OWASP
Open Web Application
Security Project

面向开发人员
2018 v 3.0

10项软件开发人员须具备的关键安全开发意识

项目领导人

KATY ANTON
JIM MANICO
JIM BIRD

关于 OWASP

Open Web Application Security Project (OWASP) 是一个 501c3 非盈利性教育慈善机构，致力于使组织机构能够设计、开发、获取、运营和维护安全的软件。所有 OWASP 的工具、文档、论坛和区域分会都是免费的，并对任何有兴趣提高应用程序安全性的人士开放。我们的网站是：www.owasp.org 。

OWASP 是一个新型组织。我们没有商业压力，使得我们能够提供无偏见、实用、低成本的应用安全信息。

OWASP 不隶属于任何技术公司。和许多开源软件项目一样，OWASP 以一种协作、开放的方式制作了许多不同种类的材料。OWASP 基金会是确保项目长期成功的非营利性组织。

关于 OWASP 中国

OWASP 中国是 OWASP 安全组织在中国区域的分部，是全球 262 个区域分部之一。OWASP 中国成立于 2008 年，在过去 11 年的发展历程中，OWASP 中国已在国内形成了超过 20 个子区域（包括：北京、上海、广东、广西、浙江、江苏、陕西、山西、四川、吉林、辽宁、黑龙江等区域），并吸引了超过 6000 名行业专家成为 OWASP 中国区会员、3 家国内企业成为 OWASP 中国企业会员单位、3 家高等院校成为 OWASP 中国学术支持单位。

前言

不安全的软件正在破坏着我们的金融、医疗、国防、能源和其他全球重要基础设施。随着我们的软件变得愈加重要、复杂且相互关联，实现应用程序安全的难度也呈指数级增长。我们已经不能再忍受相对简单的安全问题了。

目标

OWASP Top 10 Proactive Controls (OPC) 项目的目标是描述软件开发人员必须注意的最重要领域，提高开发人员对应用程序安全性的认识。我们鼓励大家使用 OWASP OPC 项目让开发人员关注和做好应用安全。开发人员可以从其他组织的错误中吸取教训。我们希望 OWASP OPC 对软件研发团队构建安全的软件有所帮助。

联系我们

若您有关于 OWASP OPC 项目的任何疑问、评论和想法，欢迎直接和我们联系，jim@owasp.org。

版权与许可

项目领导者

Katy Anton　　Jim Bird　　Jim Manico

项目贡献者

Chris Romeo	Dan Anderson	David Cybuck
Dave Ferguson	Josh Grossman	Osama Elnaggar
Colin Watson	Rick Mitchell	其他人

中文项目

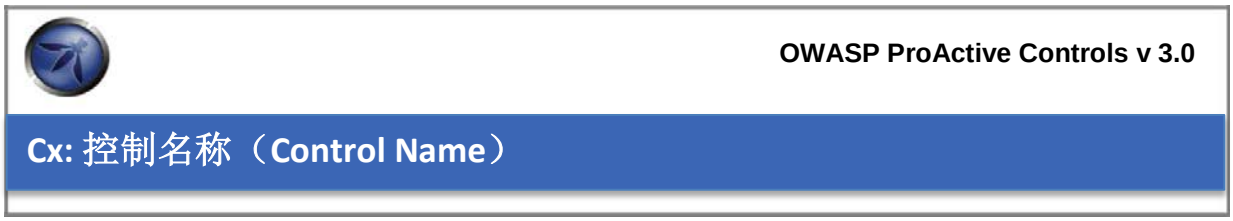
项目负责人：王颉

项目组成员：郭振新、李瑞、李永凯、秦波、张京河

注：由于中文项目组成员翻译水平有限，存在的错误敬请指正。如有任何意见或建议，可联系我们。邮箱：project@owasp.org.cn

文档结构

本文档提供了安全控制的列表。每个控制的描述如下:



描述

有关控制机制的详细描述,包括需要考虑的一些最佳实践。

实现方法

实现的最佳实践和示例,以说明如何实现每个控制机制。

预防的缺陷

-
-

已预防缺陷或已消除风险的列表（OWASP TOP 10、CWE 等）。

参考文献

-
-

提供进一步研究的参考文献列表（OWASP Cheat Sheet、Security Hardening Guidelines 等）。

工具

-
-

能把安全控制机制轻松引入或集成到软件中的工具项目集。

简介

OWASP 十大主动控制 2018 是一组每个软件开发项目都应考虑和实现的安全技术列表。本文档是为开发人员编写的,以帮助刚接触安全开发的开发人员。

本文档的主要目标之一是提供具体的实践指导，以帮助开发人员开发安全的软件。这些控制机制应在软件开发的早期阶段积极应用，以确保能产生最大的效力。

十大主动控制

本列表按重要性排序，其中第 1 个列表项是最重要的:

C1：定义安全需求（Define Security Requirements）

C2：使用安全框架和库（Leverage Security Frameworks and Libraries）

C3：安全的数据库访问（Secure Database Access）

C4：数据编码与转义（Encode and Escape Data）

C5：验证所有输入（Validate All Inputs）

C6：实现数字身份（Implement Digital Identity）

C7：实施访问控制（Enforce Access Controls）

C8：保护所有的数据（Protect Data Everywhere）

C9：实施安全日志记录和监控（Implement Security Logging and Monitoring）

C10：处理所有错误和异常（Handle All Errors and Exceptions）

本列表的创建方式

此列表最初由当前的项目负责人创建，并由几位志愿者参与。然后，本文档在全球范围内共享，因此，全球范围内的任何反馈建议都可以考虑，包括匿名建议。目前，本项目已接受了来自 OWASP 社区超百项建议。

