。

【密钥的生命周期】

密钥用于具有许多敏感功能的应用程序。例如，密钥可以用于签署JSON Web令牌(JSON Web Token，JWT)、保护信用卡、提供多种形式的身份验证以及提升其他敏感的安全功能。在管理密钥时，组织应该遵循许多规则，包括：

- 确保所有的密钥都受到保护，以防止未经授权的访问。
- 记录所有对密钥的授权访问，以用于取证。
- 将密钥存储在合适的密钥库中。
- 需要多个密钥时，只提供独立的密钥。
- 密钥支持加密算法的更改，为未来所需的变更做准备。
- 应用程序支持并能平稳地完成密钥替换操作。可以定期执行密钥替换，也可以在密钥泄密之后执行替换。

传输过程的数据保护

敏感数据，例如，密码、信用卡号、医疗记录、个人信息和商业机密都需要额外的保护，特别是敏感数据如果属于《欧盟通用数据保护条例(European General Data Protection Regulation of 2018，GDPR)》、金融数据保护规则（例如，《支付卡行业数据安全标准Payment Card Industry Data Security Standard，PCI-DSS 》）或其他法规的范畴。攻击方可以通过多种方式从web及web服务应用程序中窃取数据。例如，敏感信息在没有安全通信的情况下通过互联网发送，攻击方通过共享的无线网络连接就能够捕获与窃取用户数据。当然，攻击方可以通过SQL注入的方式从应用程序数据库窃取密码及其他凭证(Credential)，然后将窃取的信息公开。

【使用当前的加密协议】

在开发web应用程序时，请使用TLSv1.2或TLSv1.3，最好使用TLSv1.3。如果可以，研究HTTP/2或HTTP/3的使用情况，因为HTTP/2或HTTP/3能够保证使用安全的TLS版本/算法。

- 直接关闭旧协议，以防范协议降级攻击。
- 不提供HTTP服务，同时禁用HTTP和SSL压缩。
- 始终使用安全的随机数生成器(Random Number Generator，RNG)。

【指引客户端强制加密传输层(Transport)】

Web服务器可以指引用户的Web浏览器维护最低级别的传输层安全性：

- 使用响应报头（HTTP Strict-Transport-Security，HSTS）来强制执行机会加密和证书验证检查。
- 内容安全策略（content-security–policy，CSP）允许客户端将HTTP自动升级到HTTPS。
- 设置cookie时，始终使用“secure”标志以防止客户端通过HTTP传输。

密码学的与时俱进

建议密码学要与时俱进，为做到这点，通过选择加密方式，例如，使用不同算法或调整密钥大小的配置。如果应用程序需要支持高可用，请设计密钥轮换流程。

漏洞预防

- https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- https://mas.owasp.org/MASVS/controls/MASVS-CRYPTO-1/

参考资料

- OWASP Cheat Sheet: Transport Layer Protection
- Ivan Ristic: SSL/TLS Deployment Best Practices
- OWASP Cheat Sheet: HSTS
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage
- OWASP Testing Guide: Testing for TLS
- OWASP WrongSecrets : vulnerable application with example of how to NOT use secrets

工具

- https://github.com/nabla-c0d3/sslyze
- https://testssl.sh/
- SSLyze - SSL configuration scanning library and CLI tool
- SSLLabs - Free service for scanning and checking TLS/SSL configuration
- OWASP O-Saft TLS Tool - TLS connection testing tool
- GitRob - Command line tool to find sensitive information in publicly available files on GitHub
- TruffleHog - Searches for secrets accidentally committed
- Hashicorp Vault - Secrets manager
- Amazon KMS - Manage keys on AWS
- AWS Secrets Manager - Manage secrets on AWS
- Azure Key Vault - Manage keys and secrets on Azure
- Google Cloud KMS - Manage keys on Google Cloud Platform
- Google Secret Manager - Manage secrets on Google Cloud Platform

C2:正确应用加密技术

国密说明

国密法律法规

2018年，国家密码管理局发布实施密码行业标准GM/T0054-2018《信息系统密码应用基本要求》，修改完善上升为国家标准，进一步突出了其在商用密码应用标准体系中的基础性地位。2021年3月，国家市场监管总局、国家标准化管理委员会发布公告，正式发布国家标准GB/T39786-2021《信息安全技术信息系统密码应用基本要求》。贯彻落实《中华人民共和国密码法》，指导我国商用密码应用与安全性评估工作的一项基础性标准，对于规范和引导信息系统合规、正确、有效应用密码，切实维护国家网络与信息安全具有重要意义。《密码法》及相关法律法规明确了商用密码应用与安全性评估的法定要求。《信息安全技术信息系统密码应用基本要求》标准从物理和环境安全、网络和通信安全、设备和计算安全、应用和数据安全等四个方面提出了密码应用技术要求，以及管理制度、人员管理、建设运行、应急处置等密码应用管理要求。与GM/T0054-2018《信息系统密码应用基本要求》相比，该标准结合近年来商用密码应用与安全性评估工作实践对部分内容进行了优化，按照信息系统安全等级分别提出了相应的密码应用要求。

国密算法介绍

我国自主研制的SM2椭圆曲线公钥密码签名算法、SM3密码杂凑算法、SM4分组密码算法、ZUC序列密码算法、SM9标识密码算法均已成为国际标准，介绍如下：

① SM2是非对称加密算法：

基于椭圆曲线密码的公钥密码算法标准，其秘钥长度256bit，包含数字签名、密钥交换和公钥加密，用于替换RSA/DH/ECDSA/ECDH等国际算法。SM2采用的是ECC 256位的一种，其安全强度比RSA 2048位高，且运算速度快于RSA。

② SM3是一种密码杂凑算法：

用于替代MD5/SHA-1/SHA-2等国际算法，适用于数字签名和验证、消息认证码的生成与验证以及随机数的生成。在SHA-256基础上改进实现的一种算法，采用Merkle-Damgard结构，消息分组长度为512bit，输出的摘要值长度为256bit。SM3算法的安全性要高于MD5算法和SHA-1算法。

