应用安全验证标准 4.0.3

终版

2021 年 10 月

Table of Contents

卷首语

关于本标准

应用安全验证标准，是一份要求和测试应用安全的清单，可供架构师、开发人员、测试人员、安全专家、工具供应商和消费者参考，用于定义、构建、测试和验证安全的应用。

版权和许可

Version 4.0.3, 2021 年 10 月

项目负责人

Andrew van der Stock	Daniel Cuthbert	Jim Manico
Josh C Grossman	Elar Lang	

主要贡献者

Abhay Bhargav	Benedikt Bauer	Osama Elnaggar
Ralph Andalis	Ron Perris	Sjoerd Langkemper
Tonimir Kisasondi		

其他贡献者和审查者

Aaron Guzman	Alina Vasiljeva	Andreas Kurtz	Anthony Weems	Barbara Schachner
Christian Heinrich	Christopher Loessl	Clément Notin	Dan Cornell	Daniël Geerts
David Clarke	David Johansson	David Quisenberry	Elie Saad	Erlend Oftedal
Fatih Ersinadim	Filip van Laenen	Geoff Baskwill	Glenn ten Cate	Grant Ongers
hello7s	Isaac Lewis	Jacob Salassi	James Sulinski	Jason Axley
Jason Morrow	Javier Dominguez	Jet Anderson	jeurgen	Jim Newman
Jonathan Schnittger	Joseph Kerby	Kelby Ludwig	Lars Haulin	Lewis Ardern
Liam Smit	lyz-code	Marc Aubry	Marco Schnüriger	Mark Burnett
Philippe De Ryck	Ravi Balla	Rick Mitchell	Riotaro Okada	Robin Wood
Rogan Dawes	Ryan Goltry	Sajjad Pourali	Serg Belkommen	Siim Puustusmaa
Ståle Pettersen	Stuart Gunter	Tal Argoni	Tim Hemel	Tomasz Wrobel
Vincent De Schutter	Mike Jang			

如果上面的 4.0.3 致谢列表中缺少某些内容，请在 GitHub 上记录一个工单，以便在未来的更新中得到确认。

从 2008 年的 ASVS 1.0，到 2016 年的 3.0，这份应用安全验证标准，一直都站在前人的肩膀上。 直到今天，ASVS 中的大部分结构和验证项，都是最初由 Mike Boberski、Jeff Williams 和 Dave Wichers 编写的，但还有许多贡献者。 感谢那些以前参与过的人。若需查看所有早期版本贡献者的名单，请查阅每个以前的版本。

序言

欢迎来到应用安全验证标准（Application Security Verification Standard，以下简称 ASVS）4.0 版。 ASVS 是一项社区驱动的工作，旨在建立一个安全要求和控制的框架，在设计、开发和测试现代网络应用程序和网络服务时，定义所需要的功能和非功能性的安全控制措施。

4.0.3 版是对 4.0 版的第三个小补丁，旨在修正拼写错误，使需求更清晰，而不做实质性的改变（如加强或增加需求）。 然而，在我们认为合适的地方，一些要求可能被稍微削弱，一些完全多余的要求被删除（但没有重新编号）。

ASVS v4.0 是过去十年来社区努力和行业反馈的结晶。 我们试图让 ASVS 更容易地适应安全软件开发生命周期中的各种用例场景。

我们知道，对于任何 Web 应用程序标准（包括 ASVS）的内容，很可能永远不会达成 100% 的一致。 在某种程度上，风险分析总是主观的，这对试图以一刀切的标准来概括造成了一种挑战。 然而，我们希望这个最新的版本，是朝着正确方向迈出的一步，并增强了这个关键行业标准中的概念。

4.0 版的新内容

这个版本中最重要的变化，是采用了 NIST 800-63-3 数字身份指南，引入了现代的、基于证据的高级认证控制。 虽然我们预计在与高级认证标准接轨方面会有一些阻力（这个广受好评的应用安全标准是基于证据的），但我们认为标准保持一致是非常必要的。

信息安全标准应尽量减少独特要求的数量，以便遵守的组织不必决定冲突或不兼容的控制措施。 在认证和会话管理方面，《OWASP Top 10 2017》和现在的《OWASP 应用安全验证标准》，已与 NIST 800-63 保持一致。我们鼓励其他标准制定机构与我们、NIST 和其他机构合作，制定一套普遍接受的应用安全控制措施，最大限度地提高安全性和减少合规成本。

ASVS 4.0 从头到尾都进行了重新编号。新的编号方案使我们能够弥补长期消失的章节的空白，并允许我们分割较长的章节，以尽量减少开发人员或团队必须遵守的控制项。例如，如果应用程序不使用 JWT，那么会话管理中有关 JWT 的整个部分都不适用。

4.0 版本中的新内容，是对 常见弱点列举（Common Weakness Enumeration，以下简称 CWE）的全面映射，这是我们在过去十年中最普遍的功能要求之一。 CWE 映射，允许供应商和使用漏洞管理软件的人，将其他工具和先前 ASVS 版本的内容，同 4.0 及之后的版本进行匹配。 为了给 CWE 条目腾出空间，我们不得不取消“起始时间”列，由于我们完全重新编号，因此这一列意义不大。 并非 ASVS 中的每个项目都有相关的 CWE，由于 CWE 有大量的重复，我们尝试使用最常用而非最接近的匹配项。 验证控制，并不总是可以对应到等效的 CWE。 我们乐意与 CWE 社区和信息安全领域就缩小这一差距进行更广泛的讨论。

我们致力于全面满足并超越《OWASP Top 10 2017》和《OWASP Proactive Controls 2018》的要求。由于《OWASP Top 10 2017》是避免疏忽的最低要求，我们特意将 Top 10 中的所有要求（除“日志记录”外）列为 1 级控制（Level 1），让 OWASP Top 10 的采用者，更容易将实际的安全标准落地。

我们着手确保 ASVS 4.0 1 级（Level 1），是 PCI DSS 3.2.1 第 6.5 节的超集，用于应用程序设计、编码、测试、代码安全审计和渗透测试。这需要覆盖 V5 中的缓冲区溢出和不安全内存操作，以及 V14 中与不安全内存相关的编译标志，以及现有行业领先的应用程序和 Web 服务验证要求。

W 我们已经完成了 ASVS 的转变，从单一的服务器端控制，到为所有现代应用和 API 提供安全控制。 在 server-less API、移动、云、容器、CI/CD 和 DevSecOps、联邦等等的时代，我们不能继续忽视现代应用架构。 现代应用的设计，与 2009 年最初的 ASVS 发布时的设计大不相同。ASVS 必须始终着眼于未来，这样我们才能为我们的主要受众——开发者提供合理的建议。 我们已经删除了那些“假设应用程序在单个组织拥有的系统上执行”的要求。

由于 ASVS 4.0 的规模，以及我们希望成为所有其他应用安全验证标准的基线，我们已经淘汰了“移动”章节，转而支持移动应用程序安全验证标准（MASVS）。 物联网附录将出现在未来 OWASP IoT 项目的物联网 ASVS 中。我们在附录 C 中包含了物联网 ASVS 的早期预览。 感谢 OWASP Mobile 团队和 OWASP IoT 项目团队对 ASVS 的支持，期待未来与他们合作提供补充标准。

最后，我们删除和淘汰了影响较小的控制项。 随着时间的推移，ASVS 开始成为一套全面的控制措施，但并非所有控制项在开发安全软件方面都是平等的。 消除低影响项目的努力可能会更进一步。 在 ASVS 的未来版本中，通用弱点评分系统（Common Weakness Scoring System，CWSS），将有助于进一步优先考虑那些真正重要的控制和应该停用的控制。

从 4.0 版本开始，ASVS 将专注于成为领先的 Web 应用程序和服务标准，涵盖传统和现代应用架构，以及敏捷的安全实践和 DevSecOps 文化。

使用 ASVS

ASVS 有两个主要目标：

- 帮助组织开发和维护安全的应用程序。
- 允许安全服务厂商、安全工具供应商和消费者调整他们的要求和产品。

应用安全验证级别

应用程序安全验证标准（ASVS）定义了三个安全验证级别，每个级别的深度都在增加。

- ASVS Level 1 适用于低保证级别，可通过渗透测试验证。
- ASVS Level 2 适用于包含敏感数据的应用程序（需要保护），是大多数应用程序的推荐级别。
- ASVS Level 3 适用于最关键的应用程序：执行高价值交易、包含敏感医疗数据的应用程序，或任何需要最高级别信任的应用程序。

每个 ASVS 级别都包含一个安全要求的列表。其中的每一项，都可以对应到开发人员必须在软件中建立的特定安全特性和功能。

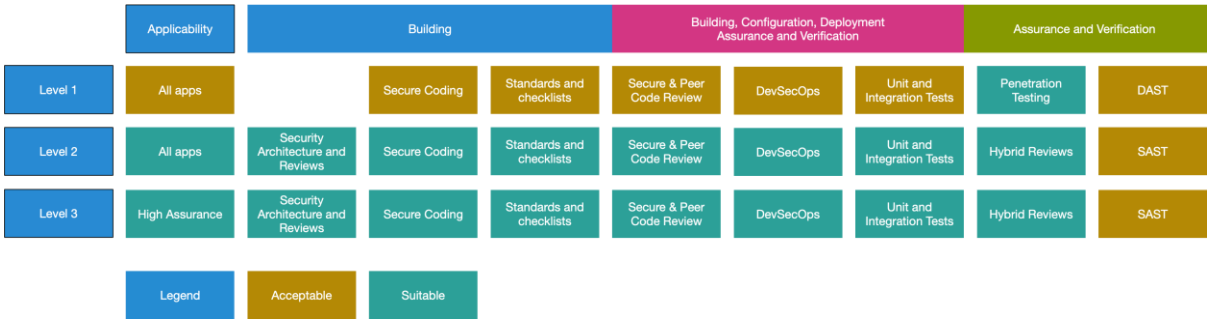

图 1 - OWASP 应用安全验证标准级别（4.0 版）

Level 1 是唯一可以由人类进行渗透测试的级别。 所有其他的等级都需要接触文档、源代码、配置以及开发人员。 然而，即使 L1 允许通过“黑盒”（无文档和无源代码）测试来验证，但它并不是有效的保证活动，应尽量避免单独采用。 恶意攻击者有大量的时间，但大多数渗透测试在几周内就会结束。 防御者需要在合理的时间内，建立安全控制，发现、解决和保护所有的弱点，检测和应对恶意攻击者。 恶意攻击者的时间基本上是无限的，只需要一处漏洞，或者一个缺少检测的弱点就可以成功。 黑盒测试，通常在开发结束时快速执行，或者根本不执行，完全无法应对这种不对称性。

在过去的 30 多年里，黑盒测试一次又一次被证明会遗漏关键的安全问题，直接导致了越来越多的大规模漏洞。 我们强烈鼓励使用广泛的安全保证和验证，包括用 Level 1 的源代码主导的（混合）渗透测试来取代渗透测试，在整个开发过程中可以充分接触到开发人员和文档。 金融监管机构不会容忍“无法接触账本、交易样本或执行控制的人员”的外部财务审计。 行业和政府必须要求软件工程领域具有相同的透明度标准。

我们强烈鼓励在开发过程中使用安全工具。 构建管道可以持续使用 DAST 和 SAST 工具，来查找容易发现的、不应该出现的安全问题。

在没有人工协助的情况下，ASVS 中超过半数的验证项无法被自动化工具和在线扫描完成。 如果需要对每次构建进行全面的自动化测试，则使用自定义单元和集成测试的组合，以及由构建发起的在线扫描。业务逻辑缺陷和访问控制测试只能使用人工协助。 这些内容应该变成单元和集成测试。

如何使用这个标准

使用应用程序安全验证标准（ASVS）的最佳方法之一，是将其作为一个蓝图，创建一个针对您的应用程序、平台或组织的安全编码检查表。 建议您根据不同的项目场景，针对其中最重要的安全要求，在定制的 ASVS 中增加关注。

Level 1 - 第一步，自动化，或全景图

如果一个应用程序能够充分防御 OWASP Top10 或其他类似检查表中提到的那些容易发现的安全漏洞，那么它就达到了 ASVS Level 1。

Level 1，是所有应用程序应争取的最低限度。 作为多个阶段工作的第一步，或是当应用程序不存储或处理敏感数据，因此不需要 L2 或 L3 的更严格的控制时，它也是有用的。 Level 1 控制可以通过工具自动检查，也可以在不访问源代码的情况下简单地手动检查。我们认为 Level 1 是所有应用程序的最低要求。

对应用程序的威胁，往往来自那些使用简单、省力的技术来识别“简单漏洞”的攻击者。 这与坚定的攻击者形成对比，后者将花费集中的精力专门针对应用程序。 如果您的应用程序处理的数据具有很高的价值，您很少会止步于 Level 1。

Level 2 - 大多数应用程序

如果一个应用程序能够充分抵御当今与软件相关的大多数风险，那么它就达到了 ASVS Level 2（级别或标准）。

Level 2 确保安全控制在应用程序中得到有效的落实。 Level 2 通常适用于处理重要 B2B 事务（B2B，Business-to-Business）的应用程序， 包括处理医疗保健信息、实施关键业务、实现敏感功能、或处理敏感资产的应用程序，或完整性是保护其业务的关键方面的行业，例如打击游戏外挂和作弊的游戏行业。

对 Level 2 应用程序的威胁，通常是熟练的和有动机的攻击者，他们专注于特定的目标，使用工具和技术有效地发现和利用应用程序中的风险。

Level 3 - 高价值、高保证或高安全性

ASVS Level 3，是 ASVS 内的最高级别验证。 这个级别通常保留给需要大量安全验证的应用，例如军事、健康、安全和关键基础设施等领域的应用。

对于执行关键功能的应用程序，组织可能需要 ASVS Level 3，在这种情况下，故障可能会严重影响组织的运作，甚至影响其生存。下面提供了有关 ASVS 3Level 3 应用的示例指导。 如果一个应用程序能够充分防御高级应用程序的安全漏洞，并显示出良好的安全设计原则，那么它就达到了 ASVS Level 3（或高级）。

与其他级别相比，ASVS Level 3 的应用程序需要对架构、编码和测试进行更深入的分析。 安全的应用程序以有意义的方式模块化的（以促进弹性，可扩展性，以及最重要的安全层），每个由网络连接和/或物理实例分开的模块，负责自己纵深防御的安全职责，这需要适当的记录。 职责包括确保机密性（例

如加密）、完整性（例如事务、输入验证）、可用性（例如优雅地处理负载）、身份验证（包括系统之间）、授权和审计（日志记录）的控制。

在实践中应用 ASVS

不同的威胁有不同的动机。某些行业有独特的信息和技术资产，以及特定领域的监管要求。

我们强烈建议，各组织根据其业务性质，深入研究其独特的风险特征，并根据该风险和业务要求，确定适当的 ASVS 级别。

如何引用 ASVS 要求

每个需求都有一个标识符，格式为 <章>.<节>.<要求>，每个元素都是一个数字，例如：1.11.3 。

- <chapter> 值对应于需求出现的章，例如：所有 1.#.# 的需求都来自 Architecture 这一章。
- <section> 值对应于该章中需求出现的小节，例如：所有 1.11.# 需求都在 Architecture 章的 Business Logic Architecture 部分。
- <requirement>值对应该章的具体要求，例如：本标准的 4.0.3 版本中的 1.11.3 是：

验证所有高价值业务逻辑流（包括身份验证、会话管理和访问控制）都是线程安全的，并且可以防止“检查时间和使用时间不一致”导致的条件竞争问题。

标识符可能会在标准版本之间发生变化，因此在其他文档、报告或工具最好使用以下格式：v<version>-<chapter>.<section>.<requirement>，其中：“version”是 ASVS 版本标签。例如：v4.0.3-1.11.3 将被理解为特指版本 4.0.3 中“架构”这一章中“业务逻辑架构”这一节的第 3 项要求。（可概括为 v<version>-<requirement_identifier>。）

注意：版本部分前面的 v 要小写。

如果使用不包括 v<version> 的标识符，那么它们应该被认为是指最新的应用安全验证标准内容。显然，随着标准的增长和变化，这将导致问题，这就是为什么作者或开发者应该将版本包括进去。

ASVS 需求列表，以 CSV、JSON 和其他可能对参考或编程有用的格式提供。

评估和认证

OWASP 对 ASVS 认证和信任标志的立场

OWASP 作为一个与供应商无关的非营利性组织，目前不认证任何供应商、验证人员或软件。

所有这类保证声明、信任标志或认证，均未经 OWASP 正式审查、注册或认证，因此依赖此类观点的组织，需要谨慎对待任何第三方的信任或声称 ASVS 认证的信任标志。