本文档的目标受众

本文档主要是写给开发人员参考学习的。但是，开发经理、产品所有人、Q/A 专业人员、项目经理和参与构建软件的任何人都可以从本文档中受益。

如何使用本文档

本文档旨在提供有关构建安全软件的初始意识。另外，本文档还为帮助推动入门性软件安全开发人员培训提供了良好的基础素材。这些控制应在所有应用程序中被一致、彻底地使用。然而，本文档应被视为一个抛砖引玉的起点，而不是一套全面的技术和做法。完整的安全开发过程应包含来自于行业标准（如：OWASP ASVS）中的全面要求，以及在软件安全开发成熟度模型(如：OWASP SAMM 和 BSIMM)中描述的一系列软件安全开发实践活动。

本项目与 OWASP Top 10 项目的关联关系

OWASP 十大主动控制项目与 OWASP Top 10 项目相似，但本项目更聚焦于防御技术和控制机制，而不是风险。本文档中的每个技术或控制都将映射到 OWASP Top 10 中的一个或多个风险项。此映射信息包含在每个控制描述的末尾。

OWASP ProActive Controls v 3.0

C1: 明确安全需求 Define Security Requirements

描述

安全需求是对所需安全功能的声明，以确保软件各不相同的安全属性均能得到实现。安全需求来源于行业标准、适用法律和历史安全漏洞。安全需求定义了新特性或者是为了解决特定的安全问题或消除潜在的漏洞，而对现有功能提出额外要求。

安全需求为应用程序提供审核安全功能的基础。统一标准的安全需求容许开发人员复用安全控制措施和最佳实践，而不是单独每个应用程序自定义安全实现。经过审核的安全需求为历史安全缺陷提供解决方案。需求的存在防止之前安全教训的重演。

OWASP ASVS 项目

OWASP 应用软件安全级别验证参考标准（ASVS）项目提供了有效的安全需求和验证标准，可以作为开发团队详细安全需求的来源。

安全需求根据更高级别的安全功能分为不同的要点。例如：ASVS 包含身份验证、访问控制、错误处理/日志记录和 Web 服务等类别。每个类别都包含一组代表可验证需求的最佳实践。

通过用户故事（user stories）和误用案例（misuse cases）扩展需求

ASVS 需求是基本的可验证声明，可以通过用户故事和误用案例加以扩展。用户故事或误用案例的优势在于它将应用程序与用户或攻击者对应用系统的具体操作结合起来，而不是描述系统为用户提供了什么。

下面是一个扩展 ASVS 3.0.1 需求的例子。在 ASVS 3.0.1 的“验证和认证需求”章节中，需求 2.19 主要关注默认密码。

> 2.19 验证应用程序框架或应用程序使用的任何组件均没有使用默认密码。(如“admin/password”)

该需求包含验证是否存在默认密码的操作，同时提供在应用程序中不应存在默认密码的附加指导。

用户故事关注系统的用户、管理员或攻击者的视角，并根据用户希望系统做什么来描述功能。用户故事的形式是“作为一个用户，我可以做 x、y 和 z”。

> 作为用户，我可以输入我的用户名和密码来访问应用程序。
>
> 作为用户，我可以输入最长为 1023 个字符的长密码。

当故事聚焦于攻击者及其行为时，就会被称为误用案例。

> 作为攻击者，我可以输入默认用户名和密码来访问系统。

上面的用户故事包含了来自 ASVS 的传统需求一致的信息，以及额外用户或攻击者的详细信息，可以帮助需求更具备可测试性。

实现方法

安全需求的成功实施包含四个步骤。该过程包括发现/选择、记录、实现和确保应用程序中新安全特性和功能得以正确实现。

发现和选择

该过程从安全需求的发现和选择开始。在此阶段，开发人员将理解来自标准来源（如 ASVS）的安全需求，并为给定的应用程序版本选择包含哪些需求。发现和抉择的关键是为本次版本或 Sprint 选择一个可供管理的安全需求数量，然后在每个 Sprint 继续迭代，以随着时间的推移增加更多的安全功能。

调查和记录

在调查和编制文档期间，开发人员根据新的一组安全需求审视当前应用程序是否满足需求，是否需要进一步开发。这类调查的交付件是将审查结果记录在案。

实现和测试

确定要实现的需求后，开发人员必须采取措施修改应用程序来添加新功能或消除不安全项。在此阶段中，开发人员需要首先确定软件设计，然后完成代码修改以满足需求。应该创建测试用例来确认新功能准确完成，或者验证之前的不安全项得到解决。

预防的缺陷

安全需求定义应用程序的安全功能。在应用程序生命周期一开始就构建更好的安全性可以防范众多类型漏洞的发生。

参考文献

- OWASP Application Security Verification Standard (ASVS)
- OWASP Mobile Application Security Verification Standard (MASVS)
- OWASP Top Ten

OWASP ProActive Controls v 3.0

C2: 使用安全的框架和软件库 Leverage Security Frameworks and Libraries

描述

具备一定安全性的安全开发代码库和软件框架，能够帮助软件开发人员防范安全相关的设计和编码缺陷。从零开始编写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现或维护安全特性。使用安全框架有助于更加有效、准确地实现安全目标。

实现的最佳实践

当在软件中引入第三方库或框架时，必须考虑以下最佳实践:

1. 使用来源可信的、被积极维护的、被大量应用程序广泛使用的库和框架。
2. 创建和维护所有第三方库的列表。
3. 主动保持库和组件的最新版本。使用像 OWASP Dependency Check 和 Retire.JS 这样的工具来识别项目依赖关系，并检查所有第三方代码是否存在已知的、公开披露的漏洞。
4. 通过封装库来减少攻击面，并且只将必需的行为暴露到软件中。

预防的缺陷

安全框架和库有助于防止各种 web 应用程序漏洞。正如 2017 年版 OWASP Top 10 中所描述"使用具有已知漏洞的组件"那样，保持这些框架和库的最新版本至关重要。

