Journal of Current Research in Engineering and Science
JCRES
Bi-Annual Online Journal (ISSN : 2581 - 611X)

Volume 5- Issue 2, Paper 41 August 2022

Importance of Software Bill of Material (SBoM) & Software Composition Analysis Tool (SCA)

Vivek Bargat
Eaton
Pune, India
vivekbargat@eaton.com

***Abstract* — Open-source components have become an essential part of today's software development process, helping organizations speed up their release cycles and push out innovative software faster. In fact, over 90% of the respondents to a survey conducted by Gartner stated that they rely on open-source components. However, open-source components also bring a new set of challenges that organizations need to address to keep their products secure and compliant.**

Gartner's first-ever report about Software Composition Analysis (SCA) — Technology Insight for Software Composition Analysis — explains why security and risk management leaders must proactively control the open-source components in their application, details the main benefits and capabilities of SCA tools, as well as the risks organizations need to look out for, and presents recommendations for using SCA tools.

Keywords—Software Bill of Material (SBoM), Software Composition Analysis Tool (SCA), DevSecOps

INTRODUCTION

Software supply chain attacks are not a new security concern, but recent high-profile attacks such as SolarWinds, CodeCov, and Kaseya have brought the topic to the forefront of cybersecurity awareness across the globe.

Gartner predicts that by 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a threefold increase from 2021.[1]

Supply chain attacks have not only increased in volume and frequency, but have also become more sophisticated, involving Advanced Persistent Threat (APT) actors. This trend, together with the potentially wide impact of a singular successful supply chain attack, requires organizations to take dedicated steps to ensure the security and integrity of both the software they build and supply to their customers, and/or the software they procure for internal usage.

Open source is extremely attractive to attackers, who implant malware directly into open-source projects to infiltrate the software supply chain. In the past, software supply chain exploits were used against publicly disclosed open-source vulnerabilities that were left unpatched. But instead of waiting for disclosures to pursue an exploit, attackers now are choosing to inject new vulnerabilities into open-source projects that feed the global supply chain – and then exploit those vulnerabilities before they are discovered.

The growth in software supply chain attacks has – not surprisingly – resulted in a firestorm of activity on the part of regulators and governmental organizations to determine how such attacks can be prevented. A key facet for all the regulatory agencies involved with securing the software supply chain is the demand for a software bill of materials (SBoM), a formal record that contains the details and supply chain relationships of various components that are used to build software.

Today's software development landscape, with short release cycles, leads software development teams to rely heavily on open source to accelerate innovation. It is important, however, that each open-source component included in an organization's projects is tracked to avoid risks for legal non- compliance and to maintain a strong security posture. In a DevSecOps environment, this tracking must be integrated into every stage of the development lifecycle.

WHAT IS SOFTWARE BILL OF MATERIAL (SBOM)

The United States' NTIA (National Telecommunications and Information Administration) [2] describes SBoMs as a nested inventory – a list of ingredients that make up software components – which provides those who produce, purchase, and operate software with information that enhances their understanding of the supply chain.

Volume 5- Issue 2, Paper 41 August 2022

Like the list of ingredients on food packaging, a Software Bills of Materials (SBoM) provides details on the different components included within a supplied product: open-source dependencies, containers, and build tools used.

Following President Biden's Executive Order [3], SBoMs will be a requirement for any vendor supplying software to the Federal Government and will likely become a cross- industry standard.

SBoMs provide critical visibility into software components and supply chains. The aim is that they can be shared without friction between teams and companies as a core part of software management for critical industries and digital infrastructure.

WHY IS THERE AN URGENCY FOR SBOM?

In the wake of the SolarWinds [4] hack and the recent Log4Shell vulnerability in Log4j [5], governments are prioritizing cybersecurity and are actively mapping out plans to ensure their departments, partners, and stakeholders are building greater cyber resilience.

The concept of an SBoM is not new, but it's garnered much more interest lately due to the U.S. Cybersecurity Executive Order and the UK Government Cyber Security Strategy:
2022 to 2030.

