Toward a Reference Architecture for Software Supply Chain Metadata Management

Nguyen Khoi Tran
CREST
The University of Adelaide
Adelaide, Australia
nguyen.tran@adelaide.edu.au

Samodha Pallewatta
CREST
The University of Adelaide
Adelaide, Australia
samodha.pallewatta@adelaide.edu.au

M. Ali Babar
CREST, The University of Adelaide
Cyber Security Cooperative Research Centre
Adelaide, Australia
ali.babar@adelaide.edu.au

ABSTRACT

An Software Supply Chain (SSC) attack combines an upstream attack, where malicious codes are injected into a software artefact via a compromised life cycle activity, and a downstream attack on the consumers who use the compromised artefact. Organisations need thorough and trustworthy visibility over the entire SSC of their software inventory to detect risks early and rapidly identify compromised assets in the event of an SSC attack. One way to achieve such visibility is through SSC metadata, machine-readable and authenticated documents describing an artefact's lifecycle, such as how it was constructed and the utilised "ingredients". Adopting SSC metadata requires organisations to procure or develop a Software Supply Chain Metadata Management system (SCM2), a suite of software tools for performing life cycle activities of SSC metadata documents such as creation, signing, distribution, and consumption. Selecting or developing an SCM2 is challenging due to the lack of a comprehensive domain model and architectural blueprint to aid practitioners in navigating the vast design space of SSC metadata terminologies, frameworks, and solutions. This paper addresses the above-mentioned challenge with a Systematisation of Knowledge about SSC metadata and SCM2, presented as a Reference Architecture (RA). The RA comprises a domain model and an architectural blueprint for SCM2 systems, constructed from the concepts and building blocks scattered across existing SSC security frameworks and standards. Our evaluation shows that the RA framework is effective for analysing existing SCM2 solutions and guiding the engineering of new SCM2.

CCS CONCEPTS

• **Software and its engineering** → *Software design engineering*; • **Security and privacy** → **Software and application security**.

KEYWORDS

Software Supply Chain, SSC Metadata, SBOM, Software Provenance, Reference Architecture, SoK

ACM Reference Format:
Nguyen Khoi Tran, Samodha Pallewatta, and M. Ali Babar. 2024. Toward a Reference Architecture for Software Supply Chain Metadata Management . In *Proceedings of Preprint*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recent years have witnessed a meteoric rise in Software Supply Chain (SSC) attacks. For instance, the 8th Annual State of the Software Supply Chain Report by Sonatype in 2023 [34] reported a 742% annual increase in SSC attacks in the past three years. An SSC attack combines an upstream attack, where malicious codes are injected into a software artefact via a compromised life cycle activity, and a downstream attack on the consumers who use the artefact [25]. Lack of oversight in an artefact's lifecycle makes upstream attacks possible, while a lack of visibility over the distribution and consumption of artefacts opens the door for downstream attacks. Therefore, securing an inventory of software assets requires a thorough and *trustworthy* visibility over its entire SSC so that upstream risks (e.g., changing of a project's maintainer) can be detected early and vulnerable assets can be identified quickly in the event of an upstream attack. In this context, the SSC of a software artefact contains not only the dependencies between its producers and consumers [6, 10, 23] but also the relationship between various actors involved in its life cycle (e.g., develop, build, test, distribute) [7, 11, 25, 33].

The visibility into the SSC of a software artefact can be provided by SSC metadata - the information about an artefact's life cycle, such as how it was constructed and the "ingredients" utilised in its construction [37]. This information can be *implied* by SSC tools, such as commit history and contributor list of a GitHub repository, or *manually written* by artefact producers, such as the `package.json` file of a NodeJS package. However, this information generally does not follow a predefined data model or is authenticated, making it challenging for other software tools to interpret and trust it for SSC visibility and security purposes. Therefore, *a new class of formal SSC metadata that is machine-readable and authenticated has emerged.* This paper focuses on this SSC metadata class. For brevity, we use the term SSC metadata to address this SSC metadata class rather than the implicit and manually written metadata types.

An example of SSC metadata is *Software Bill of Material (SBOM)*, a formal and machine-readable list of ingredients that make up software components [15]. Other examples include *provenance* statements that describe how and by whom a software artefact was produced [33] and *software attestation*, which are digitally signed statements about a software artefact [11]. These SSC metadata documents are constructed based on a predefined standard such as Software Package Data Exchange (SPDX),[1] CycloneDX[2], and in-toto attestation framework[3]. The use of digital signature and, optionally, notarisation mechanisms (e.g., Supply Chain Integrity, Transparency and Trust (SCITT) [8]) helps ensuring the authenticity and verifiability of these metadata documents. The mentioned advantages could enable SSC metadata to form a "foundation data layer" on which further security tools, practices, and assurances can be built and applied, thus converting metadata into intelligence and actions [25, 37]. Another motivation for adopting SSC metadata is compliance to regulations such as the US President's Executive Order (EO) 14028 on Improving the Nation's Cybersecurity issued on May 12, 2021 [27], which mandates software suppliers to provide SBOM of their products.

SSC metadata requires software tools for generating, signing, distributing, and consuming them. We denote this software suite as a **S**oftware Supply **C**hain **M**etadata **M**anagement system **(SCM2)**. Adopting SSC metadata requires practitioners to assemble or develop an SCM2. This activity is challenging for two reasons. The first challenge lies in the *diversity and overlap of terminology and framework related to SSC metadata*. As we described above, SSC metadata standards and frameworks vary not only in syntax but also in semantics and content, which are tied to how they define an SSC and the aspects of an SSC they aim to describe. Navigating this design space requires clear communication with stakeholders to identify and align information needs with the SSC metadata type. Such communication relies on a shared understanding of SSC metadata, which can be facilitated by *a domain model*. Unfortunately, such a comprehensive domain model for SSC metadata has not been established. In our experience working on industry projects related to SSC metadata, we found that most discussions required establishing a conceptual foundation. These first-hand experiences motivated our research presented in this paper.

The second challenge lies in the diversity of software solutions making up the design space of an SCM2. For instance, various proprietary application security platforms can now generate SSC metadata documents for an existing software portfolio, usually in conjunction with binary analysis and software composition analysis tools. Many open-source tools (e.g., Syft[4]) cover specific aspects of SSC metadata's life cycle, such as generating and signing SSC metadata documents. Both proprietary platforms and open-source tools differ regarding their supported SSC metadata types and standards. They also vary in capability and correctness. The accuracy and completeness of the generated SBOMs caused by tooling competence have been a shared concern among practitioners, according to a recent empirical study about SBOM adoption [37]. When navigating this vast design space to construct an SCM2, practitioners would benefit from an *architectural blueprint* that specifies the system's context, scope, and functional decomposition. Such an architectural blueprint would provide a framework for identifying, assessing, and selecting platforms and tools to construct an SCM2. Even though SSC metadata appears in most existing SSC security frameworks (e.g., SLSA [33], SCVS [26], SSF [7]), an architectural blueprint of an SCM2 that covers the entire life cycle of all classes of both SBOM, provenance, and attestations has not been established.

The described challenges highlighted the need for a Systematisation of Knowledge (SoK) of SSC metadata. This paper presents such an SoK in the form of a Reference Architecture (RA) for SCM2. An RA can be considered a template for software systems, providing a common vocabulary, taxonomy, architectural vision, and building blocks to guide the development of future systems [13]. RAs have traditionally been constructed to provide SoK and guidance for Service-oriented computing [3], Cloud computing [12], Big data [28], and Internet of Things [36], where a diverse and decentralised set of stakeholders are required to interoperate and cooperate. Today's SSC metadata management exhibits similar multiplicity due to the diversity of frameworks, standards, tool sets, and stakeholders, making RA a suitable solution.