这并不影响组织提供此类保证服务，只要他们不要求官方的 OWASP 认证。

认证组织指南

应用程序安全验证标准，可以用作应用程序的公开验证，包括对关键资源的开放和自由访问（如架构师和开发人员、项目文档、源代码），对测试系统的认证访问（包括对每个角色的一个或多个帐户的访问），特别是 L2 和 L3 验证。

从历史上看，渗透测试和安全代码审查都包含“异常”问题——即只有未通过的测试项才会出现在最终报告中。 认证组织必须在任何报告中包括验证的范围（特别是某个关键组件不在范围内时，如 SSO 身份验证）、验证结果的摘要，包括通过的和未通过的测试，并清楚地说明如何解决未通过的测试。

某些验证要求可能不适用于被测试的应用程序。例如，如果你向客户提供无状态的服务层 API 而没有客户端实现，那么“V3-会话管理”中的许多要求就不能直接使用。 在这种情况下，认证机构仍可声称完全符合 ASVS 的要求，但必须在报告中明确说明被排除的验证要求不适用的原因。

保留详细的工作底稿、屏幕截图或视频、可靠地重复利用一个问题的脚本，以及测试的电子记录，如拦截代理日志和相关的笔记（如清理清单），被认为是标准的行业惯例，哪怕是对于最可疑的开发人员来说，它们也能作为调查结果的证明。 仅仅跑一个工具并报告故障是不够的，这根本不能提供充分的证据，证明所有认证级别的问题都经过了彻底的测试。 在有争议的情况下，应该有足够的证据，来证明每一个经过验证的需求确实被测试过。

测试方法

认证机构可自由选择适当的测试方法，但应在报告中注明。

根据所测试的应用程序和验证需求，可以使用不同的测试方法来获得相似的结果置信度。 例如，要验证应用程序输入验证机制的有效性，可以通过手动渗透测试或通过源代码来分析。

自动化安全测试工具的作用

鼓励使用自动化渗透测试工具以提供尽可能多的覆盖范围。

仅使用自动渗透测试工具，是不可能完全完成 ASVS 验证的。虽然 L1 中的绝大多数需求可以使用自动化测试来执行，但总体上，绝大多数需求并不适合自动化渗透测试。

请注意，随着应用安全行业的成熟，自动化和手动测试之间的界限已经变得模糊。 自动化工具通常由专家手动调整，而手动测试人员通常会利用各种自动化工具。

渗透测试的作用

在 4.0 版本中，我们决定让 L1 完全可渗透测试，而无需访问源代码、文档或开发人员。 OWASP Top 10 2017 A10 要求的两个日志记录项目，将需要访谈、屏幕截图或其他证据，就像它们在 OWASP Top 10 2017 中的一样。 然而，在无法获得必要信息的情况下进行测试，并不是一种理想的安全验证方式，因为它不仅错过了审查来源、识别威胁和缺失控制的可能性，还会错过在更短的时间内进行更彻底测试的可能。

在可能的情况下，执行 L2 或 L3 评估时，需要访问开发人员、文档、代码，以及访问具有非生产数据的测试应用程序。 在这些级别进行的渗透测试，需要这种级别的访问，我们称之为 “混合审查” 或 “混合渗透测试”。

ASVS 的其他用途

除了用于评估应用程序的安全性外，我们还确定了 ASVS 的许多其他潜在用途。

作为详细的安全架构指南

应用程序安全验证标准的更常见用途之一，是作为安全架构师的资源。 Sherwood 应用业务安全架（Sherwood Applied Business Security Architecture，SABSA）缺少大量的信息，而这些信息是完成一次彻底的应用安全架构审查所必需的。 ASVS 可以用来填补这些空白，让安全架构师为常见问题选择更好的控制措施，如数据保护模式和输入验证策略。

作为现有安全编码 Checklists 的替代品

许多组织可以从采用 ASVS 中受益，通过选择三个级别中的一个，或通过 fork ASVS，在特定领域改变每个应用风险级别的要求。 我们鼓励这种 fork，只要保持可追溯性，因此，如果一个应用程序已经通过了标准版本中的“要求 4.1”，那么也就通过了 fork 版本中的这个要求。

作为自动化单元和集成测试的指南

ASVS 的设计是高度可测试的，唯一的例外是架构和恶意代码要求。 通过构建单元和集成测试，对相关的滥用情况进行 fuzz 测试，应用程序几乎可以在每次构建中进行自我验证。 例如，可以为登录控制器制作额外的测试，测试常见的默认用户名参数、帐户枚举、暴力破解、LDAP 注入、SQL 注入以及 XSS。同样地，对密码参数的测试，应该包括常用密码、密码长度、空字节注入、移除参数、XSS 等。

用于安全开发培训

ASVS 还可用于定义安全软件的特征。 许多“安全编码”课程只是带有少量编码技巧的道德黑客课程。 这不一定能帮助开发人员编写更安全的代码。 相反，安全开发课程可以使用 ASVS，重点关注 ASVS 中的主动控制，而不是前 10 项不该做的负面事情。

作为敏捷应用安全的驱动程序

在敏捷开发过程中，为了获得安全的产品，ASVS 可以作为框架来定义团队需要实施的特定任务。 一种可能的方法是：从 Level 1 开始，根据指定级别的 ASVS 要求，验证特定应用程序或系统，查找缺少哪些项目，并在待办事项中提出特定工单/任务。 这有助于对具体任务进行优先排序（梳理），并使安全在敏捷开发中可见。 这也可用于确定组织中审计和审查任务的优先；其中，特定的 ASVS 要求，可以作为团队成员审查、重构或审计的驱动因素，并可以记录到最终的待办清单中。

作为指导安全软件采购的框架

ASVS 是一个很好的框架，可以帮助确保安全软件的采购或定制开发服务的采购。 买方可以简单地设定一个要求，即他们希望采购的软件必须按照 ASVS 的 Level x 来开发，并要求卖方证明该软件满足 ASVS 的 x 级。

V1 架构、设计和威胁建模

控制目标

在许多组织中，安全架构几乎已成为一门失传的艺术。 在 DevSecOps 时代，企业架构师的日子已经过去。应用安全领域必须迎头赶上，采用敏捷安全原则，同时将领先的安全架构原则重新介绍给软件从业者。 A 架构不是一种实施，而是一种思考问题的方式，它可能有许多不同的答案，而没有一个单一的“正确”答案。 很多时候，安全被视为不灵活的，要求开发人员以特定方式修复代码，而开发人员可能知道解决问题的更好方法。 对于架构来说，没有单一的、简单的解决方案，尝试寻找这种方案，是对软件工程领域的一种损害。

一个 Web 应用程序的具体实现，很可能在其生命周期中不断被修改，但整体架构可能很少改变，而是缓慢发展。 安全架构也是一样的，身份验证——我们今天需要、明天需要、五年后也需要。 如果我们今天做出合理的决定，选择和复用符合架构的解决方案，那么就可以节省大量的精力、时间和金钱。 例如，十年前，多因素认证很少被实施。

如果开发人员已经在单一的“安全标识提供程序模型”上有所投入（例如 SAML 联邦认证），身份提供者可以更新以纳入新的要求（例如 NIST 800-63 标准），同时不改变原始应用程序的接口。 如果许多应用程序共享相同的安全架构和组件，那么它们都将同时从这次升级中受益。 然而，SAML 并不总是最好或最合适的身份验证解决方案——随着需求的变化，可能需要替换为其他解决方案。 像这样的更改要么很复杂，成本高到需要完全重写，要么在没有安全架构的情况下完全不可能。

在本章中，ASVS 涵盖了任何良好安全架构的主要方面：可用性、保密性、完整性、不可抵赖性和隐私。 这些安全原则中的每一条，都必须适用并内置于所有应用程序中。 “左移”至关重要，从安全编码 Checklists、指导和培训、编码和测试、构建、部署、配置和操作开始，到后续的独立测试，确保所有安全控制存在且功能正常。 这最后一步，曾经是我们作为一个行业所做的一切，但当开发人员每天数十次或数百次地推送代码时，就已经不够了。 应用安全专业人员必须跟上敏捷技术的步伐，这意味着要适应开发人员的工具，学习编码，并与开发人员一起工作，而不是在其他人离开的几个月后再批评项目。

V1.1 安全软件开发生命周期

#	说明	L1	L2	L3	CWE
1.1.1	验证使用安全的软件开发生命周期，在开发的各个阶段解决安全问题。 (C1)		✓	✓	
1.1.2	验证在每次设计变更或 sprint 计划中使用威胁建模，以识别威胁、计划对策、促进适当的风险响应，并指导安全测试。		✓	✓	1053
1.1.3	验证所有用户信息和功能是否包含功能安全约束，例如 “作为一个用户，我应该能够查看和编辑我的个人资料。我不应该能够查看或编辑其他人的资料”		✓	✓	1110

#	说明	L1	L2	L3	CWE
1.1.4	验证应用程序所有的信任边界、组件和重要数据流的文档，判断其合理性。		✓	✓	1059
1.1.5	验证应用程序的高级架构及远程连接服务涉及的定义和安全分析。 (C1)		✓	✓	1059
1.1.6	验证集中、简单（设计）、安全、经过审查、和可重复使用的安全控制措施的实施情况，以避免重复、缺失、无效或不安全的控制措施。 (C10)		✓	✓	637
1.1.7	向所有开发人员和测试人员，验证安全编码 Checklist、安全需求、指南或策略的可用性。		✓	✓	637

V1.2 认证架构

在设计身份验证时，如果攻击者可以通过拨打客服电话，回答常见的问题来重置帐户，那么是否具有强大硬件支持的多因素身份验证（MFA）并不重要。 在证明身份时，所有的认证途径必须具有相同的强度。

#	说明	L1	L2	L3	CWE
1.2.1	验证应用程序所有的组件、服务和服务器，是否使用了唯一或特殊的低权限操作系统帐户。 (C3)		✓	✓	250
1.2.2	验证应用组件之间（包括 API、中间件和数据层）的通信是否经过验证。组件只具有最低的必要权限。 (C3)		✓	✓	306
1.2.3	验证应用程序是否使用已知安全的单一认证机制，可以扩展到强身份验证，并有足够的日志记录和监控，来检测帐户滥用或违规行为。		✓	✓	306
1.2.4	验证所有的认证途径和身份管理 API，都实现了一致的认证安全控制强度，以便收敛应用程序的风险。		✓	✓	306

V1.3 会话管理架构

这是未来架构需求的占位符。

V1.4 访问控制架构

#	说明	L1	L2	L3	CWE
1.4.1	验证受信任的实施点（如访问控制网关、服务器和 Serverless 函数）是否实施了访问控制。切勿在客户端实施访问控制。		✓	✓	602
1.4.2	[已删除，不可操作]				
1.4.3	[已删除，与 4.1.3 重复]				

#	说明	L1	L2	L3	CWE
1.4.4	验证应用程序使用单一的、经过严格审查的访问控制机制，来访问受保护的数据和资源。 所有请求都必须通过这个单一机制，以避免复制、粘贴或不安全的替代路径。 (C7)		✓	✓	284
1.4.5	验证是否使用基于属性/特征的访问控制，即代码应检查用户对某一特征/数据项的授权，而不仅仅是他们的角色。 权限仍应依照不同角色进行分配。 (C7)		✓	✓	275

V1.5 输入和输出架构

在 4.0 中，我们已经不再把 “服务器端” 作为一个表示信任边界的术语。 信任边界仍然令人担忧——在不受信任的浏览器或客户端设备上做决定，很容易被绕过的。 然而，在今天的主流架构部署中，信任的执行点已经发生了巨大的变化。 因此，在 ASVS 中使用 “受信任的服务层” 这一术语时，我们描述的都是受信任的执行点，无论其位置如何，如微服务、Serverless API、服务器端、具有安全启动的客户端设备上的受信任的 API、合作伙伴或外部 API 等等。

这里的“不受信任的客户端”一词，是指呈现表示层的客户端技术，通常称为“前端”技术。 这里的术语“序列化”不仅表示通过网络发送数据（如一个数组的值或获取 JSON 结构），还指传递可以包含逻辑的复杂对象。

#	说明	L1	L2	L3	CWE
1.5.1	验证输入和输出要求，明确规定如何根据类型、内容以及适用的法律、法规和其他政策规定，来操作和处理数据。		✓	✓	1029
1.5.2	验证在与不受信任的客户进行通信时，不使用序列化。 如果无法做到这一点，请确保执行足够的完整性控制（如果发送敏感数据，可能还要进行加密），以防止反序列化攻击，包括对象注入。		✓	✓	502
1.5.3	验证输入验证是否在可信的服务层上执行。 (C5)		✓	✓	602
1.5.4	验证输出编码是否发生在其预期的解释器附近（或由解释器进行）。 (C4)		✓	✓	116

V1.6 加密架构

应用程序需要设计强大的加密架构，以根据其分类保护数据资产。加密所有东西，是浪费；不加密任何东西，是法律上的疏忽。 在架构或顶层设计、设计冲刺（Design Sprint）或架构高峰期，通常需要取得一种平衡。 一边设计加密技术，一边进行开发迭代，这样的安全实施，其成本不可避免地要比一开始就做简单的构建要高得多。

架构要求是整个代码库的内在要求，因此很难进行单元或集成测试。架构需求需要在整个编码阶段的编码标准中加以考虑，并应在安全架构、代码审查或复盘会议中加以审查。

#	说明	L1	L2	L3	CWE
1.6.1	验证是否有明确的加密密钥管理政策，以及加密密钥的生命周期是否遵循密钥管理标准，如 NIST SP 800-57。		✓	✓	320
1.6.2	验证密码服务的消费者是否通过使用密钥库或基于 API 的替代方案，来保护密钥材料和其他机密。		✓	✓	320
1.6.3	验证所有的密钥和密码是否可替换的，并且是重新加密敏感数据的明确定义流程的一部分。		✓	✓	320
1.6.4	验证架构是否将客户端机密（例如对称密钥、密码或 API 令牌）视为不安全的，并且从不使用它们来保护或访问敏感数据。		✓	✓	320

V1.7 错误、日志和审计架构

#	说明	L1	L2	L3	CWE
1.7.1	验证整个系统是否使用了通用的日志记录格式和方法。 (C9)		✓	✓	1009
1.7.2	验证日志是否安全地传输到远程系统，以便进行分析、检测、报警和升级。 (C9)		✓	✓	

V1.8 数据保护和隐私架构

#	说明	L1	L2	L3	CWE
1.8.1	验证所有敏感数据都已识别并归入保护级别。		✓	✓	
1.8.2	验证所有保护级别都具有一套相关的保护要求，如加密要求、完整性要求、保留、隐私和其他机密性要求，并在架构中应用这些要求。		✓	✓	

V1.9 通信架构

#	说明	L1	L2	L3	CWE
1.9.1	验证应用程序对组件之间的通信进行加密，特别是当这些组件处于不同的容器、系统、站点或云提供商时。 (C3)		✓	✓	319
1.9.2	验证应用组件是否验证了通信链接中每一方的真实性，以防止中间人攻击。例如，应用程序组件应校验 TLS 证书链。		✓	✓	295

V1.10 恶意软件架构

#	说明	L1	L2	L3	CWE
1.10.1	验证是否使用了源代码控制系统，以及有程序确保签入时附带问题或变更单。源代码控制系统应该具有访问控制和可识别的用户，以追溯任何的更改。		✓	✓	284

V1.11 业务逻辑架构

#	说明	L1	L2	L3	CWE
1.11.1	验证所有应用组件在其提供的业务或安全功能方面的定义和文档。		✓	✓	1059
1.11.2	验证所有高价值的业务逻辑流，包括认证、会话管理和访问控制，不共享不同步的状态。		✓	✓	362
1.11.3	验证所有高价值的业务逻辑流，包括身份验证、会话管理和访问控制都是线程安全的，并能抵抗检查时间和使用时间不同步时的条件竞争。			✓	367

V1.12 安全上传架构

#	说明	L1	L2	L3	CWE
1.12.1	[已删除，与 12.4.1 重复]				
1.12.2	验证用户上传的文件——如果需要显示或从应用中下载，是通过二进制流下载，或从无关的域（如云文件存储桶）提供。实施合适的内容安全策略（CSP），以减少来自上传文件的 XSS 向量或其他攻击的风险。		✓	✓	646

