OWASP
Open Web Application
Security Project

Standard

软件组件验证标准

SCVS

Software Component Verification Standard

Version 1.0

卷首语

软件组件验证标准（Software Component Verification Standard）是一组按照控制域分类的控制项。架构师、开发人员、安全人员、法务人员和合规人员可以使用这些控制项来定义、构建和验证软件供应链的完整性。

版权

版本

V 1.0, 2020 年 6 月 25 日发布

英文项目组

该项目的灵感来自于 OWASP 应用程序安全验证标准（ASVS）。感谢本项目的参与者。

项目组长	• Steve Springett
参与人员及审核人员	• Dave Russo • Garret Fick • JC Herz • John Scott • Mark Symons • Pruthvi Nallapareddy • Bryan Garcia

中文项目组

参与人员及审核人员	• 肖文棣 • 许飞 • 尹杰 • 汪杰 • 王颉

第一部分

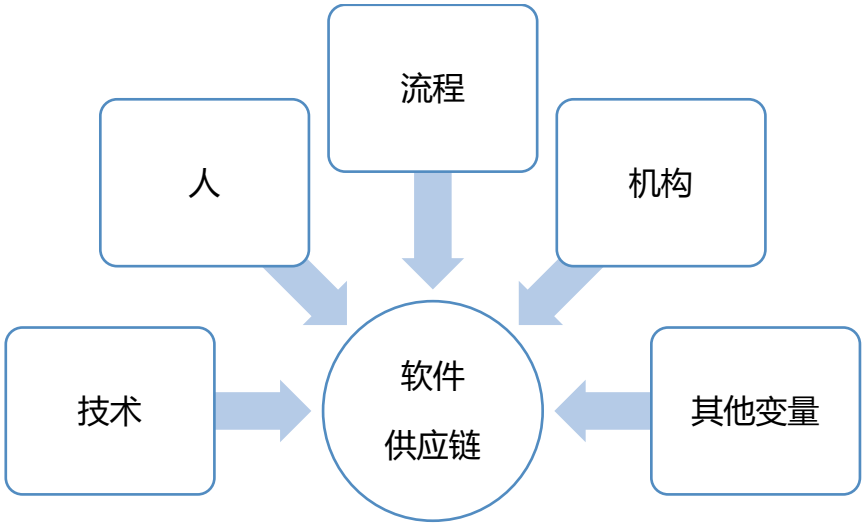
项目介绍

1 前言

欢迎使用软件组件验证标准 (SCVS) 1.0 版。 SCVS 是一项社区驱动的工作，旨在建立一个用于识别活动、控制和最佳实践的框架，这有助于识别和降低软件供应链中的风险。

管理软件供应链中的风险对于减少系统脆弱的攻击面，以及将技术债务衡量为补救措施的障碍非常重要。

衡量和改进软件供应链保障是成功的关键。 具有供应链可见性的组织能够更好地保护其品牌、增加信任、缩短上市时间以及发生供应链事件时的管理成本。



提高供应链保障的标准需要风险管理人员、任务负责人以及法律和采购等业务部门的积极参与，这些部门传统上不参与技术实施。

确定风险接受标准不是企业工具可以解决的问题：由风险管理人员和业务决策者根据系统暴露、监管要求和受约束的财务和人力资源来综合评估。在内部无法实现或使开发或采购陷入停顿的任务构成了它们自身的安全和制度风险。

SCVS 旨在逐步实施，并允许组织随着时间的推移逐步实施不同级别的控制。

2 使用 SCVS

SCVS 有以下目标：

- 开发一套通用的活动、控制和最佳实践，以降低软件供应链中的风险；
- 确定一个基线和路径，以使软件供应链安全保障变得成熟。

2.1 控制域

软件供应链管理有六个控制域包含多个控制项，控制域包括：

2.2 软件组件验证级别

软件组件验证标准定义了三个验证级别。 更高的级别包括额外的控制。

- SCVS 1 级适用于满足基本分析形式的低保障要求；
- SCVS 2 级适用于需要额外分析或尽职调查的中度敏感软件；
- SCVS 3 级适用于因数据敏感性或软件使用而产生的高保障要求。