Our proposed RA for SCM2 comprises *a domain model of SCC metadata* and an *architectural blueprint for SCM2 systems*. The domain model (Section 3.1) presents the key concepts related to SSC metadata and the relationships between them. The domain model also describes the life cycle of SSC metadata and the related actors. The architectural blueprint describes the scope and positioning of SCM2 within an SSC (Section 3.2) and the function units making up an SCM2 instance (Section 3.3). We also discuss some possible variants of the blueprint that practitioners can use to adapt the blueprint to their concrete use cases and requirements. The RA was constructed from the concepts and building blocks scattered across existing SSC security frameworks and standards, according to the empirically grounded design methodology proposed by Galster and Avgeriou [9]. We demonstrated the correctness and utility of the RA by mapping the SCM2 elements of five prominent SSC security tools onto the concepts and architecture of the RA. Based on the architectural mapping, we discussed the state of practice of SCM2 and presented a reference instantiation of SCM2 from the existing open-source tools.

2 METHODOLOGY

We adopted the empirically grounded design methodology proposed by Galster and Avgeriou [9] to construct an RA for SCM2. This methodology was chosen because it provides a systematic approach to constructing RAs from practice-proven concepts and building blocks and evaluating the resultant RAs in terms of validity and applicability. The methodology comprises six steps (Figure 1).

Step 1 - Determine RA type: This is a crucial decision that determines the design strategy and data collection method for constructing an RA. Based on Angelov's classification system for RAs [2], we classified our RA as a *classical*, *facilitation* RA intended for use by *multiple organizations* in an *industry cross-cutting context*. This means that our RA would be based on experiences from preexisting systems, with the primary goal of facilitating the design and development of new systems. It would be used by multiple organizations, both within and outside the software industry, that require access to SSC metadata.

Step 2 - Determine Design Strategy: We adopted a hybrid design strategy for our SCM2 RA. This approach utilises the existing

[1]https://spdx.dev/
[2]https://cyclonedx.org/
[3]https://in-toto.io/
[4]https://github.com/anchore/syft

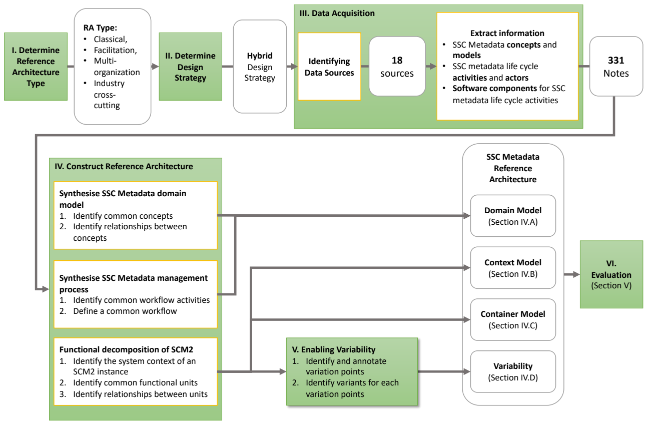
I. Determine Reference Architecture Type
RA Type:
Classical,
Facilitation,
Multi-organization
Industry cross-cutting
II. Determine Design Strategy
Hybrid Design Strategy
III. Data Acquisition
Identifying Data Sources
18 sources
Extract information
SSC Metadata concepts and models
SSC metadata life cycle activities and actors
Software components for SSC metadata life cycle activities
331 Notes
IV. Construct Reference Architecture
Synthesise SSC Metadata domain model
1. Identify common concepts
2. Identify relationships between concepts
Synthesise SSC Metadata management process
1. Identify common workflow activities
2. Define a common workflow
Functional decomposition of SCM2
1. Identify the system context of an SCM2 instance
2. Identify common functional units
3. Identify relationships between units
V. Enabling Variability
1. Identify and annotate variation points
2. Identify variants for each variation points
SSC Metadata Reference Architecture
Domain Model (Section IV.A)
Context Model (Section IV.B)
Container Model (Section IV.C)
Variability (Section IV.D)
VI. Evaluation (Section V)

Figure 1: Methodology for constructing the reference architecture for SSC Metadata life cycle management

architecture constructs from industry-driven SSC security frameworks and peer-reviewed academic articles as inputs for the RA construction.

Step 3 - Data Acquisition: This step has two activities. Firstly, we *identified the data sources for constructing the RA*, which include peer-reviewed articles and industry-driven SSC security frameworks and architectures related to SCM2. The frameworks and architectures were identified based on the authors' experience in industry projects on SSC security. We also expanded our selection by using the snowballing sampling technique. If a framework had multiple revisions, we selected the latest version. Through this process, we identified 12 industry-driven reports and standards and four peer-reviewed articles which, together, specify **eleven SSC security frameworks** that served as inputs for our RA. Table 1 presents an overview of the chosen frameworks, focusing on their coverage of SSC metadata and its life cycle. We classify the coverage into four classes: "0" denotes that a framework does not offer any details about an SSC metadata life cycle stage, "1" indicates that it provides an overall description, "2" indicates that it describes workflow activities and actors, and "3" denotes that it provides functional decomposition and describes involved software components. For brevity, we make the details of these frameworks available in the online appendix of paper [4].

Table 1: SCM2 Support of existing SSC Security Frameworks

Framework/ Architecture	Scope	Supported SSC Metadata Types	Supported SSC Metadata Standards	Metadata Life Cycle Coverage		
				Generation	Sharing	Consumption
SEI SCRM [1, 29, 30]	SSC Concepts	N/A	N/A	N/A	N/A	N/A
In-toto Attestation Framework [11]	SSC Metadata	Attestation Provenance (Partially)	In-toto Attestation Specification	3	1	3
Software Delivery Governance [31]	SSC Metadata	Provenance	N/A	1	3	1
SCVS [26]	SSC Metadata	SBOM	SPDX	1	0	1
SBOM Generation Playbook [15, 16, 21, 22]	SSC Metadata	SBOM	Any	2	1	1
SBOM Sharing Playbook [19, 35]	SSC Metadata	SBOM	Any	0	2	1
SBOM Consumption Playbook [20]	SSC Metadata	SBOM	Any	0	1	2
SBOM Tool Classification [18]	SSC Metadata	SBOM	Any	1	0	1
SSF Architecture [7]	SSC Metadata	Attestation	Any	3	3	0
SCITT [8]	SSC Metadata	Attestation	SCITT Transparent Statement	1	3	2
SLSA [33]	SSC Metadata	Provenance Attestation	SLSA Provenance Model, SLSA Software Attestation Model	1	1	1

The second activity was *extracting information*. The authors independently analysed the identified reports, standards, and peer-reviewed articles and extracted relevant information regarding SSC metadata, SSC Metadata life cycle activities and actors, and software components introduced to carry out SSC metadata life cycle activities. The extracted information are captured in **331 digital note cards** and stored in a version-controlled repository. The authors collaboratively organised the note cards into a knowledge graph incrementally based on weekly discussion sessions. We leveraged Obsidian[5], a personal knowledge base system, as the tool to capture digital notes and construct the knowledge graph. The resulting knowledge graph can be found in the online appendix of paper [4].

Step 4 - Reference Architecture Construction: Our RA for SCM2 includes a domain model of SSC metadata and an architectural blueprint for SCM2 instances. The domain model was derived from the knowledge graph constructed in the previous step, focusing on static relationships between concepts related to SSC metadata, SSC artefacts, SSC activities, and SSC actors. To complement the static view of the domain model, we also developed a life cycle model of SSC metadata. This model was based on the union of SSC metadata life cycle activities (e.g., generation, signing, distribution) introduced by different frameworks and architectures.

The architectural blueprint for SCM2 was also constructed from the *union* of architectural building blocks related to SSC metadata life cycle activities. We chose union rather than intersection because no existing frameworks covered SSC metadata types and life cycle activities completely. By finding the union, we can achieve more extensive coverage.

We utilised the *C4 architectural view model* [5] to document the constructed RA. The C4 model visualises a software architecture as a set of diagrams, showing a hierarchy of abstraction from high-level system context to the concrete classes and objects making up system components. Based on the information availability, we constructed and documented the RA at two abstraction levels: system context and containers (functional units).

Step 5 - Enabling Variability: Creating specific software architectures from an RA may introduce variations, such as modifying or removing functional units. An RA can support variability by specifying the variation points (architectural elements where changes might occur) and variants (the possible options for those variation points). We identified the variation points of the RA for SCM2 based on the SSC metadata life cycle activities and functional units where the analysed frameworks and architectures diverged. The various approaches and architectural constructs utilised by these frameworks and architectures represent the variants of the identified variation points. We used textual annotation to describe the variability of the proposed RA.