③ SM4是分组加密算法：

分组对称密码算法用于替代DES/AES等国际算法。SM4算法与AES算法具有相同的密钥长度、分组长度，都是128bit。

④ SM9是基于标识的非对称密码算法：

用椭圆曲线对实现的基于标识的数字签名算法、密钥交换协议、密钥封装机制和公钥加密与解密算法，包括数字签名生成算法和验证算法，并给出了数字签名与验证算法及其相应的流程。并提供了相应的流程。可以替代基于数字证书的PKI/CA体系。

⑤ ZUC序列密码算法：

又称祖冲之算法，以3GPP(3rdGenerationPartnershipProject)机密性算法EEA3和完整性算法EIA3为核心，是由中国自主设计的加密算法。成为3GPPLTE第三套加密标准核心算法。

C3:验证所有输入并处理异常

描述

输入验证是一种编程技术，确保只有格式正确的数据才能进入软件系统组件。当注入攻击针对客户端（例如基于JavaScript的攻击）时，Web服务器可以在将攻击方提供的数据转发给客户端之前对数据进行转义或者编码处理。

如果应用程序将数据输入错误地解释为可执行命令，则可能会发生注入攻击，这种情况通常发生在忘记实施输入验证或实施错误时。例如，假设一个Web应用程序接受用户输入的电子邮件地址。电子邮件地址则是预期的“数据”。攻击方会寻找方法让应用程序将（假设的）数据作为命令执行。不同的注入攻击针对不同的领域：

- 当攻击方诱导应用程序将用户输入的数据解释为SQL命令（或其一部分）时，就会发生SQL注入攻击。注入的命令随后会在数据库服务器内执行。
- 远程命令注入（RCE）是应用程序错误地将用户数据当作在Web应用程序服务器或主机上执行的命令。服务器端模板注入是另外一种在应用服务器内执行注入的例子。
- 当Web应用程序接受了用户数据并导致JavaScript注入时，数据会作为代码强制执行。通常注入的JavaScript代码会在其他用户使用Web浏览器访问时执行，因此这类攻击并不是直接攻击Web服务器，而是攻击其他用户。

语法与语义的有效性

应用程序应在使用任何数据（包括将数据回显给用户）之前，检查数据的语法与语义有效性（按顺序）。

- **语法有效性：**表示数据需要以符合预期的形式出现。例如，应用程序可能允许用户选择一个四位数的“账户ID”来执行某些操作。应用程序应假定用户正在输入SQL注入载荷，并检查用户输入的数据是否精确为四位数字长度，且仅由数字组成（此外应正确使用参数化查询）。
- **语义有效性：**包括仅接受应用程序功能和上下文可接受范围内的输入。例如，选择日期范围时，开始日期必须早于结束日期。

实施

防止注入攻击通常基于纵深防御方法，并结合输入过滤、输出转义以及使用强化机制。前两者仅依赖于实施的安全措施，而后者主要依赖于客户端支持，例如，在防御跨站脚本攻击XSS时，无论使用哪种Web浏览器，从输入中过滤XSS并在服务器端转义输出数据都可以防止XSS；添加内容安全策略CSP也可以防止XSS，但前提是用户的浏览器支持此策略。因此，安全性绝不能仅仅依赖于可选的强化措施。

防止恶意数据进入系统

永远不要信任提供的数据！务必筛查所有数据是否存在恶意模式，或者更稳妥的方式是，将所有数据与允许列表进行比对检查。以下是一些建议措施：

【允许列表和拒绝列表】

常规有两种执行语法验证方法，通常为允许列表和拒绝列表：

- 拒绝列表（Denylisting）验证尝试检查给定数据是否不包含“已知恶意”内容。例如，Web应用程序可能会阻止包含< SCRIPT >的输入以帮助防止跨站脚本攻击XSS。但是，这种防御可以通过小写的script标签或混合大小写的script标签来绕过。
- 允许列表（Allowlisting）验证尝试检查给定数据是否符合一组“已知良好”规则。例如，用于美国州的允许列表验证规则是将两字母代码代表一个有效美国州。

允许列表是推荐的最低限度的方法。拒绝列表容易出错，可以通过各种规避技术绕过，当依赖于自身时可能会很危险。即使攻击方经常可以绕过拒绝列表，但拒绝列表有助于检测明显的攻击。因此，允许列表通过确保数据具有正确的语法和语义有效性来帮助限制攻击面，但拒绝列表有助于检测并可能阻止明显的攻击。

【客户端验证和服务器端验证】

始终在服务器端执行输入验证以确保安全。虽然客户端验证在功能和安全方面都有用，但由于攻击方很容易绕过客户端验证，应用程序的安全性绝不应仅依赖于客户端验证。例如，JavaScript验证可能会提醒用户某个字段必须由数字组成，但服务器端应用程序必须验证提交的数据是否仅由该功能所需范围内的有效数字组成。此外，同时使用客户端和服务器端验证的另一个好处是，可以记录下来任何服务器端验证的警告，从而有助于发现潜在的攻击方的攻击，由于攻击方已经绕过客户端的验证。

【正则表达式】

正则表达式提供了一种检查数据是否匹配特定模式的方法。下面是一个基础示例，正则表达式定义了一个验证用户名的允许列表规则。

```
^\[a-z0-9_\]{3,16}$
```

此正则表达式仅允许使用小写字母、数字和下划线字符。用户名的长度也限制为 3 到 16 个字符。

① 注意：潜在的拒绝服务风险

创建正则表达式时需谨慎。设计不良的正则表达式可能会导致潜在的拒绝服务情况（即ReDoS）。可以使用各种工具对正则表达式进行测试，确保其不易受到ReDoS攻击。

② 注意：正则表达式的复杂性

正则表达式只是实现验证的一种方式。对于某些开发人员来说，正则表达式可能难以维护或理解。其他验证替代方案包括以编程方式编写验证方法，这对于某些开发人员来说可能更易于维护。

【意外的用户输入（批量分配）】

一些框架支持自动将HTTP请求参数绑定到应用程序使用的服务器端对象。此自动绑定功能可能允许攻击方更新不应修改的服务器端对象。攻击方可能利用此功能修改访问控制级别或绕过应用程序的预期业务逻辑。这种攻击有多个名称，包括：批量分配、自动绑定和对象注入。例如，如果用户对象有一个字段privilege，该字段指定用户在应用程序中的权限级别，恶意用户可以查找修改用户数据的页面，并在发送的HTTP参数中添加 “privilege=admin”。