2.2.1 SCVS 1 级

SCVS 1 级奠定的是基础。 此级别侧重于实施最佳实践，采用现代软件工程实践即可实现 1 级 1，例如：

- 创建具有完整和准确软件清单的软件物料清单；
- 利用持续集成产生可重复的构建；
- 使用公开可用的工具和情报对第三方组件进行分析。

2.2.2 SCVS 2 级

SCVS 2 级扩展了 1 级能力的广度。 2 级适用于具有现有风险管理框架和监管或者类似的合同要求的软件密集型企业和组织。 2 级还扩大了利益相关者的数量，包括那些具有非技术角色的相关人。 采用第 2 级可能需要额外的资源和领域专业知识才能实现。

2.2.3 SCVS 3 级

SCVS 3 级扩展了 2 级能力的深度。 3 级适用于有安全要求的关键基础设施和系统。供应链中的可审计性和端到端透明度需要在这些系统以及生产和维护它们的组织中保持高度安全状态。

2.3 标准使用

使用软件组件验证标准的最佳方法之一，是将其用作评估方法和对逐级递增改进的方法。根据您的用例定制 SCVS 将更加关注对您组织最重要的安全要求。 组织可以选择采用更高级别的某些控制而不实施所有更高级别的控制。

SCVS 的一个用例是评估内部软件开发流程和能力。 另一个是评估供应商，包括开源软件维护者、合同软件开发商和商业软件供应商。

2.3.1 内部能力评估

SCVS 提供了一种结构化的方法来评估和验证内部软件功能和流程，并确定可以改进可验证性的领域。 由于 SCVS 的分层和主题结构，同一组织可以在验证类别内或跨类别处于不同级别。 根据给定系统所需的保障级别，系统批准可能需要不同级别的 SCVS 控制。

SCVS 控制可以是自动化的，并为持续的系统批准提供基础。鉴于软件组成的变化比较快，特别是对于在持续集成管道中开发的功能，保障的连续性需要持续验证而不是一次性验证。

2.3.2 供应商评估

SCVS 提供了一种标准化的方法来评估合同或外包软件供应商提供的供应链透明度，软件客户可以要求供应商提供基于软件开发工作流程中存在的文档和元数据或者类似的提供的软件可交付成果，或要求提供软件物料清单 (SBOM)，以建立供应链可见性并区分供应商。

由于 SCVS 的分层和主题结构，它可以用于分析备选方案或者类似的全部或部分用于评估采购提案。不同级别的 SCVS 控制可用于确定采购资格，或作为对提案进行评分的一个要素。

供应商提供的软件物料清单和其他验证数据使客户能够持续监控供应商软件中存在的风险，以实现可见性，而不管供应商提供的通知或更新如何。该数据的可审计性使合同条款的制定和执行能够在指定的时间段内修复已知漏洞。

2.4 应用 SCVS

软件组件验证标准强调可以通过自动化实施或验证的控制。控制域并非特定于单个开发团队。控制域代表整个组织的利益相关者，包括：软件开发人员、安全和风险管理人员以及采购部门。 所有利益相关者的积极参与对于衡量和改善网络态势是必要的。一旦组织确定了当前的成熟度基线，就可以确定目标和时间表以提高成熟度，并设计可以通过自动化实施或验证控制的方法。

当在达到更高 SCVS 级别控制要求时，可能要更多地依赖于业务流程，而不是可用的技术方法。

3 评估和认证

3.1 OWASP 对 SCVS 认证和信任标志的立场

OWASP 作为供应商中立的非营利组织，不对任何供应商、验证者或软件进行认证。

所有此类保障声明、信任标记或认证都未经 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或声称 SCVS 认证信任标记的信任。

这并不应阻止相关组织提供此类保障服务，只要它们不要求 OWASP 官方认证即可。

3.2 认证软件组件供应链指南

验证软件供应链与 SCVS 合规性的推荐方法是执行"开卷式"审查，这意味着审核员有权访问关键资源，例如：法务、采购、构建工程师、开发人员、仓库、文档和带有源代码的构建环境。

认证组织必须在任何报告中包括：验证范围（特别是如果任何实践超出范围）、验证结果摘要，并明确说明如何解决和改进结果。 如有争议，应有足够的支持性证据证明每一项经过验证的做法确实已得到满足。

只要披露了控制级别，评估人员和软件供应商就可以将评估描述为使用 SCVS 控制执行。

3.2.1 自动验证的作用

只要有可能，应使用自动化工具来验证 SCVS 中详述的控制，以提高效率和准确性。 某些控制无法通过自动化方法进行验证。但是，对于可以的控制，如果可以通过其他方式验证结果，则鼓励自动化方式。