Step 6 - Evaluation: Following the design methodology, we evaluated the correctness and usefulness of the constructed RA by mapping prominent SSC security solutions to the SSC metadata concepts and architectural blueprint defined by the constructed RA. We will provide more details about the evaluation methodology in Section 4.

[5]https://obsidian.md

3 SCM2 REFERENCE ARCHITECTURE

3.1 Domain Model

3.1.1 SSC Metadata Concepts. We organised the concepts relevant to SSC metadata and the relationship between them into four clusters (Figure 2). Because SSC metadata provides information about artefacts generated and consumed by SSC activities, we include concepts about SSC activities and artefacts as the first two clusters.

SSC Activities: An SSC can be considered a series of coordinated *SSC activities* performed by a network of actors in order to create and distribute software products to end-users [7, 11, 25, 31, 33]. On the producer side, SSC activities include *development*, *build*, *test*, and *release*. The activities on the consumer side after retrieving a software product include *verification* (e.g., checking digests and verifying the product against its metadata), *installation*, and *maintaining* the software product throughout its life span.

SSC artefacts: SSC activities consumes and produces *software artefacts*. The existing SSC security frameworks defined artefacts in various ways, from any "immutable blob of data" [33] to "item that is moving along the supply chain" [8] to the principal output of a secure software factory that would be consumed by downstream users [7]. These diverse definitions reflect the differences in scope and focus of the SSC security frameworks. In our domain model, we classify SSC artefacts into input and output artefacts. The input artefacts include *software materials* [11] such as *source code*, build *pipeline definition*, external *build parameters* [33] and *dependency*, which are other artefacts that need to be collected before building a software [7]. The input artefacts also include cryptographic material such as certificates, tokens, and signing keys [7]. The output artefact, also known as *build output*, contains the output produced by a build process [33]. *Software components*, the focus of SBOM-focused reports by NTIA [15] and Core Infrastructure Initiative [14], can be considered fine-grained build outputs that can be called by another software. *Software packages*, the focus of SLSA framework [33], can be a collection of software components that are published for use by others. A *software product* can be considered a synonym or a super-set of software packages.

SSC metadata provides machine-readable and authenticated description of SSC artefacts and activities. At the core of a SSC metadata document is a collection of *statements*, consisting of a set of *predicates* about a *subject* such as an SSC activity or artefact [11]. The predicates can describe various types of information:

- *Software Bill of Material (SBOM* is a machine-readable nested inventory of components within a package or product [15]. SBOM can be written in an interoperable format such as SPDX, CycloneDX, and SWID. The US National Telecommunications and Information Administration (NTIA) has published a list of required and recommended data fields of an SBOM document [17]. SBOM can be further enriched by *Vulnerability Exploitability eXchange (VEX)* documents, which integrate vulnerability information with component data from SBOM [7, 15, 35].
- *Provenance* describes how an artefact was produced. It can be considered a claim that some entity produced one or more artefacts by executing a build pipeline definition according to some parameters, possibly using some other artefacts as input [33]. An alternative definition put forward by the SCVS framework [26] considers provenance as the origin and chain of custody of a software component.
- *SSC Layout* is a document providing an ordered list of steps, requirements for the steps, and actors responsible for each step, that requires to be carried out in the SSC to create a software product [11]. It can be considered a super-set of the build pipeline definition.

Software attestation is generated when the statements are authenticated by a digital signature. By signing an attestation, the authors of an SSC metadata document claim that the predicates (e.g., SBOM, provenance) about a subject (e.g., a software package) is true [32]. A trusted authority countersigns an attestation to prove that the attestation was created on a particular date and time and it was valid, according to some predefined criteria. A countersigned attestation can be denoted as a *transparent statement* [8].

SSC Metadata Actors are responsible for the life cycle activities of SSC metadata documents. They can be organised into three types. *SSC Metadata Producers* handle the creation and signing of SSC metadata. This actor is also known as *issuer* in SCITT [8] and *author* in NTIA supplier playbook [21] and CISA SBOM sharing life cycle [35]. Metadata producer can be a project owner [11] or package producers [33], who are responsible for defining the layout of the supply chain of a software artefact. Metadata producers can also be functionaries (e.g., developers, build systems etc.) that handle individual steps in the life cycle of a software artefact [11].

SSC Metadata Consumers retrieve and utilise SSC metadata [20, 33, 35]. A *verifier* is a type of consumers who leverages SSC metadata to verify the authenticity and integrity of software artefacts [8, 11, 33]. The inspection process of the artefacts also creates more metadata, such as VEX. Other types of consumers include *insurers* and *auditors* who leverage SSC metadata to assess an organisation for regulatory or insurance purposes.

SSC Metadata Providers [8] operate the intermediary services that bridge SSC metadata producers and consumers. Providers can also act as a notary, vouching for the existence and the correctness of SSC metadata documents registered by producers [8]. A *SSC metadata auditor* is a special type of auditors who assesses the trustworthiness of the providers based on the log or provenance of the SSC metadata documents distributed by providers [8].

3.1.2 SSC Metadata Life Cycle. The life cycle of SSC metadata comprises ten activities that can be organised into generation, sharing, and consumption phase. Figure 3 depicts the life cycle activities and the actors handling those activities. Where the analysed frameworks and architectures diverge, we present their alternative activities as variations.

SSC Metadata Generation Phase consists of 3 main activities starting with *SSC metadata creation*. This includes the creation of different types of metadata (i.e., SBOM, provenance, attestation, VEX) by individuals or automated tools. The supplier playbook by NTIA [21] specifies this activity in the context of SBOM and introduces a three-step process that includes identifying software components for delivery, acquiring their data, and capturing the