工具

- OWASP Dependency Check，用于识别项目组件依赖并检查公开披露的漏洞
- Retire.JS，用于 JavaScript 库的扫描器

OWASP ProActive Controls v 3.0

C3: 安全的数据库访问 Secure Database Access

描述

本节介绍对所有数据存储的安全访问，包括关系数据库和 NoSQL 数据库。需要考虑的方面主要有：

1.安全查询
2.安全配置
3.安全认证
4.安全通信

安全查询

当非可信用户的输入以不安全的方式动态添加到 SQL 查询中时，通常通过基本字符连接引起 SQL 注入。SQL 注入是最危险的应用程序安全风险之一。SQL 注入很容易被利用，并可能导致整个数据库被盗、被擦除或被修改。存在上述漏洞的应用程序，甚至可以被用来让托管数据库的操作系统执行危险命令，从而使攻击者在您的网络上获得立足点。

为了消减 SQL 注入风险，我们应阻止不受信任的输入被解释为 SQL 命令的一部分。实现这一点的最佳方法是使用名为“参数化查询”的编程技术。这种防御应当应用于 SQL、OQL 以及存储过程构造。

建议通过 http://bobby-tables.com 和 OWASP Cheat Sheet on Query Parameterization，来获取针对 ASP、ColdFusion、C#、Delphi、.NET、Java、Perl、PHP、PL/SQL、PostgreSQL、Python、Ruby 和 Scheme 等语言实现参数化查询的例子。

参数化查询的注意事项

数据库查询语句中的某些位置是不可参数化的。而且对于每个数据库供应商，这些位置是不同的。当遇到无法绑定到参数化查询语句的参数时，一定要非常小心地执行精确匹配验证或手动转义。此外，使用参数化查询在很大程度上对性能有积极的影响，

然而在某些数据库实现中的特定参数化查询会对性能产生负面影响。一定要测试查询语句的性能，特别是带有大量 like 或文本搜索功能的复杂查询语句。

安全配置

数据库管理系统并不总是提供“默认安全”配置。必须注意确保启用并且正确配置了数据库管理系统（DBMS）和托管平台提供的安全控制措施。对于常见的 DBMS，有许多可用的标准、指南和基线指导如何实现安全配置。

安全认证

对数据库的任何访问都应该通过恰当的认证。对 DBMS 的认证应该以安全的方式完成，并只能通过安全通道进行身份验证。认证凭据应被合理的保护并且确保可用。

安全通信

大多数 DBMS 支持多种安全的（已验证的、加密的）和不安全的（未验证的、未加密的）通信方法（如：服务、API 等）。根据“保护所有的数据”控制原则，对应上述情况的一个良好实践便是，仅使用安全的通信选项。

预防的缺陷

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

参考资料

- OWASP Cheat Sheet: Query Parameterization
- Bobby Tables: A guide to preventing SQL injection
- CIS Database Hardening Standards

OWASP ProActive Controls v 3.0

C4: 数据编码与转义 Encode and Escape Data

描述

编码和转义是用于阻断注入攻击的防御技术。编码（通常也被称为“输出编码”）是指将特殊字符转换为某种不同但等价的、并且能在目标解释器中无害的形式。例如，在写入 HTML 页面时，将字符“<”转换为字符串“<”。转义是指在字符或字符串之前添加一个特殊字符，以避免被错误解释。例如，在“'”(双引号)字符之前添加一个“\”字符，以便将其解释为文本，而不是结束字符串。

输出编码最好是在内容传递给目标解释器之前应用。如果在处理请求时过早地执行此防御，则编码或转义可能会干扰程序其他部分中内容的使用。例如，如果 HTML 在数据库存储数据前转义内容，而 UI 对该数据自动进行第二次转义，那么由于重复转义，内容将无法正确显示。

上下文输出编码

上下文输出编码是阻止 XSS 攻击的一种重要的安全编程技术。在构建用户接口时，上下文输出编码，在不受信任的数据动态添加到 HTML 的最后一刻执行。编码类型将取决于文档中显示或存储数据的位置（或上下文）。用于构建安全用户接口的不同编码类型，包括 HTML 实体编码、HTML 属性编码、JavaScript 编码和 URL 编码。

Java 编码样例

OWASP Java 编码器可提供上下文输出编码，例子参见：OWASP Java Encoder Project Examples。

.NET 编码样例

从.NET 4.5 开始，反跨站脚本库就已经是.NET 框架的一部分，但是，默认情况下并不开启这个功能。可以通过使用 web.conf 设置，来指定使用此库中的 Antixssencoder，作为整个应用的默认编码器。一旦使用，对输出的上下文进行编码非常重要——这意味着可以使用 AntiXSSEncoder 库中的正确函数对数据在适当的位置进行编码。

PHP 编码样例

Zend Framework 2

在 Zend Framework 2 (ZF2)中，Zend\Escaper 类可以被用来对输出进行编码。对上下文编码的例子参见 Context-specific escaping with zend-escaper。

其他类型的编码和注入防御

编码与转义可用于使其他形式的注入内容失效。例如，在向操作系统命令添加输入时，编码可以使某些特殊的元字符免于执行。这被称为“OS 命令转义”、“shell 转义”或其他类似的名称。上述防御方法可以用来阻止“命令注入”漏洞。

还有其他形式的转义可用于阻止注入，例如 XML 属性转义阻止各种形式的 XML 和 XML 路径注入，以及 LDAP 区别名称（distinguished name）转义可用于阻止各种形式的 LDAP 注入。

字符编码和标准化

Unicode 编码是一种以多字节存储字符的方法。只要容许输入数据，就使用 Unicode 格式输入从而得以伪装恶意代码并执行各种攻击。RFC2279 引用了许多可以对文本进行编码的方法。

标准化是一种将系统数据转换为简单或标准形式的方法。Web 应用程序通常使用字符标准化来确保存储或显示时所有内容都是相同的字符类型。

为了防止与规范化相关的攻击，意味着应用程序在输入格式错误的 Unicode 和其他错误格式的字符表示形式时应该也是安全的。

预防的缺陷

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)
- OWASP Mobile_Top_10_2014-M7 Client Side Injection

参考资料

- XSS - General information
- OWASP Cheat Sheet: XSS Prevention - Stopping XSS in your web application
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

工具

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper - examples of contextual encoding

