Introduction of Blockchain system in SBOM technology

Advanced Topics in Computer and Network Security

Federico Sanguinetti
University of Padua

Marco Pasca
University of Padua

Abstract

The increasing complexity of modern software and supply chains resulted in the need of finding a technology which can guarantee visibility and a clear vision of the cybersecurity risks. The introduction of the SBOM software products from CISA, in the USA, has pointed a solution in this way but has brought with it several other challenges related to their use and their adaptability. In particular, most of them are related to integrity and privilege management due to the different composition of the supply chains and the software development in various companies. This study proposes a new model based on the management of the SBOM software through the blockchain technology.

Keywords

SBOM, blockchain, model, supply chain, software, transaction

1. Introduction

Modern software systems involve increasingly complex and dynamic supply chains, which as known, represent the backbone of the development environments. As a consequence of that, a lack of systemic visibility into the composition and functionality of these systems can result in substantial cybersecurity risk, besides the possibility of increased costs of development, procurement and maintenance. The direction, which was decided to take in this regard, was the one of transparency, which can improve different aspects of the software development process, such as, the ability to identify vulnerable software components; the reduction of unplanned and unproductive work; the identification of suspicious software components. Talking in general this process can also help the vendors to easily differentiate themselves in the market and with less effort. To address the problem of poor supply chain transparency, the NTIA (National Telecommunications and Information Administration) convened a multistakeholder process which has resulted, for the main aspect, in the creation of a model for software component information that can be universally shared among the industry ecosystems. The model defines this component as a SBOM: a Software Bill Of Materials. To scale this model globally, it is necessary to address the difficult problem of universally identifying and defining certain aspects of software components. So, a subsidiary goal was to select a core, baseline set of attributes necessary to identify components with sufficient relative uniqueness. Another goal was to capture SBOM applications and consider what additional, optional attributes and external elements might be needed beyond the baseline set. Since the introduction of this new component in the supply chain, some concerns about its use and efficiency have come out, such as:

- How to verify the source of SBOM data and possible related alterations.
- How to control the development flow of the SBOM, including who can access and modify it and how can he do that.
- How to adapt the transparency of the SBOM to the different needs of a company.

In this paper our goal is to propose a model which implements the blockchain technology to address all of these aspects, based on the efficiency and high-level of integrity and security a blockchain network can bring to this scenario. Also the decentralized approach which this technology adopt is capable in our opinion to handle the granularity and the diversification that a SBOM product needs, since its large target use on the software market.

2. What is a SBOM?

A SBOM is a formal machine-readable inventory software components and dependencies, information about those components, and their hierarchical relationships. A SBOM should be: comprehensive, open or proprietary software, widely available or access restricted. On the other hand, a SBOM should include baseline attributes with the ability to uniquely identify individual components in a standard data format. Talking about the benefits and use cases related to this artifact, the reduction in costs, security risks, licensing and software product compliance certainly stand out. All these features result in an improved software development, which comprehends supply chain management, vulnerability management, assets management and high assurance processes. An SBOM is effectively a nested inventory, a list of ingredients that make up software components. In particular it is possible to group a SBOM in three minimum elements which are: data fields, automation support, practice and processes. The first one refers to the baseline information about each component that should be tracked; the second one refers to the automatic generation and machine readability to allow for scaling across the software ecosystem; the last one defines the operations of SBOM requests, generation and use including frequency, depth, known unknowns, distribution and delivery, access control and accommodation mistakes In the next sections will be given a more precise description of what are the main topics related to the SBOMs, such as attributes, processes and roles.

2.1. SBOM Attributes

The primary purpose of an SBOM is to uniquely and unambiguously identify software components and their relationships to one another. Therefore, one necessary element of an SBOM system is a set of baseline attributes that can be used to identify components and their relationships. An SBOM system for format that follows the guidance and framing proposed in this document must support these baseline attributes. An SBOM system or format may support additional attributes. There are also cases where certain attributes may not be available or applicable or may not materially contribute to component identification. [3]
A SBOM is composed of 8 principle attributes:

- Author name: author of the SBOM.
- Timestamp: date and time when the SBOM was last updated.
- Supplier name: name or other identifier of the supplier of a component in an SBOM entry.
- Component name: name or other identifier of a component.
- Version: version of a component.
- Component hash (recommended but not required): cryptographic hash of a component.
- Unique identifier: additional information to help uniquely define a component.
- Relationship: association between SBOM components.