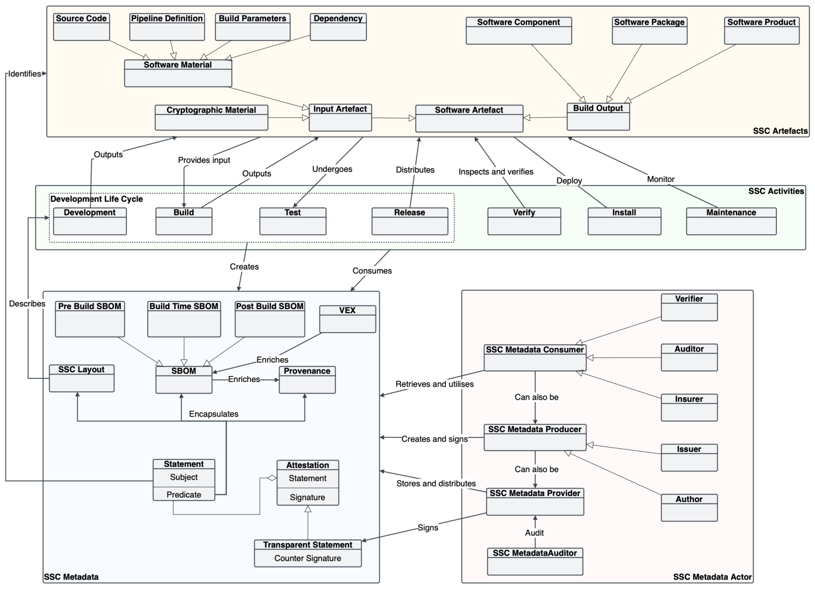

Figure 2: Domain Model for SSC and SSC Metadata Concepts

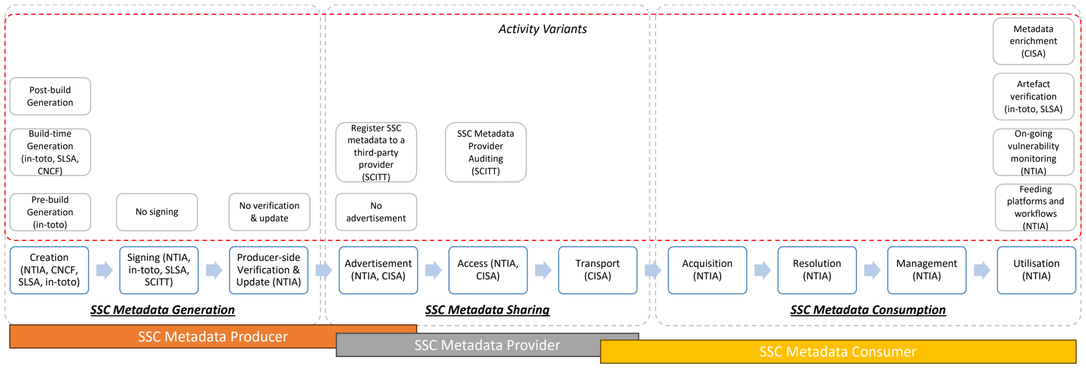
Activity Variants
Post-build Generation
Build-time Generation (in-toto, SLSA, CNCF)
Pre-build Generation (in-toto)
No signing
No verification & update
Register SSC metadata to a third-party provider (SCITT)
SSC Metadata Provider Auditing (SCITT)
No advertisement
Metadata enrichment (CISA)
Artefact verification (in-toto, SLSA)
On-going vulnerability monitoring (NTIA)
Feeding platforms and workflows (NTIA)
Creation (NTIA, CNCF, SLSA, in-toto)
Signing (NTIA, in-toto, SLSA, SCITT)
Producer-side Verification & Update (NTIA)
Advertisement (NTIA, CISA)
Access (NTIA, CISA)
Transport (CISA)
Acquisition (NTIA)
Resolution (NTIA)
Management (NTIA)
Utilisation (NTIA)
SSC Metadata Generation
SSC Metadata Sharing
SSC Metadata Consumption
SSC Metadata Producer
SSC Metadata Provider
SSC Metadata Consumer

Figure 3: Life cycle of SSC Metadata

data in an SBOM format. [15] further prescribes that SBOM authors must include SBOMs of all the dependencies or create those SBOMs on a best-effort basis should such information is not available. The in-toto attestation framework [11] prescribes that all "functionaries", actors participating in the creation of software artefacts, must record metadata after finishing their activity. The SLSA 1.0RC specification [33] also requires that build processes automatically generate provenance attestation describing how the artefact was actually built (by whom, using which process or command, with which input artefacts) in order to satisfy the SLSA level one requirement in the build track. Similarly, the Secure Software Factory framework [7] also describes automated metadata generation that is carried out by a software component embedded into build pipelines.

There are three variants of carrying out the SSC metadata creation activity based on their time of creation within SSC, namely *pre-build* generation, *build-time* generation, and *post-build* generation [21]. Pre-build generation activity creates metadata about source code, such as static dependency declaration in the form of `requirements.txt` in Python projects and `package.json` in NodeJS projects. The SSC layout introduced by the in-toto framework [11] is also a product of a pre-build generation activity. The build-time generation activity generates SSC metadata during or before the end of the build process. It is the most common variant of SSC metadata creation activity, featured in most frameworks including in-toto [11], SLSA 1.0RC [33], and Secure Software Factory [7]. The final variant is post-build generation, which captures metadata such as VEX from automated tests and auditing activities performed on the newly built artefacts.

The second activity in the generation phase is *signing* the generated SSC metadata [21]. This activity is required by the in-toto framework [11] and the SCITT architecture [8]. The second and third levels of the build track of SLSA 1.0RC specification [33] also require SSC metadata to be digitally signed. It should be noted that signing SSC metadata is not mandatory. For instance, the supplier handbook [21] considers metadata signing an optional activity. The level one of the build track of SLSA 1.0RC also does not require signing metadata. Therefore, we introduce *"no signing"* as a variant of the signing activity.

The third activity is *verifying and updating* the content of the created metadata. This activity was introduced in the NTIA's supplier playbook [21] as a remedy for potential errors due to the SBOM creation software tools and errors inherited from upstream SBOMs. These challenges regarding the correctness of SBOM tools were also noted in a prior empirical study [37]. Interestingly, we could not find the provider-side verification and update of metadata in any other frameworks and architectures. Therefore, we introduce *"no verification and update"* as a variant.

SSC Metadata Sharing Phase bridges the SSC metadata producers and consumers [19, 35]. The first activity in this phase is *"advertisement"*, which is performed by SSC metadata producers in order to inform consumers of the availability of SSC metadata documents and how to access them [19]. Producers also use the advertisement activity to notify consumers of corrections or updates of a previously published SSC metadata document. Producers can handle the advertisement themselves or *registering SSC metadata documents with a third party provider* for distribution [8, 19]. Alternatively, producers can *omit the advertisement.* In this case, consumers would query producers for SSC metadata of the received software artefacts.

The second activity is *access* in which consumers' requests for SSC metadata are authorised according to some policies specified by the producer. These policies are defined in a fine-grained manner, limiting access to specific versions or potions of SSC metadata documents [35]. The responsibility for assessing and enforcing access policies lies with either producers or providers who host the SSC metadata documents. If a third-party provider is utilised, the *provider auditing* activity could be performed.

The metadata sharing phase concludes with *transport* SSC metadata documents from a producer or a provider to the authorised consumers using a secure mechanism [19, 35]. According to [35], transfer can be done from single point to single point or single point to multi point based on the targeted consumers.

SSC Metadata Consumption Phase comprises four activities [20]. The first activity is the *acquisition* of SSC metadata documents via the sharing mechanisms provided by producers or providers. The second activity is *resolution*, in which a consumer maps subjects of the statements in an SSC metadata document onto software artefacts within their inventory to establish a link between them. This step might also involve resolving transitive dependency identified in an SBOM.

The third activity is applying *content management* technique on the acquired SSC metadata documents, such as defining and enforcing policies about their storage, data retention and life cycle. If the SSC metadata documents are not utilised in any enterprise and IT workflow, the consumption phase ends here. Otherwise, the last activity is *utilisation* where the acquired SSC metadata documents are *fed into platforms or workflows* (e.g., Application Security Posture Management, Software Asset Management), utilised for *on-going vulnerability monitoring*, employed for in artefact verification processes [11, 33]. An organisation can also *enrich* an acquired SSC metadata document, such as by appending their own VEX, and share it with other consumers.

3.2 Context Model

The SSC Metadata Management System (SCM2) is a suite of software tools for SSC metadata actors to carry out the life cycle activities described above (Figure 3). The system context diagram in Figure 4 presents the positioning and scope of SCM2, reflected via its interactions with three types of actors and eight types of external systems.

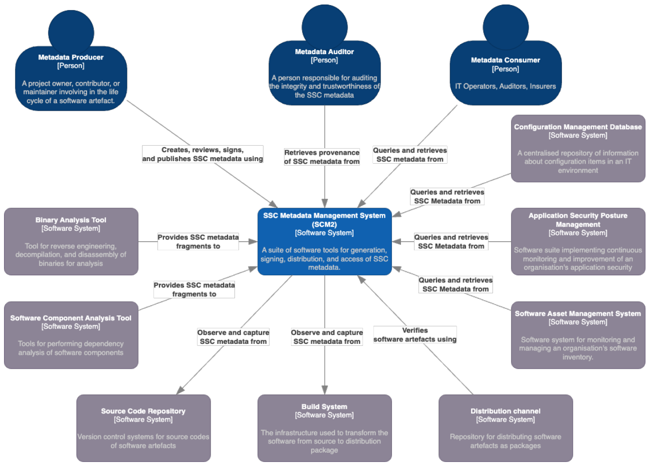
Metadata Producer
[Person]
A project owner, contributor, or maintainer involving in the life cycle of a software artefact.
Metadata Auditor
[Person]
A person responsible for auditing the integrity and trustworthiness of the SSC metadata
Metadata Consumer
[Person]
IT Operators, Auditors, Insurers
Configuration Management Database
[Software System]
A centralised repository of information about configuration items in an IT environment
Creates, reviews, signs, and publishes SSC metadata using
Retrieves provenance of SSC metadata from
Queries and retrieves SSC metadata from
Queries and retrieves SSC Metadata from
Binary Analysis Tool
[Software System]
Tool for reverse engineering, decompilation, and disassembly of binaries for analysis
Provides SSC metadata fragments to
SSC Metadata Management System (SCM2)
[Software System]
A suite of software tools for generation, signing, distribution, and access of SSC metadata.
Queries and retrieves SSC Metadata from
Application Security Posture Management
[Software System]
Software suite implementing continuous monitoring and improvement of an organisation's application security
Provides SSC metadata fragments to
Queries and retrieves SSC Metadata from
Software Component Analysis Tool
[Software System]
Tools for performing dependency analysis of software components
Observe and capture SSC metadata from
Observe and capture SSC metadata from
Verifies software artefacts using
Software Asset Management System
[Software System]
Software system for monitoring and managing an organisation's software inventory.
Source Code Repository
[Software System]
Version control systems for source codes of software artefacts
Build System
[Software System]
The infrastructure used to transform the software from source to distribution package
Distribution channel
[Software System]
Repository for distributing software artefacts as packages

Figure 4: Context model of SCM2

The actors and external systems interacting with SCM2 can be organised into two groups, namely producer and consumer. The producer group contains the metadata producers, binary analysis tools, software component analysis tools (SCA), source code repositories, and build systems. From the perspective of SCM2, the first three actors and external systems are push-based producers because they push fragments of SSC metadata to SCM2 for constructing SSC metadata documents. On the other hand, source code repositories and build systems are pull-based producers because SCM2 observes and pulls metadata fragments from these systems when constructing SSC metadata document for an artefact.