WHO OWNS/MANAGES AN SBOM?

In the past, SBoMs were used primarily by compliance teams for audits, license monitoring, and to comply with industry-specific regulations. However, with the rise of software supply chain attacks, SBoMs have become critical for security and development teams alike.

Security teams need visibility into what risks exists, the potential impact, the issues that need to be prioritized, and a path for remediating any impending risks.

Development teams can use SBoMs to keep a pulse on the open-source, commercial, and custom-built software components that they use across the applications they develop, manage, and operate.

From a cross-functional perspective, SBoMs provide a means to manage dependencies, identify security issues for remediation early, and ensure that an organization is meeting the standards set in its security posture.

WHY DO WE NEED SBOM

The heightened awareness of supply chain security and SBoMs are a good thing but organizations that produce, choose, operate software might still be questioning their usefulness. They may agree to supply them but fail to see downstream benefits. So why should organizations insist on SBoMs from your software suppliers? Why should you start creating them for your products or the applications you are using today? Consider these benefits:

Lower costs – You can greatly reduce time to mitigate security risks with an effective detection process. This saves not just time, but even the costs incurred due to these risks.

Identifying and avoiding vulnerabilities in reused components in your own developed software and software purchased by your organization.

Managing software supply chain risk to remove and reduce the unknown security risk in reused software. SBoMs provide data for business decisions on software purchases and open-source reuse.

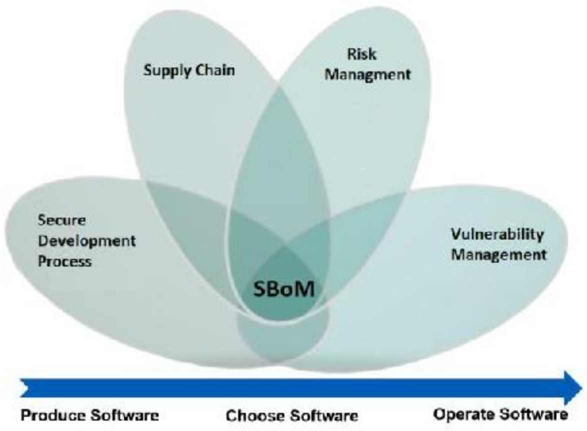


Fig. 1. Importance of SBoM for any organisation dealing with software

Improved security and downstream benefits that come with risk management and mitigation. Avoiding and catching security risks before they become embedded in the product pays huge dividends during development and deployment of your products.

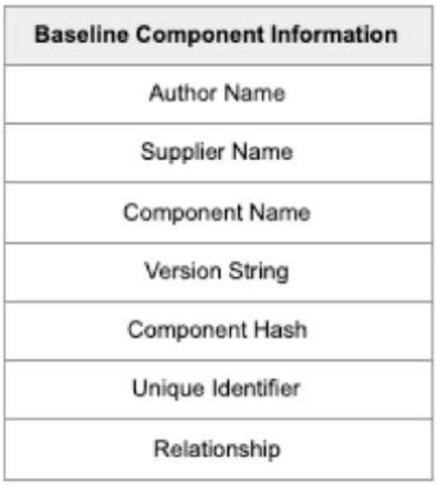
Common understanding of software assets that comes with a standardized SBoM amongst software developers, suppliers, and open-source projects. SBoMs become a way to communicate software contents and dependencies within and outside an organization.

Security teams can leverage SBoMs to confirm that the software they're using, purchasing, or building is free from known vulnerabilities and components with unacceptable licenses. This in turn can provide evidence that an organization is meeting compliance requirements. And as mentioned above, SBoMs will be especially important for software vendors that sell to the U.S. government due to the requirements in the executive order.

COMPONENTS OF SBOM

Author - The person or an organization that drafts the metadata. Although not necessary, the author is usually the supplier of the software component.

Component - The name of the software component, which is usually given by the supplier. It will also include multiple names, if any, of the component.