OWASP ProActive Controls v 3.0

C5: 验证所有输入 Validate All Inputs

描述

输入验证是一种确保只有正确格式的数据才能进入软件系统组件的编程技术。

语法和语义的有效性

应用程序应该在以任何方式使用数据之前（包括显示给用户），检查数据语法和语义的有效性（按此顺序）。

语法有效性表示数据采用预期的形式。例如：应用程序可以允许用户选择一个 4 位的"账户 ID"去执行某种操作。应用程序应该假设用户正在输入一个 SQL 注入的 payload，并且应该检查用户输入的数据是否恰好为 4 位数字，并且只由数字组成(除了使用适当的查询参数之外)。

语义有效性包括只接受给定应用程序功能和上下文中可接受范围内的输入。例如，当选择时间范围时，选择的开始日期必须在结束日期之前。

白名单 vs 黑名单

执行输入语法验证有两种常用的方法，通常称为黑名单和白名单：

- 黑名单或者黑名单验证尝试检查输入的数据是否包含"已知非法"的内容。例如，Web 应用程序为了防止 XSS 可能会阻止包含确定大写字符<SCRIPT>的输入。但是使用小写或混合大小写的 script 标签可以绕过这种防御。
- 白名单或者白名单验证尝试检查输入的数据是否符合一组"已知合法"的规则。例如，美国州名缩写的白名单验证规则是必须是两个字符且必须是有效的美国州名缩写之一。

当构建安全软件时，建议采用最小限度的白名单的方式。黑名单很容易出错，并且可以通过各种规避技术绕过，如果依赖于黑名单本身则会很危险。尽管黑名单常常可以被绕过，但它通常有助于检测明显的攻击。因此，白名单通过确保数据具有正确的语法和语义的有效性来帮助限制攻击，黑名单有助于检测并且可能阻止明显的攻击。

客户端和服务器端验证

为了安全起见，输入验证必须始终在服务器端进行。虽然客户端验证对于功能性和某些安全性目的都很有用，但通常可以被轻松绕过。这使得服务器端验证对安全性更加重要。例如，JavaScript 验证可能会警告用户某个特定字段必须由数字组成，但是服务器端应用程序必须验证提交的数据只由符合该功能的数字范围内的数字组成。

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。让我们从一个基础的例子开始。

下面的正则表达式用于定义一个验证用户名的白名单。

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

这个正则表达式只允许小写字母、数字和下划线字符。用户名的长度也被限制为 3 到 16 个字符。

警告:潜在的拒绝服务（DoS）

在创建正则表达式时应小心。设计不良的正则表达式可能导致潜在的拒绝服务条件（即 ReDoS）。各种工具都可以测试正则表达式是否不容易受到 ReDoS 的攻击。

警告:复杂性

正则表达式只是完成验证的一种方法。对于某些开发人员来说，正则表达式很难维护或理解。其他验证方法包括以编程的方式编写验证方法，这对某些开发人员来说更容易维护。

输入验证的限制

输入验证并不总是使数据“安全”，因为某些形式的复杂输入可能是“合法的”，但仍然是危险的。例如，一个合法的电子邮件地址可能包含 SQL 注入攻击，或者一个合法的 URL 可能包含跨站脚本攻击(XSS)。除了输入验证之外，应该始终对数据进行额外的防御，比如查询参数化或转义。

验证序列化数据的挑战

某些形式的输入非常复杂，验证只能最低限度地保护应用程序。例如，反序列化不可信的数据或被攻击者操纵的数据是危险的。唯一安全的体系结构模式是不接受来自不可信源的序列化对象，或者只对简单数据类型进行有限容量的反序列化。您应该避免处理序列化的数据格式，并尽可能使用更容易保护的格式，例如使用 JSON。

如果这是不可能的，那么在处理序列化数据时应考虑一系列的验证防御机制。

- 实现序列化对象的完整性检查或加密，以防止恶意对象创建或数据篡改。
- 在对象创建之前的反序列化过程中执行严格的类型约束；典型的代码被期望为一组可定义的类。目前，这种技术已经证明了是可绕过的。
- 隔离反序列化的代码，使其在非常低的权限环境（如临时容器）中运行。
- 日志中记录安全性的反序列化异常和失败，例如传入的类型不是预期的类型，或者反序列化引发了异常。
- 限制或监视来自反序列化容器或服务器的传入和传出的网络连接。
- 监视反序列化，如果用户不断反序列化，则发出警报。

意外用户输入(批量赋值)

一些框架支持将 HTTP 请求参数自动绑定到应用程序使用的服务器端对象中。这个自动绑定功能允许攻击者更新不打算修改的服务器端对象。攻击者可以使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。

此攻击有许多名称，包括：批量赋值、自动绑定、对象注入。

一个简单的例子：如果用户对象具有权限字段，该权限字段指定应用程序中用户的权限级别，恶意用户可以查找修改用户数据的页面，并将“权限=管理员”(privilege=admin)添加到发送的 HTTP 参数中。如果以不安全的方式启用自动绑定，则用户的服务器端对象将进行相应地修改。

可以使用两种方法来处理这个问题：

- 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，但是为每个页面或功能设置白名单规则，定义哪些字段允许自动绑定。

更多的例子请查看：OWASP Mass Assignment Cheat Sheet

验证和净化 HTML

思考一个需要从用户接受 HTML 的应用程序（通过 WYSIWYG 编辑器，该编辑器将内容显示为 HTML 或有功能直接接受输入的 HTML）。在这种情况下，验证或转义将不起作用。

- 正则表达式的表达能力不足以理解 HTML5 的复杂性。
- 编码或转义 HTML 将没有帮助，因为它将导致 HTML 无法正确显示。

因此，您需要一个能够解析和清理 HTML 格式文本的库，有关 HTML 净化的更多信息，请查看 XSS Prevention Cheat Sheet on HTML Sanitization。

库和框架中的验证功能

所有语言和大多数框架都提供了验证的库或函数，应该利用这些库或函数来验证数据。验证库通常包括常见数据类型、长度要求、整数范围、“是否为空”的检查等。许多验证库和框架允许您定义自有的正则表达式或逻辑，以实现自定义的验证，从而允许程序员在整个应用程序中利用该功能。验证功能的示例包括 PHP 的 filter functions 或 Java 的 Hibernate Validator 。HTML 净化包括 Ruby on Rails sanitize method, OWASP Java HTML Sanitizer 和 DOMPurify.