The SSC metadata generation process of SCM2 can be initiated in two ways. Firstly, it can be triggered by the build process as a part of a CI/CD pipeline. This workflow is suggested by most frameworks and particularly relevant to the scenarios where the SSC metadata producer of a software artefact is also the developer and project owner of an artefact. The second approach is to have the process triggered manually by SSC metadata producer, when the need for SSC metadata arises, such as to document their software inventory with SBOM documents. This workflow is particularly relevant to organisations that primarily consume rather than produce software artefacts. The producers might rely on SCA and binary analysis tools to extract the necessary information about their software artefacts. Due to the separation of concern, we decided to place SCA and binary analysis tools outside the scope of SCM2. It means that SCM2 does not contain the ability to perform composition analysis or reverse engineering binaries. It only contains the ability to generate SSC metadata document from the output of these tools.

The second group of actors and external systems of SCM2 consists of metadata consumers, configuration management databases, application security posture management systems, software asset management systems, distribution channels, and metadata auditors. All actors and external systems in this group besides auditors query and retrieve SSC metadata documents from SCM2 for various purposes, such as verifying an incoming software artefact before installation. The metadata auditor, on the other hand, consumes the provenance of the SSC metadata documents in order to assess the trustworthiness of the SSC metadata and the SCM2 itself.

3.3 Container Model

The container diagram in Figure 5 presents the distribution of the functionality of an SCM2 system across 13 containers. It should be noted that a container in the C4 architectural view model denotes a separately runnable unit that executes code or stores data rather than a software container (e.g., Docker). We defer the binding of containers to specific technologies to the design time when architects instantiate concrete SCM2 architecture from the RA. The containers of an SCM2 system organised into five groups according to the SSC metadata life cycle phase in which they manage.

SSC metadata generation group contains three containers. *SSC Metadata Editor* provides a graphical interface for reviewing and editing SSC metadata document. This tool belongs to the type "Edit" of the category "Produce" in the tool classification taxonomy by NTIA [18]. It supports SSC metadata producers in authoring pre-build metadata (e.g., in-toto's SSC layout metadata [11]), supplying post-build metadata and verifying the auto-generated SSC metadata documents [21].

Pipeline Observer captures information about tasks in a build pipeline, such as fetching source code and dependencies, building and testing artefacts. The captured evidence is the input for generating SSC metadata documents. Pipeline observer was introduced in the Secure Software Factory reference architecture [7].

SSC Metadata Generator creates SSC metadata document from the input information according to a prescribed standard such as SPDX. It should be noted that most existing tools combine the metadata generator with a Software Composition Analysis tool or a pipeline observer (e.g., by integrating the generator tool into a build pipeline). In this reference architecture, we model the metadata generator as a separate function unit to follow the separation of concern principle and facilitate the reuse of the software logic for generating SSC metadata document for different standards and templates.

SSC metadata signing group consists of one container: *SSC Metadata Signer.* The signer provides metadata producers with the ability to sign the generated SSC metadata documents, effectively turning them into authenticated statements about a software artefact (software attestations). For example, the "cosign" component[6] of the "sigstore" project can be leveraged as an SSC metadata signer by utilising its built-in support for the in-toto attestation framework.

SSC metadata publishing group comprises two containers. *SSC Metadata Publisher* provides metadata producers with the functionality for publishing and advertising the generated SSC metadata documents. This functionality reflects and implements the "advertisement" phase described in the SBOM sharing life cycle [19, 35]. The publisher can also implement the notarising workflow described by the SCITT architecture [8] where an external authority verifies and countersigns a submitted SSC metadata document to attest for its trustworthiness. The client-side of sigstore's rekor[7] can be utilised to implement the publisher logic.

SSC Metadata Access Control provides metadata providers with the functionality for specifying access control policies of the SSC metadata documents that they publish. It also assesses and authorises requests for SSC metadata documents. This container was introduced based on the requirement for access control mechanism outlined in the SBOM sharing life cycle [19, 35].

SSC metadata sharing group consists of four containers. *SSC Metadata Repository* stores the generated SSC metadata documents. This container corresponds to the "artefact repository" component in the Secure Software Factory reference architecture [7]. *SSC Metadata Access Service* acts as a wrapper around the metadata repository, providing the metadata consumers and external systems with the ability to query and retrieve SSC metadata document. This container implements the "access" and "transport" phases of the SBOM sharing life cycle [19, 35]. It relies on the SSC metadata access control container to assess and authorise incoming requests.

Registry is a verifiable, append-only data store for the records regarding the notarisation of SSC metadata documents. This container was introduced in the SCITT architecture [8]. The public instance of Rekor[8] represents an example of a registry. *Transparency Service* validates and notarises the submitted SSC metadata documents and record the processin the registry. This container was introduced in the SCITT architecture [8].

SSC metadata consumption group consists of three containers. *SSC Metadata Validator* provides the ability to validate the syntax of SSC metadata documents to ensure their adherence to a given standard. This component was introduced by the [20]. *SSC Metadata Verifier* provides the ability to verify the information contained within a SSC metadata document. This component was introduced by the [20]. *SSC Metadata Resolver* implements the resolution step introduced in [20]. It identifies artefacts described in SSC metadata document and resolves transitive dependency.

3.4 Variability

This section discusses the variability of SCM2 and presents some prominent variants in terms of the selection, deployment, governance of containers that practitioners can choose when instantiating a concrete SCM2 system for their use case. For brevity, we present variation points and variants in five groups that correspond to architectural design decisions as follows.

Monolithic vs Distributed: This design decision impacts the deployment and interaction between containers making up an SCM2 instance. The result of this decision leads to two SCM2 variants which we denote as *monolithic* and *distributed.* A *monolithic SCM2* operates and scales as a single unit. For instance, the system might contain a core framework that orchestrates SSC metadata life cycle activities and exposes hooks for integration with modules that implement the functionalities of the containers in Figure 5. The