对于更高级别的保障控制，可以使用自动化方法进行独立验证。

第二部分

控制项说明

V1：软件清单要求

V1-1 控制目标

拥有一份软件开发过程中所有使用组件的准确清单，是进行进一步分析的基本要求。 以下控制包含单个应用程序清单、组织清单，和在采购新软件或系统时让软件变得透明的方法。

组件标识因组件所属的生态系统而异。 因此，为了所有清单，可以使用标识符（例如包 URL）来标准化和规范托管依赖项的命名约定。

组织范围内的所有第一方（内部）和第三方（包括开源）组件清单，可以提高组织内的透明度，促进软件标准化和重用，并允许快速分析影响。

V1-2 验证要求

序号	描述	1 级	2 级	3 级
1.1	所有直接依赖和间接依赖组件及其版本在构建完成时都是已知的。	✓	✓	✓
1.2	使用了包管理器以管理所有第三方二进制组件。	✓	✓	✓
1.3	以机器可读格式提供所有第三方组件的准确软件清单。	✓	✓	✓
1.4	为公开或商业可用的应用程序生成软件物料清单。	✓	✓	✓
1.5	对于新的采购，明确要求了软件物料清单。		✓	✓
1.6	软件物料清单持续维护并适用于所有系统。			✓
1.7	组件以一致的机器可读格式进行唯一标识。	✓	✓	✓

1.8	组件类型在整个软件清单中都是已知的。			✓
1.9	组件功能在整个软件清单中都是已知的。			✓
1.10	所有组件都知道其来源或出处。			✓

V2：软件物料清单 (SBOM) 要求

V2-1 控制目标

在构建管道中自动创建准确的软件物料清单 (SBOM) 是成熟开发流程的一项指标。 SBOM 应该是机器可读的格式。 每种格式都有不同的功能和它们擅长的用例。SBOM 采用的一部分是确定最适合特定目的的用例和功能。虽然整个组织的 SBOM 格式标准化可能是可取的，但可能需要采用多个标准化的格式来满足功能、合同、合规性或监管要求。

V2-2 验证要求

序号	描述	1 级	2 级	3 级
2.1	使用结构化、机器可读的软件物料清单 (SBOM) 格式。	✓	✓	✓
2.2	SBOM 创建是自动化的和可重复的。		✓	✓
2.3	每个 SBOM 都有一个唯一的标识符。	✓	✓	✓
2.4	SBOM 已由出版商、供应商或认证机构签署。		✓	✓
2.5	存在 SBOM 签名验证。		✓	✓
2.6	执行 SBOM 签名验证。			✓
2.7	SBOM 带有时间戳。	✓	✓	✓
2.8	对 SBOM 进行风险分析。	✓	✓	✓
2.9	SBOM 包含了 SBOM 所描述所有组件的完整且准确的清单。	✓	✓	✓

2.10	SBOM 包含其所描述资产或应用程序的所有测试组件的准确清单。		✓	✓
2.11	SBOM 包含了 SBOM 所描述资产或软件的元数据。		✓	✓
2.12	组件标识符来自其原生生态系统（如果适用）。	✓	✓	✓
2.13	组件的来源以一种一致的、机器可读的格式（例如：PURL）标识。			✓
2.14	SBOM 中定义的组件具有准确的许可证信息。	✓	✓	✓
2.15	SBOM 中定义的组件具有有效的 SPDX 许可证 ID 或表达式（如果适用）。		✓	✓
2.16	SBOM 中定义的组件具有有效的版权声明。			✓
2.17	SBOM 中定义的、已从原始版本修改的组件，具有详细的出处和谱系信息。			✓
2.18	SBOM 中定义的组件具有一个或多个文件哈希（SHA-256、SHA-512 等）。			✓

V3：构建环境要求

V3-1 控制目标

软件构建管道可能由源代码仓库、包仓库、持续集成和交付过程、测试程序，以及支持这些功能的网络基础设施和服务共同组成。管道中的每个系统都可能提供一个入口点，其中的缺陷、故障和错误配置可能会危及软件供应链。强化所涉及的系统并实施最佳实践，可降低系统失陷的可能性。