如果自动绑定以不安全的方式启用，则攻击方可以相应地修改代表用户的服务器端对象。

处理自动绑定问题可以采用两种方法：

- 避免直接绑定输入，而是使用数据传输对象（DTOs）。
- 启用自动绑定，但为每个页面或功能设置允许列表规则，定义哪些字段允许自动绑定。

更多示例请参见OWASP Mass Assignment Cheat Sheet。

输入验证的局限性

输入验证虽然重要，但并非万无一失的安全保障，因为某些复杂的输入形式，即使输入数据在形式上看起来是有效的，但也可能隐藏潜在的风险。例如，一个看似合法有效的电子邮件地址可能包含SQL注入攻击，或一个看似正常有效的URL可能包含跨站脚本攻击XSS。除了输入验证外，还应采取额外的防御措施来确保数据的安全性，如参数化查询或转义。

使用支持数据和命令分离的机制

即使恶意数据通过了输入检查，应用程序也可以通过避免将这些恶意数据作为命令/代码执行来防止注入攻击。实现这一目标可以采用多种措施，其中大多数依赖于具体的安全技术。例如：

- 通过 SQL 使用关系数据库时，请使用预处理语句。。当攻击方能够提供精心构造的输入数据，这些数据能够“逃逸”出通过字符串拼接创建的SQL命令时，则通常会发生SQL注入攻击。使用预处理语句允许计算机自动对输入数据进行编码，使其无法从命令模板中“逃逸”。
- 使用对象关系映射（ORM）时，虽然它们的间接层可能会阻止常见的SQL注入，但针对ORM的特殊构造的攻击有时仍然可能成功。
- 服务器端模板注入（SSTI）使用服务器端的模板引擎动态生成内容，然后展示给用户。为了降低风险，SSTI引擎通常允许配置沙箱，即仅允许执行有限数量的方法。
- 使用用户输入作为参数执行系统命令容易遭受注入攻击。应尽可能地避免将用户输入直接用于执行系统命令。

【JavaScript注入攻击】

JavaScript注入攻击（也称为跨站脚本攻击，简称XSS）是一种特殊情况。注入的恶意代码通常在受害者的浏览器中执行。通常，攻击方试图从浏览器中窃取用户的会话信息，而不是直接执行命令（如在服务器端做的那样）。除了服务器端的输入过滤和输出转义外，还可以采取多种客户端强化措施（这些措施还可以防止基于 DOM 的 XSS 的特殊情况，因为基于DOM的XSS不涉及服务器端逻辑，因此无法过滤恶意代码）：

- 使用httpOnly标记敏感的cookie，使JavaScript无法访问。
- 利用内容安全策略（CSP）减少JavaScript攻击的攻击面。
- 使用默认安全框架，如Angular。

【验证并清理HTML】

假设有一个应用程序，当用户需要通过WYSIWYG编辑器输入内容（表示为HTML）或直接输入HTML代码时，验证或转义可能不足以确保安全性，将不再适用。

- 正则表达式无法满足HTML5的复杂性。
- 编码或转义HTML将不起作用，因为会导致HTML无法正确渲染。

因此，需要一个库来解析和清理HTML格式的文本。有关HTML清理的更多信息，请参阅XSS Prevention Cheat Sheet on HTML Sanitization。

特殊情况：反序列化过程中验证数据

某些形式的输入非常复杂，验证只能为应用程序提供有限程度的保护。例如，不受信任的反序列化数据或攻击方可以操纵的数据都存在着严重的安全风险。唯一安全的架构模式是不接受来自不受信任来源的序列化对象，或仅对简单数据类型进行有限容量的反序列化。系统更应避免处理序列化数据格式，并尽可能使用更易于保护的格式（例如 JSON）。

当无法避免时，请考虑在处理序列化数据时进行一系列验证防御：

- 实施完整性检查和序列化对象的加密，以防止恶意对象创建或数据篡改。
- 在对象创建前的反序列化过程中实施严格的类型约束；通常代码期望的是一组可定义的类。但是有证据表明，这个技术方法可以绕过。
- 隔离反序列化代码，使其在权限极低的环境中运行，如临时容器。
- 记录安全反序列化异常和失败，例如当传入类型不是预期类型时，或反序列化抛出异常时。
- 限制或监控从反序列化容器或服务器的入站和出站网络连接。
- 监控反序列化过程，如果用户持续进行反序列化操作，则发出警报。

漏洞预防

- 输入验证减少了应用程序的攻击面，并且有时会使针对应用程序的攻击变得更困难。
- 输入验证是一种专为某些形式的数据提供安全保护的技术，以抵御某些特定的攻击手段，但并不能作为普遍适用的安全规则来依赖。
- 输入验证不应是用于防止XSS、SQL 注入和其他攻击的主要方法。
- 2023 CWE Top 25 - 3 Improper Neutralization of Special Elements used in an SQL Command（‘SQL Injection’）SQL 命令中使用特殊元素的不当中和（“SQL 注入”）。
- 2023 CWE Top 25 - 5 Improper Neutralization of Special Elements used in an OS Command（‘OS Command Injection’）操作系统命令中使用特殊元素的不当中和（“操作系统命令注入”）。
- 2023 CWE Top 25 - 16 Improper Neutralization of Special Elements used in a Command（‘Command Injection’）命令中使用的特殊元素的不当中和（“命令注入”）。
- 2023 CWE Top 25 - 23 Improper Control of Generation of Code（‘Code Injection’）对代码生成的不当控制（“代码注入”）。

参考资料

关于输入验证：

- OWASP Cheat Sheet: Input Validation OWASP 备忘单：输入验证
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs OWASP 备忘单：iOS - 通过不受信任的输入做出安全决策
- OWASP Testing Guide: Testing for Input Validation OWASP 测试指南：输入验证测试
- Injection Prevention Cheat Sheet 注入预防备忘单
- Injection Prevention Cheat Sheet in Java Java注入防护备忘单
- Hardening with CSP: CSP with Google 使用CSP进行强化：Google的CSP
- Deploying CSP in Single Page Applications 单页应用中部署CSP

工具

帮助进行输入验证的工具：

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate ValidatorApache Commons ValidatorPHP’ s filter functions