V1.13 API 架构

这是未来架构需求的占位符。

V1.14 配置架构

#	说明	L1	L2	L3	CWE
1.14.1	通过明确的安全控制、防火墙规则、API 网关、反向代理、基于云的安全组或类似机制，验证不同信任级别的组件的隔离情况。		✓	✓	923
1.14.2	验证二进制签名、可信连接和经过验证的接口，以将二进制文件部署到远程设备。		✓	✓	494
1.14.3	验证构建管道是否对过期或不安全的组件发出警告并采取适当的行动。		✓	✓	1104

#	说明	L1	L2	L3	CWE
1.14.4	验证构建管道是否包含自动构建和验证应用安全部署的构建步骤，特别是当应用基础设施是软件定义时，例如云环境构建脚本。		✓	✓	
1.14.5	验证应用程序部署是否在网络级别进行了充分的沙盒化、容器化或隔离，以延迟和阻止攻击者攻击其他应用程序，尤其是当攻击者执行敏感或危险操作时（如反序列化）。 (C5)		✓	✓	265
1.14.6	验证应用程序未使用不受支持、不安全或不推荐的客户端技术，如 NSAPI 插件、Flash、Shockwave、ActiveX、Silverlight、NACL 或客户端 Java applets。		✓	✓	477

参考文献

有关更多信息，请参阅：

- OWASP Threat Modeling Cheat Sheet
- OWASP Attack Surface Analysis Cheat Sheet
- OWASP Threat modeling
- OWASP Software Assurance Maturity Model Project
- Microsoft SDL
- NIST SP 800-57

V2 认证

控制目标

认证是建立或确认某人（或某物）的真实性，并且个人或设备的声明是正确的，可防止假冒，并防止恢复或拦截密码。

当 ASVS 首次发布时，用户名+密码是最常见的认证形式（除高安全系统以外）。多因素身份验证（MFA）在安全界被普遍接受，但在其他地方很少需要。随着密码泄露次数的增加，认为用户名在某种程度上是保密的，而密码则是未知的这种想法使得许多安全控制无法成立。例如，NIST 800-63 将用户名和基于知识的身份验证（KBA）视为公共信息，将 SMS 和电子邮件通知视为"受限"的认证类型，而密码是预先泄露的。这一现实使基于知识的认证器、短信和电子邮件恢复、密码历史、复杂性和轮换控制变得毫无用处。这些控制措施不总那么有用，经常迫使用户每隔几个月就想出一些弱的密码，但是随着 50 多亿用户名和密码泄露事件的公布，现在是时候继续前进了。

在 ASVS 的所有章节中，认证和会话管理章节的变化最大。采用有效的、以证据为基础的领先实践，对许多人来说将是挑战，这完全没问题。现在我们必须开始向未来的后密码时代过渡。

NIST 800-63 - 现代的、基于证据的认证标准

NIST 800-63b 是一种现代的、基于证据的标准，代表了可用的最佳建议，无论其适用性如何。该标准对世界各地的所有组织都有帮助，但与美国机构和与美国机构打交道的机构尤其相关。

NIST 800-63 的术语一开始可能有点令人困惑，特别是你只习惯于用户名+密码认证的话。现代认证的进步是必要的，所以我们必须引入将来会变得司空见惯的术语，但我们确实理解：在行业落实这些新术语之前，理解这些新术语的困难。我们在本章末尾提供了一个词汇表，以提供帮助。我们重新表述了许多要求，以满足要求的意图，而不只是拘泥于文字。例如，当 NIST 在本标准中使用"记忆秘密"时（memorized secret），ASV 使用术语"密码"（password）。

ASVS V2 身份验证、V3 会话管理以及在较小程度上的 V4 访问控制，已被调整为符合 NIST 800-63b 控制项的一个子集，主要围绕常见的威胁和经常被利用的认证弱点。如果需要完全遵守 NIST 800-63，请参考 NIST 800-63。

选择合适的 NIST AAL 级别

应用程序安全验证标准（ASVS），已尝试将 ASVS L1 对应到 NIST AAL1 要求，将 L2 对应到 AAL2，将 L3 对应到 AAL3。然而，ASVS Level 1 作为"基本"的控制，不一定是验证应用或 API 的正确 AAL 级别。例如，如果该应用是 L3 应用或有 AAL3 的监管要求，则应在 V2 和 V3 会话管理章节选择 L3。应根据 NIST 800-63b 指南选择符合 NIST 标准的认证保证级别（AAL），如 NIST 800-63b 第 6.2 节 中的 *Selecting AAL*。

图例

应用程序总是可以超过当前级别的要求，特别是如果现代认证是在应用程序的路线图上。以前，ASVS 要求强制 MFA。NIST 不要求强制 MFA。因此，我们在本章中使用了一个可选的指定，以表明 ASVS 鼓励但不要求控制的地方。本标准自始至终使用了以下图示：

标记	说明
	不要求
o	建议，但不要求
✓	要求

V2.1 密码安全

在 NIST 800-63 中，密码被称为“记忆的秘密”（Memorized Secrets），包括密码、PIN、解锁图案、选择正确的小猫或其他图像元素以及密码短语。 它们通常被认为是“您知道的东西”，并且通常用作单因素身份认证工具。 继续使用单因素认证有很大的风险，包括互联网上披露的数十亿有效用户名和密码、默认或弱密码、彩虹表和最常见密码的有序字典。

应用程序应强烈鼓励用户注册多因素认证，并应允许用户重新使用他们已经拥有的令牌，如 FIDO 或 U2F 令牌，或链接到提供多因素认证的凭证服务提供商。

凭据服务提供商（CSP）为用户提供联合身份。 用户通常会拥有多个 CSP 的多个身份，例如使用 Azure AD、Okta、Ping identity 或 Google 的企业身份，或使用 Facebook、Twitter、Google 或微信的普通用户，这只是一些常见的可能。 这份清单并不是对这些公司或服务的认可，而只是鼓励开发者考虑用户有许多既定身份的现实。 组织应该考虑与现有的用户身份整合，根据 CSP 的身份证明强度的风险状况，组织应考虑与现有用户身份集成。 例如，政府机构不太可能接受社交媒体身份作为敏感系统的登录名，因为很容易伪造或丢弃身份，而移动游戏公司可能需要与主要社交媒体平台整合，以扩大他们的活跃玩家群。

#	说明	L1	L2	L3	CWE	NIST §
2.1.1	验证用户设置的密码长度至少为 12 个字符（多个空格合并后）。 (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	验证是否允许 64 个字符以上的密码，并拒绝超过 128 个字符的密码。 (C6)	✓	✓	✓	521	5.1.1.2
2.1.3	验证不进行密码截断。然而，连续的多个空格可以被单个空格代替。 (C6)	✓	✓	✓	521	5.1.1.2
2.1.4	验证密码中是否允许使用任何可打印的 Unicode 字符，包括语言中立字符，例如空格和表情符号。	✓	✓	✓	521	5.1.1.2
2.1.5	验证用户可以更改其密码。	✓	✓	✓	620	5.1.1.2
2.1.6	验证密码更改功能是否需要用户的当前密码和新密码。	✓	✓	✓	620	5.1.1.2

#	说明	L1	L2	L3	CWE	NIST §
2.1.7	验证在账户注册、登录和密码更改过程中提交的密码，是否出现在被泄露过的密码中，这些密码可以是本地的（如符合系统密码策略的前 1000 个或 10000 个最常见的密码），也可以使用外部 API。 如果使用 API，应使用零知识证明或其他机制，以确保纯文本密码不被发送或用于验证密码的违反状态。 如果密码被泄露，应用程序必须要求用户设置一个新的未被泄露的密码。 (C6)	✓	✓	✓	521	5.1.1.2
2.1.8	验证是否提供了密码强度表，以帮助用户设置更强的密码。	✓	✓	✓	521	5.1.1.2
2.1.9	验证是否有限制允许的字符类型的密码组成规则。对大写或小写、数字或特殊字符不应有任何要求。 (C6)	✓	✓	✓	521	5.1.1.2
2.1.10	验证没有定期更换凭证或密码历史的要求。	✓	✓	✓	263	5.1.1.2
2.1.11	验证是否允许 “粘贴” 功能、浏览器密码辅助工具和外部密码管理器。	✓	✓	✓	521	5.1.1.2
2.1.12	验证用户可以选择临时查看整个屏蔽的密码，或者在没有内置功能的平台上临时查看密码的最后输入的字符。	✓	✓	✓	521	5.1.1.2

注意：允许用户查看密码或临时查看最后一个字符的目的，是为了提高凭证输入的可用性，尤其是在使用更长的密码、口令和密码管理器时。 包含该要求的另一个原因，是为了防止测试报告不必要地要求组织重写内置平台密码字段的行为，从而保持这种现代用户友好的安全体验。

V2.2 通用身份验证器的安全性

身份验证器的敏捷性，对于面向未来的应用程序至关重要。 重构应用程序验证器以允许用户根据偏好添加额外的验证器，并允许以有序的方式停用已弃用或不安全的验证器。

NIST 将电子邮件和 SMS 视为 “受限”的身份验证器类型，它们很可能在未来的某个时候从 NIST 800-63 以及 ASVS 中删除。 应用程序应计划一个不需要使用电子邮件或短信的路线图。

#	说明	L1	L2	L3	CWE	NIST §
2.2.1	验证反自动化控制的措施能够有效地缓解被泄露的凭证测试、暴力破解和账户锁定攻击。 这些控制措施包括阻止最常见的泄露密码、软锁定、速率限制、验证码、每次尝试后逐渐增加的间隔时间、IP 地址限制，或基于风险的限制，例如位置、设备上的首次登录、最近解锁账户的尝试等类似情况。 验证单个帐户每小时的失败尝试次数不超过 100 次。	✓	✓	✓	307	5.2.2 / 5.1.1.2 / 5.1.4.2 / 5.1.5.2

#	说明	L1	L2	L3	CWE	NIST §
2.2.2	验证弱身份验证器（例如 SMS 和电子邮件）的使用，仅限于二次验证和批准交易，而不是作为更安全的认证方法的替代。 验证是否在弱方法之前提供了更强的方法，用户是否意识到风险，或者是否采取了适当的措施来限制帐户泄露的风险。	✓	✓	✓	304	5.2.10
2.2.3	验证在更新认证信息（如凭证重置、电子邮件或地址变更、从未知或风险地点登录）后向用户发送安全通知。 最好使用推送通知——而不是短信或电子邮件，但在没有推送通知的情况下，只要通知中没有披露敏感信息，短信或电子邮件也是可以接受的。	✓	✓	✓	620	
2.2.4	验证对网络钓鱼的抗冒充性，如使用多因素认证、有意图的加密设备（如有推送认证的连接密钥），或在更高的 AAL 级别，客户端证书。			✓	308	5.2.5
2.2.5	验证当凭证服务提供者（CSP）和验证认证的应用程序分开时，两个端点之间有相互认证的 TLS（mTLS）。			✓	319	5.2.6
2.2.6	验证抗重放性，是否通过强制使用一次性密码（OTP）设备、加密认证器或查询代码。			✓	308	5.2.8
2.2.7	通过要求输入 OTP 令牌或用户发起的动作（如按下 FIDO 硬件钥匙的按钮）来验证认证意图。			✓	308	5.2.9

V2.3 身份验证器生命周期

认证器是密码、软令牌、硬件令牌和生物识别设备。认证器的生命周期对应用程序的安全至关重要——如果任何人都可以在没有身份证明的情况下自行注册一个账户，那么对身份断言的信任就会很低。 对于像 Reddit 这样的社交媒体网站，这是完全可以的。 对于银行系统来说，更加注重凭证和设备的注册和发放（对应用程序的安全至关重要）。

注意：密码不要有最长的使用寿命，也不要进行密码轮换。 应检查密码是否已泄露，而不是定期更换。

#	说明	L1	L2	L3	CWE	NIST §
2.3.1	验证系统生成的初始密码或激活码应该是安全随机生成的，应该至少有 6 个字符的长度，可以包含字母和数字，并在短时间内过期。这些初始秘密不得被允许成为长期密码。	✓	✓	✓	330	5.1.1.2 / A.3
2.3.2	验证是否支持注册和使用用户提供的认证设备，如 U2F 或 FIDO 令牌。		✓	✓	308	6.1.3

#	说明	L1	L2	L3	CWE	NIST §
2.3.3	验证更新指令的发送时间是否足够，以更新有时间限制的认证器。		✓	✓	287	6.1.4

V2.4 凭证存储

架构师和开发人员在构建或重构代码时，应遵守本节。本节内容只能使用源代码审查或通过安全单元或集成测试来完全验证。渗透测试不能识别这里面的任何问题。

经批准的单向密钥推导功能列表详见 NIST 800-63 B 第 5.1.1.2 节，以及 BSI Kryptographische Verfahren: Empfehlungen und Schlussellängen (2018)。除了上面这些选择，还可以根据最新的国家或地区选择算法和密钥长度标准。

此部分无法进行渗透测试，因此控制措施未标记为 L1。但是，如果证书被盗，这部分对证书的安全至关重要，因此如果为架构或编码指南或源代码审查清单 Fork ASVS，请在您的私有版本中将这些控制措施放回 L1。

#	说明	L1	L2	L3	CWE	NIST §
2.4.1	验证密码是以一种可以抵抗离线攻击的形式存储的。密码应使用认可的单向密钥推导或密码散列函数进行加盐和散列。密钥推导和密码散列函数，在生成密码散列时，将密码、盐和计算成本作为输入。 (C6)		✓	✓	916	5.1.1.2
2.4.2	验证盐的长度至少为 32 位，并且是任意选择的，以减少存储的哈希值之间的碰撞。对于每个凭证，应存储唯一的盐值和由此产生的哈希值。 (C6)		✓	✓	916	5.1.1.2
2.4.3	验证如果使用 PBKDF2，迭代次数应在验证服务器性能允许的范围内，一般至少为 100,000 次迭代。 (C6)		✓	✓	916	5.1.1.2
2.4.4	验证如果使用 bcrypt，工作系数应在验证服务器性能允许的范围内尽量大，最小为 10。 (C6)		✓	✓	916	5.1.1.2
2.4.5	验证是否执行了密钥派生函数的额外迭代，使用的是只有验证者知道的秘密盐值。使用经批准的随机位生成器 [SP 800-90Ar1] 生成盐值，并至少提供 SP 800-131A 最新修订版中规定的最低安全强度。秘密盐值应与散列密码分开存储（例如，在像硬件安全模块这样的专用设备中）。		✓	✓	916	5.1.1.2

在提到美国标准时，可以根据需要使用地区或当地标准来代替或补充美国标准。

V2.5 凭证恢复

#	说明	L1	L2	L3	CWE	NIST §
2.5.1	验证系统生成的初始激活或恢复密码，不会以明文形式发送给用户。 (C6)	✓	✓	✓	640	5.1.1.2
2.5.2	验证密码提示或基于知识的身份验证（所谓的“密码保护问题”）不存在。	✓	✓	✓	640	5.1.1.2
2.5.3	验证密码凭据恢复不会以任何方式泄露当前密码。 (C6)	✓	✓	✓	640	5.1.1.2
2.5.4	验证共享或默认帐户不存在（例如“root”、“admin”或“sa”）.	✓	✓	✓	16	5.1.1.2 / A.3
2.5.5	验证如果更改或替换了身份验证因素，则用户会收到此事件的通知。	✓	✓	✓	304	6.1.2.3
2.5.6	验证忘记密码以及其他恢复路径，使用了安全的恢复机制，例如基于时间的 OTP（TOTP）或其他软令牌、移动推送或其他离线恢复机制。 (C6)	✓	✓	✓	640	5.1.1.2
2.5.7	验证如果 OTP 或多因素身份验证因素丢失，身份证明的执行水平与注册时相同。		✓	✓	308	6.1.2.3

V2.6 查找密码认证

查找密码（译者注：Look-up secrets，类似于密码保护卡），是预先生成的秘密代码列表，类似于交易授权号码（TAN）、社交媒体恢复代码，或包含一组随机值的网格。 这些被安全地分配给用户。这些查询代码只使用一次，一旦全部使用，查询的秘密列表就会被丢弃。这种类型的认证器被认为是“你拥有的东西”。

#	说明	L1	L2	L3	CWE	NIST §
2.6.1	验证查找密文只能使用一次。		✓	✓	308	5.1.2.2
2.6.2	验证查询秘密有足够的随机性（112 位熵），如果少于 112 位熵，则用唯一的随机 32 位盐进行加盐，并用认可的单向散列进行散列。		✓	✓	330	5.1.2.2
2.6.3	验证查找秘密能够抵抗离线攻击，例如可预测的值。		✓	✓	310	5.1.2.2