预防的缺陷

- 输入验证减少了应用程序的攻击面，有时会使针对应用程序的攻击更加困难。
- 输入验证是一种为特定形式的数据提供安全性的技术，特定于某些攻击，并且不能作为一般性安全规则可靠地应用。
- 输入验证不应该被用作防御 XSS, SQL Injection 和其它攻击的主要方法。

参考资料

- OWASP Cheat Sheet: Input Validation
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs

- OWASP Testing Guide: Testing for Input Validation

工具

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- PHP's filter functions

OWASP ProActive Controls v 3.0

C6: 实现数字身份 Implement Digital Identity

描述

数字身份是用户(或其他主体)在进行在线交易时的唯一标识。身份认证是验证个人或实体是否是他们所声称的身份的过程。会话管理是服务器维护用户身份认证状态的过程，以便用户可以继续使用系统而无需重新进行身份认证。《NIST Specila Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management》为实现数字身份、身份认证和会话管理控制提供了可靠的指导。

下面是一些安全实现的建议。

认证等级

NIST 800-63b 描述了身份认证保证的三个等级，被称为身份认证保证等级(AAL)。AAL 等级 1 保留给不包含个人识别信息（PII）或其他私有数据的低风险应用程序。AAL 等级 1 只需要单因素的身份认证，通常通过使用密码来进行身份认证。

等级 1：密码

密码非常非常重要。我们需要策略，我们需要安全的存储它们，我们有时候需要允许用户重置它们。

密码要求

密码至少应符合下列规定:

- 如果同时使用多因素身份认证(MFA)和其他控制，则长度至少为 8 个字符。如果没有使用 MFA，则应该增加到至少 10 个字符;
- 所有 ASCII 字符以及空格字符在密码中都应该是可接受的;
- 鼓励使用长密码和口令;
- 删除复杂性要求，因为这些要求的有效性是有限的。相反，建议采用 MFA 或更长的密码长度;

- 确保使用的密码不是在以前已经泄露的常用密码中。您可以选择阻止前 1000 或 10000 个最常见的密码，这些密码满足上述长度要求，并且可以在密码泄露列表中找到。下面链接包含最常见的密码：https://github.com/danielmiessler/SecLists/tree/master/Passwords

实现安全密码恢复机制

应用程序通常具有一种机制，在忘记密码的情况下，用户可以访问他们的帐户。一个设计优秀的密码恢复功能的工作流将使用多因素身份认证元素。例如，它可能会问一个安全问题 —— 用户知道的东西，然后将生成的令牌发送到一个设备 —— 用户拥有的东西。

更多详细信息，请查看 Forgot_Password_Cheat_Sheet 和 Choosing_and_Using_Security_Questions_Cheat_Sheet 。

实现安全密码存储

为了提供强大的身份认证控制，应用程序必须安全地存储用户凭据。此外，应该设置加密控制，防止在凭证(例如密码)受到泄露时攻击者立即获得该信息。

用于密码存储的 PHP 示例

下面是一个在 PHP 中使用 password_hash()函数（从 5.5.0 版本开始可用）进行安全密码散列计算的示例，默认使用 bcrypt 算法。该示例使用的工作因子为 15。

```
<?php
 $cost = 15;
 $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" => $cost] );
?>
```

更多信息请查看 OWASP Password Storage Cheat Sheet 。

等级 2: 多因素身份认证

NIST 800-63b AAL 等级 2 是为包含“自我声明的个人识别信息（PII）或在线提供的其他个人信息”的高风险应用程序而预留的。在 AAL 等级 2 中，需要进行多因素身份认证，包括动态口令（OTP）或实现其他形式的多因素身份认证。

多因素身份认证（MFA）要求用户通过以下组合来证明自己，从而确保用户是他们声称的用户：

- 你知道的东西 – 密码或个人识别码（PIN）
- 你拥有的东西 – 令牌或电话
- 你自己 – 生物特征，比如指纹

将密码作为唯一因素提供了较弱的安全性。多因素解决方案要求攻击者必须获取多个元素来使用服务进行身份认证，从而提供了更健壮的解决方案。

值得注意的是，当使用生物识别技术作为单一的身份认证因素时，它并不被认为是数字身份认证可接受的方式。它们可以在网上获得，也可以在某人知情或不知情的情况下，用照相手机给他拍照(比如面部图像)，从某人触摸的物体上提取(比如潜在的指纹)，或者用高分辨率图像捕捉(比如虹膜图案)。生物识别只能作为使用物理身份认证器(您拥有的东西)进行多因素身份认证的一部分。例如，访问一个多因素一次性密码(OTP)设备，该设备将为用户生成手动输入的一次性密码用来进行认证。

等级 3:基于加密的身份认证

当系统被攻破后可能导致人身伤害、重大经济损失、损害公共利益或涉及民事或刑事违规时，则必须使用 NIST 800-63b 认证保证等级 3 (AAL3)。AAL3 要求是“基于通过加密协议拥有密钥的证据”的身份认证。这种类型的身份认证用于实现保证最强等级的身份认证。这通常是通过硬件加密模块完成的。

会话管理

一旦用户初始化身份认证成功，应用程序就可以选择在有限的时间内跟踪和维护这种身份认证状态。这将允许用户继续使用应用程序，而不必对每个请求进行重新身份认证。跟踪此用户状态称为会话管理。

会话的生成和过期

在会话中跟踪用户状态，此会话通常存储在服务器上，用于传统的基于 Web 的会话管理。然后向用户提供会话标识符，以便用户能够识别哪个服务器端会话包含正确的用户数据。客户端只需要维护这个会话标识符，它还将敏感的服务器端会话数据与客户端隔离。