测试注入攻击的工具：

- Sqlmap.py
- OWASP ZAP-based scans Helping with Hardening:
- CSP Evaluator

C4:使用安全架构模式

描述

专家们将自己最佳实践的智慧结晶以易于理解的方式分享，称之为安全架构模式。架构模式是可重复使用的，可以应用于多个应用程序。

要将解决方案视为一个模式，该解决方案必须具备以下特点：

1. 安全架构模式必须解决安全问题。
2. 安全架构模式不能依赖于特定供应商或技术。
3. 安全架构模式必须展示如何缓解威胁。
4. 安全架构模式必须使用标准化术语描述威胁和控制措施，以便于重复使用。

架构模式是一种使用标准解决方案，而不是创建自定义解决方案来解决问题的方法。安全架构模式是一种经过审查并针对已知安全威胁进行强化的标准解决方案。

实施

1. 确定需要解决的问题。
2. 考虑可用的安全架构模式目录。
3. 为设计选择一个安全架构模式。
4. 实施安全架构模式。

漏洞预防

- 业务逻辑漏洞：这些模式有助于在构建应用程序时规避复杂且经常忽视的业务逻辑缺陷。
- OWASP Top 10 2021-A04（不安全设计）：安全架构模式直接针对与不安全设计相关的风险进行缓解，这是OWASP强调的一个关键问题。

参考资料

- https://securitypatterns.io/what-is-a-security-pattern/
- https://owasp.org/www-pdf-archive/Vanhilst_owasp_140319.pdf
- https://cheatsheetseries.owasp.org/cheatsheets/Microservices_based_Security_Arch_Doc_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.html

工具

- https://securitypatterns.io/what-is-a-security-pattern/ ↩

C5:默认安全配置

描述

“默认安全”意味着产品在出厂时就具备抗击常见攻击手段的能力，而无需额外付费。让应用程序初始就安全的好处在于，减轻了开发人员在如何保护系统安全方面的负担，提供了一个已经安全的产品。 “默认安全 ”减少了以安全方式部署产品所需的工作量，并增强了产品长期保持安全的信心。

实施

在现代云应用程序中，当开发人员在构建应用程序时，也在为其应用程序构建基础架构，在编写代码的同时做出基础架构决策，包括安全关键配置。应用程序部署在通过代码创建和配置的基础架构上，即基础架构即代码（IaC）、使用应用级别（包括Web服务器和数据库）、容器级别、功能即服务级别或基础架构级别的配置。例如，在Web应用程序中，文件夹权限需要遵循最小权限原则来限制资源访问权限。当web和移动应用程序在生产环境中部署时，应禁用调试功能。

当开发人员整合基础架构组件时，以下几点十分重要：

1. 根据最小权限原则实施配置--例如：确保云存储（S3 或其他）配置为私有，并仅限用户有限时间内访问。
2. 默认情况下拒绝访问，通过允许列表进行授权。
3. 使用已经通过软件包和组件漏洞扫描的容器镜像，并确保这些镜像是从私有容器注册表中提取的。
4. 优先使用声明式基础架构配置，而不是手动配置活动。在底层应用上，利用基础架构即代码模板对云和内部部署基础架构进行自动调配和配置。在高层应用上，利用策略即代码（Policy-as-Code）来执行包括权限分配在内的策略。注：使用声明性格式允许策略类似源代码一样进行管理：嵌入源代码管理系统、版本控制、访问控制、变更管理等。
5. 流量加密--默认实现，或不要在关键位置使用未加密的通信渠道。

持续配置验证

作为软件开发的一部分，开发人员需要确保应用软件在应用程序级别的默认配置是安全的。例如：

- 定义基础架构的代码应遵循最小权限原则。
- 应禁用不需要的配置和功能，如帐户和演示功能。

漏洞预防

- OWASP Top 10 2021 A05 – Security Misconfiguration

参考资料

- OWASP Cheat Sheet: Infrastructure as Code Security Cheatsheet
- OWASP ASVS: Application Security Verification Standard V14 Configuration
- Cloud security guidance - NCSC.GOV.UK

工具

- Tfsec - open source static analysis for your Terraform templates 用于分析Terraform模板的开源静态分析工具。
- Terrascan - scan for Infrastructure-as-Code vulnerabilities 用于扫描基础设施即代码漏洞。
- Checkov - Scan for open-source and Infrastructure-as-Code vulnerabilities 用于扫描开源和基础设施即代码漏洞。
- Scout Suite is an open source multi-cloud security-auditing tool which currently supports: Amazon Web Services, Microsoft Azure, Google Cloud Platform 一个开源的多云安全审计工具，目前支持：亚马逊网络服务、微软Azure、谷歌云平台。
- prowler
- Cloudmapper
- Snyk - Scan for open-source, code, container, and Infrastructure-as-Code vulnerabilities 用于扫描开源、代码、容器和基础设施即代码漏洞。
- Trivy - Scan for open-source, code, container, and Infrastructure-as-Code vulnerabilities 用于扫描开源、代码、容器和基础设施即代码漏洞。
- KICS - Scan for Infrastructure-as-Code vulnerabilities 用于扫描基础设施即代码漏洞。
- Kubescape - Scan for Kubernetes vulnerabilities 用于扫描Kubernetes漏洞。
- Kyverno - Securing Kubernetes using Policies 用策略来保障Kubernetes安全。

C6:确保组件安全

描述

在软件开发中，利用库和框架是软件开发中的常见做法。具有嵌入式安全功能的安全库和软件框架可帮助软件开发人员预防安全相关的设计和实施缺陷。从头开始编写应用程序的开发人员可能没有足够的知识、时间或预算来正确实施或维护安全功能。利用安全框架（包括开源和供应商提供的）有助于更高效、更准确地实现安全目标。