V3-2 验证要求

序号	描述	1 级	2 级	3 级
3.1	应用程序使用可重复的构建。	✓	✓	✓
3.2	存在有关如何构建应用程序以及重复构建说明的文档。	✓	✓	✓
3.3	应用程序使用持续集成构建管道。	✓	✓	✓
3.4	应用程序构建管道禁止在执行构建的作业之外更改构建。		✓	✓
3.5	应用程序构建管道禁止更改包管理设置。		✓	✓
3.6	应用程序构建管道禁止在作业构建脚本的上下文之外执行任意代码。		✓	✓
3.7	应用程序构建管道只能对版本控制系统中维护的源代码执行构建。	✓	✓	✓
3.8	应用程序构建管道禁止在构建期间更改 DNS 和网络设置。			✓

3.9	应用程序构建管道禁止更改信任证书存储。			✓
3.10	应用程序构建管道强制执行身份验证，默认为拒绝。		✓	✓
3.11	应用程序构建管道强制执行授权，默认为拒绝。		✓	✓
3.12	应用程序构建管道需要特别关注修改系统设置的检查点。			✓
3.13	应用程序构建管道维护所有系统更改的可验证审计日志。			✓
3.14	应用程序构建管道维护所有构建作业更改的可验证审计日志。			✓
3.15	应用程序构建管道需要维护整个堆栈更新、修补和重新认证以供使用的节奏。		✓	✓
3.16	对编译器、版本控制客户端、开发实用程序和软件开发工具包是否存在篡改、木马或恶意代码进行分析和监控。			✓
3.17	对源代码或二进制文件的所有构建操作，都是已知且明确定义。	✓	✓	✓
3.18	每次构建都会记录所有第一方和第三方组件的校验和。	✓	✓	✓
3.19	每当打包或分发这些组件时，所有组件的校验和都可以带外访问和交付。		✓	✓
3.20	未使用的直接依赖和间接依赖组件得到识别。			✓
3.21	未使用的直接依赖和间接依赖组件已从应用程序中删除。			✓

V4：包管理要求

V4-1 控制目标

旨在重用的开源组件通常发布到特定生态系统的包仓库。许多构建系统都存在集中仓库，包括：Maven、.NET、NPM 和 Python。组织内部的仓库可以额外提供第一方组件（内部组件）的重用以及对受信任第三方组件的访问。

包管理器通常在构建过程中被调用，并负责解析组件版本和从仓库中检索组件。

虽然使用包管理器和集中式仓库具有巨大的业务、技术和安全优势，但它们通常是攻击者的目标。实施最佳实践可以显着降低软件供应链中受损的风险。

V4-2 验证要求

序号	描述	1 级	2 级	3 级
4.1	从包仓库中检索二进制组件。	✓	✓	✓
4.2	包仓库内容与开源组件的权威来源一致。	✓	✓	✓
4.3	包仓库需要强身份验证。		✓	✓
4.4	包仓库支持多因素认证组件发布。		✓	✓
4.5	包仓库组件已发布，并具有多因素身份验证。			✓
4.6	包仓库支持安全事件报告。		✓	✓
4.7	包仓库自动生成安全事件报告。			✓
4.8	包仓库通知发布者安全问题。		✓	✓

4.9	包仓库通知用户安全问题。			✓
4.10	包仓库提供了一种将组件版本与版本控制中的特定源代码相关联的可验证方式。		✓	✓
4.11	更新组件时，包仓库可进行审计。	✓	✓	✓
4.12	包仓库需要代码签名才能将包发布到生产仓库。		✓	✓
4.13	包管理器在从远程仓库检索包时验证包的完整性。	✓	✓	✓
4.14	包管理器在从文件系统检索包时验证包的完整性。	✓	✓	✓
4.15	包仓库强制对所有交互使用 TLS。	✓	✓	✓
4.16	包管理器验证 TLS 证书链到仓库，并在验证失败时以安全的方式显示失败。	✓	✓	✓
4.17	包仓库需要或者类似的在发布组件之前执行静态代码分析，并使结果可供其他人使用。			✓
4.18	包管理器不执行组件代码。	✓	✓	✓
4.19	包管理器以机器可读的形式记录包的安装过程。	✓	✓	✓

V5：组件分析要求

V5-1 控制目标

组件分析是识别使用第三方和开源软件组件的潜在风险领域的过程。每个组件，无论是直接的还是传递的，都是分析的对象。通过使用第三方软件继承的风险，可能直接影响依赖它们的应用程序或系统。