以下是构建或实施会话管理解决方案时要考虑的一些控制：

- 确保会话 id 是长的，唯一的并且随机的。
- 应用程序应该生成一个新的会话，或者至少在身份认证和重新身份认证期间更新会话 id。
- 应用程序应该在一段不活动时间和每个会话的绝对最大生命周期之后实现空闲超时，在此之后用户必须重新进行身份认证。超时时间的长度应该与受保护数据的价值成反比。

详细信息请查看 Session Management Cheat Sheet 。ASVS 第 3 节涵盖了额外的会话管理需求，见《软件安全开发指南—应用软件安全级别验证参考标准》第 5 章。

浏览器 Cookie

浏览器 cookie 是 Web 应用程序存储用于实现标准会话管理技术的会话标识符的常用方法。下面是使用浏览器 cookie 时需要考虑的一些防御措施。

- 当使用浏览器 cookie 来跟踪已经通过身份认证的用户的会话时，这些 cookie 应该只能够被最小范围的一组域和路径访问，并且应该标记为在到会话有效期时或之后不久过期。
- 应该设置“secure”属性，以确保传输只能通过安全通道(TLS)完成。
- 应该设置“HttpOnly”属性，以防止通过 JavaScript 访问 cookie。
- 在 cookie 中添加“samesite”属性可以防止一些现代浏览器发送带有跨站请求的 cookie，并且可以提供针对跨站请求伪造（CSRF）和信息泄漏攻击的保护。

令牌

服务器端会话可能会限制某些形式的身份认证。“无状态服务”允许客户端出于性能目的对会话数据进行管理，这样它们的服务器就可以减少存储和验证用户会话的负担。这些“无状态”应用程序生成一个短期访问令牌，可以使用该令牌对客户端请求进行身份认证，而无需在初始身份认证之后发送用户的凭据。

JWT（JSON Web 令牌）

JSON Web Token (JWT)是一个开放标准(RFC 7519)，它定义了一种紧凑且自包含的方式，应用于 JSON 对象在各方之间安全地传输信息。可以验证和信任此信息，因为它是数字

签名过的。JWT 令牌是在身份认证期间创建的，并在任何处理之前由服务器(或多个服务器)进行验证。

然而，在初始创建之后，服务器通常不会保存 JWT。JWT 通常创建并传递给客户端，而不需要由服务器以任何方式保存。令牌的完整性是通过使用数字签名来维护的，因此服务器之后可以验证 JWT 仍然有效，并且从创建以来没有被篡改。

这种方法既是无状态的，又是可移植的，因为客户端和服务器技术可以是不同的，但仍然可以交互的。

警告

数字身份、身份认证和会话管理是非常重要的主题。我们只触及了数字身份这个话题的皮毛。由于大多数身份解决方案的复杂性，请确保使用最有能力的工程师进行维护。

预防的缺陷

- OWASP Top 10 2017 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication

参考文献

- 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
- OWASP Cheat Sheet: IOS Developer
- OWASP Testing Guide: Testing for Authentication
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

工具

- Daniel Miessler: Most commonly found passwords

OWASP ProActive Controls v 3.0

C7: 实施访问控制 Enforce Access Controls

描述

访问控制（或授权）是授予或拒绝用户、程序或进程的特定请求的过程。访问控制还包含授予和撤销这些特权的行为。

应该注意的是，授权（验证对特定功能或资源的访问）并不等同于验证（验证身份）。

访问控制功能依据访问控制系统的复杂性，通常会跨越软件的许多领域。例如，出于可伸缩性目的管理访问控制元数据或构建缓存通常是访问控制系统中需要构建或管理的附加组件。

这里需要讨论如下几种不同类型的访问控制方法设计。

- 自主访问控制（DAC）是一种根据主体的身份及知其所需的内容（如用户、进程）或对象所属组来限制客体的访问（如文件、实体数据）的访问控制方法。
- 强制访问控制（MAC）是一种根据系统资源中包含的信息的敏感性（由标签表示）以及用户访问信息的正式授权（即许可）来限制对系统资源的访问的一种手段。
- 基于角色的访问控制（RBAC）是一种用于控制对资源的访问模型，该模型对资源的允许操作通过角色进行标识，而不是单独对客体进行标识。
- 基于属性的访问控制（ABAC）将根据用户的任意属性和对象的任意属性，以及可能被全局识别并与现有策略更相关的环境条件，授予或拒绝用户请求。

访问控制设计原则

在应用程序开发的初始阶段，应考虑以下“积极的”访问控制设计要求。

（1）提前设计访问控制

当您选择了一个特定的访问控制设计模式，在您的应用程序中再使用一种新的模式重新设计访问控制通常是困难且耗时的。访问控制作为应用程序安全设计的主要领域之一，必须预先实施彻底的设计，特别是在处理多租户和横向（数据相关的）访问控制等需求时。

访问控制设计可以从简单的开始，通常会成长为一个复杂的、功能强大的安全控制。在评估软件框架的访问控制能力时，请确保您的访问控制功能允许为您的特定访问控制功能需求实施定制化。

（2）强制所有请求通过访问控制检查

确保所有请求都通过某种访问控制的验证层。像 Java 过滤器或其他自动请求处理机制之类的技术是理想的编程构件，将有助于确保所有请求都通过某种访问控制的检查。

（3）默认拒绝

默认拒绝是一个原则，即，如果请求不被特别允许，则它将被拒绝。此规则在应用程序代码中有许多方法可以实现。如下是一些例子：

1. 应用程序代码在处理访问控制请求时可能会抛出错误或异常。在这种情况下，访问控制应始终保持拒绝。
2. 当管理员创建一个新用户或用户注册一个新帐户时，默认情况下该帐户的访问配置应为最小访问权限或无访问权限。
3. 向应用程序添加新功能时，在正确配置该功能之前，应拒绝所有用户使用该功能。