Baseline Component Information
Author Name
Supplier Name
Component Name
Version String
Component Hash
Unique Identifier
Relationship

Fig. 2. Components of SBoM. Diagram courtesy of the National Telecommunications and Information Administration (NTIA) [6]

Volume 5- Issue 2, Paper 41 August 2022

Component Relationship - A description of how two or more components are related. It is either described using the relationship type 'includes' or 'included in.' It is common for components to be related across the supply chain.

Component Hash - For easy identification of a component, it is defined by a unique cryptographic hash. This is to identify the precise and unmodified version of the component.

Supplier - Indicates the name of the person or the organization that owns or has supplied the software component.

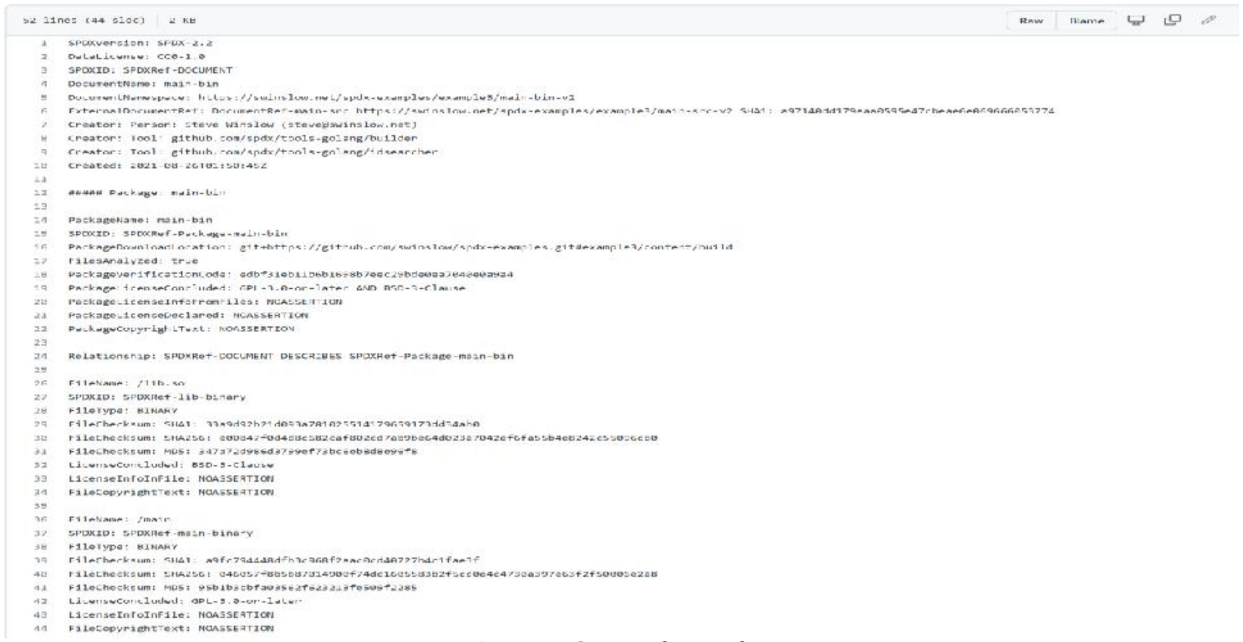
Unique identifier - Unique identifiers help you in determining components in key database like NVD [7]. Some identifiers are Software Identification (SWID) Tags and Common Platform Enumeration (CPE).

Version string - This defines the version of the software and is often given by the supplier.

FORMATS OF SBOM

The National Telecommunications and Information Administration (NTIA) suggests three formats for generating the SBoM inventory list that identifies software entities and their associated metadata:

- The Linux Foundation's **Software Package Data Exchange (SPDX)** is an open standard for creating an SBoM inventory list containing all software components, component licenses, copyrights, and security references.