Relating to the relationship we can have 4 different types:

- Unknown: default setting. There is not yet any claim, knowledge, or assertion about upstream components.
- None: there are no immediate upstream relationships. As defined by the supplier, the component has no upstream components.
- Partial: there is at least one immediate upstream relationship and may or may not be others. The relationship that are known, are listed.
- Known: the complete set of immediate upstream relationships is known and listed.

A SBOM can also be described by additional elements, such as authenticity and integrity.

2.2. SBOM Recommended Data Fields

The following data fields are recommended for consideration, especially for higher levels of security. [4]

2.2.1. Hash of the Component

Robust identifiers are important for mapping the existence of a component to relevant sources of data. A cryptographic hash would provide a foundational element to assist in this mapping. Hashes also offer confidence that a specific component was used. There are some situations when a hash may not be possible, or convey relatively little value. If component information was obtained from a tool that did not have direct access to the underlying component, then the component author may not be able to credibly determine the exact bits used, and so be unable to generate a hash.

2.2.2. Lifecycle Phase

The data about software components can be collected at different stages in the software lifecycle. Due to unique features of each of these stages, the SBOM may have some differences depending on when and where the data was created. So simply noting how this data was captured, will be helpful for consumption and data management.

2.2.3. Other Component Relationship

The minimum elements of SBOM are connected through a single type of relationship: dependency. That is, X is included in Y. This relationship is implied in the SBOM graph structure. Other than that, we can use “derivation” or “descendancy” dependencies.

2.2.4. License Information

Helps organizations to track the licenses and terms of their diverse software components.

2.3. SBOM Process

This section describes how to create and exchange SBOM information from three stakeholder perspectives: those who produce, choose, and operate software.[3]

2.3.1. How

To create a SBOM, the suppliers define components that the supplier creates themselves, produces baseline and any additional attributes for these components and enumerates all directly included components.

2.3.2. When

A SBOM should be created when new component is released, and should be update when component changes (should also change when new SBOM information becomes available even if the components themselves have not changed). When there is a change of an existing component, there are two possibilities of threating those component: treat it as a separate, new component added to the existing SBOM or to create a new one.

2.3.3. Exchange

Exchange is a very important concept for SBOMs. The primary exchange is between the supplier and the consumer. Due to variety of software, it’s unlikely that one SBOM exchange mechanism will suffice. Some existing formats, namely SWID and SPDX, are provided as additional files as part of a component distribution or delivery.

2.3.4. Rules

Participants in a SBOM system have to follow a set of network rules so that SBOM systems function at scale. A SBOM must list at least one primary component, which defines the subject of the SBOM. An SBOM lists components that can be:

- Originally created by supplier who is authoritative source of software.
- Integrated as a component from an upstream supplier who also provides a SBOM.
- Integrated as a component from an upstream supplier who does not provide a SBOM

2.4. SBOM Roles

As mentioned in the previous section, SBOM is defined by three stakeholder perspectives: those who produce, choose, and operate software.[2]

1. **Produce**: the person/organization that creates a software component or software for use by others [write/create/assemble/package]
2. **Choose**: the person/organization that decides the software/products/suppliers for use [purchase/acquire/source/select/approve]
3. **Operate**: the person/organization that operates the software component [uses/monitor/maintain/defend/respond]

2.4.1. Produce Software

A SBOM can help a software supplier produce their software in the following ways:

- Reduce unplanned/unscheduled work
- Reduce code bloat
- Understand dependencies
- Know and comply with the license obligations
- Monitor components for vulnerabilities
- End of life
- Make code easier to review
- Blacklist of banned components
- Provide a SBOM to a costumer

2.4.2. Choose Software

Choosing software includes the selection and acquisition of software products. These tasks differ from organization to organization but have common steps between them. Since the choice is usually a long-lasting commitment, it is very important that the decision to acquire one software product or component vs. another is made with forethought and planning. For this reason, several functional teams could be involved, including end users, finance, legal, and technical support are often involved The acquisition process can be formal or informal and may include steps such as reviewing requirements, market surveys, evaluating suppliers and products, and the purchasing and receiving to the actual acquisition of the software. A SBOM can help an organization configure, maintain, and administer its software in the following ways:

- Identify potentially vulnerable components
- More targeted security analysis
- Verify the sourcing
- Compliance with policies
- Aware of end-of-life components
- Verify some claims
- Understand software's integration
- Pre-purchase and pre-installation planning
- Market signal