V2.7 带外验证器

在过去，常见的带外验证器是包含密码重置链接的电子邮件或短信。攻击者利用这种薄弱的机制来重置他们尚未控制的账户，例如接管一个人的电子邮件账户并重新使用任何发现的重置链接。 处理带外验证有更好的方法。

安全的带外验证器是可以通过安全的二级信道与验证器进行通信的物理设备。 例子包括向移动设备推送通知。这种类型的验证器被认为是“你拥有的东西”。 当用户希望进行认证时，验证应用程序通过与认证器的连接，直接或间接地通过第三方服务向带外认证器发送一个消息。 该消息包含一个认证代码（通常是一个随机的六位数或一个模式化的批准对话框）。 验证应用程序等待通过主通道接收验证码，并将接收到的值的哈希值与原始验证码的哈希值进行比较。如果它们匹配，则带外验证器可以认为用户已通过身份验证。

ASVS 假定只有少数开发者会开发新的带外认证器，如推送通知，因此以下 ASVS 控制措施适用于验证器，如认证 API、应用程序和单点登录实现。 如果开发一个新的带外认证器，请参考 NIST 800-63B § 5.1.3.1。

不安全的带外认证器，如电子邮件和 VOIP 是不允许的。 PSTN 和 SMS 认证目前受到 NIST 的“限制”，应该被弃用，以支持推送通知或类似的方式。 如果你需要使用电话或短信的带外认证，请参见 NIST 800-63B § 5.1.3.3。

#	说明	L1	L2	L3	CWE	NIST §
2.7.1	验证默认情况下不提供短信或 PSTN 等带外的明文认证器，并首先提供推送通知等更强的替代方案。	✓	✓	✓	287	5.1.3.2
2.7.2	验证带外验证器在 10 分钟后将带外验证请求、代码或令牌过期。	✓	✓	✓	287	5.1.3.2
2.7.3	验证带外验证器身份验证请求、代码或令牌仅可使用一次，并且仅可用于原始身份验证请求。	✓	✓	✓	287	5.1.3.2
2.7.4	验证带外验证器和验证器是否通过安全的独立信道进行通信。	✓	✓	✓	523	5.1.3.2
2.7.5	验证带外验证器只保留认证代码的散列版本。		✓	✓	256	5.1.3.2
2.7.6	验证初始验证码是否由安全随机数生成器生成，包含至少 20 位熵（通常为 6 位数字随机数即可）。		✓	✓	310	5.1.3.2

V2.8 一次性验证器

单因素一次性密码（OTP），是显示持续变化的伪随机一次性挑战的物理或软令牌。 这些设备使网络钓鱼（冒充）变得困难，但并非不可能。 这种类型的身份验证器被认为是“你拥有的东西”。 多因素令牌，类似于单因素 OTP，但需要输入有效的 PIN 码、生物识别解锁、USB 插入、NFC 配对或一些附加值（例如交易签名计算器）才能创建最终 OTP。

#	说明	L1	L2	L3	CWE	NIST §
2.8.1	验证基于时间的 OTP 在过期前有确定的使用寿命	✓	✓	✓	613	5.1.4.2 / 5.1.5.2
2.8.2	验证用于验证提交的 OTP 的对称密钥是否被高度保护，例如使用硬件安全模块或基于安全操作系统的密钥存储。		✓	✓	320	5.1.4.2 / 5.1.5.2

#	说明	L1	L2	L3	CWE	NIST §
2.8.3	验证 OTP 的生成、播种和验证是否使用了经过批准的加密算法。		✓	✓	326	5.1.4.2 / 5.1.5.2
2.8.4	验证基于时间的 OTP 在有效期内只能使用一次。		✓	✓	287	5.1.4.2 / 5.1.5.2
2.8.5	验证如果基于时间的多因素 OTP 令牌在有效期内被重复使用，将被记录并拒绝，同时向设备持有者发送安全通知。		✓	✓	287	5.1.5.2
2.8.6	验证物理单因素 OTP 生成器在被盗或其他损失的情况下可以被撤销。确保撤销在登录会话中立即生效，无论在何处。		✓	✓	613	5.2.1
2.8.7	验证生物特征身份验证器仅限于用作次要因素，与“你拥有的东西”和“你知道的东西”一起使用。		o	✓	308	5.2.3

V2.9 密码验证器

加密安全密钥是智能卡或 FIDO 密钥，用户必须将加密设备插入或配对到计算机上才能完成身份验证。验证者立即向加密设备或软件发送挑战随机数，设备或软件根据安全存储的加密密钥计算出响应。

对单因素密码设备和软件的要求，和多因素密码设备和软件的要求是一样的，都是证明密码认证者拥有认证因素。

#	说明	L1	L2	L3	CWE	NIST §
2.9.1	验证用于验证的加密密钥是否安全存储并防止泄露，例如使用可信平台模块（TPM）或硬件安全模块（HSM），或可以使用这种安全存储的操作系统服务。		✓	✓	320	5.1.7.2
2.9.2	验证质询随机数的长度至少为 64 位，并且在统计学上是唯一的，或在加密设备的生命周期内是唯一的。		✓	✓	330	5.1.7.2
2.9.3	验证在生成、播种和验证中使用经批准的加密算法。		✓	✓	327	5.1.7.2

V2.10 服务认证

此部分不可渗透测试，因此没有任何 L1 要求。但是，如果用于架构、编码或安全代码审查，请假设软件（就像 Java 密钥库一样）是 L1 的最低要求。 在任何情况下都不允许明文存储秘密。

注：

- HSM，硬件安全模块（Hardware Security Module）
- OS assisted，操作系统协助

#	说明	L1	L2	L3	CWE	NIST §
2.10.1	验证服务内机密不依赖于不变的凭据，例如密码、API 密钥或具有特权访问权限的共享帐户。		OS assisted	HSM	287	5.1.1.1
2.10.2	验证如果服务身份验证需要密码，则使用的服务帐户不是默认凭据（例如，root/root 或 admin/admin 是安装过程中某些服务的默认设置）。		OS assisted	HSM	255	5.1.1.1
2.10.3	验证存储的密码是否有足够的保护，以防止离线恢复攻击，包括本地系统访问。		OS assisted	HSM	522	5.1.1.1
2.10.4	验证密码、与数据库和第三方系统的集成、种子和内部机密以及 API 密钥都得到安全管理，不包含在源代码中或存储在源代码存储库中。 这种存储应能抵御离线攻击。建议使用安全的软件密钥存储（L1）、硬件 TPM 或 HSM（L3）来存储密码。		OS assisted	HSM	798	

美国机构的其他要求

美国机构对 NIST 800-63 有强制性要求。 一直以来，应用安全验证标准都是对几乎 100%的应用采取 80%的控制措施，而不是最后 20%的高级控制或那些有限的适用性。 因此，ASVS 是 NIST 800-63 的一个严格的子集，特别是对于 IAL1/2 和 AAL1/2 分类，但不够全面，特别是关于 IAL3/AAL3 分类。

我们强烈敦促美国政府机构全面审查和实施 NIST 800-63。

术语表

术语	含义
CSP	凭证服务提供者也称为身份提供者
Authenticator	验证密码、令牌、MFA、联合断言等的代码。
Verifier	“通过使用认证协议，验证申请人对一个或两个认证器的拥有和控制，来验证申请人的身份的实体。 为此，验证者可能还需要验证将认证器与用户的标识符联系起来的凭证，并检查其状态”
OTP	一次性密码
SFA	单因素身份验证，例如“您知道的东西”（记忆的秘密、密码、密码短语、pin），“您的特征“（生物特征识别、指纹、面部扫描），或“您拥有的东西“（OTP 令牌、智能卡等加密设备），
MFA	多因素认证，包括两个或多个单因素

参考文献

有关更多信息，请参阅：

- NIST 800-63 - Digital Identity Guidelines
- NIST 800-63 A - Enrollment and Identity Proofing
- NIST 800-63 B - Authentication and Lifecycle Management
- NIST 800-63 C - Federation and Assertions
- NIST 800-63 FAQ
- OWASP Testing Guide 4.0: Testing for Authentication
- OWASP Cheat Sheet - Password storage
- OWASP Cheat Sheet - Forgot password
- OWASP Cheat Sheet - Choosing and using security questions

V3 会话管理

控制目标

任何基于 Web 的应用程序或有状态的 API 的核心组成部分之一，是它控制和维护与之交互的用户或设备的状态的机制。会话管理将无状态协议变为有状态协议，这对于区分不同的用户或设备至关重要。

确保经过验证的应用程序满足以下高级会话管理需求:

- 会话对每个人都是唯一的，不能被猜测或共享。
- 会话在不再需要时失效，在不活动期间超时。

如前所述，这些要求已被调整为符合 NIST 800-63b 控制的一个子集，重点关注常见的威胁和经常被利用的认证弱点。 以前的验证要求已被淘汰、去重，或者在大多数情况下已调整为与强制性 NIST 800-63b 要求的意图保持高度一致。

安全验证要求

V3.1 基本会话管理安全

#	说明	L1	L2	L3	CWE	NIST §
3.1.1	验证应用不会在 URL 参数中显示会话令牌。	✓	✓	✓	598	

V3.2 会话绑定

#	说明	L1	L2	L3	CWE	NIST §
3.2.1	验证应用程序在用户身份验证时，生成新的会话令牌。 (C6)	✓	✓	✓	384	7.1
3.2.2	验证会话令牌具有至少 64 位的熵。 (C6)	✓	✓	✓	331	7.1
3.2.3	验证应用程序仅使用安全方法在浏览器中存储会话令牌，例如适当的 cookie 保护（参见第 3.4 节）或 HTML 5 会话存储。	✓	✓	✓	539	7.1
3.2.4	验证会话令牌是使用批准的加密算法生成的。 (C6)		✓	✓	331	7.1

会话管理必须使用 TLS 或其他安全传输通道。这在“通信安全”一章中有介绍。

V3.3 会话终止

会话超时已经与 NIST 800-63 保持一致，它允许比传统安全标准所允许的更长的会话超时。组织应查看下表，如果应用的风险需要更长的超时，NIST 值应是会话空闲超时的上限。

在此上下文中，L1 是 IAL1/AAL1，L2 是 IAL2/AAL3，L3 是 IAL3/AAL3。对于 IAL2/AAL2 和 IAL3/AAL3，空闲超时时间越短，表示注销或重新认证恢复会话的时间越短。

#	说明	L1	L2	L3	CWE	NIST §
3.3.1	验证注销和到期是否会使会话令牌无效，以便后退按钮或下游依赖方不会恢复经身份验证过的会话。 (C6)	✓	✓	✓	613	7.1
3.3.2	如果认证器允许用户保持登录状态，请验证在活跃使用或空闲一段时间过后，定期进行重新认证。 (C6)	30 天	12 小时 或 30 分钟不活动，可选 2FA	12 小时 或 15 分钟不活动，使用 2FA	613	7.2
3.3.3	验证应用程序是否提供了在成功更改密码（包括通过密码重置/恢复）后终止所有其他活动会话的选项，并且这在应用程序、联合登录（如果存在）和任何依赖方中都是有效的。		✓	✓	613	
3.3.4	验证用户能够查看并（在重新输入登录凭证后）注销当前的所有活动会话和设备。		✓	✓	613	7.1

V3.4 基于 Cookie 的会话管理

#	说明	L1	L2	L3	CWE	NIST §
3.4.1	验证基于 cookie 的会话令牌是否设置了'Secure'属性。 (C6)	✓	✓	✓	614	7.1.1
3.4.2	验证基于 cookie 的会话令牌是否设置了'HttpOnly'属性。 (C6)	✓	✓	✓	1004	7.1.1
3.4.3	验证基于 cookie 的会话令牌是否使用了'SameSite'属性，以限制跨站点请求伪造攻击（CSRF）的风险。 (C6)	✓	✓	✓	16	7.1.1
3.4.4	验证基于 cookie 的会话令牌是否使用'__Host-'前缀，这样 cookie 只会被发送到最初设置 cookie 的主机。	✓	✓	✓	16	7.1.1
3.4.5	验证如果应用程序在一个域名下发布，而其他应用程序设置或使用会话 cookie（这可能会泄露会话 cookie），则在基于 cookie 的会话令牌中设置路径属性（Path），尽可能使用最精确的路径。 (C6)	✓	✓	✓	16	7.1.1

V3.5 基于令牌的会话管理

基于令牌的会话管理包括 JWT、OAuth、SAML 和 API 密钥。其中，API 密钥公认较弱，不应该在新代码中使用。

#	说明	L1	L2	L3	CWE	NIST §
3.5.1	验证该应用允许用户撤销与链接应用建立信任关系的 OAuth 令牌。		✓	✓	290	7.1.2
3.5.2	验证应用程序使用会话令牌，而不是静态 API 密码或密钥，旧的实现除外。		✓	✓	798	
3.5.3	验证无状态会话令牌是否使用数字签名、加密等对策，来防止篡改、封装、重放、空密码和密钥替换等攻击。		✓	✓	345	

V3.6 联合重认证

本节与那些编写依赖方（RP）或凭证服务提供商（CSP）代码的人员有关。如果依赖于实现这些功能的代码，请确保正确处理这些问题。

#	说明	L1	L2	L3	CWE	NIST §
3.6.1	验证依赖方（RP）是否指定了凭证服务提供商（CSP）的最长身份验证时间，并且如果用户在该期间内未使用会话，CSP 是否会重新验证用户。			✓	613	7.2.1
3.6.2	验证凭证服务提供商（CSP）通知依赖方（RP）最后一次认证事件，以便 RP 确定他们是否需要重新认证用户。			✓	613	7.2.1

V3.7 针对会话管理漏洞的防御措施

已知的一些会话管理漏洞，与会话的用户体验（UX）相关。以前，根据 ISO 27002 要求，ASVS 需要阻止多个并发会话。 现在，阻止并发会话已不再合适，不仅因为现代用户有许多设备，或者应用程序是没有浏览器会话的 API，还因为在大多数这些实现中，最后一个身份验证者获胜，这通常是攻击者。本小节提供了使用代码阻止、延迟和检测会话管理攻击的主要指导。

半开放攻击的描述

在 2018 年初，一些金融机构遭到了攻击者所谓的“半开放攻击”（half-open attacks）。这个术语在行业中一直存在。攻击者以不同的专有代码库攻击了多家机构，实际上，在同一个机构中似乎也有不同的代码库。这种“半开放攻击”利用了许多现有认证、会话管理和访问控制系统中常见的设计模式缺陷。

攻击者通过试图锁定、重置或恢复一个凭证来开始半开放攻击。流行的会话管理设计模式，在未认证、半认证（密码重置、忘记用户名）和完全认证的代码之间，重用用户配置文件会话对象/模型。 这种设计模式会填充一个有效的会话对象或令牌，其中包含受害者的个人资料，包括密码哈希值和角色。如果访问控制检查在控制器或路由器没有正确验证用户是完全登录，攻击者将能够冒充用户。攻击手段可能包括：将用户的密码更改为已知值、更新电子邮件地址以执行有效的密码重置、禁用多因素身份验证或注册新的 MFA 设备、暴露或更改 API 密钥等。

#	说明	L1	L2	L3	CWE	NIST §
3.7.1	在允许任何敏感交易或帐户修改之前，验证应用程序确保完整、有效的登录会话，或要求重新验证（二次验证）。	✓	✓	✓	306	

参考文献

有关更多信息，请参阅：

- OWASP Testing Guide 4.0: Session Management Testing
- OWASP Session Management Cheat Sheet
- Set-Cookie Host- prefix details

V4 访问控制

控制目标

授权是一个概念，即只允许那些被允许使用资源的人访问资源。确保经过验证的应用程序满足以下高级要求：

- 访问资源的人员持有有效凭据才能这样做。
- 用户与一组明确定义的角色和权限相关联。
- 角色和权限元数据受到保护，不会被重放或篡改。

安全验证要求

V4.1 通用访问控制设计

#	描述	L1	L2	L3	CWE
4.1.1	验证应用程序是否在受信任的服务层上执行访问控制规则，尤其是在有客户端访问控制并且可能被绕过的情况下。	✓	✓	✓	602
4.1.2	验证访问控制所使用的所有用户和数据属性以及策略信息，不能被最终用户操纵，除非得到特别授权。	✓	✓	✓	639
4.1.3	验证是否存在最小权限原则——用户应该只能访问他们拥有特定授权的功能、数据文件、URL、控制器、服务和其他资源。这意味着防止欺骗或特权提升。 (C7)	✓	✓	✓	285
4.1.4	[已删除，与 4.1.3 重复]				
4.1.5	验证访问控制安全，在发生异常时是否失效。 (C10)	✓	✓	✓	285