尽可能优先使用框架现有的安全功能，而不是引入另一个第三方库，因为后者需要定期更新和维护。最好是让开发人员利用已经在使用的工具和库，而不是强迫再引入另一个库。

在将第三方库或框架集成到开发的软件时，必须考虑以下两类最佳实践：

1. 识别可信任的库和框架，以集成到软件中。
2. 监控和更新软件包，确保软件不会受到第三方组件可能带来的安全漏洞的影响。

实施

以下是对两类最佳实践的进一步详细说明，旨在保障软件安全：

识别可信库

以下列出了可以用来选择软件库或框架的一些标准。以下要求并不是一份详尽的清单，但可以为开发者在选择可信的库或框架时提供一个良好的基础和参考。

1. **来源：** 从官方来源下载推荐的安全库，使用安全链接，并优先选择已签名的软件包，以减少包含经过篡改的恶意组件的风险（参见 A08:2021--软件和数据完整性故障）。
2. **流行度：** 使用那些拥有庞大社区支持的并且许多应用程序都在用的库和框架，可以参考GitHub上包源码仓库获得的星星数量，以及包管理器中的下载数量等数据。
3. **活跃度：** 确保库和框架得到积极维护，并及时解决问题。
4. **成熟度：** 使用稳定版本。处于早期开发阶段的项目会给软件带来更高风险。
5. **复杂性：** 一个大型、复杂、有大量依赖关系的库更难集成到软件中。此外，依赖关系越多表明未来需要升级的次数越多，以确保所有依赖关系都是最新的和安全的。
6. **安全性：** 如果软件包是开源的，可以使用静态应用程序安全测试（SAST）或软件构成分析（SCA）来帮助识别恶意代码或安全漏洞，然后再将其集成到软件包。

确保组件安全的最佳实践

新的安全漏洞每天都会披露出来，并发布在NIST国家漏洞数据库(NVD)等公共数据库中。NVD使用Common Vulnerability and exposure (CVE)识别公开的已知漏洞。此外，公共数据库中披露的漏洞攻击方式可以让攻击方将攻击自动化。因此，必须定期检查以确保软件不存在已知的安全漏洞。

1. 维护一个所有第三方组件的清单目录：建议从构建流水线内部自动创建SBOM（软件物料清单）。SBOM 包含所有使用的第三方依赖项及其版本，可通过各种供应链管理工具自动监控。

2. 执行持续检查：将 SBOM 与定期或持续监控工具（如 OWASP dependency-track）结合使用，自动检测已知的公开披露的漏洞。
3. 尽早并经常验证组件安全性：在软件开发的早期阶段集成 SCA 工具，以便从软件开发生命周期的每个阶段了解软件及其依赖关系的安全漏洞数量和严重程度。
4. 主动更新库和组件：更新软件必须是一项定期执行的任务，贯穿应用程序或产品从概念到退市的整个生命周期。

漏洞预防

安全的框架和库可帮助预防各种网络应用程序的漏洞。按照vulnerable and outdated components with known vulnerabilities Top 10 2021中所述，保持框架和库的更新是至关重要的。

参考资料

- OWASP Cheat Sheet: Third Party Javascript Management
- OpenSSF Scorecard - Security health metrics for Open Source

工具

- OWASP Dependency-Check - to identify project dependencies and check for publicly disclosed vulnerabilities 用于识别项目依赖项并检查公开披露的漏洞。
- OWASP Dependency-Track - periodically monitor SBOM files for new vulnerabilities 定期监控SBOM文件以发现新漏洞。
- Retire.JS scanner for JavaScript libraries 扫描器用于JavaScript库。
- Renovate for automated dependencies updates
- Harbor : an open source registry that secures artifacts with policies and rolebased access control 一个开源注册表，通过策略和基于角色的访问控制保护构件。

C7:实施数字身份

描述

数字身份（Digital Identity）是个人、组织（或者其他主体）在在线交易时的独特表示。验证个人或者实体身份真实性的过程称为身份**认证（Authentication）**。

会话管理（Session management）是指服务器维持用户认证状态的过程，允许用户能够在无需重新验证的情况下持续访问系统。

数字身份、认证和会话管理均属于复杂且深入的话题，本节仅对数字身份主题提供了基础性探讨。为确保能妥善处理这些复杂性问题，组织应当指派具备高度专业能力的工程师来主导身份解决方案的实施。美国国家标准与技术研究院（NIST）发布的特别出版物800-63B：《数字身份指南：认证和生命周期管理》为数字身份、认证及会话管理控制的执行提供了宝贵的指导。以下是一些安全实施建议，旨在确保应用程序中执行强有力的数字身份控制。

身份验证保证级别

NIST 800-63B 描述了三种级别的身份认证保证，称为身份认证保证级别(AAL)。

【AAL级别1：密码（Passwords）】

适用于不包含个人身份信息（PII）或者其他敏感私人数据的低风险应用程序。在AAL级别1中，仅需要单因素认证，通常通过密码（即你知道的信息）来实现。密码（或者一般凭证）的安全至关重要，涉及到安全存储方式（如使用密钥派生函数等技术）以及相应的流程，例如拥有安全的密码重置流程。

【AAL级别2：多因素认证（Multi-Factor Authentication，MFA）】

适用于包含“自我声明的个人身份信息（PII）或者其他在线提供的个人信息”的高风险应用。在AAL级别2中，需要采用多因素认证，包括一次性密码（OTP）或者其他形式的多因素实现。

多因素认证（MFA）通过要求用户使用以下组合来确认自己的身份，确保用户身份的真实性：

- 你所知道的——密码或PIN码
- 你所拥有的——令牌或者手机，当使用手机时，请使用遵循标准化协议（如FIDO2）的标准认证应用程序。
- 你所具备的——生物特征，尽管使用如指纹等生物特征作为认证因素可以提供额外的安全性，但仅依赖生物特征作为单独的认证因素可能安全性较低。

多因素认证方案要求攻击方必须获取多个验证因素，才能与服务完成身份验证，从而提供了更强的安全性。需要注意的是，生物特征用作单一认证因素时，不能作为数字认证可信赖的秘密信息。因为可能在不知情的情况下通过在线获取、拍照（例如，面部图像），或者从某人接触的物体上提取信息（例如，隐形指纹），或者捕获高分辨率图像（例如，虹膜图案）。因此，生物特征应仅作为多因素身份验证中的一个环节，并与物理认证器（即你拥有）结合使用。例如，用户可以通过多因素一次性密码（OTP）设备生成一次性密码，并手动输入至验证器以完成身份验证。

【AAL级别3：基于加密的认证(Cryptographic Based Authentication)】

当受损系统的影响可能导致人身伤害、重大经济损失、损害公共利益或者涉及民事、刑事违规时，需要实施NIST 800-63B指南中的第三级认证保证（AAL3）。AAL3级别要求采用“基于通过加密协议验证密钥持有”的身份验证类型。该认证类型通常通过硬件加密模块实现，旨在提供最高级别的认证保证。在Web应用程序开发中，通常推荐采用WebAuthn或PassKeys等技术。