2.4.3. Operate Software

The last stakeholder perspective includes all the installation, configuration, maintenance and administration responsibilities necessary once the software package or component has been selected and acquired. A SBOM can help an organization choose their software in the following ways:

- Organization can quickly evaluate whether it's using the component
- Drive independent mitigations
- Make more informed risk-based decisions
- Alerts about potential end-of-life
- Better support compliance and reporting requirements
- Reduce costs

2.5. SBOM Ecosystem Benefits

The adoption of SBOM can bring significant near-term and subsequent benefits which emerge for the entire ecosystem. In a supply chain each member is potentially a link (figure 1), therefore as new links participate, the entire chain can benefit from various advantages. [2]
More specifically, an SBOM can improve the health of the entire software ecosystem in these ways:

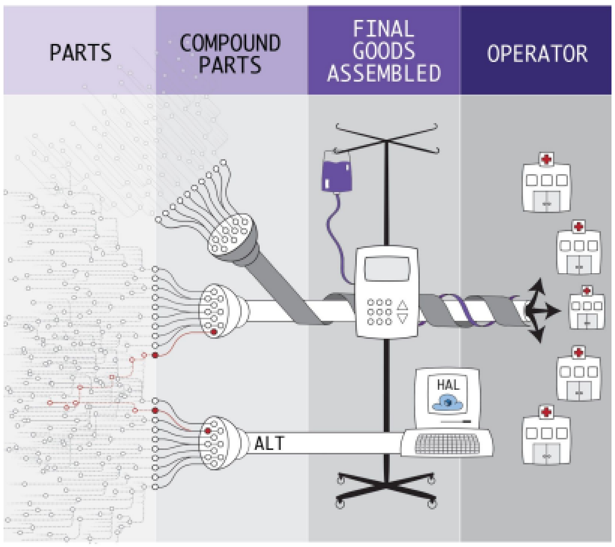
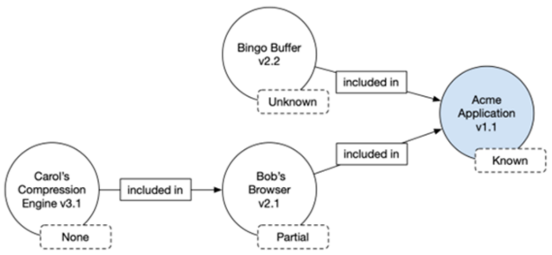
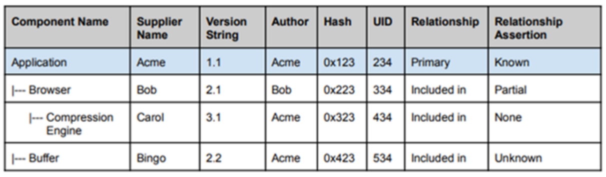


Figure 1: Software Supply Chain Ecosystem





Component Name	Supplier Name	Version String	Author	Hash	UID	Relationship	Relationship Assertion
Application	Acme	1.1	Acme	0x123	234	Primary	Known
|--- Browser	Bob	2.1	Bob	0x223	334	Included in	Partial
|--- Compression Engine	Carol	3.1	Acme	0x323	434	Included in	None
|--- Buffer	Bingo	2.2	Acme	0x423	534	Included in	Unknown

Figure 2: Conceptual SBOM table with upstream relationship assertions

- Accelerate vulnerability management
- Amplified "nerd "immunity""
- Selecting for better suppliers
- Weathering suppliers going away
- Combined/cumulative value

2.6. SBOM Example

To further illustrate the relationships described in the previous section, consider this SBOM example. In Figure 2 is shown one approach to viewing SBOM information and relationships. This is only a conceptual representation and not specific formats like SPDX, CycloneDX, or SWID. [3]

3. SBOM Types

This section is dedicated to the presentation of the various types of existing SBOM. For each type, there is going to be a short description and the benefits and limitations linked to each of them are described. [1]

3.1. *Design*

SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact. Typically derived from a design specification, RFP, or initial concept.

Benefits: highlights incompatible and recommended component ahead of the process

Limitations: difficult to generate

3.2. *Source*

SBOM created directly from the development environment, source files, and included dependencies used to build an product artifact. Typically generated from software composition analysis (SCA) tooling, with manual clarifications.

Benefits: Provide visibility without access to build process. Can facilitate remediation of vulnerabilities at the source.