V4.2 操作级访问控制

#	描述	L1	L2	L3	CWE
4.2.1	验证敏感数据和 API 的保护，防止针对创建、读取、更新和删除记录的不安全直接对象引用（IDOR）攻击，如创建或更新别人的记录，查看每个人的记录或删除所有记录。	✓	✓	✓	639
4.2.2	验证应用程序或框架是否实施了强大的反 CSRF 机制来保护经过身份验证的功能，以及有效的反自动化或反 CSRF 保护无需身份验证的功能。	✓	✓	✓	352

V4.3 其他访问控制注意事项

#	描述	L1	L2	L3	CWE
4.3.1	验证管理界面使用适当的多因素认证，防止未经授权的使用。	✓	✓	✓	419
4.3.2	验证目录浏览被禁用，除非特意需要。此外，应用程序不应允许披露文件或目录元数据，例如 Thumbs.db、.DS_Store、.git 或.svn 文件夹。	✓	✓	✓	548
4.3.3	验证应用程序对低价值的系统有额外的授权（如升级或自适应认证），对高价值的应用程序进行职责分离，以根据应用程序和过去的欺诈风险执行反欺诈控制。		✓	✓	732

参考文献

有关更多信息，请参阅：

- OWASP Testing Guide 4.0: Authorization
- OWASP Cheat Sheet: Access Control
- OWASP CSRF Cheat Sheet
- OWASP REST Cheat Sheet

V5 验证、过滤和编码

控制目标

最常见的 Web 应用程序安全漏洞，是在没有任何输出编码的情况下，直接使用来自客户端或环境的输入（缺乏正确的验证）。 这一弱点导致了 Web 应用程序中几乎所有的重大漏洞，例如跨站点脚本（XSS）、SQL 注入、解释器注入、语言环境/Unicode 攻击、文件系统攻击和缓冲区溢出。

确保经过验证的应用程序满足以下高级要求：

- 输入验证和输出编码架构有一条约定的管道来防止注入攻击。
- 输入数据是强类型的，经过验证，范围或长度检查，或者在最坏的情况下，经过消毒或过滤。
- 输出结果根据数据的上下文进行编码或转义，尽可能地接近解释器。

随着现代网络应用架构的发展，输出编码比以往任何时候都更重要。在某些情况下很难提供健壮的输入验证，因此使用更安全的 API，如参数化查询、自动转义的模板框架或精心选择的输出编码，对应用程序的安全性至关重要。

V5.1 输入验证

正确实施的输入验证控制，使用白名单列表和强数据类型，可以消除 90% 以上的所有注入攻击。长度和范围检查可以进一步减少这种情况。在应用程序架构、设计冲刺（Design Sprint）、编码以及单元和集成测试期间，需要构建安全输入验证。 尽管其中许多项目在渗透测试中找不到，但不实施它们的结果通常可以在 V5.3 - 输出编码和注入预防要求 中找到。建议开发人员和安全代码审查人员将本小节作为基础（正如所有项目都需要满足 L1 那样），以防止注入。

#	描述	L1	L2	L3	CWE
5.1.1	验证应用程序是否有 HTTP 参数污染攻击的防御措施，特别是当应用程序框架没有区分请求参数的来源（GET、POST、cookies、请求头或环境变量）。	✓	✓	✓	235
5.1.2	验证框架是否能防止批量参数分配攻击，或者应用程序是否有对策来防止不安全的参数分配，如将字段标记为私有等类型。 (C5)	✓	✓	✓	915
5.1.3	验证所有输入（HTML 表单字段、REST 请求、URL 参数、HTTP 请求头、cookies、批处理文件、RSS 源等）都使用“白名单”（允许列表）。 (C5)	✓	✓	✓	20
5.1.4	验证结构化数据是强类型的，并根据定义的模式进行验证，包括允许的字符、长度和模式（如信用卡号码、电子邮件地址、电话号码，或验证两个相关字段是否合理，如检查郊区和邮政编码是否匹配）。 (C5)	✓	✓	✓	20
5.1.5	验证 URL 重定向和转发的目标地址都在白名单中，或者在重定向到可能不受信任的内容时显示警告。	✓	✓	✓	601

V5.2 过滤和沙盒化

#	描述	L1	L2	L3	CWE
5.2.1	验证所有来自“所见即所得”编辑器或类似的不受信任的 HTML 输入，都已经通过 HTML 过滤库或框架功能，进行了适当的净化。 (C5)	✓	✓	✓	116
5.2.2	验证非结构化数据是否经过消毒处理，以执行安全措施，如允许的字符集和长度。	✓	✓	✓	138
5.2.3	验证应用程序在传递给邮件系统之前，对用户的输入进行过滤，以防止 SMTP 或 IMAP 注入。	✓	✓	✓	147
5.2.4	验证应用程序是否避免使用 eval()或其他动态代码执行功能。在没有其他选择的情况下，任何被包含的用户输入必须在执行前进行过滤或沙箱处理。	✓	✓	✓	95
5.2.5	验证应用程序是否对相关的用户输入进行过滤或沙箱处理，来防止模板注入攻击。	✓	✓	✓	94
5.2.6	验证应用程序是否通过验证或净化不受信任的数据或 HTTP 文件元数据（如文件名和 URL 输入字段），并使用协议、域、路径和端口的白名单，来防止 SSRF 攻击。	✓	✓	✓	918
5.2.7	验证应用程序是否过滤、禁用或沙盒处理了用户提供的可扩展矢量图（SVG）脚本内容，特别是与内联脚本产生的 XSS 有关的内容，以及外部对象。	✓	✓	✓	159
5.2.8	验证应用程序是否对用户提供的模板语言内容（脚本或表达式，如 Markdown、CSS 或 XSL 样式表、BBCode 或类似内容）进行过滤、禁用或沙盒处理。	✓	✓	✓	94

V5.3 输出编码和预防注入

靠近或邻近当前解释器的输出编码，对应用程序的安全至关重要。通常情况下，输出编码并不持久化，而是用于在适当的输出环境中使输出安全，以便立即使用。不进行输出编码，将最终形成一个不安全、可注入的应用程序。

#	描述	L1	L2	L3	CWE
5.3.1	验证输出编码是否与所需的解释器和环境相关。例如，根据 HTML 值、HTML 属性、JavaScript、URL 参数、HTTP 头、SMTP 等上下文的要求，使用专门的编码器，特别是来自不可信任的输入（如带有 Unicode 或单引号的名字，如ねこ或 O'Hara）。 (C4)	✓	✓	✓	116

#	描述	L1	L2	L3	CWE
5.3.2	验证输出编码是否保留了用户选择的字符集和地域，从而使任何 Unicode 字符点都能得到有效和安全的处理。 (C4)	✓	✓	✓	176
5.3.3	验证上下文感知，最好是自动——或者最差也是手动——转义输出，以防止反射、存储或基于 DOM 的 XSS。 (C4)	✓	✓	✓	79
5.3.4	验证数据选择或数据库查询（如 SQL、HQL、ORM、NoSQL）是否使用参数化查询、ORM、实体框架，或以其他方式防止数据库注入攻击。 (C3)	✓	✓	✓	89
5.3.5	验证在没有参数化或更安全机制的情况下，使用特定上下文的输出编码来防止注入攻击，例如使用 SQL 转义来防止 SQL 注入。 (C3, C4)	✓	✓	✓	89
5.3.6	验证应用程序是否可以防止 JSON 注入攻击、JSON eval 攻击和 JavaScript 表达式评估。 (C4)	✓	✓	✓	830
5.3.7	验证应用程序可以防止 LDAP 注入漏洞，或者已经实施了特定的安全控制来防止 LDAP 注入。 (C4)	✓	✓	✓	90
5.3.8	验证应用程序是否能防止操作系统命令注入，以及操作系统调用是否使用参数化的操作系统查询或使用上下文命令行输出编码。 (C4)	✓	✓	✓	78
5.3.9	验证应用程序是否能防止本地文件包含（LFI）或远程文件包含（RFI）攻击。	✓	✓	✓	829
5.3.10	验证应用程序是否能防止 XPath 注入或 XML 注入攻击。 (C4)	✓	✓	✓	643

注意：使用参数化查询或转义 SQL 并不总是足够的；表和列名、ORDER BY 等不能被转义。若在这些字段中转义用户提供的数据，会导致查询失败或 SQL 注入。

注意：SVG 格式在几乎所有情况下都明确允许 ECMA 脚本，所以可能无法完全阻止所有的 SVG XSS 向量。如果需要上传 SVG，我们强烈建议将这些上传的文件作为 text/plain 提供，或者使用一个单独的“用户提供内容”域，以防止成功的 XSS 接管应用程序。

V5.4 内存、字符串和非托管代码

以下要求仅在应用程序使用系统语言或非托管代码时适用。

#	描述	L1	L2	L3	CWE
5.4.1	验证应用程序是否使用内存安全字符串、更安全的内存复制和指针运算，以检测或防止堆栈、缓冲区或堆溢出。		✓	✓	120
5.4.2	验证格式化字符串不接受潜在的有害输入，并且是常量。		✓	✓	134
5.4.3	验证运用了符号、范围和输入验证技术来防止整数溢出。		✓	✓	190

V5.5 预防反序列化

#	描述	L1	L2	L3	CWE
5.5.1	验证序列化对象是否使用完整性检查或加密，以防止恶意对象的创建或数据篡改。 (C5)	✓	✓	✓	502
5.5.2	验证应用程序正确限制 XML 解析器，使其只使用最严格的配置，并确保禁用不安全的功能，如解析外部实体，以防止 XML 外部实体注入（XXE）攻击。	✓	✓	✓	611
5.5.3	验证自定义代码和第三方库（如 JSON、XML 和 YAML 解析器）禁止或限制不受信任数据的反序列化。	✓	✓	✓	502
5.5.4	验证在浏览器或基于 JavaScript 的后端解析 JSON 时，使用 JSON.parse 来解析 JSON 文档。不使用 eval() 来解析 JSON。	✓	✓	✓	95

参考文献

有关更多信息，请参阅：

- OWASP Testing Guide 4.0: Input Validation Testing
- OWASP Cheat Sheet: Input Validation
- OWASP Testing Guide 4.0: Testing for HTTP Parameter Pollution
- OWASP LDAP Injection Cheat Sheet
- OWASP Testing Guide 4.0: Client Side Testing
- OWASP Cross Site Scripting Prevention Cheat Sheet
- OWASP DOM Based Cross Site Scripting Prevention Cheat Sheet
- OWASP Java Encoding Project
- OWASP Mass Assignment Prevention Cheat Sheet
- DOMPurify - Client-side HTML Sanitization Library
- XML External Entity (XXE) Prevention Cheat Sheet

有关自动转义的更多信息，请参阅：

- Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems
- AngularJS Strict Contextual Escaping
- AngularJS ngBind
- Angular Sanitization
- Angular Security
- ReactJS Escaping
- Improperly Controlled Modification of Dynamically-Determined Object Attributes

有关反序列化的更多信息，请参阅：

- OWASP Deserialization Cheat Sheet

- OWASP Deserialization of Untrusted Data Guide

V6 存储密码学

控制目标

确保经过验证的应用程序满足以下高级要求：

- 所有加密模块以安全的方式失效，并且正确处理错误。
- 使用合适的随机数发生器。
- 密钥访问被安全地管理。

V6.1 数据分类

最重要的资产是由应用程序处理、存储或传输的数据。始终执行隐私影响评估，对任何存储数据的数据保护需求进行正确分类。

#	说明	L1	L2	L3	CWE
6.1.1	验证受监管的私人数据在静止状态下是否被加密存储，如个人身份信息（PII）、敏感个人信息或经评估可能受制于欧盟 GDPR 的数据。		✓	✓	311
6.1.2	验证受监管的健康数据在静止状态下是否被加密存储，如医疗记录、医疗设备详情或去匿名化的研究记录。		✓	✓	311
6.1.3	验证受监管的金融数据在静止状态下是否被加密存储，如金融账户、违约或信用记录、税务记录、工资记录、受益人或去匿名化的市场或研究记录。		✓	✓	311

V6.2 算法

密码学的最新进展意味着以前安全的算法和密钥长度不再安全或足以保护数据。因此，应该可以改变算法。

虽然这一部分不容易进行渗透测试，但开发人员应该把这一整节视为强制性的，即使在大多数项目中 L1 都没有要求。

#	说明	L1	L2	L3	CWE
6.2.1	验证所有的加密模块即使在故障时也是安全的，并且处理错误的方式不会使 Padding Oracle 攻击得逞。	✓	✓	✓	310
6.2.2	验证使用业界认可或政府批准的加密算法、模式和库，而不是自定义编码的加密技术。 (C8)		✓	✓	327
6.2.3	验证加密初始化向量、密码配置和分组模式是否使用最新建议进行安全配置。		✓	✓	326

#	说明	L1	L2	L3	CWE
6.2.4	验证随机数、加密或散列算法、密钥长度、轮次、密码或模式，可以在任何时候重新配置、升级或交换，以防止密码中断。 (C8)		✓	✓	326
6.2.5	验证不使用已知不安全的分组模式（如 ECB 等）、填充模式（如 PKCS#1 v1.5 等）、小块大小的密码（如 Triple-DES、Blowfish 等）和弱散列算法（如 MD5、SHA1 等），除非需要向后兼容。		✓	✓	326
6.2.6	验证随机数、初始化向量和其他一次性使用的数字，不得与特定的加密密钥使用超过一次。生成方法必须适合所使用的算法。		✓	✓	326
6.2.7	验证加密数据是否通过签名、认证的密码模式或 HMAC 进行身份验证，以确保密文不会被未经授权的一方更改。			✓	326
6.2.8	验证所有的密码操作都是恒定时间的，在比较、计算或返回中没有“短路”操作，以避免信息泄漏。			✓	385

V6.3 随机值

真正的伪随机数生成（PRNG）很难实现。通常，如果过度使用，系统内良好的熵源不但很快耗尽，而且随机性较小的源会导致可预测的密钥和秘密。

#	说明	L1	L2	L3	CWE
6.3.1	验证所有的随机数、随机文件名、随机 GUID 和随机字符串，都是使用加密模块认可的加密安全随机数生成器生成的，而这些随机值旨在不被攻击者猜测。		✓	✓	338
6.3.2	验证是否使用 GUID v4 算法和加密安全伪随机数生成器（CSPRNG）创建了随机 GUID。使用其他伪随机数生成器创建的 GUID 可能是可预测的。		✓	✓	338
6.3.3	验证应用程序即使在处于高负载下时也使用适当的熵创建随机数，或者应用程序在这种情况下优雅地降级。			✓	338

V6.4 密钥管理

虽然这一部分不容易进行渗透测试，但开发人员应将整个部分视为强制性的，即使大多数项目中都缺少 L1 的要求。

#	说明	L1	L2	L3	CWE
6.4.1	验证秘密管理解决方案，如钥匙库，用于安全地创建、存储、控制对秘密的访问和销毁。 (C8)		✓	✓	798

#	说明	L1	L2	L3	CWE
6.4.2	验证密钥材料是否未暴露给应用程序，而是使用一个隔离的安全模块（如保险库）进行加密操作。 (C8)		✓	✓	320

参考文献

有关更多信息，请参阅：

- OWASP Testing Guide 4.0: Testing for weak Cryptography
- OWASP Cheat Sheet: Cryptographic Storage
- FIPS 140-2

V7 错误处理和日志记录

控制目标

错误处理和记录的主要目标，是为用户、管理员和事件响应团队提供有用的信息。目标不是创建大量日志，而是创建高质量的日志，信息多于丢弃的噪声。

高质量的日志往往包含敏感数据，必须按照当地的数据隐私法或指令进行保护。这应该包括：

- 除非特别要求，否则不收集或记录敏感信息。
- 确保所有记录的信息被安全地处理，并根据其数据分类进行保护。
- 确保日志不会被永久存储，而是有一个尽可能短的生命周期。

如果日志包含私人或敏感数据（各国对这些数据的定义各不相同），则日志将成为应用程序所持有的最敏感信息之一，因此对攻击者本身来说非常有吸引力。

同样重要的是，要确保应用程序安全地故障，而且错误不会泄露不必要的信息。

V7.1 日志内容

记录敏感信息是很危险的——日志本身就成了机密，这意味着它们需要被加密，成为保留政策的对象，并且必须在安全审计中被披露。确保只有必要的信息被保存在日志中，当然没有支付、凭证（包括会话令牌）、敏感或个人身份信息。