会话管理：客户端与服务器端会话

HTTP协议本身是一种无会话协议：请求之间不共享数据。但在日常浏览网页的体验中并非如此，比如用户登录网站后，因为在HTTP协议上实现了会话管理机制，所以在后续的请求中仍会保持登录状态。用户完成初次认证后，应用程序可以选择在一定的时间内跟踪并保持这种认证状态，这将允许用户继续使用应用程序，而无需每次请求都重新进行身份验证。这种状态跟踪称为会话管理。会话管理大致可以分为客户端会话管理和服务器端会话管理两种类型。客户端会话管理是将所有会话数据存储在客户端，并随每次请求发送给服务器。而服务器端会话管理则是在服务器上保存会话数据（例如，数据库），并仅向客户端发送单一会话标识。客户端随后在每次请求时仅需发送该会话标识，由服务器根据此标识符从服务器端存储中检索会话数据。

从安全角度来看，服务器端会话技术具有多重优势：

- 数据不直接存储在客户端：在处理敏感数据时，直接存储在客户端可能带来风险。因此，客户端会话管理策略必须确保存储在客户端的数据没有篡改。
- 客户端和服务器之间传输的数据量更少（这与网络带宽的增加无关）。
- 服务器端会话管理允许会话失效，例如，用户可以注销其所有会话。默认情况下，应始终使用服务器端会话管理。

实施

使用密码场景

【密码要求】

密码至少应符合以下要求：

- 如果同时使用多因素认证（MFA）和其他控制措施，则密码长度至少应为8个字符。如果无法使用MFA，则应将密码长度增加到至少10个字符。
- 所有可打印的ASCII字符以及空格字符都应当可以作为记忆的密码使用。
- 鼓励使用长密码和密码短语。
- 由于复杂性要求的效果有限，建议不再强制执行密码复杂性规则。相反，建议采用多因素认证（MFA）或增加密码长度。
- 确保所使用的密码不是已泄露的常用密码。在满足上述长度要求的情况下，禁止使用泄露密码列表中前1000或10000个最常见的密码。可参考以下链接获取常见泄露密码列表：https://github.com/danielmiessler/SecLists/tree/master/Passwords
- 实行强制密码轮换，避免由于长期使用相同密码而导致的潜在安全风险。

【实施安全的密码恢复机制】

应用程序通常具备一种机制，允许忘记密码的用户恢复对其账户的访问权限。设计良好的密码找回功能会采用多因素认证的方式。例如，可能会询问用户安全问题（用户所知道的信息），然后向设备（用户所拥有东西）发送生成的令牌。更多详细信息，请参阅Forgot_Password_Cheat_Sheet和Choosing_and_Using_Security_Questions_Cheat_Sheet。

【实施安全的密码存储】

为确保提供强有力的认证控制，应用程序必须安全存储用户凭证。同时，应采取加密措施，保证在凭证（如密码）泄露的情况下，攻击方也无法立即访问此信息。更多详细信息，请参阅OWASP Password Storage Cheat Sheet。

服务器端会话管理

服务器端会话管理通常采用HTTP cookie实现，这类Cookie用于存储会话标识符。当请求新会话时，服务器会创建新的会话标识符，并将其发送给客户端（如浏览器）。随后的每个请求都会将会话标识符从客户端传回服务器，服务器再利用该会话标识符在服务器端数据库中查找会话数据。

【会话生成与过期】

在服务器端会话管理中，用户的状态信息需要有效地跟踪。在传统的基于Web的会话管理方式中，这些会话信息一般存储于服务器。服务器分配单一会话标识符给用户，用户通过该标识符来识别并访问服务器端存储的与其相关的正确数据。客户端仅需要保留该会话标识符，就可以避免将敏感的服务器端会话数据暴露给客户端。在构建或者实施会话管理方案时，需要考虑以下一些控制措施：

- 确保会话ID足够长、唯一且随机，即具有高熵。
- 应用程序应在认证和重新认证期间创建新的会话。
- 应用程序应设置一段时间不活动后的空闲超时机制，并为每个会话设置绝对最大有效期限制，超时这个时间后用户必须重新认证。超时长度应与受保护数据的价值成反比。

更多详细信息，请参阅Session Management Cheat Sheet 。ASVS 第3部分涵盖了其他会话管理要求。

客户端会话管理

服务器端会话在某些认证场景中可能存在局限性。出于性能考虑， “无状态服务”允许在客户端管理会话数据，从而减轻服务器存储用户会话的负担。这类“无状态”应用程序通常会生成单个短期的访问令牌，该令牌包含用户当前所有的访问权限，并在随后的所有请求中使用该令牌。为确保安全，必须采用加密技术防止客户端篡改令牌中的权限信息。当客户端请求服务器操作时，需要附上此访问令牌，服务器则验证该令牌的完整性，确认后从令牌中提取权限信息。这些权限信息将用于后续权限检查。

【JWT（JSON Web Tokens）】

JSON Web Token（JWT）是一种开放标准 （RFC 7519），该标准定义了一种紧凑且自含的方式，用于在各方之间以 JSON 对象的形式安全地传输信息。此信息可以验证和信任，只要该信息是经过受信任的机构数字签名的。JWT令牌是在认证过程中创建的，并在处理任何请求之前由单个或者多个服务器完成验证。但服务器在初始创建JWT后通常不会保存JWT，而是创建后发送给客户端。然后通过数字签名维护令牌的完整性，以确保服务器可以验证JWT的有效性及自创建以来没有改变。这种方法即无状态又具有可移植性，即使客户端和服务器使用不同技术仍能交互。需要注意的是，使用JWT时，要确保采用的JWT与当前在用的签名算法一致。否则，可能遭受攻击方利用NULL算法伪造JWT，实施MAC-vs-Signature混淆攻击，或者提供自定义JWS密钥签名的风险。因此发放JWT时，必须使用安全的私钥签名，因为每个输出的JWT都可能暴露给潜在的攻击方，存在离线破解的风险。为了保障安全，需要定期轮换密钥。