1) 已知漏洞

漏洞情报有多种公共和商业来源。当漏洞被发布到国家漏洞数据库 (NVD) 等服务时，或者以其他方式记录在公共缺陷跟踪器、提交日志或其他公共资源中时，漏洞就会为人所知。

2) 组件版本的流通历史

如果指定了组件版本，则可以确定组件是否已过期或生命周期结束。过时和报废组件更容易受到攻击，并且不太可能作为优秀实体得到支持。由于互操作性问题，过时的组件可能会减慢系统修复速度。使用最新的组件可以减少暴露时间，并且可能包括对未公开漏洞的补救。

3) 组件类型

框架和库具有独特的升级挑战和相关风险。抽象、耦合和架构设计模式可能会影响使用给定组件类型的风险。库、框架、应用程序、容器和操作系统是常见的组件类型。

4) 组件功能

识别和分析每个组件的用途可能会揭示具有重复或类似功能的组件的存在。通过最小化每个功能的组件数量并为每个功能选择最高质量的组件，可以降低潜在风险。

5) 组件数量

使用开源的运营和维护成本可能会随着每个新组件的采用而增加。可以预计，随着时间的推移，维护不断增长组件集的能力会降低。对于有时间限制的团队来说尤其如此。

6) 许可证

第三方和开源软件通常根据一个或多个许可证发布。选择的许可证可能允许也可能不允许某些类型的使用，包含分发要求或限制，或者如果组件被修改，则需要特定的操作。使用具有与组织目标或能力相冲突的许可证的组件可能会给业务带来风险。

V5-2 验证要求

序号	描述	1 级	2 级	3 级
5.1	使用静态分析工具分析组件。	✓	✓	✓
5.2	在组件使用前，使用静态分析工具分析组件。		✓	✓
5.3	在每次升级组件时，执行静态分析。		✓	✓
5.4	使用自动化流程，以识别使用到的第三方及开源组件的所有公开漏洞。	✓	✓	✓
5.5	使用自动化流程，以识别已确认数据流的可利用性。			✓
5.6	使用自动化流程，以识别非特定组件的版本。	✓	✓	✓
5.7	使用自动化流程，以识别过时的组件。	✓	✓	✓
5.8	使用自动化流程，以识别寿命终止或停止支持的组件。			✓
5.9	使用自动化流程，以识别组件类型。		✓	✓
5.10	使用自动化流程，以识别组件功能。			✓
5.11	使用自动化流程，以识别组件数量。	✓	✓	✓
5.12	使用自动化流程，以识别组件许可。	✓	✓	✓

V6：谱系和来源要求

V6-1 控制目标

确定软件原始来源和监管链，以便在原始来源或监管链受到损害时管理系统风险。对于内部包管理器和仓库，维护导入组件的谱系和来源数据非常重要。

V6-2 验证要求

序号	描述	1 级	2 级	3 级
6.1	源代码和二进制组件的来源是可验证的。		✓	✓
6.2	对于源代码和二进制组件，监管链是否可审计。			✓
6.3	已修改组件的出处是已知的并被记录。	✓	✓	✓
6.4	组件修改的谱系是已记录并可验证的。		✓	✓
6.5	已修改组件是唯一标识的，并且与原始组件不同。		✓	✓
6.6	已修改组件的分析精度与未修改的组件维持在同一水平。	✓	✓	✓
6.7	已修改组件的特有风险可被分析，并与修改的变量特定关联。	✓	✓	✓

第三部分

参考资料

附录 A：开源政策指导

以下几点应被视为组织在使用它们后所产生成功和最佳实践的建议。 它们不是 SCVS 的一部分。

1) 所有使用开源软件的组织都应该有一个开源政策。

2) 开源政策得到跨职能利益相关者的支持和执行。

3) 开源政策应解决：

- 组件的年龄基于其发布日期；
- 可以接受多少主要或次要的旧版本修订；
- 通过自动化保持组件持续更新的指南；
- 具有已知漏洞组件的排除标准；
- 更新有风险组件的平均修复时间标准；
- 对使用已终止或支持终止组件的限制；
- 供应商选择或排除标准；
- 基于使用的可接受许可证列表；
- 禁用组件列表；
- 向生产组件的社区提供修改的机制和权限。