V7.1 涵盖了 OWASP Top 10 2017:A10. 由于 2017:A10 和本节不可通过渗透测试来验证，重要的是：

- 开发人员要确保完全遵守本节，就像所有项目都被标记为 L1 一样。
- 渗透测试人员通过访谈，屏幕截图或断言来验证 V7.1 中所有项目的完全合规性。

#	描述	L1	L2	L3	CWE
7.1.1	验证应用程序不记录凭证或支付细节。会话令牌应该只以不可逆的散列形式存储在日志中。 (C9, C10)	✓	✓	✓	532
7.1.2	验证应用程序不会记录当地隐私法或相关安全政策规定的其他敏感数据。 (C9)	✓	✓	✓	532
7.1.3	验证应用程序是否记录安全相关事件，例如成功和失败的认证事件、访问控制失败、反序列化失败和输入验证失败的事件。 (C5, C7)		✓	✓	778
7.1.4	验证每个日志事件都包含必要的信息，以便在事件发生后详细调查时间轴。 (C9)		✓	✓	778

V7.2 日志处理

及时的日志记录对于事件的审计、研判和升级是至关重要的。确保应用程序的日志是清晰的，可以很容易地在本地监测和分析，或将日志发送到远程监控系统。

V7.2 涵盖了 OWASP Top 10 2017:A10. 由于 2017:A10 和本节不可通过渗透测试来验证，重要的是：

- 开发人员要确保完全遵守本节，就像所有项目都被标记为 L1 一样。
- 渗透测试人员通过访谈，屏幕截图或断言来验证 V7.2 中所有项目的完全合规性。

#	描述	L1	L2	L3	CWE
7.2.1	验证所有的认证决策都被记录下来，不存储敏感的会话令牌或密码。这应该包括安全调查所需的具有相关元数据的请求。		✓	✓	778
7.2.2	验证是否可以记录所有访问控制决策并记录所有失败的决策。这应包括安全调查所需的具有相关元数据的请求。		✓	✓	285

V7.3 日志保护

可以轻易修改或删除的日志，对于调查和起诉毫无用处。日志的披露可能会暴露有关应用程序或其包含的数据的内部细节。在保护日志免遭未经授权的披露、修改或删除时，必须小心谨慎。

#	描述	L1	L2	L3	CWE
7.3.1	验证所有日志组件是否对数据进行了适当的编码，以防止日志注入。(C9)		✓	✓	117
7.3.2	[已删除，与 7.3.1 重复]				
7.3.3	验证安全日志是否受到保护，防止未授权的访问或修改。(C9)		✓	✓	200
7.3.4	验证时间源是否同步到正确的时间和时区。如果系统是全球性的，强烈考虑只用 UTC 来记录，以协助事件后的取证分析。(C9)		✓	✓	

注意：日志编码（7.3.1）很难使用自动化工具和渗透测试进行测试和审查，但架构师、开发人员和源代码审查人员应将其视为 L1 要求。

V7.4 错误处理

错误处理的目的，是允许应用程序提供与安全有关的事件，用于监控、研判和升级。目的不是为了创建日志。当记录安全相关的事件时，要确保日志有其目的，并且可以被 SIEM 或分析软件识别。

#	描述	L1	L2	L3	CWE
7.4.1	验证在发生意外或安全敏感错误时，是否显示通用信息，可能带有支持人员可以用于调查的唯一 ID。(C10)	✓	✓	✓	210
7.4.2	验证整个代码库是否使用了异常处理（或类似功能），以说明预期和非预期的错误情况。(C10)		✓	✓	544
7.4.3	验证是否定义了“最后手段”的错误处理程序，以捕获所有未处理的异常。(C10)		✓	✓	431

注意：某些语言，例如 Swift 和 Go ——以及通过常见的设计实践——许多函数式语言，不支持异常或最后的事件处理程序。在这种情况下，架构师和开发人员应该使用一种模式、语言或框架友好的方式，来确保应用程序能够安全地处理异常、意外或安全相关的事件。

参考文献

有关更多信息，请参阅：

- OWASP Testing Guide 4.0 content: Testing for Error Handling
- OWASP Authentication Cheat Sheet section about error messages

V8 数据保护

Control Objective

健全的数据保护有三个关键因素。机密性、完整性和可用性（CIA）。这个标准假定数据保护是在一个可信的系统上执行的，比如服务器，它已经被加固并有足够的保护措施。

应用程序必须假设所有的用户设备都以某种方式受到损害。如果应用程序在不安全的设备上传输或存储敏感信息，如共享电脑、手机和平板电脑，应用程序有责任确保存储在这些设备上的数据是加密的，不能轻易地被非法获取、改变或披露。

确保经过验证的应用程序满足以下高水平的数据保护要求：

- 机密性。数据应受到保护，在传输过程中和存储时都不会被未经授权的观察或披露。
- 完整性。应保护数据不被未经授权的攻击者恶意创建、更改或删除。
- 可用性。数据应按要求提供给授权的用户。

V8.1 通用数据保护

#	说明	L1	L2	L3	CWE
8.1.1	验证应用程序保护敏感数据不被缓存在负载均衡和应用程序缓存等服务器组件中。		✓	✓	524
8.1.2	验证在服务器上所存储敏感数据的所有缓存或临时副本是否受到保护（防止未经授权的访问），或在被授权用户访问后清除/失效。		✓	✓	524
8.1.3	验证应用程序尽量减少请求中的参数数量，如隐藏字段、Ajax 变量、cookies 和请求头。		✓	✓	233
8.1.4	验证应用程序能够检测并提醒异常的请求数量，例如按 IP、用户、每小时或每天的总数，或其它对应用程序有意义的指标。		✓	✓	770
8.1.5	验证是否对重要数据进行了定期备份，是否对数据进行了测试恢复。			✓	19
8.1.6	验证备份的安全存储，防止数据被盗或损坏。			✓	19

V8.2 客户端数据保护

#	说明	L1	L2	L3	CWE
8.2.1	验证应用程序设置足够的“禁止缓存”头，以便敏感数据不会在现代浏览器中被缓存。	✓	✓	✓	525
8.2.2	验证存储在浏览器存储（例如 localStorage、sessionStorage、IndexedDB 或 cookie）中的数据不包含敏感数据。	✓	✓	✓	922

#	说明	L1	L2	L3	CWE
8.2.3	在客户端或会话终止后，验证经过身份验证的数据已从客户端存储（例如浏览器 DOM）中清除。	✓	✓	✓	922

V8.3 敏感私有数据

本小节有助于保护敏感数据免遭未经授权的创建、读取、更新或删除，尤其是批量数据。

遵守本节意味着遵守 V4 访问控制，尤其是 V4.2。例如，为了防止个人敏感信息的未授权的更新或泄露，需要遵守 V4.2.1。请遵守本节和 V4 以获得全面覆盖。

注意：隐私法规和法律，例如澳大利亚隐私原则 APP-11 或 GDPR，直接影响应用程序必须如何处理敏感个人信息的存储、使用和传输的实施。范围从严厉的处罚到简单的建议。请查阅您当地的法律法规，并根据需要咨询专业的隐私专家或律师。

#	说明	L1	L2	L3	CWE
8.3.1	验证敏感数据是在 HTTP 消息正文或请求头中被发送到服务器，以及 HTTP 请求方法的查询字符串参数都不包含敏感数据。	✓	✓	✓	319
8.3.2	验证用户是否有途径按需删除或导出自己的数据。	✓	✓	✓	212
8.3.3	验证向用户提供了关于收集和使用其个人信息的明确语言，并且在以任何方式使用这些数据之前，用户已勾选了同意。	✓	✓	✓	285
8.3.4	验证应用程序创建和处理的所有敏感数据是否已被识别，并确保已制定了如何处理敏感数据的策略。 (C8)	✓	✓	✓	200
8.3.5	如果数据是根据相关数据保护指令收集的或（应用）要求记录访问日志，验证访问敏感数据是否被审计（不记录敏感数据本身）。		✓	✓	532
8.3.6	为了减少内存转储攻击，一旦不再需要内存中的敏感信息，请检查该敏感信息是否会被覆盖（使用 0 或随机数）。		✓	✓	226
8.3.7	验证需要加密的敏感信息或私有信息是否使用经过批准的机密性和完整性算法加密。 (C8)		✓	✓	327
8.3.8	验证敏感的个人信息是否符合数据保留分类，以便自动、按计划或根据情况需要删除旧数据或过时数据。		✓	✓	285

在考虑数据保护时，首要的考虑应该围绕批量提取、修改或过度使用。例如，许多社交媒体系统只允许用户每天添加 100 个新好友，但这些请求来自哪个系统并不重要。银行平台可能希望阻止每小时超过 5 笔的、转移超过 1000 欧元的外部交易。每个系统的要求可能非常不同，所以决定“异常”必须考虑威胁模型和商业风险。重要的标准是检测、遏制，或者最好是阻止这种异常批量行为的能力。

参考文献

有关更多信息，请参阅：

- Consider using Security Headers website to check security and anti-caching headers
- OWASP Secure Headers project
- OWASP Privacy Risks Project
- OWASP User Privacy Protection Cheat Sheet
- European Union General Data Protection Regulation (GDPR) overview
- European Union Data Protection Supervisor - Internet Privacy Engineering Network

V9 通讯

控制目标

确保经过验证的应用程序满足以下高级要求：

- 要求 TLS 或强加密，与内容的敏感性无关。
- 遵循最新指南，包括：
  - 配置建议
  - 首选算法和密码
- 避免使用弱的或即将被废弃的算法和密码，除非是最后的手段。
- 禁用已废弃或已知不安全的算法和密码。

在这些要求范围内：

- 了解业界对安全 TLS 配置的建议，因为它经常变化（往往是由于现有算法和密码的灾难性破坏）。
- 使用最新版本的 TLS 配置审查工具，来配置首选顺序和算法选择。
- 定期检查你的配置，以确保安全通信始终存在并有效。

V9.1 客户端通信安全

确保所有客户端消息都通过加密网络发送，使用 TLS 1.2 或更高版本。 使用最新的工具定期检查客户端配置。

#	描述	L1	L2	L3	CWE
9.1.1	验证所有客户端连接都使用了 TLS，并且不会降级到不安全或未加密的通信。 (C8)	✓	✓	✓	319
9.1.2	使用最新的 TLS 测试工具，验证是否只启用了强密码套件，并将最强的密码套件设置为首选。	✓	✓	✓	326
9.1.3	验证只启用最新推荐版本的 TLS 协议，如 TLS 1.2 和 TLS 1.3。最新版本的 TLS 协议应该是首选项。	✓	✓	✓	326

V9.2 服务器通信安全

服务器通信不仅仅是 HTTP。与其他系统的安全连接，例如监控系统、管理工具、远程访问和 ssh、中间件、数据库、大型机、合作伙伴或外部源系统——必须到位。所有这些都必须加密，以防止“外面安全，里面被轻易截获”。

#	描述	L1	L2	L3	CWE
9.2.1	验证与服务器的连接是否使用受信任的 TLS 证书。在使用内部生成或自签名证书的情况下，必须将服务器配置为只信任特定的内部 CA 和特定的自签证书。所有其他的都应该被拒绝。		✓	✓	295
9.2.2	确认所有入站和出站连接都使用了 TLS 等加密通信，包括管理端口、监控、身份验证、API 或 Web 服务调用、数据库、云、serverless、大型机、外部和合作伙伴的连接。服务器不得回退到不安全或未加密的协议。		✓	✓	319
9.2.3	验证所有外部系统中与敏感信息/功能相关的加密连接，均已通过身份验证。		✓	✓	287
9.2.4	验证是否启用并配置了正确的证书吊销，例如在线证书状态协议（OCSP）Stapling。		✓	✓	299
9.2.5	验证是否记录了后端 TLS 连接失败（的事件）。			✓	544

参考文献

有关更多信息，请参阅：

- OWASP – TLS Cheat Sheet
- OWASP - Pinning Guide
- 关于 “TLS 的批准模式“ 的说明:
  - 在过去，ASVS 提到了美国标准 FIPS 140-2，但作为一个全球标准美国标准的应用可能充满困难、矛盾或混乱。
  - 实现第 9.1 节的更好方法是审查指南，如 Mozilla's Server Side TLS or generate known good configurations，并使用已知最新的 TLS 评估工具来获得所需的安全等级。

V10 恶意代码

控制目标

确保代码满足以下高级要求：

- 恶意活动得到安全和适当的处理，不会影响应用程序的其余部分。
- 没有定时炸弹或其他基于时间的攻击。
- 不会向恶意或未经授权的目的地“打电话回家”。
- 没有后门、复活节彩蛋、Salami 攻击、rootkit 或攻击者可以控制的未授权代码。

发现恶意代码是否定的证明，这是不可能被充分验证的。应尽最大努力，确保代码没有固有的恶意代码或不需要的功能。

V10.1 代码完整性

对恶意代码的最佳防御是“信任，但要验证”。在许多司法管辖区，将未经授权或恶意的代码片段引入代码，通常是刑事犯罪。策略和过程应明确对恶意代码的制裁。

首席开发人员应该定期检查代码签入，特别是那些可能去访问时间、I/O 或网络功能的代码签入。

#	描述	L1	L2	L3	CWE
10.1.1	验证是否使用了代码分析工具，可以检测潜在的恶意代码，如时间函数、不安全的文件操作和网络连接。			✓	749

V10.2 恶意代码搜索

恶意代码极为罕见，难以检测。手动逐行审查代码可以帮助寻找逻辑炸弹，但即使是最有经验的代码审查员也很难找到恶意代码，哪怕事先知道它们的存在。

如果不能完全接触到源代码，包括第三方库，就不可能遵守本节的规定。

#	描述	L1	L2	L3	CWE
10.2.1	验证应用程序的源代码和第三方库不包含未经授权的回连或数据收集功能。如果存在这样的功能，在收集任何数据之前，要获得用户的操作许可。		✓	✓	359
10.2.2	验证应用程序不会对隐私相关的功能或传感器（例如联系人、摄像头、麦克风或位置）要求不必要或过度的权限。		✓	✓	272
10.2.3	验证应用程序的源代码和第三方库不包含后门，如硬编码或额外的未记录的账户或密钥、代码混淆、未记录的二进制 blobs、rootkits 或反调试、不安全的调试特性，或其他过时、不安全或隐藏的功能，一旦被发现可能会被恶意使用。			✓	507

#	描述	L1	L2	L3	CWE
10.2.4	通过搜索日期和时间相关函数，来验证应用程序源代码和第三方库不包含时间炸弹。			✓	511
10.2.5	验证应用程序源代码和第三方库不包含恶意代码，例如 salami 攻击、逻辑绕过或逻辑炸弹。			✓	511
10.2.6	验证应用程序的源代码和第三方库不包含复活节彩蛋或任何其他潜在的冗余功能。			✓	507

V10.3 应用程序完整性

应用程序被部署后，恶意代码仍然可以被插入。应用程序需要保护自己免受常见的攻击，例如执行来自不受信任来源的未签名代码或子域接管。

本节内容的实现，很可能是操作性和持续性的。

#	描述	L1	L2	L3	CWE
10.3.1	验证如果应用程序具有客户端或服务器自动更新功能，则应通过安全通道获得更新，并进行数字签名。更新代码必须在安装或执行更新之前验证更新的数字签名。	✓	✓	✓	16
10.3.2	验证应用程序是否采用了完整性保护，如代码签名或子资源完整性。应用程序不得从不受信任的来源加载或执行代码，例如从不可信任的来源或互联网加载模块、插件、代码或库。	✓	✓	✓	353
10.3.3	如果应用程序依赖 DNS 条目或 DNS 子域，例如过期的域名、过时的 DNS 指针或 CNAME、公共源代码库中过期的项目或临时的云 API 接口、serverless 功能或存储桶（*autogen-bucket-id*.cloud.example.com）或类似情况，则验证该应用程序是否具有防止子域接管的措施。保护措施可以包括确保定期检查应用程序使用的 DNS 名称是否过期或改变。	✓	✓	✓	350

参考文献

有关更多信息，请参阅：

- Hostile Subdomain Takeover, Detectify Labs
- Hijacking of abandoned subdomains part 2, Detectify Labs