[6]https://github.com/sigstore/cosign

[7]https://github.com/sigstore/rekor

[8]https://rekor.sigstore.dev

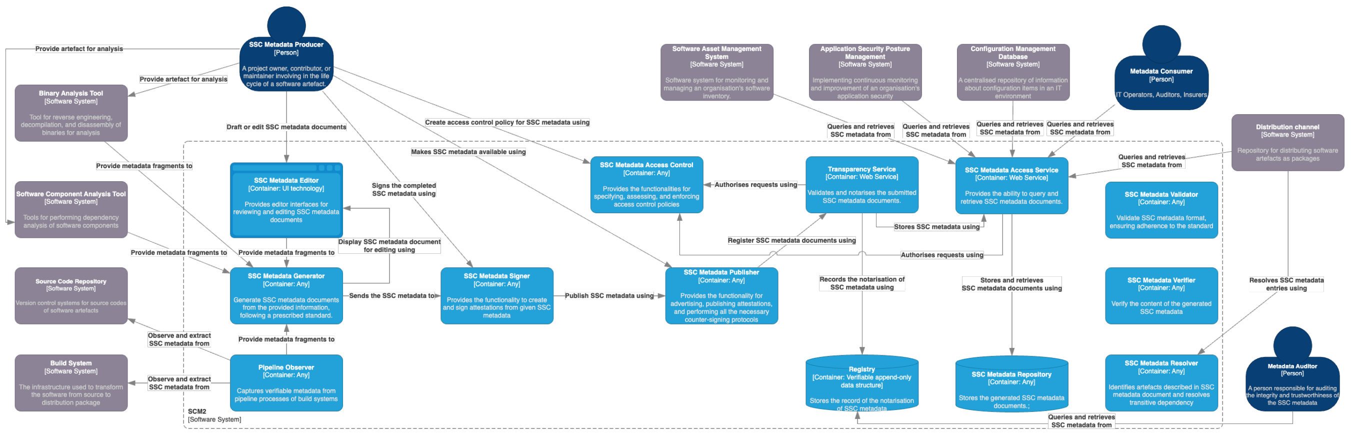
SSC Metadata Producer
[Person]
A project owner, contributor, or maintainer involving in the life cycle of a software artefact.
Provide artefact for analysis
Provide artefact for analysis
Binary Analysis Tool
[Software System]
Tool for reverse engineering, decompilation, and disassembly of binaries for analysis
Draft or edit SSC metadata documents
Create access control policy for SSC metadata using
Makes SSC metadata available using
Provide metadata fragments to
Software Component Analysis Tool
[Software System]
Tools for performing dependency analysis of software components
SSC Metadata Editor
[Container: UI technology]
Provides editor interfaces for reviewing and editing SSC metadata documents
Signs the completed SSC metadata using
Display SSC metadata document for editing using
Provide metadata fragments to
Provide metadata fragments to
Source Code Repository
[Software System]
Version control systems for source codes of software artefacts
SSC Metadata Generator
[Container: Any]
Generate SSC metadata documents from the provided information, following a prescribed standard.
Sends the SSC metadata to
SSC Metadata Signer
[Container: Any]
Provides the functionality to create and sign attestations from given SSC metadata
Publish SSC metadata using
SSC Metadata Publisher
[Container: Any]
Provides the functionality for advertising, publishing attestations, and performing all the necessary counter-signing protocols
Observe and extract SSC metadata from
Provide metadata fragments to
Build System
[Software System]
The infrastructure used to transform the software from source to distribution package
Observe and extract SSC metadata from
Pipeline Observer
[Container: Any]
Captures verifiable metadata from pipeline processes of build systems
SCM2
[Software System]
Software Asset Management System
[Software System]
Software system for monitoring and managing an organisation's software inventory.
Application Security Posture Management
[Software System]
Implementing continuous monitoring and improvement of an organisation's application security
Configuration Management Database
[Software System]
A centralised repository of information about configuration items in an IT environment
Metadata Consumer
[Person]
IT Operators, Auditors, Insurers
Queries and retrieves SSC metadata from
Queries and retrieves SSC metadata from
Queries and retrieves SSC metadata from
Queries and retrieves SSC metadata from
Distribution channel
[Software System]
Repository for distributing software artefacts as packages
Queries and retrieves SSC metadata from
SSC Metadata Access Control
[Container: Any]
Provides the functionalities for specifying, assessing, and enforcing access control policies
Authorises requests using
Transparency Service
[Container: Web Service]
Validates and notarises the submitted SSC metadata documents.
SSC Metadata Access Service
[Container: Web Service]
Provides the ability to query and retrieve SSC metadata documents.
SSC Metadata Validator
[Container: Any]
Validate SSC metadata format, ensuring adherence to the standard
Stores SSC metadata using
Register SSC metadata documents using
Authorises requests using
Records the notarisation of SSC metadata using
Stores and retrieves SSC metadata documents using
SSC Metadata Verifier
[Container: Any]
Verify the content of the generated SSC metadata
Resolves SSC metadata entries using
Registry
[Container: Verifiable append-only data structure]
Stores the record of the notarisation of SSC metadata
SSC Metadata Repository
[Container: Any]
Stores the generated SSC metadata documents.;
SSC Metadata Resolver
[Container: Any]
Identifies artefacts described in SSC metadata document and resolves transitive dependency
Metadata Auditor
[Person]
A person responsible for auditing the integrity and trustworthiness of the SSC metadata
Queries and retrieves SSC metadata from

Figure 5: Container model of SCM2

framework and its modules might run within a process or a software container. Scaling of such monolithic SCM2 might be achieved by creating replicas of the system and placing them behind a load balancer.

On the other hand, a *distributed SCM2* comprises multiple units that can operate autonomously and be deployed on separated machines. The containers described in Figure 5 can be mapped directly onto these units. The microservice architecture can be applied to design the architecture of the distributed SCM2 system.

Centralised vs Decentralised: This design decision reflects the relationship between the entities owning and operating a SCM2 system. It influences the security requirements of interaction between SCM2's containers and the need for decentralisation of the SSC metadata registry and repository. A *centralised SCM2* system operates within the trust domain of the system owner. On the other hand, a *decentralised SCM2* system spans multiple trust domains. For instance, the system might have multiple separated SSC metadata producers and SSC metadata providers. In this context, producer's containers cannot trust provider's containers implicitly, and therefore authentication and authorisation mechanisms must be introduced. The SSC metadata registry and repository might need to be replicated and maintained by the consensus of a group of SSC metadata providers (e.g., by leveraging a distributed ledger) to mitigate the lack of trust in a single entity.

Formal metadata distribution vs Informal sharing: This design decision reflects the degree of separation between SSC metadata producers and SSC metadata consumers. When consumers are also producers, such as when an organisation applies a software composition analysis tool to construct SBOM for their software inventory, containers related to the advertisement, access, and transport of SSC metadata can be omitted. We also denote this variant as *single-user SCM2* system. When consumers and providers are separate entities connected via a supply chain relationship, formal distribution and access control mechanisms are necessary. We denote this variant as *multi-user SCM2* system.

Signing: This design decision reflects the maturity of SSC security procedures and SSC metadata adoption within an organisation. For instance, the signing of SSC metadata is only required from level two requirement of both the build track of SLSA [33] and the SBOM control family of SCVS [26]. We denote the variant of SCM2 without the SSC metadata signer container as *no-sign SCM2* system.

Notarisation: This design decision reflects the maturity of the SSC metadata sharing mechanism. Notarisation includes the countersigning of SSC metadata documents performed by a trusted authority and the recording of such countersigning. For instance, the SCITT architecture [8] presents a notarisation protocol. An *unnotarised SCM2* system omits the transparency service and the registry, sending the published SSC metadata documents to the repository directly.

4 EVALUATION

This section evaluates the correctness and utility of our proposed RA for SCM2 by mapping existing SSC security solutions on the RA. We should note that we focus only on these tools' SCM2-related functionality and components. The following sections present our methodology for selecting SSC security solutions. Based on the mapping results, we discussed the utility of our proposed RA, some observations regarding the state of SCM2 reflected by the analysed SSC security solutions, and a reference instantiation of SCM2 based on the current open-source tools.

4.1 Methodology

We selected prominent SSC security solutions that support SCM2 functionalities for our evaluation. We identified prominent SSC security solutions by finding the union among grey literature articles that list and compare SSC security solutions. We gathered these articles by leveraging Google as the search engine with the search phrase "Software Supply Chain Security Solutions OR SBOM Tools" and excluding sponsored search results. This process resulted in *34 solutions*. We ranked the solutions by the number of mentions and filtered them based on their support for SCM2 functionality and the availability of their technical documentation. The above protocol resulted in five SSC security solutions we utilise for the architecture mapping (Table 2).