浏览器 Cookies

浏览器Cookie是Web应用程序存储会话标识符的常见方法，适用于实现标准会话管理技术的Web应用程序。以下是使用浏览器Cookie时需要考虑的一些防御措施：

- 当使用浏览器Cookie来跟踪已认证用户的会话时，应限制这些Cookie的可访问的域和路径最小化范围，且设置在会话结束时或之后立即失效。
    ① 如果在设置Cookie时没有指定域名，将默认使用当前网站的域名。
    ② 当设置Cookie的路径时，浏览器只有在请求的路径与声明的路径相匹配时才会发送Cookie。这有助于确保某个应用程序的Cookie不会被同一服务器上不同路径的其他应用程序访问。但这种保护机制存在缺陷：如果“other”应用程序存在跨站脚本攻击（XSS）漏洞，攻击方可以通过嵌入iframe来绕过这一“路径”限制。
- 应设置“secure”标志，以确保仅通过安全通道（TLS）传输。
- 应设置HttpOnly标志以防止Cookie通过JavaScript访问。
- 为Cookie添加“samesite”属性可以防止某些现代浏览器在跨站请求时自动发送Cookie，从而提供对跨站请求伪造和信息泄露攻击的防护。

漏洞预防

- A07:2021 – Identification and Authentication Failures
- OWASP Mobile Top 10 2016-M4- Insecure Authentication

参考资料

- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Choosing and Using Security Questions
- OWASP Cheat Sheet: Session Management
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

工具

- Daniel Miessler: Most commonly found passwords

C8:协助浏览器守护用户安全

描述

浏览器是大多数用户访问网络的入口。因此，采用强有力的安全措施以保护用户免受各种威胁至关重要。本节概述了可以采用的技术和策略，以增强浏览器的安全性。

尽管目前主要关注传统的网络浏览器，但值得提醒的是，还有许多其他类型的客户端程序，比如API客户端和智能电视。

机会主义安全与浏览器支持

网络浏览器采取的安全措施本质上是机会主义的：网络应用无法验证浏览器是否遵循给定的指导原则。因此，这些安全措施应当始终作为额外的（可选的）强化措施，旨在进一步增加攻击方难度。

另外，网络浏览器必须做到真正支持网络应用提供的安全指导。由于不同浏览器及其版本对安全指导的支持程度各异，可以通过访问网站如https://caniuse.com来查询特定网络浏览器（版本）是否支持某项功能。随着时间的推移，浏览器支持的安全特性可能会发生变化。例如，所有主流浏览器都已经移除了X-XSS-Protection响应头；浏览器默认设置也可能发生变化，如Referrer-Policy；甚至现有响应头的语义也可能随时间而变化，如X-Content-Type-Options。

尽管浏览器功能的变化可能带来问题，但新版浏览器通常提供更多的安全特性，并且这些特性可能已经默认启用。显式设置这些安全头部（headers）可以统一不同浏览器的行为，简化维护工作。

一个完成受损的浏览器可能不遵循安全指导。但如果攻击方完全控制了浏览器，意味着攻击方将拥有比忽略安全指导更具破坏性的攻击手段。

实施

通常，网络应用可以通过两种特定于安全标头的方式来指示网络浏览器处理安全事项：HTTP标头和HTML标签。

如果应用多次指定安全指令，其处理方式依据具体的安全头部特性而定。例如，重复设置X-Frame-Options头部会使其保护失效，而重复设置Content-Security-Policy头部则会让策略更严格，从而增强安全性。以下是一些潜在的加固措施列表：

配置浏览器以防止信息泄露

当浏览器通过未加密的通道（使用HTTP而非HTTPS）传输信息或者一开始即发送过多信息（如通过Referer头部）时，就可能发生信息泄露。以下是一些降低信息泄露风险的方法：

【HTTP严格传输安全】

确保浏览器仅通过HTTPS访问该网站，防止SSL剥离攻击。

【内容安全策略】

该策略可以指示浏览器自动将HTTP连接升级为HTTPS。此外，还可以利用如form-src指令来防止表单向外部网站传输数据。

【引用策略】

当用户在网页间跳转时，浏览器发出的HTTP请求中会包含当前URL，而此URL可能含有敏感信息。通过使用Referrer-Policy，网站可以统一浏览器的行为，并选择哪些信息应在网站之间传输。

【Cookie的secure标志】

虽然不是HTTP标头信息，但此安全标志对于防范信息泄露至关重要。如果设置了Secure标志，浏览器将不会通过未加密的HTTP传输发送Cookie。

减少跨站脚本攻击（XSS）的潜在影响

基于Javascript的XSS攻击在近几十年来一直很普遍。为了降低该漏洞潜在的风险，浏览器提供了丰富的防御机制，以降低XSS攻击的潜在影响：

【内容安全策略】

CSP是一种强而有效的工具，能广泛抵御包括跨站脚本攻击（XSS）和数据注入在内的多种攻击。严格的CSP策略可以有效禁用内联JavaScript和样式，显著提升攻击方注入恶意内容的难度。

【主机白名单】

禁止所有第三方JavaScript的加载，可以大幅度缩小攻击面，并阻止利用第三方库中存在的漏洞。

【可信类型】

这是一种浏览器API，通过确保仅有安全的数据类型才能插入到文档对象模型(DOM)中，有效防御基于DOM的跨站脚本安全漏洞。

【Cookie的httpOnly属性】

虽然不属于HTTP标头信息，但启用此属性可阻止Javascript访问特定的Cookie，尤其推荐对会话Cookie使用此属性，以增强安全性。

防止点击劫持

点击劫持又称为界面伪装攻击，通过在合法网页上覆盖恶意界面以混淆用户。用户认为在与合法网站交互，而实际上是在与恶意网站交互。

【X-Frame-Options】

此头部通过限制内容不允许嵌入在其他站点以防止点击劫持攻击。XFO的使用较为敏感，重复使用会失效。

【内容安全策略】

通过其多个frame-*指令，提供细粒度控制能力，哪些站点允许嵌入当前站点，以及当前站点允许嵌入哪些其他站点的内容。

控制浏览器的高级特性

现代浏览器功能不仅限于呈现HTML页面，还能与多个系统组件交互（例如，摄像头、麦克风、USB设备等）。尽管许多网站不使用这些高级功能，但攻击方可以滥用这些功能来执行恶意操作。