（4）最小特权原则

确保所有用户、程序或进程仅被授予最小或必要访问权限。应注意那些不提供细粒度访问控制配置功能的系统。

（5）不要硬编码角色

许多应用程序框架默认的访问控制方法为基于角色的访问控制（RBAC）。通常会找到包含了这种性质检查的应用程序代码。

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

在编码中要小心这种基于角色的编程。 它具有以下限制或危险。

- 基于角色的编程是脆弱的。代码中很容易创建不正确或丢失的角色检查。
- 基于角色的编程不允许多租户。为了使基于角色的系统对不同的客户具有不同的规则，需要为每个客户分配代码或添加检查等极端措施。
- 基于角色的编程不允许使用特定于数据的访问控制规则。
- 具有许多访问控制检查的大型代码库可能难以审核或验证整个应用程序访问控制策略。

相反，请考虑以下访问控制编程方法：

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

基于属性或特征的访问控制检查，是构建设计良好且功能丰富的访问控制系统的起点。这种类型的编程还允许随着时间的推移实现更强大的访问控制定制能力。

（6）记录所有访问控制事件

应记录所有访问控制失败情况，这些情况可能表示恶意用户正在探测应用程序的漏洞。

预防的缺陷

- OWASP Top 10 2017-A5-Broken Access Control
- OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication

参考文献

- OWASP Cheat Sheet: Access Control
- OWASP Cheat Sheet: iOS Developer - Poor Authorization and Authentication
- OWASP Testing Guide: Testing for Authorization

工具

- OWASP ZAP with the optional Access Control Testing add-on

OWASP ProActive Controls v 3.0

C8: 保护所有的数据 Protect Data Everywhere

描述

密码、信用卡号、健康记录、个人信息和商业机密等敏感数据需要额外保护。特别是如果这些数据属于隐私法（欧盟通用数据保护法规 GDPR），PCI 数据安全标准（PCI DSS）等财务数据保护规则或其他法规。

攻击者可以通过多种方式从 web 或 web 服务应用程序中窃取数据。 例如，如果通过互联网发送的敏感信息没有通信安全性，则攻击者可以通过共享无线连接上查看并窃取其他用户的数据。 此外，攻击者可以通过 SQL 注入攻击从应用程序数据库中窃取密码和其他凭据，并将该信息公开给公众。

数据分类

对系统中的数据进行分类并确定每个数据属于哪个级别的敏感度至关重要。不同类别的数据可以映射到每个敏感级别所需的保护规则。例如，可以将不敏感的公共营销信息分类为可放置在公共网站上的公共数据。将信用卡号分类为个人用户数据，并需要在存储或传输时被加密。

传输数据加密

在通过任何网络传输敏感数据时，都应该考虑端到端通信安全性（或传输加密）。TLS 是迄今为止最常用且最被广泛支持的通信安全加密协议。它被许多类型的应用程序（web、webservice、mobile），在网络中进行的安全通信时被使用。同时，必须正确的配置 TLS 才能有效的保护通信安全。

传输层安全性的最主要好处是保护 Web 应用程序数据免受未经授权披露和修改，特别是传输在客户端（Web 浏览器）和 Web 应用程序服务器之间、Web 应用程序服务器和后端之间、及其他非基于浏览器的企业组件之间。

静态数据加密

敏感数据管理的第一条规则是尽可能避免存储敏感数据。如果您必须存储敏感数据请确保数据已经通过某种方式实施加密保护，以避免未经授权的披露和修改。

密码学（或加密）是信息安全的更高级主题之一，其理解需要更多的学校教育和经验。密码学很难做到十分正确，因为当今有许多的加密算法，每种方法都有其优点和缺点，需要 Web 解决方案的架构师和开发人员彻底理解。此外由于重要的密码学研究通常基于高等数学和数论，故为理解提供了严重的障碍。

强烈建议不要从头开始构建加密功能，而是使用经同行评审的开放式解决方案，如 Google Tink 项目，Libsodium，以及内置于许多软件框架和云服务中的安全存储功能。

移动应用程序：安全本地存储

移动应用程序特别容易发生数据泄漏，因为移动设备经常丢失或被盗，但设备中却经常包含敏感数据。

一般情况下，移动设备上只应存储所需的最小数据。但是，如果您必须在移动设备上存储敏感数据，那么敏感数据应该存储在每个移动操作系统特定的数据存储目录中。在安卓设备上是 Android 的密钥库，在 iOS 设备上是 iOS 的密钥链。

密钥生命周期

密钥在应用程序中被大量用于处理敏感信息的功能。例如，密钥可用于签署 JWT、保护信用卡、提供各种形式的身份验证以及便利其他敏感的安全功能。在管理密钥时应遵循一些规则包括：

- 确保保护任何密钥不受未经授权的访问。
- 将密钥存放在适当的保密空间中，如下文所述。
- 需要多个密钥时使用独立的密钥。
- 在需要时建立对更改算法及密钥的支持。
- 构建应用程序功能以处理密钥的轮换。

应用程序保密管理

应用程序包含许多“保密”的信息需要实施安全操作。这些包括：证书、SQL 连接密码、第三方服务帐户凭据、密码、SSH 密钥、加密密钥等。未经授权披露或修改这些保密信息可能导致系统完全陷入危险。在管理应用程序保密信息时，请考虑以下内容：

- 不要将保密信息存储在代码、配置文件中或通过环境变量进行传递。使用像 GitRob 或 TruffleHog 这样的工具来扫描代码回收保密信息。
- 将密钥和其他应用程序级别的密钥保存在 KeyWhiz 或 Hashicorp's Vault project 或 Amazon KMS 等密钥库中，以便在运行时提供安全存储和对应用程序级别机密的访问。

预防的缺陷

- OWASP Top 10 2017 - A3: Sensitive Data Exposure
- OWASP Mobile Top 10 2014-M2 Insecure Data Storage

参考文献

- 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

工具

- 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
- KeyWhiz - Secrets manager
- Hashicorp Vault - Secrets manager
- Amazon KMS - Manage keys on Amazon AWS