Limitations: Can highlight components (which may have vulnerabilities) that never run or are compiled out in deployed code. May not include dynamic components. May require references to other SBOMs.

3.3. *Build*

SBOM generated as part of the process of building the software to create a releasable artifact (e.g., executable or package) from data such as source files, dependencies, built components, build process ephemeral data, and other SBOMs. Typically generated as part of a build process. May consist of integrated intermediate Build and Source SBOMs for a final release artifact SBOM.

Benefits: Increases confidence that the SBOM representation of the product artifact is correct, due to the information available during the build. Provides visibility into more components than just source code. Increased trust due to the SBOM signing.

Limitations: Potentially have to change the build process to generate this SBOM. Highly dependent on the build environment. May be difficult to capture indirect and/or runtime dependencies, also can have problems with dynamic links.

3.4. *Analyzed*

SBOM generated through analysis of artifacts (e.g., executables, packages, containers, and virtual machine images) after its build. Such analysis generally requires a variety of heuristics. In some contexts, this may also be referred to as a “3rd party” SBOM. Typically generated through analysis of artifacts by 3rd party tooling.

Benefits: Provides visibility without an active development environment. Does not need access to the build process. Can help verify SBOM data from other sources. May find hidden dependencies missed by other SBOM type creation tools.

Limitations: May be from to omissions, errors, or approximations if the tool is unable to decompose or recognize the software components precisely. May depend on heuristics or context-specific risk factors.

3.5. *Deployed*

SBOM provides an inventory of software that is present on a system. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior in a (potentially simulated) deployment environment. Typically generated by recording the SBOMs and configuration information of artifacts that have been installed on systems.

Benefits: Highlights software components installed on a system, including other configurations and system components used to run an application.

Limitations: May require changing install and deploy processes to generate. May not reflect the software’s runtime environment, as components may reside in inaccessible code.

3.6. *Runtime*

SBOM generated through instrumenting the system running the software, to capture only components present in the system, as well as external call-outs or dynamically loaded components. In some contexts, this may also be referred to as an “Instrumented” or “Dynamic”

SBOM. Typically generated from tooling interacting with a system to record the artifacts present in a running environment and/or that have been executed.

Benefits: Provides visibility to understand what is in use when the system is running, including dynamically loaded components and external connections. Can include detailed info about whether components are active and what parts are used.

Limitations: Requires the system to be analysed while running, which may require additional overhead. Some detailed information may be available only after the system has run for a period of time until the complete functionality has been exercised.

4. SBOM Attacks

In this section we will analyse how SBOM handles vulnerabilities into a software product, and how this, summed to what we have said in the initial statement may lead to possible attacks or problems to the supply chain. Let us firstly start on the mechanism with which the SBOMs manage the risk into software: while vulnerabilities are a key component of understanding risk, not all vulnerabilities put organization at risk. Some vendor data suggests that a relatively small percentage of vulnerable components have a security impact in the environment where the software is deployed. In the SBOM context, focusing on upstream vulnerable components that have been declared not to have impact on the downstream software will waste time and resources. To avoid this kind of situation SBOMs rely to a piece of software called "Vulnerability Exploitability eXchange" or VEX file. In this document all the vulnerabilities are classified with the following aliases:

- Not Affected
- Affected
- Fixed
- Under investigation.

A SBOM consumer may also be concerned about verifying the source of the SBOM data and confirming it was not altered. In the software world, integrity and authenticity are most often supported through signatures and public key infrastructure. Those supplying and requesting SBOM are encouraged to explore options to both sign SBOMs and very tamper-detection. Now based on this, we can basically have two types of attack against SBOM:

- *Tamper attacks*, or integrity-based attakcs, which aim to modify in an unauthorized way the content of the SBOM and cause disease in the supply chain management;
- *Resource attacks*, which aim to alter the VEX files and structures SBOMs rely on.

As we will see, the introduction of a Blockchain network has been thought to specifically counter this problems, since the maintenance of integrity is one of the strengths of this technology.

Our Proposal

As seen in the previous section, the major issues related to SBOMs are integrity and privilege management related to general functioning. To overcome those problems, we chose to propose a model based on blockchain technology, which we recall has strengths in its being a decentralized and highly secure system with regard to transactions. Among the various types of SBOMs presented, the given model was made for the Source type, as it appears to be the most basic and used among from the proposed one.