Table 2: Architectural Mapping of SSC Security Solutions

Container	**Scribe**	**Chainguard**	**Anchore**	**Snyk**	**FOSSA**
SSC Metadata Editor	View/ Review and Merge (GUI)	View (GUI)	View (GUI)	View (API, CLI)	View (GUI)
Metadata Generator	SBOM from Container images and Source Code (CycloneDX) VeX related to SBOM SLSA Provenance	SBOM from Container Images (SPDX and CycloneDX) Signed Commits	SBOM from Container Images and Source Code (SPDX and CycloneDX)	SBOM from Container images (SPDX and CycloneDX)	SBOM from source code (SPDX and CycloneDX)
Pipeline Observer	✓	✓	-	-	-
Metadata Signer	Utilizes Sigstore tools which support in-toto-Attestations	Utilizes Sigstore tools which support in-toto-Attestations	Utilizes Syft SBOM tool which supports in-toto-Attestations	-	-
Metadata Publisher	Pub/Sub method	Through invites	-	-	Shareable links
Transparency Service	sigstore Rekor	sigstore Rekor	-	-	-
Metadata Access Control	Pub/Sub method	IAM-based access	-	-	-
Metadata Repository	✓ Cloud Storage, OCI Registry, Local directory	✓ OCI Regsitry	✓ PostgreSQL	-	✓
Registry	Rekor Transparency Log	Rekor Transparency Log	-	-	-
Metadata Access Service	✓	✓	✓	-	✓
SSC Metadata Validator	-	-	-	-	-
SSC Metadata Verifier	✓	✓	-	-	-
SSC Metadata Resolver	✓	✓	✓	✓	✓
Deployment	-	-	Distributed	-	Monolithic
Governance	Centralised	Centralised	Centralised	Centralised	Centralised

4.2 Architecture Mapping

Scribe Platform: Scribe[9] helps software producers and consumers manage risks within their supply chains. It helps them generate evidence such as SBOMs to prove the safety and compliance of their software against regulations and standards such as SLSA. The platform implements the pipeline observer by leveraging the *integration with build systems* like GitHub Actions and Jenkins. Scribe also supports *generating SSC metadata from container images*, such as by leveraging the SBOM generation feature of Docker's BuildKit. Based on inputs from build systems and container image analysis, Scribe can *generate SBOM in CycloneDX format and SLSA-compliant provenance statements*. It also *supports the generation of VeX* to enrich the generated SBOMs. Regarding signing, Scribe supports in-toto attestation by leveraging the `cosign` tool from the `sigstore` toolset. Scribe also supports SSC metadata countersigning by leveraging `Rekor` from the `sigstore` toolset to implement both the registry and transparency service. Regarding SSC metadata storage and distribution, Scribe supports storing SSC metadata in a cloud storage service or a locally deployed OCI-based container registry. A Pub/Sub protocol implements the advertisement and access control of SSC metadata. On the consumer side, Scribe supports verification of the software attestation by leveraging `sigstore` tools.

Chainguard Enforce Platform: The Chainguard Enforce[10] is an SSC security solution focusing on container workloads. It continuously inspects the SSC metadata of container images within a Kubernetes cluster to evaluate and enforce user-defined policies. Whilst Chainguard Enforce focuses on the consumption of SBOM for policy enforcement, it can also *generate SBOM in SPDX and CycloneDX format* for container images that lack an associated SBOM by leveraging Syft[11], an open-source SCA tool for container images. Chainguard Enforce does not extract SSC metadata fragments from build systems. Thus, it *does not include a pipeline observer*. Chainguard Enforce utilises *an OCI-based container registry as the repository* for both containers and the associated SBOM. Producers advertise the existence of SBOM by sending invitations with invite codes. Access is granted via an *Identify and Access Management (IAM) model*. Similar to the Scribe platform, Chainguard Enforce utilises `cosign` and `rekor` from the `sigstore` suite for implementing the signer, registry, and transparency service. `cosign` is also used to verify the signature of the container images.

Anchore Platform: Anchore[12] is an SCM2 focusing on generating and consuming SBOM of container images. Anchore revolves around two open-source tools: Syft for generating SBOM from container images and Grype for consuming SBOM for vulnerability scanning. Anchore does not feature a pipeline observer because it does not capture information about tasks in the build pipeline but triggers the generation of SBOMs and vulnerability analysis upon commits. Anchore utilises PostgreSQL as the data store and supports on-prem databases or integration with external services like Amazon RDS. We could not identify support for SSC metadata signing, notarisation, and signing.

Snyk Platform: Snyk[13] is a developer security platform that provides vulnerability detection across the software life cycle. The platform has been extended to support SBOM generation for external consumption. Snyk can *generate SBOM in SPDX and CycloneDX* format. While the Snyk platform has a GUI, it mainly supports viewing vulnerability reports, whereas viewing the generated SBOMs is carried out by accessing a REST API through the curl command line tool or using the Snyk CLI. We could not identify support for SSC metadata signing, notarisation, and signing.

FOSSA Platform: FOSSA[14] is an open-source management platform focusing on providing visibility into the licenses and vulnerability of the open source dependencies of a software inventory. FOSSA supports generating SBOM in SPDX and CycloneDX format. It leverages SCA tools to provide fragments to the SBOM generation process. In the SaaS mode of FOSSA, SBOMs are hosted in cloud storage that serves as a repository. Producers can advertise the generated SBOMs by sharing an access link to the repository with consumers. We could not identify support for SSC metadata signing, notarisation, and signing.

[9]https://scribesecurity.com/scribe-platform/
[10]https://www.chainguard.dev/chainguard-enforce
[11]https://github.com/anchore/syft
[12]https://anchore.com/opensource/
[13]https://snyk.io/
[14]https://fossa.com/

4.3 Discussions

The architecture mappings above demonstrated the correctness and usability of our RA for SCM2. The mappings also reveal some interesting insights about the state of practice of SCM2 and its divergence from academic literature and the existing frameworks.

Containers rather than packages: Three out of five analysed SSC security solutions generate and consume SSC metadata, particularly SBOM, at the software container level. This focus aligns with the cloud native computing paradigm. Nevertheless, it starkly contrasts with the academic literature (e.g., [24]) and our experiences in academia-industry collaborative projects, which tend to focus on software packages distributed via package repositories such as NPM and PyPI. An advantage of a container-centric approach is the ease of SSC metadata sharing. The generated SSC metadata documents can be embedded within container images and thus distributed via the same distribution channels. By embedding the SSC metadata inside container images, one digital signature can be used to secure both the artefact (container image) and the SSC metadata document. Furthermore, if a container image carries its corresponding SSC metadata, the resolution step would be simplified as consumers no longer need to link SSC metadata documents with their corresponding software artefacts.

Consumers are producers: Beside Snyk, the analysed SSC security solutions focus on the consumption of SSC metadata for vulnerability detection and policy enforcement. When SSC metadata is generated, it is usually performed in a reserve engineering manner (e.g., analysing an existing container image to extract information and create an SBOM) by the consumers. In other words, the SCM2 implemented by the analysed SSC security solutions belong to the centralised and single-user variant. They exist as utilities of an organisation rather than a diverse ecosystem of SSC metadata producers, providers, and consumers that jointly produce and consume SSC metadata described in the SCITT architecture [8] and the SBOM network concept [15].

A focus on SBOM: All analysed SSC security solutions focus on SBOM written in SPDX and CycloneDX format rather than software provenance statements in either SLSA or in-toto provenance model. Even when attestation was utilised, it was used to wrap around SBOM content, such as by using in-toto SPDX predicates[15]. The centralised and single-user design focusing on the consumption of SSC metadata might have contributed to the focus on SBOM rather than provenance.

Attestation and notarisation are not commonplace practices: Signing and notarisation of SSC metadata are not universally supported. Such a state of practice aligns with the requirement levels specified by SLSA and SCVS, which put attestation at a less immediate level than SSC metadata generation. When signing and notarisation were performed, the primary implementation was the `sigstore` software suite.

Reference Instantiation of SCM2: Based on the identified open-source tools, we assembled the following reference instantiation of an SCM2. It covers the end-to-end life cycle of all three types of SSC metadata (SBOM, Provenance and Attestation) from the following SSC layout: software supplier committing the source code to a Version Control System, which is then input into a CI/CD build pipeline to generate a container image as output and publishes it to a Container Image Registry along with SSC metadata related to it. Figure 6 presents the binding of SCM2 containers to open-source tools and technologies.