OWASP ProActive Controls v 3.0

C9: 实施安全日志记录和监控 Implement Security Logging and Monitoring

描述

日志记录的目的是让大多数开发人员用于程序调试和诊断。安全日志记录也是一个相同的基本概念：在应用程序的运行状态下记录安全信息。监控是使用各种形式的自动化工具对应用程序的安全日志进行实时审查。相同的工具和模式可用于操作，调试和安全目的。

安全日志的好处

安全日志记录可用于：

1) 服务于入侵检测系统。
2) 取证分析和调查。
3) 满足监管合规要求。

安全日志的实现

以下是安全日志记录实现的最佳实践：

- 遵循系统和公司内部的通用日志记录格式和方法。常见日志记录框架的一个示例是 Apache Logging Services，它有助于保持 Java、PHP、.NET 和 C ++应用程序之间日志记录的一致性。
- 日志不要记录太多，也不要太少。例如，请确保始终记录时间戳和标识信息，包括源 IP 和用户 ID。但注意不要记录私人或机密数据。
- 密切关注跨节点的时间同步问题，保证时间戳一致。

对入侵检测和响应的日志记录

使用日志记录来标识用户恶意行为的活动。需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。
- 提交的数据涉及对不应修改数据的更改（选择列表，复选框或其他有限制的列表组件）。

- 违反服务器端访问控制规则的请求。
- 更全面的检测点列表请看这里。

当您的应用程序遇到此类活动时，您的应用程序应至少记录下日志并将其标记为高严重性问题。理想情况下，您的应用程序还应能响应已被识别的攻击，例如使用户的会话无效并锁定用户的帐户。响应机制应允许软件对已识别的攻击进行实时处理。

安全日志记录设计

日志记录解决方案必须以安全的方式进行构建和管理。安全日志记录设计应遵循以下原则:

- 在记录之前需要编码并验证任意危险字符，以防止日志注入攻击和日志伪造攻击。
- 不记录敏感信息。例如，请勿记录密码，会话 ID，信用卡或社会保险号。
- 保护日志完整性。攻击者可能会试图篡改日志。 因此，应考虑日志文件的读写权限和日志审核的用户权限。
- 将日志从分布式系统转发到中心化、安全的日志记录服务。这样如果一个节点受到攻击，可以确保日志数据不会丢失。这也有助于日志的集中监控。

参考文献

- OWASP AppSensor Detection Points - Detection points used to identify a malicious user probing for vulnerabilities or weaknesses in application.
- OWASP Log injection
- OWASP Log forging
- OWASP Cheat Sheet: Logging How to properly implement logging in an application
- OWASP Development Guide: Logging
- OWASP Code Review Guide: Reviewing Code for Logging Issues

工具

- OWASP Security Logging Project

C10: 处理所有错误和异常 Handle all Errors and Exceptions

- Apache Logging Services

描述

异常处理是一种编程概念，它允许应用程序以各种方式响应不同的错误状态（如网络关闭或数据库连接失败等）。正确地处理异常和错误对代码的可靠性和安全性至关重要。

错误和异常处理可能发生在应用程序的各个位置，包括关键业务逻辑以及安全功能和框架代码。

从入侵检测的角度来看，错误处理也很重要。应用程序在受到特定攻击时，可能会触发错误，而这些错误信息有助于检测正在进行中的恶意攻击。

错误处理的失误

多伦多大学的研究员发现：错误处理中很小的失误或者忘记处置也可能导致分布式系统的灾难性故障。

错误处理过程的失误可能会导致不同类型的安全漏洞。

- **信息泄漏:**在错误消息中泄漏的敏感信息可能会无意中帮助攻击者。例如，在错误消息中返回堆栈跟踪信息或其他内部详细错误信息，可以为攻击者提供有关运行环境的信息。甚至在处理不同错误状态下产生的极小差异（如，在登陆认证错误上返回“无效用户”或“无效密码”）可以为攻击者提供有价值的线索。综上所述，请务必记录错误的详细信息，用于取证和调试的目的。但不要公开相关信息，尤其是对外部客户端。
- **TLS 绕过：**Apple goto "fail bug"漏洞是因为在错误处理代码时，控制流出现错误，导致苹果系统上 TLS 连接完全泄露。
- **DOS**：缺少基本的错误处理机制可能导致系统关闭。这通常是一个攻击者相当容易利用的漏洞。其他错误处理的问题可能会导致 CPU 或磁盘使用量增加，从而导致系统性能下降。

积极的建议

- 以集中方式管理异常以避免代码中重复出现 try/catch 块。确保在应用程序内正确处理所有的意外行为。
- 确保向用户显示的错误消息不会泄漏关键数据，但仍然要足够详细、能正确响应用户请求。
- 记录异常信息要提供足够信息，确保可以支持测试、取证或应急响应团队快速了解问题。
- 仔细测试并验证错误处理代码。

参考文献

- OWASP Code Review Guide: Error Handling
- OWASP Testing Guide: Testing for Error Handling
- OWASP Improper Error Handling
- CWE 209: Information Exposure Through an Error Message
- CWE 391: Unchecked Error Condition

工具

- Error Prone，Google 的静态分析工具，用于发现 Java 开发人员在处理错误中的常见失误。
- Netflix's Chaos Monkey，是在运行时发现错误的著名自动化工具之一，它故意禁用系统实例以确保整体服务能够正确恢复。

结束语

本文档应该被视为一个开始，而不是一整套完整的技术和实践。我们想再次强调，文档旨在提供有关构建安全软件的初步意识。

有助于构建安全应用软件的后续步骤包括:

(1) 了解 Web 应用程序安全的一些常见风险，请查看 OWASP Top 10 和 OWASP Mobile Top 10。

(2) 根据主动控制 C1, 安全开发程序应该包含 1 份标准的安全要求综合列表。例如: OWASP (Web) ASVS 和 OWASP (Mobile) MASVS。

(3) 从更宏观的角度理解安全软件程序的核心构建模块，请查看 OWASP OpenSAMM project.

如果您对项目领导团队有任何疑问，请邮件联系我们。邮件列表为:

https://lists.owasp.org/mailman/listinfo/owasp_proactive_controls.

OWASP
Open Web Application
Security Project

面向开发人员