4.1. Role based-Scheme

For the management of the roles and privileges that nodes have within the network, we relied on a hierarchical type model (see figure 3) in which each node belonging to the system, has access to the information of the nodes of the same level and to the information of the child nodes. Access to information for each node is defined by its privileges, which are assigned according to its membership group and with descendant order, respectively: root, subordinates, children. This means, that in order to obtain a specific permission, it is necessary for the child node to make a request to the parent node.

4.2. Transaction

In the protocol we devised, the concept of transaction within a blockchain is taken and adapted to the software product, keeping the role-based scheme mentioned earlier as a reference. In particular, the concept of generic transaction is linked with that of SBOM update. With the term "update" we mean any modification/insertion/removal of an SBOM. Read access is not considered part of a transaction, as it

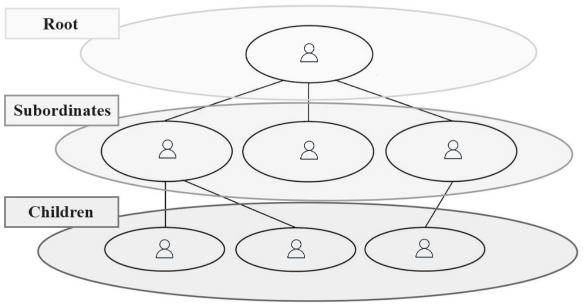


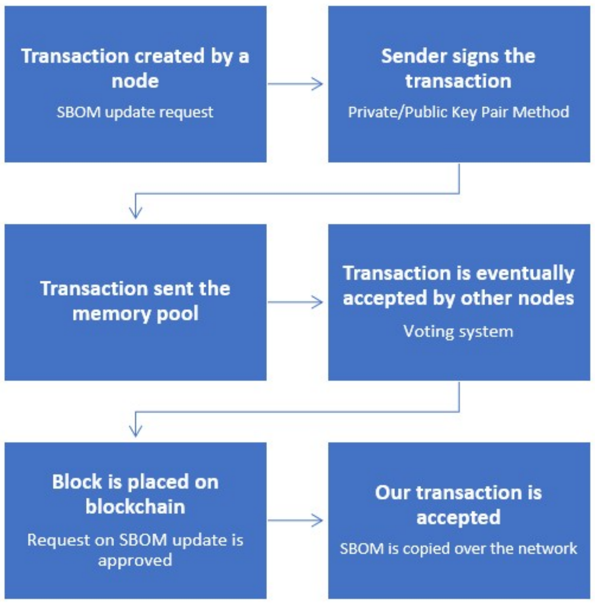
Figure 3: Role-based scheme

does not modify the system data. For this reason, this type of access must be authorized separately through role-base access control.

Wanting to go more specific, the content of a transaction is determined by the type of the update and the id of the component to be updated. All references to the hash of the previous block and the id of the current block, which are present in all blockchains, must be also added to the content. The next subsections will discuss the details of the protocol we have implemented.

4.3. Protocol operation

The scheme we illustrated in figure 4 consists of 6 steps. The first step begins following an update request from a node, which creates a transaction. The node subsequently, signs the transaction through its Private Key, a transaction that is then broadcast to all nodes in the reference layer in the network. The received transaction can then be accepted by miners using their public key, which we recall are generally nodes capable of providing permissions, and grouped within a block. The acceptance by miners is done following a voting system which is necessary for validating the transaction and placing the block within the blockchain. The moment the transaction is confirmed, it is accepted and therefore is propagated within the network. The design choice of not including a read permission within a transaction is due to the structure of the blockchain itself, as once inserted, a block cannot be removed. For this reason, but also with general validity, unnecessary or intermediate insertions within a design phase should be avoided as they would lead to unnecessary overheads and consumption of resources.


Transaction created by a node
SBOM update request
Sender signs the transaction
Private/Public Key Pair Method
Transaction sent the memory pool
Transaction is eventually accepted by other nodes
Voting system
Block is placed on blockchain
Request on SBOM update is approved
Our transaction is accepted
SBOM is copied over the network
Figure 4: Enter Caption

References

[1] CISA. Types of software bill of material (sbom) documents, 2023.

[2] NTIA. Roles and benefits for sbom across the supply chain, 2019.

[3] NTIA. Framing software component transparency: Establishing a common software bill of materials (sbom), 2021.

[4] NTIA. The minimum elements for a software bill of materials (sbom), 2021.