V11 业务逻辑

控制目标

确保经过验证的应用程序满足以下高级要求：

- 业务逻辑流程是串行的，按顺序处理的，并且不能被绕过。
- 业务逻辑包括检测和防止自动化攻击，如连续的小额资金转移，或一次添加上百万个好友等。
- 高价值的业务逻辑流已经考虑了滥用情况和恶意行为者，并有防止欺骗、篡改、信息披露和特权提升攻击的保护措施。

V11.1 业务逻辑安全

业务逻辑安全对每个应用程序来说都是非常独特的，因此没有通用的检查表。业务逻辑安全必须设计成能够抵御可能的外部威胁——它不能使用 Web 应用防火墙或安全通信来添加。我们建议在设计冲刺（Design Sprint）期间使用威胁建模，例如使用 OWASP Cornucopia 或类似工具。

#	描述	L1	L2	L3	CWE
11.1.1	验证应用程序仅按串行顺序处理同一用户的业务逻辑流，不会跳过步骤。	✓	✓	✓	841
11.1.2	验证应用程序将只处理业务逻辑流，所有步骤都在现实的人工时间内处理，即事务不会提交得太快。	✓	✓	✓	799
11.1.3	验证应用程序是否对特定的业务操作或交易有适当的限制，并在每个用户的基础上正确执行。	✓	✓	✓	770
11.1.4	验证应用程序具有反自动化的控制手段，以防止过度调用，如大量数据泄露、业务逻辑请求、文件上传或拒绝服务攻击。	✓	✓	✓	770
11.1.5	验证应用程序是否具有业务逻辑限制或验证，以防止可能的业务风险或威胁（使用威胁建模或类似方法识别）。	✓	✓	✓	841
11.1.6	验证应用程序是否存在 TOCTOU（Time Of Check to Time Of Use）问题 或敏感操作的其他条件竞争问题。		✓	✓	367
11.1.7	验证应用程序是否从业务逻辑角度监控异常事件或活动。例如，尝试执行无序的操作或普通用户永远不会尝试的操作。 (C9)		✓	✓	754
11.1.8	验证应用程序在检测到自动化攻击或异常活动时，具有可配置的警报。		✓	✓	390

参考文献

有关更多信息，请参阅：

- OWASP Web Security Testing Guide 4.1: Business Logic Testing

- 反自动化可以通过多种方式实现，包括使用 OWASP AppSensor 和 OWASP Automated Threats to Web Applications
- OWASP AppSensor 也可以帮助进行攻击检测和响应。
- OWASP Cornucopia

V12 文件和资源

控制目标

确保经过验证的应用程序满足以下高级要求：

- 不受信任的文件数据应以安全的方式进行相应处理。
- 从不可信任的来源获得的不可信任的文件数据被存储在 Web 目录之外，并具有有限的权限。

V12.1 文件上传

尽管 zip 炸弹很容易使用渗透测试技术进行测试，但它们被认为是 L2 及以上级别，以鼓励设计和开发时考虑仔细的人工测试，并避免对拒绝服务场景进行自动化或不熟练的手动渗透测试。

#	描述	L1	L2	L3	CWE
12.1.1	确认应用程序不会接受可能会填满存储空间或导致拒绝服务的大文件。	✓	✓	✓	400
12.1.2	验证应用程序在解压缩文件前，根据允许的最大解压缩尺寸和最大文件数检查压缩文件（如 zip，gz，docx，odt）。		✓	✓	409
12.1.3	验证文件大小配额和每个用户的最大文件数是否被强制执行，以确保单个用户不能用过多的文件或过大的文件填满存储。		✓	✓	770

V12.2 文件完整性

#	描述	L1	L2	L3	CWE
12.2.1	验证从不可信任的来源获得的文件，根据文件的内容，验证其是否为预期类型。		✓	✓	434

V12.3 文件执行

#	描述	L1	L2	L3	CWE
12.3.1	验证系统或框架文件系统不直接使用用户提交的文件名元数据，并且使用 URL API 来防止路径遍历。	✓	✓	✓	22
12.3.2	验证用户提交的文件名元数据是否经过验证或忽略，以防止通过本地文件包含（LFI） 泄露、创建、更新或删除本地文件。	✓	✓	✓	73
12.3.3	验证用户提交的文件名元数据是否经过验证或忽略，以防止通过远程文件包含（Remote File Inclusion， RFI）或服务器端请求伪造攻击（server - Server Side Request Forgery， SSRF）泄露或执行远程文件。	✓	✓	✓	98

#	描述	L1	L2	L3	CWE
12.3.4	验证应用程序通过验证或忽略用户提交的 JSON、JSONP 或 URL 参数中的文件名来防止反射文件下载（RFD），响应的 Content-Type 头应该设置为 text/plain，而 Content-Disposition 头应该有一个固定的文件名。	✓	✓	✓	641
12.3.5	验证未受信任的文件元数据不直接用于系统 API 或库，以防止操作系统命令注入。	✓	✓	✓	78
12.3.6	验证应用程序不包含或执行不可信任来源的功能，如未经验证的内容分发网络、JavaScript 库、node npm 库或服务器端 DLL。		✓	✓	829

V12.4 文件存储

#	描述	L1	L2	L3	CWE
12.4.1	验证从不受信任的来源获得的文件是否存储在 Web 根目录之外，并具有有限的权限。	✓	✓	✓	552
12.4.2	验证从不受信任的来源获得的文件是否已被防病毒扫描程序扫描，以防止上传和提供已知的恶意内容。	✓	✓	✓	509

V12.5 文件下载

#	描述	L1	L2	L3	CWE
12.5.1	验证网络层是否被配置为只提供具有特定文件扩展名的文件，以防止意外信息和源代码泄漏。例如，除非有需要，应阻止提供备份文件（如 .bak）、临时工作文件（如.swp）、压缩文件（.zip、.tar.gz 等）以及其他编辑人员常用的扩展名。	✓	✓	✓	552
12.5.2	验证对上传文件的直接请求永远不会作为 HTML/JavaScript 内容执行。	✓	✓	✓	434

V12.6 SSRF 保护

#	说明	L1	L2	L3	CWE
12.6.1	验证 Web 或应用程序服务器是否配置了资源或系统的白名单列表，服务器可以向其发送请求或加载数据/文件。	✓	✓	✓	918

参考文献

有关更多信息，请参阅：

- File Extension Handling for Sensitive Information
- Reflective file download by Oren Hafif
- OWASP Third Party JavaScript Management Cheat Sheet

V13 API 和 Web Service

控制目标

确保经验证使用可信服务层 API（通常使用 JSON 或 XML 或 GraphQL）的应用程序具有:

- 对所有网络服务进行充分的认证、会话管理和授权。
- 对所有从较低信任级别传入较高信任级别的输入参数进行验证。
- 有效地对所有 API 类型进行安全控制，包括云和 Serverless API。

请将本章与其他章节结合起来阅读；我们不再重复说明认证或 API 会话管理问题。

V13.1 通用 Web Service 安全

#	描述	L1	L2	L3	CWE
13.1.1	验证所有应用程序组件使用相同的编码和解析器，以避免利用不同的 URI 或文件解析特性的攻击（这些解析特性可能被用于 SSRF 和 RFI 攻击）。	✓	✓	✓	116
13.1.2	[已删除，与 4.3.1 重复]				
13.1.3	验证 API URL 不公开敏感信息，例如 API 密钥、会话令牌等。	✓	✓	✓	598
13.1.4	验证授权决策是同时在 URI（由程序性或声明性的控制器或路由安全执行）和资源层面（由基于模型的权限执行）做出的。		✓	✓	285
13.1.5	验证包含意外或缺少内容类型的请求是否通过适当的响应头拒绝（HTTP 响应状态 406 Unacceptable 或 415 Unsupported Media Type）。		✓	✓	434

V13.2 RESTful Web Service

JSON 模式验证还处于标准化的草案阶段（见参考文献）。当考虑使用 JSON 模式验证（这是 RESTful web services 的最佳实践）时，可以考虑将这些额外的数据验证策略与 JSON 模式验证结合使用:

- 对 JSON 对象进行解析验证，例如是否有缺失或多余的元素。
- 使用标准的输入验证方法对 JSON 对象的值进行验证，如数据类型、数据格式、长度等。
- 以及正式的 JSON 模式验证。

一旦 JSON 模式验证标准被正式确定，ASVS 将更新这方面的建议。仔细监测正在使用的任何 JSON 模式验证库，因为它们需要定期更新，直到标准正式化并且从参考实现中消除错误。

#	描述	L1	L2	L3	CWE
13.2.1	验证启用的 RESTful HTTP 方法对用户或操作来说是有效的选择，例如防止普通用户在受保护的 API 或资源上使用 DELETE 或 PUT。	✓	✓	✓	650
13.2.2	验证 JSON 模式验证是否到位，并在接受输入之前进行验证。	✓	✓	✓	20

#	描述	L1	L2	L3	CWE
13.2.3	通过使用以下至少一项或多项来验证使用 cookie 的 RESTful Web services 是否受到跨站点请求伪造（CSRF）的保护：双重提交 cookie 模式、CSRF 随机数或 Origin 请求头检查。	✓	✓	✓	352
13.2.4	[已删除，与 11.1.4 重复]				
13.2.5	验证 REST 服务明确检查传入的 Content-Type 是否为预期类型，如 application/xml 或 application/json。		✓	✓	436
13.2.6	验证消息头和有效载荷是可信的，在传输过程中没有被修改。在许多情况下，要求对传输进行强加密（仅 TLS）可能就足够了，因为它同时提供保密性和完整性保护。每条信息的数字签名可以在传输保护的基础上，为高安全性的应用提供额外的保证，但也带来了额外的复杂性和风险，需要权衡利弊。		✓	✓	345

V13.3 SOAP Web Service

#	描述	L1	L2	L3	CWE
13.3.1	验证是否进行了 XSD 模式验证以确保 XML 文档格式正确，然后在对该数据进行任何处理之前验证每个输入字段。	✓	✓	✓	20
13.3.2	验证消息负载是否使用 WS-Security 进行签名，以确保客户端和 service 之间的可靠传输。		✓	✓	345

注意：由于针对 DTD 存在 XXE 攻击问题，因此不应使用 DTD 验证，并且根据 V14 配置 中规定的要求，禁用框架 DTD 评估。

V13.4 GraphQL

#	描述	L1	L2	L3	CWE
13.4.1	验证是否使用查询白名单或深度和数量限制的组合，来防止昂贵的嵌套查询，导致对 GraphQL 或数据层表达式的拒绝服务（DoS）。对于更高级的场景，应该使用查询成本分析。		✓	✓	770
13.4.2	验证 GraphQL 或其他数据层的授权逻辑应在业务逻辑层，而不是 GraphQL 层实现。		✓	✓	285

参考文献

有关更多信息，请参阅：

- OWASP Serverless Top 10
- OWASP Serverless Project
- OWASP Testing Guide 4.0: Configuration and Deployment Management Testing

- OWASP Cross-Site Request Forgery cheat sheet
- OWASP XML External Entity Prevention Cheat Sheet - General Guidance
- JSON Web Tokens (and Signing)
- REST Security Cheat Sheet
- JSON Schema
- XML DTD Entity Attacks
- Orange Tsai - A new era of SSRF Exploiting URL Parser In Trending Programming Languages

V14 配置

控制目标

确保经过验证的应用程序具有：

- 一个安全的、可重复的、可自动化的构建环境。
- 加固第三方库、依赖和配置管理，使应用不包括过时的或不安全的组件。

应用程序开箱即用配置应该是安全的，可以放在互联网上，这意味着安全的开箱配置。

V14.1 构建和部署

构建管道是可重复安全性的基础——每次发现不安全的东西时，都可以在源代码、构建或部署脚本中解决它，并自动进行测试。我们强烈鼓励使用自动化的构建管道来执行安全和依赖检查，这些检查会警告或破坏构建，以防止已知的安全问题被部署到生产环境中。不定期执行的手动步骤会直接导致可避免的安全错误。

随着行业向 DevSecOps 模式的转变，必须确保部署和配置的持续可用性和完整性，以实现“已知良好”的状态。在过去，如果一个系统被入侵，需要几天到几个月的时间来证明没有进一步的入侵发生。今天，随着软件定义的基础设施、零停机时间的快速 A/B 部署和自动化容器构建的出现，自动和持续地构建、加固和部署一个“已知良好”的替代品来替代任何被入侵的系统，是有可能的。

如果传统模式仍然存在，那么就必须采取手动步骤来加固和备份该配置，以便及时用高完整性的、未受损害的系统快速替换受损害的系统。

遵守本节的规定要求一个自动化的构建系统，以及对构建和部署脚本的访问。

#	描述	L1	L2	L3	CWE
14.1.1	验证应用程序的构建和部署过程是以安全和可重复的方式进行的，如 CI / CD 自动化、自动配置管理和自动部署脚本。		✓	✓	
14.1.2	验证编译器标志的配置是否配置为启用所有可用的缓冲区溢出保护和警告，包括堆栈随机化、数据执行保护，并在发现不安全的指针、内存、格式字符串、整数或字符串操作时中断构建。		✓	✓	120
14.1.3	验证服务器配置是否按照应用程序服务器和所使用框架的建议进行了加固。		✓	✓	16
14.1.4	验证应用程序、配置和所有依赖项是否可以使用自动部署脚本重新部署、在合理的时间内根据记录和测试的运行手册构建，或者及时从备份中恢复。		✓	✓	
14.1.5	验证授权管理员可以验证所有安全相关配置的完整性，以发现篡改行为。			✓	

V14.2 依赖

依赖管理，对于任何类型应用程序的安全运行都至关重要。未能及时更新过时的或不安全的依赖，是迄今为止最大和最昂贵攻击的根本原因。

注意：在 L1， 14.2.1 的合规性与客户端和其他库、组件的观察或检测有关，而不是更准确的构建时静态代码分析或依赖分析。这些更准确的技术可根据需要在访谈中发现。

#	描述	L1	L2	L3	CWE
14.2.1	验证所有组件都是最新的，最好是在构建或编译时使用依赖检查工具。 (C2)	✓	✓	✓	1026
14.2.2	验证所有不需要的功能、文档、示例应用程序和配置均已被删除。	✓	✓	✓	1002
14.2.3	应用资产，例如 JavaScript 库、CSS 或网页字体，如果被托管在外部的内容分发网络（CDN）或供应商，则验证使用子资源完整性（SRI）来验证该资产的完整性。	✓	✓	✓	829
14.2.4	验证第三方组件来自预先定义的、可信的和持续维护的资源库。 (C2)		✓	✓	829
14.2.5	验证是否维护了正在使用中的所有第三方库的软件材料清单（SBOM）。 (C2)		✓	✓	
14.2.6	验证通过沙盒或封装第三方库来减少攻击面，只将必需的行为暴露在应用程序中。 (C2)		✓	✓	265

V14.3 意外安全泄露

应加强生产配置以防止常见攻击，例如调试控制台，提高跨站点脚本（XSS）和远程文件包含（RFI）攻击的门槛，并消除琐碎的信息发现“漏洞”，这是许多渗透测试报告中不受欢迎的标志。 其中许多问题很少被评为重大风险，但它们可跟其他漏洞联系在一起。如果这些问题在默认情况下不存在，那就提高了大多数攻击的门槛。

#	描述	L1	L2	L3	CWE
14.3.1	[已删除，与 7.4.1 重复]				
14.3.2	验证 Web 或应用服务器和应用框架的调试模式在生产中是否被禁用，以消除调试功能、开发人员控制台和非预期的安全披露。	✓	✓	✓	497
14.3.3	验证 HTTP 标头或 HTTP 响应的任何部分不暴露系统组件的详细版本信息。	✓	✓	✓	200

V14.4 HTTP 安全标头

#	描述	L1	L2	L3	CWE
14.4.1	验证每个 HTTP 响应都包含一个 Content-Type 头。如果内容类型是 text/* 、 /+xml 和 application/xml ， 还要指定一个安全的字符集（如 UTF-8， ISO-8859-1）。内容必须与提供的 Content-Type 头相匹配。	✓	✓	✓	173
14.4.2	验证所有 API 响应是否包含 Content-Disposition: attachment; filename="api.json" 标头（或内容类型的其他适当文件名）。	✓	✓	✓	116
14.4.3	验证内容安全策略（CSP）响应标头是否到位，有助于减轻对 HTML、DOM、JSON 和 JavaScript 注入漏洞等 XSS 攻击的影响。	✓	✓	✓	1021
14.4.4	验证所有响应是否包含 X-Content-Type-Options: nosniff 标头。	✓	✓	✓	116
14.4.5	验证所有响应和所有子域中是否包含 Strict-Transport-Security 标头，例如 Strict-Transport-Security: max-age=15724800; includeSubdomains。	✓	✓	✓	523
14.4.6	验证是否包含合适的 Referrer-Policy 标头，以避免通过 Referer 标头将 URL 中的敏感信息暴露给不受信任的各方。	✓	✓	✓	116
14.4.7	验证网络应用程序的内容在默认情况下不能被嵌入第三方网站，只有在必要时，才允许使用合适的 Content-Security-Policy: frame-ancestors 和 X-Frame-Options 响应头嵌入确切的资源。	✓	✓	✓	1021