附录 B：词汇表

序号	英文词汇表	中文对照
1	Chain of custody Auditable documentation of point of origin as well as the method of transfer from point of origin to point of destination and the identity of the transfer agent.	**监管链** 可审计的原产地文件以及从原产地到目的地的转运方法以及转运代理的身份。
2	Component function The purpose for which a software component exists. Examples of component functions include parsers, database persistence, and authentication providers.	**组件功能** 软件组件存在的目的。组件功能的示例包括解析器、数据库持久性和身份验证提供程序。
3	Component type The general classification of a software components architecture. Examples of component types include libraries, frameworks, applications, containers, and operating systems.	**组件类型** 软件组件架构的一般分类。 组件类型的示例包括库、框架、应用程序、容器和操作系统。
4	CycloneDX A software bill of materials specification designed to be lightweight and security-focused.	CycloneDX 一种软件物料清单规范，旨在实现轻量级和安全性。
5	Direct dependency - A software component that is referenced by a program itself.	**直接依赖** 程序本身引用的软件组件。
6	Package manager - A distribution mechanism that makes software artifacts discoverable by requesters.	**包管理器** 一种分发机制，使请求者可以发现软件工件。
7	Package URL (PURL)	包 URL (PURL)

	An ecosystem-agnostic specification which standardizes the syntax and location information of software components.	一种与生态系统无关的规范，它标准化了软件组件的语法和位置信息。
8	Pedigree Data which describes the lineage and/or process for which software has been created or altered.	**谱系** 描述创建或更改软件的谱系或者类似过程的数据。
9	Point of origin The supplier and associated metadata from which a software component has been procured, transmitted, or received. Package repositories, release distribution platforms, and version control history are examples of various points of origin.	**来源点** 采购、传输或接收软件组件的供应商和相关元数据。 包仓库、发布分发平台和版本控制历史是各种来源的示例。
10	Procurement The process of agreeing to terms and acquiring software or services for later use.	**采购** 同意条款和获取软件或服务以供以后使用的过程。
11	Provenance The chain of custody and origin of a software component. Provenance incorporates the point of origin through distribution as well as derivatives in the case of software that has been modified.	**出处** 软件组件的监管链和来源。 出处包含通过分发的原点以及已修改软件的衍生品。
12	Software bill of materials (SBOM) A complete, formally structured, and machine-readable inventory of all software components and associated metadata, used by or delivered with a given piece of software.	**软件物料清单** (SBOM) 一个完整的、正式结构化的、机器可读的所有软件组件和相关元数据的清单，供给定软件使用或与给定软件一起交付。
13	Software Identification (SWID) An ISO standard that formalizes how software is tagged.	**软件标识** (SWID) 一种 ISO 标准，它正式确定了软件的标记方式。

14	Software Package Data Exchange (SPDX) A Linux Foundation project which produces a software bill of materials specification and a standardized list of open source licenses.	软件包数据交换 (SPDX) 一个 Linux 基金会项目，它产生软件物料清单规范和开源许可证的标准化列表。
15	Third-party component Any software component not directly created including open source, "source available", and commercial or proprietary software.	**第三方组件** 一任何非直接创建的软件组件，包括开源、“可用源”以及商业或专有软件。
16	Transitive dependency A software component that is indirectly used by a program by means of being a dependency of a dependency.	**传递依赖（间接依赖）** 通过作为依赖的依赖而被程序间接使用的软件组件。

附录 C：参考文献

1) OWASP 项目

- OWASP Packman
- OWASP Software Assurance Maturity Model (SAMM)

2) 社区项目

- Open Source Security Coalition - Threats, Risks, and Mitigations in the Open Source Ecosystem

3) 其他

- InnerSource
- Cybersecurity Maturity Model Certification (CMMC)
- NIST 800-53 Security and Privacy Controls for Federal Information Systems and Organizations
- NIST 800-161 Supply Chain Risk Management Practices for Federal Information Systems and Organizations
- NIST 800-171 Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations
- NTIA Documents on Software Bill of Materials
- Model Procurement Contract Language Addressing Cybersecurity Supply Chain Risk
- Guide on Cybersecurity Procurement Language in Task Order Requests for Proposals for Federal Facilities
- Energy Sector Control Systems Working Group (ESCSWG)

4) SBOM 格式

- CycloneDX
- SPDX
- SPDX XML
- ISO/IEC 19770-2:2015 (SWID)