[15]https://github.com/in-toto/attestation/blob/main/spec/predicates/spdx.md

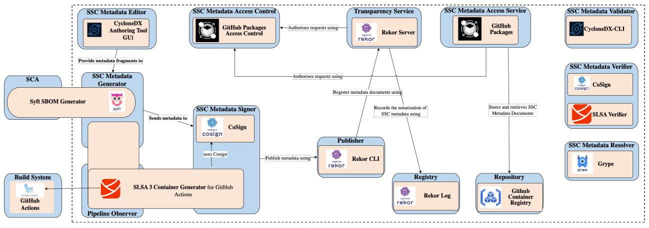
SSC Metadata Editor
CycloneDX Authoring Tool GUI
SSC Metadata Access Control
GitHub Packages Access Control
Authorises requests using
Transparency Service
Rekor Server
SSC Metadata Access Service
GitHub Packages
SSC Metadata Validator
CycloneDX-CLI
Provide metadata fragments to
SSC Metadata Generator
Authorises requests using
SSC Metadata Verifier
CoSign
SLSA Verifier
SCA
Syft SBOM Generator
Register metadata documents using
Records the notarisation of SSC metadata using
Stores and retrieves SSC Metadata Documents
Sends metadata to
SSC Metadata Signer
CoSign
uses Cosign
Publish metadata using
Publisher
Rekor CLI
SSC Metadata Resolver
Grype
Build System
GitHub Actions
SLSA 3 Container Generator for GitHub Actions
Pipeline Observer
Registry
Rekor Log
Repository
Github Container Registry

Figure 6: Reference Instantiation of RA

5 CONCLUSIONS

SSC metadata can provide machine-readable and authenticated visibility into the SSC of a software asset inventory, aiding software consumers in protecting themselves against SSC attacks. This paper presented a Systematisation of Knowledge about SSC metadata in the form of an RA for SCM2. Our RA provides practitioners with a comprehensive domain model and an architectural blueprint for communicating SSC metadata requirements with stakeholders and assembling a suitable SCM2. We demonstrated the correctness and utility of the proposed RA by mapping the architecture of five SSC security solutions with SCM2 onto the RA. The architecture mappings also revealed insights about the state of practice, showing a consumer-driven, SBOM-centric approach to SCM2 and SSC metadata. Because SSC security and SCM2 should be a collaborative effort, for our future work, we plan to develop a software suite based on the RA to facilitate a decentralised and multi-user ecosystem around SCM2 where SSC metadata would be generated from the point of origin, authenticated and notarised by a federation of transparency services and discovered and retrieved by authorised consumers via secure channels. Such a decentralised and multi-user SCM2 ecosystem could serve as a "foundation data layer" on which further security tools, practices, and assurances can be built.

REFERENCES

[1] C J Alberts, A J Dorofee, R Creel, R J Ellison, and C Woody. 2011. A Systemic Approach for Assessing Software Supply-Chain Risk. In *2011 44th Hawaii International Conference on System Sciences*. IEEE. https://doi.org/10.1109/hicss.2011.36
[2] Samuil Angelov, Paul Grefen, and Danny Greefhorst. 2009. A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. IEEE. https://doi.org/10.1109/wicsa.2009.5290800
[3] A. Arsanjani, Liang-Jie Zhang, M. Ellis, A. Allam, and K. Channabasavaiah. 2007. S3: A Service-Oriented Reference Architecture. *IT Professional* 9, 3 (may 2007), 10–17. https://doi.org/10.1109/mitp.2007.53
[4] Nguyen Khoi Tran; Samodha Pallewatta; M. Ali Babar. 2023. SCM2-RA Online Appendix. https://github.com/CREST-Adelaide/SSCTrust-SCM2-RA.
[5] Simon Brown. 2018. The C4 model for visualising software architecture. *Context, Containers, Components, and Code. URl: https://c4model. com/.(accessed: 09.12. 2022)* (2018).
[6] Brian Chess, Fredrick DeQuan Lee, and Jacob West. 2007. Attacking the build through cross-build injection: how your build process can open the gates to a trojan horse. *Fortify Software* (2007), 24–25.
[7] CNCF. 2022. *The Secure Software Factory: A reference architecture to securing the software supply chain.* Technical Report.

[8] Henk Birkholz; Antoine Delignat-Lavaud; Cedric Fournet; Yogesh Deshpande. 2023. *An Architecture for Trustworthy and Transparent Digital Supply Chains*. Technical Report.

[9] Matthias Galster and Paris Avgeriou. 2011. Empirically-grounded reference architectures. In *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS*. ACM. https://doi.org/10.1145/2000259.2000285

[10] Jack Greenfield and Keith Short. 2003. Software factories. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM. https://doi.org/10.1145/949344.949348

[11] in toto. 2017. *in-toto Specification v0.9*. Technical Report.

[12] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, Dawn Leaf, et al. 2011. NIST cloud computing reference architecture. *NIST special publication* 500, 2011 (2011), 1–28.

[13] Gerrit Muller. 2008. A reference architecture primer. *Eindhoven Univ. of Techn., Eindhoven, White paper* (2008).

[14] Frank; Jessica Wilkerson; James Dana; Nagle and Jennifer L. Hoffman. 2020. *Vulnerabilities in the Core: Preliminary Report and Census II of Open Source Software*. Technical Report.

[15] NTIA. 2021. *Framing Software ComponentTransparency: Establishing a Common-Software Bill of Materials (SBOM)*. Technical Report.

[16] NTIA. 2021. *How-To Guide for SBOM Generation*. Technical Report.

[17] NTIA. 2021. *The Minimum Elements For a Software Bill of Materials (SBOM)*. Technical Report.

[18] NTIA. 2021. *SBOM Tool Classification Taxonomy*. Technical Report.

[19] NTIA. 2021. *Sharing and Exchanging SBOMs*. Technical Report.

[20] NTIA. 2021. *Software Consumers Playbook: SBOM Acquisition, Management, and Use*. Technical Report.

[21] NTIA. 2021. *Software Suppliers Playbook: SBOM Production and Provision*. Technical Report.

[22] NTIA. 2021. *Survey of Existing SBOM Formats and Standards*. Technical Report.

[23] Roy Oberhauser and Rainer Schmidt. 2007. Improving the Integration of the Software Supply Chain via the Semantic Web. In *International Conference on Software Engineering Advances (ICSEA 2007)*. IEEE. https://doi.org/10.1109/icsea.2007.42

[24] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves (Eds.). Springer International Publishing, Cham, 23–43.

[25] Chinenye Okafor, Taylor R. Schorlemmer, Santiago Torres-Arias, and James C. Davis. 2022. SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. ACM. https://doi.org/10.1145/3560835.3564556

[26] OWASP. 2020. OWASP Software Component Verification Standard Version 1.0. https://owasp.org/www-project-software-component-verification-standard/.

[27] The President. 2021. Executive Order 14028 of May 12, 2021: Improving Nation's Cybersecurity. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

[28] Pekka Pääkkönen and Daniel Pakkala. 2015. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research* 2, 4 (dec 2015), 166–186. https://doi.org/10.1016/j.bdr.2015.01.001

[29] SEI. 2010. *Evaluating and Mitigating Software Supply Chain Security Risks*. Technical Report.

[30] SEI. 2010. *Software Supply Chain Risk Management: From Products to Systems of Systems*. Technical Report.

[31] Kapil Singi, Jagadeesh Chandra Bose R P, Sanjay Podder, and Adam P. Burden. 2019. Trusted Software Supply Chain. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. https://doi.org/10.1109/ase.2019.00141

[32] SLSA. 2021. SLSA Specificaion Version 0.1. https://slsa.dev/spec/v0.1/onepage.

[33] SLSA. 2023. SLSA Specification Version 1.0 RC. https://slsa.dev/spec/v1.0-rc1/onepage.

[34] Sonatype. 2023. *The 8th Annual State of the SoftwareSupply Chain Report*. Technical Report.

[35] Jeremiah Trent Stoddard, Michael Adam Cutshaw, Tyler Williams, Allan Friedman, and Justin Murphy. 2023. *Software Bill of Materials (SBOM) Sharing Lifecycle Report*. Technical Report.

[36] Michael Weyrich and Christof Ebert. 2016. Reference Architectures for the Internet of Things. *IEEE Software* 33, 1 (jan 2016), 112–116. https://doi.org/10.1109/ms.2016.20

[37] Boming Xia, Tingting Bi, Zhenchang Xing, Qinghua Lu, and Liming Zhu. 2023. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead. (2023). https://doi.org/10.48550/ARXIV.2301.05362