V14.5 HTTP 请求头验证

#	描述	L1	L2	L3	CWE
14.5.1	验证应用服务器只接受应用/API 使用的 HTTP 方法，包括预检请求的 OPTIONS，并对使应用上下文无效的请求进行记录/警告。	✓	✓	✓	749
14.5.2	验证提供的 Origin 标头是否不用于身份验证或访问控制决策，因为 Origin 标头很容易被攻击者更改。	✓	✓	✓	346
14.5.3	验证跨域资源共享（CORS）的 Access-Control-Allow-Origin 标头是否使用受信任域和子域的严格白名单匹配。并且不支持'null'源。	✓	✓	✓	346
14.5.4	验证由受信任的代理或 SSO 设备添加的 HTTP 标头（例如 bearer 令牌）是否已通过应用程序的身份验证。		✓	✓	306

参考文献

有关更多信息，请参阅：

- OWASP Web Security Testing Guide 4.1: Testing for HTTP Verb Tampering

- 将 Content-Disposition 添加到 API 响应，有助于防止许多基于客户端和服务器之间的 MIME 类型误解的攻击，并且“filename”选项特别有助于防止 Reflected File Download attacks.
- Content Security Policy Cheat Sheet
- Exploiting CORS misconfiguration for BitCoins and Bounties
- OWASP Web Security Testing Guide 4.1: Configuration and Deployment Management Testing
- Sandboxing third party components

附录 A：词汇表

- **地址空间布局随机化**（ASLR, Address Space Layout Randomization）- 一种技术，使内存损坏的漏洞利用更加困难。
- **允许列表** - 允许的数据或操作的列表，例如输入验证时允许执行的字符列表。
- **应用程序安全** - 应用程序级安全性侧重于分析构成开放系统互连参考模型（OSI 模型）应用层的组件，而不是侧重于例如底层操作系统或连接网络。
- **应用安全验证** – 根据 OWASP ASVS 对一个应用进行技术评估。
- **应用安全验证报告** - 记录验证者对某一特定应用的总体结果和支持性分析的报告。
- **认证** - 对应用用户所声称身份的验证。
- **自动化验证** – 使用漏洞签名来发现问题的自动化工具（动态分析工具、静态分析工具或两者兼有）。
- **黑盒测试** - 它是一种软件测试方法，在不窥视其内部结构或工作原理的情况下检查应用程序的功能。
- **组件** - 一个独立的代码单元，有相关的磁盘和网络接口，与其他组件进行通信。
- **跨站脚本**（XSS, Cross-Site Scripting）- 通常在网络应用中发现的一种安全漏洞，允许在内容中注入客户端脚本。
- **加密模块** - 实现加密算法或生成加密密钥的硬件、软件或固件。
- **常见弱点列举**（CWE, Common Weakness Enumeration）- 一个社区开发的常见软件安全弱点列表。它是一种通用语言，是软件安全工具的衡量标准，也是弱点识别、缓解和预防工作的基准。
- **设计验证** - 对应用程序的安全架构进行技术评估。
- **动态应用安全测试**（DAST, Dynamic Application Security Testing）- 技术旨在检测应用程序在运行状态下的安全漏洞。
- **动态验证** - 使用自动化工具，使用漏洞签名，在应用程序的执行过程中发现问题。
- **快速在线身份认证**（FIDO, Fast IDentity Online）- 一组认证标准，允许使用各种不同的身份验证方法，包括生物识别、可信平台模块（TPM）、USB 安全令牌等。
- **全球唯一标识符**（GUID, Globally Unique Identifier）– 在软件中作为标识符使用的唯一参考号。
- **超文本传输协议**（HTTPS）– 分布式、协作式、超媒体信息系统的应用协议。它是万维网数据通信的基础。
- **硬编码密钥** – 存储在文件系统中的加密钥匙，无论是代码、注释还是文件。
- **硬件安全模块**（HSM, Hardware Security Module）- 硬件组件，能够以受保护的方式存储加密密钥和其他密码。
- **Hibernate 查询语言**（HQL）- 一种查询语言，在外观上类似于 Hibernate ORM 库使用的 SQL。

- **输入验证** - 对未受信任的用户输入的规范化和验证。
- **恶意代码** - 在应用程序所有者不知情的情况下，在开发过程中将代码引入到应用程序中，从而规避了应用程序的预期安全策略。这与病毒或蠕虫等恶意软件不同！
- **恶意软件** - 在应用程序用户或管理员不知情的情况下，在运行期间被引入到应用程序的可执行代码。
- **OWASP**（OWASP，Open Web Application Security Project）– 开放网络应用安全项目（OWASP）是一个全球自由开放的社区，致力于提高应用软件的安全性。我们的使命是使应用安全“可见”，以便人们和组织能够对应用安全风险做出明智的决定。见： See: https://www.owasp.org/
- **一次性密码**（OTP）- 唯一生成的密码，可在单一场合中使用。
- **对象关系映射**（ORM）- 一种系统，用于允许使用应用兼容的对象模型，在应用中引用和查询基于关系/表的数据库。
- **PBKDF2**（PBKDF2，Password-Based Key Derivation Function 2）- 一种特殊的单向算法，用于从输入文本（如密码）和额外的随机盐值中创建一个强大的加密密钥，因此，如果产生的值被存储（而不是原始密码），则可用于使密码更难被离线破解。
- **个人可识别信息**（PII）- 是指可单独使用或与其他信息一起使用的信息，可用于识别、联系或定位一个人，或用于识别一个人的背景。
- **与位置无关的可执行文件**（PIE）- 放置在主存储器某处的机器代码体，无论其绝对地址如何，都能正确执行。
- **公钥基础设施**（PKI）- 将公钥与实体的各自身份结合起来的一种安排。绑定是通过在证书机构（CA）注册和颁发证书的过程建立的。
- **公共交换电话网**（PSTN）- 传统的电话网络，包括固定电话和移动电话。
- **依赖方**（RP，Relying Party）- 指依赖用户对单独的认证提供者进行认证的应用程序。该应用程序依赖于该身份验证提供者提供的某种令牌或一组签名断言，来相信用户就是他们所说的那个人。
- **静态应用安全测试**（SAST）- 一套分析应用源代码、字节码和二进制文件的技术，用于了解表明存在安全漏洞的编码和设计场景。SAST 解决方案在非运行状态下从“内部”分析一个应用程序。
- **软件开发生命周期**（SDLC）- 软件从最初的需求到部署和维护的一步步发展过程。
- **安全架构** - 应用程序设计的抽象，确定和描述安全控制的位置和方式，同时也确定和描述用户和应用程序数据的位置和敏感性。
- **安全配置** - 应用程序的运行时配置，影响安全控制的使用方式。
- **安全控制** - 执行安全检查（如访问控制检查）或在调用时产生安全效果（如生成审计记录）的功能或组件。
- **服务器端请求伪造**（SSRF）- 滥用服务器上的功能，通过更改在服务器上运行的代码会读取或提交数据的 URL，来读取或更新内部资源的攻击。

- **单点登录验证**（SSO）- 这发生在用户登录到一个应用程序，然后就自动登录到其他应用程序，而无需重新认证。例如，当你已登录到 Google 时，在访问其他谷歌服务，如 YouTube、谷歌文档和 Gmail 时，你将自动登录。
- **SQL 注入**（SQLi）- 一种代码注入技术，用于攻击数据驱动的应用程序，其中恶意的 SQL 语句被插入到一个入口点。
- **SVG** - 可扩展矢量图形
- **基于时间的 OTP** - 一种生成 OTP 的方法，将当前的时间作为生成密码的算法的一部分。
- **威胁建模** - 一种技术，包括开发越来越精细的安全架构，以确定威胁代理、安全域、安全控制以及重要的技术和商业资产。
- **传输层安全**（TLS）- 通过网络连接提供通信安全的加密协议。
- **信任平台模块**（TPM，Trusted Platform Module）- 一种 HSM，通常连接到较大的硬件组件，如主板，并作为该系统的"信任根"。
- **双因素认证**（2FA）- 这为账户登录增加了第二层认证。
- **通用第二因素**（U2F）- 由 FIDO 创建的标准之一，专门用于允许 USB 或 NFC 安全密钥作为第二认证因素使用。
- **URI/URL/URL 分片** – 统一资源标识符是用于标识 web 资源名称或 web 资源的字符串。统一资源定位符通常用作对资源的引用。
- **验证者** – 根据 OWASP ASVS 要求审核应用程序的人员或团队。
- **所见即所得**（WYSIWYG，What You See Is What You Get）- 一种富文本的内容编辑器，显示内容在渲染时的实际效果，而不是显示用于管理渲染的编码。
- **X.509 证书** - X.509 证书是一种数字证书，它使用广泛接受的国际 X.509 公钥基础设施（PKI）标准，来验证公钥是否属于证书中包含的用户、计算机或服务身份。
- **XML 外部实体**（XXE，XML eXternal Entity）- 一种 XML 实体，可以通过声明的系统标识访问本地或远程内容。这可能会导致各种注入攻击。

附录 B：参考文献

以下 OWASP 项目最可能对本标准的用户/采用者有用：

OWASP 核心项目

1. OWASP Top 10 项目：https://owasp.org/www-project-top-ten/
2. OWASP 网络安全测试指南：https://owasp.org/www-project-web-security-testing-guide/
3. OWASP 主动控制：https://owasp.org/www-project-proactive-controls/
4. OWASP 安全知识框架：https://owasp.org/www-project-security-knowledge-framework/
5. OWASP 软件保障成熟度模型（SAMM）：https://owasp.org/www-project-samm/

OWASP Cheat Sheet 系列项目

该项目 有许多与 ASVS 中的不同主题相关的备忘单。

可以在此处找到到 ASVS 的映射： https://cheatsheetseries.owasp.org/cheatsheets/IndexASVS.html

移动安全相关项目

1. OWASP 移动安全项目：https://owasp.org/www-project-mobile-security/
2. OWASP Mobile Top 10 风险：https://owasp.org/www-project-mobile-top-10/
3. OWASP 移动安全测试指南和移动应用安全验证标准：https://owasp.org/www-project-mobile-security-testing-guide/

OWASP 物联网相关项目

1. OWASP 物联网项目：https://owasp.org/www-project-internet-of-things/

OWASP Serverless 项目

1. OWASP Serverless 项目：https://owasp.org/www-project-serverless-top-10/

其他

同样，以下网站最有可能对本标准的用户/采用者有用

1. SecLists Github: https://github.com/danielmiessler/SecLists
2. MITRE 常见弱点列举: https://cwe.mitre.org/
3. PCI 安全标准委员会: https://www.pcisecuritystandards.org
4. PCI 数据安全标准（DSS）v3.2.1 要求和安全评估程序：https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
5. PCI 软件安全框架 - 安全软件要求和评估程序：https://www.pcisecuritystandards.org/documents/PCI-Secure-Software-Standard-v1_0.pdf

6. PCI 安全软件生命周期（Secure SLC）要求和评估程序：
https://www.pcisecuritystandards.org/documents/PCI-Secure-SLC-Standard-v1_0.pdf

附录 C: 物联网验证要求

本章原本是在 main 分支中，但考虑到 OWASP IoT 团队已完成的工作，所以在该主题上维护两个不同的线程没有意义。对于 4.0 版本，我们将其移到附录中，并敦促所有需要此功能的人使用主要的 OWASP IoT 项目

控制目标

嵌入式/IoT 设备应该满足：

- 通过在受信任的环境中实施安全控制，在设备内拥有与服务器中相同级别的安全控制。
- 存储在设备上的敏感数据，应使用硬件支持的存储（如安全元件）以安全的方式完成。
- 从设备传输的所有敏感数据，都应利用传输层安全。

安全验证要求

#	说明	L1	L2	L3	起始时间
C.1	验证应用层调试接口，如 USB、UART 和其他串行变体，是否被禁用或受到复杂密码的保护。	✓	✓	✓	4.0
C.2	验证加密密钥和证书对于每个单独的设备都是唯一的。	✓	✓	✓	4.0
C.3	验证嵌入式/IoT 操作系统（如果适用）是否启用了内存保护控制（如 ASLR 和 DEP）。	✓	✓	✓	4.0
C.4	验证是否禁用了 JTAG 或 SWD 等片上调试接口，或者是否启用并正确配置了可用的保护机制。	✓	✓	✓	4.0
C.5	验证是否已实施并启用受信任的执行（如果在设备 SoC 或 CPU 上可用）。	✓	✓	✓	4.0
C.6	验证敏感数据、私钥和证书是否安全存储在 Secure Element、TPM、TEE (Trusted Execution Environment)中，或使用强加密保护。	✓	✓	✓	4.0
C.7	验证固件应用程序使用传输层安全，保护传输中的数据。	✓	✓	✓	4.0
C.8	验证固件应用程序验证与服务器连接的数字签名。	✓	✓	✓	4.0
C.9	验证无线通信鉴权。	✓	✓	✓	4.0
C.10	验证无线通信是否通过加密通道发送。	✓	✓	✓	4.0
C.11	验证任何被禁止的 C 函数，都被替换成适当的安全函数。	✓	✓	✓	4.0

#	说明	L1	L2	L3	起始时间
C.12	验证每个固件都有一个软件材料清单，其中包括第三方组件、版本和已公布的漏洞。	✓	✓	✓	4.0
C.13	验证所有代码，包括第三方二进制文件、库、框架都经过审查，以防止硬编码凭据（后门）。	✓	✓	✓	4.0
C.14	通过调用 shell 命令封装器、脚本或安全控制，来防止操作系统命令注入，验证应用程序和固件组件不受操作系统命令注入的影响。	✓	✓	✓	4.0
C.15	验证固件应用程序将数字签名固定到可信服务器。		✓	✓	4.0
C.16	验证是否存在防篡改或篡改检测功能。		✓	✓	4.0
C.17	验证是否启用了芯片制造商提供的任何可用的知识产权保护技术。		✓	✓	4.0
C.18	验证安全控制是否到位，以阻止固件逆向工程（例如，删除冗长的调试符号）。		✓	✓	4.0
C.19	验证设备在加载前校验启动镜像的签名。		✓	✓	4.0
C.20	验证固件更新过程不会受到“检查时间与使用时间”攻击（译者注：time-of-check vs time-of-use attacks）。		✓	✓	4.0
C.21	验证设备在安装前使用代码签名并校验固件升级文件。		✓	✓	4.0
C.22	验证设备不能被降级到有效固件的旧版本（防回滚）。		✓	✓	4.0
C.23	验证嵌入式设备使用了密码学安全的伪随机数生成器（例如，使用芯片提供的随机数生成器）。		✓	✓	4.0
C.24	验证固件能够按照预定的时间表，执行自动固件更新。		✓	✓	4.0
C.25	验证设备在检测到篡改或收到无效信息时，能擦除固件和敏感数据。			✓	4.0
C.26	验证只使用了支持禁用调试接口（如 JTAG、SWD）的微控制器。			✓	4.0
C.27	验证只使用了提供实质性保护的微控制器，以防止“去封装”（译者注：de-capping, decapsulation）和侧信道攻击。			✓	4.0
C.28	验证敏感导线不暴露在印刷电路板的外层。			✓	4.0
C.29	验证芯片间的通信是加密的（如主板到子板的通信）。			✓	4.0
C.30	验证设备使用代码签名并在执行前验证代码。			✓	4.0
C.31	验证保存在内存中的敏感信息一旦不再需要，就立即用零值覆盖。			✓	4.0

#	说明	L1	L2	L3	起始时间
C.32	验证固件应用程序利用内核容器在应用程序之间进行隔离。			✓	4.0
C.33	验证安全编译器标志，例如 -fPIE, -fstack-protector-all, -Wl,-z,noexecstack, -Wl,-z,noexecheap 已配置到固件构建中。			✓	4.0
C.34	验证微型控制器是否配置了代码保护（如果适用）。			✓	4.0

参考文献

有关更多信息，请参阅：

- OWASP Internet of Things Top 10
- OWASP Embedded Application Security Project
- OWASP Internet of Things Project
- Trudy TCP Proxy Tool