【权限策略】

该策略允许网站告知浏览器网站不会使用某些特定功能。例如，网站可以声明自己不会使用捕捉用户音频的功能。这种情况下，即便攻击方成功注入恶意代码，也无法操控浏览器执行音频采集等未经授权操作。

防止CSRF攻击

CSRF攻击利用了网络浏览器与网站之间的信任关系。

【同源Cookie】

设置Cookie的SameSite属性可以减少跨源信息泄露的风险，并有助于防止跨站请求伪造攻击（CSRF）。

漏洞预防

实施以上浏览器防御措施可以有效减少一系列漏洞，包括但不限于：

- 跨站脚本攻击 (XSS)
- 跨站请求伪造 (CSRF)
- 点击劫持
- 通过不安全的传输窃取数据
- 会话劫持
- 滥用未预期的浏览器硬件访问（麦克风、摄像头等）

参考资料

- Content Security Policy (CSP)
- OWASP Secure Headers Project
- Fetch Metadata Request Headers
- https://caniuse.com/

工具

- Web Check
- Security Headers
- Mozilla Observatory

C9:实现安全日志记录和监控

描述

日志记录对于大多数开发人员，是一个早已用于调试和诊断的概念。而安全日志记录是一个同样基本的概念：在应用程序运行期间记录安全信息。监控则是通过各种自动化形式实时查看应用程序和安全日志。相同的工具和模式可以用于操作、调试和安全目的。

安全日志记录的好处

安全日志记录可以用于以下目的：

- 支持入侵检测系统的运作。
- 协助进行取证分析与调查。
- 确保满足法规合规要求。

入侵检测与响应的日志记录

使用日志记录来识别用户可能存在的恶意行为。需要记录的潜在恶意活动包括：

- 提交的数据超出预期的数值范围。
- 提交的数据涉及不应该修改的数据（例如选择列表、复选框或者其他有限输入组件）。
- 请求违反服务器端访问控制规则。

Application Logging Vocabulary Cheat Sheet提供了更全面的可能检测点列表。当应用程序遇到此类活动时，应用程序至少应该记录这些活动并将其标记为高严重性问题。理想情况下，应用程序还应该对可能的攻击做出响应，例如使用户会话失效并锁定用户账户。响应机制允许软件实时对可能识别的攻击做出反应。

安全日志设计

日志解决方案必须以安全的方式构建和管理。安全日志设计可能包括以下内容：

- 仅允许预期字符和/或基于目标进行输入编码，以防止日志注入攻击log injection。首选方法是日志解决方案执行输入转义而不是丢弃数据：否则日志解决方案可能会丢弃后续分析所需的数据。
- 请勿记录敏感信息。例如，请勿记录密码、会话ID、信用卡或者身份证号码。
- 保护日志完整性。攻击方可能会试图篡改日志。因此，应该考虑日志文件的权限和日志更改的审计。
- 将分布式系统的日志转发到集中、安全的日志服务。这不仅能确保在单一节点受到攻击时日志数据不会丢失，还能支持集中化或自动化的监控管理。

实施

以下是安全日志实施的最佳实践列表：

- 在系统内部和组织的各个系统之间应遵循统一的日志格式和方法。例如，Apache Logging Services 是一种常用的日志框架，有助于在 Java、PHP、.NET 和 C++ 应用程序之间提供日志一致性管理。

- 日志记录不能过多或者过少。例如，确保始终记录时间戳和识别信息，包括源IP和用户ID，但要同时应谨慎处理，避免记录敏感私人信息（如用户名）或者机密数据（如业务数据），除非已经采取了额外的保护措施。
- 密切关注节点之间的时间同步，确保时间戳一致。

漏洞预防

- 针对登录机制的暴力破解攻击

参考资料

- Logging Cheat Sheet
- OWASP Logging Guide

工具

- N/A

C10:阻止服务器端请求伪造 (SSRF)

描述

虽然注入攻击通常直接针对受害服务器本身，但服务器端请求伪造 (SSRF) 攻击则是试图强制服务器代表攻击方执行请求。这种攻击方式对攻击方来说有着显著的利益：出站请求将以受害服务器的身份执行，因此攻击方可能以提升的权限级别执行操作。以下是一些示例：

- 如果可以在DMZ内的服务器进行SSRF攻击，那么攻击方无需通过外围防火墙即可访问DMZ内的其他服务器。
- 许多服务器在本地上运行服务，通常不进行任何身份验证/授权。SSRF攻击方可以利用这些服务来进一步攻击系统。
- 如果使用单点登录 (SSO)，SSRF可以用来从服务器中提取令牌/票证/哈希。

实施

有多种方法可以预防SSRF：

- 输入验证
- 如果必须执行出站请求，检查并确保请求目标在允许列表中
- 如果使用XML，确保安全配置解析器以防止XXE攻击。在执行输入验证时要注意Unicode and other Character transformations。

漏洞预防

- A10:2021 – Server-Side Request Forgery (SSRF)

参考资料

- Server-Side Request Forgery Prevention Cheat Sheet 服务器端请求伪造预防备忘单
- A New Era of SSRF - Exploiting URL Parser in Trending Programming Languages! SSRF 新时代 - 利用流行编程语言中的URL解析器！

工具

- SSRFmap

关于OWASP

开源Web应用程序安全项目(Open Web Application Security Project, OWASP)是一个501c3非营利性教育慈善机构，致力于帮助组织能够设计、开发、获取、运营和维护安全的软件。OWASP提供的所有工具、文档、论坛和分会活动均免费开放，旨在吸引所有对提升应用程序安全性感兴趣的个人或组织，详细信息可访问www.owasp.org。

OWASP作为一种新型组织机构，始终坚持商业中立性，确保所提供的应用程序安全信息公正、实用且具成本效益。OWASP不隶属于任何技术公司，与许多开源软件项目类似，OWASP以协作和开放的方式产生了多种类型的材料。OWASP基金会是一个非营利性实体，确保项目的长期成功。

文档补充说明

- C2:正确应用加密技术，增加“国密说明”部分，简略介绍国密法律法规以及主流国密算法。
- C10:阻止服务器端请求伪造（SSRF），增加漏洞预防“A10:2021 – Server-Side Request Forgery (SSRF)”。