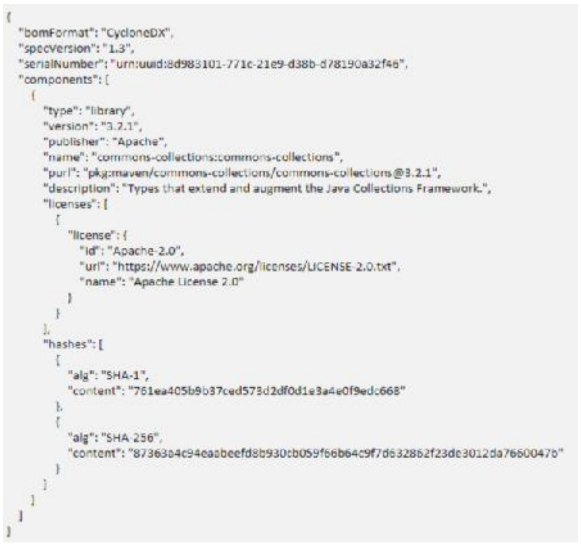


```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: main-bin
DocumentNamespace: https://swinslow.net/spdx-examples/example3/main-bin-v1
ExternalDocumentRef: DocumentRef-main-src https://swinslow.net/spdx-examples/example3/main-src-v2 SHA1: a97140dd179eaa0595e47cbeae6e069666053274
Creator: Person: Steve Winslow (steve@swinslow.net)
Creator: Tool: github.com/spdx/tools-golang/builder
Creator: Tool: github.com/spdx/tools-golang/idsearcher
Created: 2021-08-26T01:58:45Z

##### Package: main-bin

PackageName: main-bin
SPDXID: SPDXRef-Package-main-bin
PackageDownloadLocation: git+https://github.com/swinslow/spdx-examples.git#example3/content/build
FilesAnalyzed: true
PackageVerificationCode: edbf31eb11b6b1698b7eec29bd00a3784000a924
PackageLicenseConcluded: GPL-3.0-or-later AND BSD-3-Clause
PackageLicenseInfoFromFiles: NOASSERTION
PackageLicenseDeclared: NOASSERTION
PackageCopyrightText: NOASSERTION

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-main-bin

FileName: /lib.so
SPDXID: SPDXRef-lib-binary
FileType: BINARY
FileChecksum: SHA1: 33a9d32b21d093a78102551417963917 3dd54ab0
FileChecksum: SHA256: e00847f0d4d8c582caf802cd7e89b864d02387042ef6fa55b4e8242c55056c50
FileChecksum: MD5: 347a72d986d8799ef73bce0b8d8e96f8
LicenseConcluded: BSD-3-Clause
LicenseInfoInFile: NOASSERTION
FileCopyrightText: NOASSERTION

FileName: /main
SPDXID: SPDXRef-main-binary
FileType: BINARY
FileChecksum: SHA1: a9fc794448dfb3c968f2eac0cd40727b4c1fae1f
FileChecksum: SHA256: 046057f8b5887814902f74dc16d55838 2f5cd0e4c4730a397863f2f50000e288
FileChecksum: MD5: 95b1b3cbfa03562f622213fe509f2285
LicenseConcluded: GPL-3.0-or-later
LicenseInfoInFile: NOASSERTION
FileCopyrightText: NOASSERTION
```

Fig. 3. Sample SPDX format of SBoM

- The OWASP **CycloneDX** is a lightweight SBoM standard for creating a complete inventory of first- and third-party software components for risk analysis.

Volume 5- Issue 2, Paper 41 August 2022



```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.3",
  "serialNumber": "urn:uuid:8d983101-771c-21e9-d38b-d78190a32f46",
  "components": [
    {
      "type": "library",
      "version": "3.2.1",
      "publisher": "Apache",
      "name": "commons-collections:commons-collections",
      "purl": "pkg:maven/commons-collections/commons-collections@3.2.1",
      "description": "Types that extend and augment the Java Collections Framework.",
      "licenses": [
        {
          "license": {
            "id": "Apache-2.0",
            "url": "https://www.apache.org/licenses/LICENSE-2.0.txt",
            "name": "Apache License 2.0"
          }
        }
      ],
      "hashes": [
        {
          "alg": "SHA-1",
          "content": "761ea405b9b37ced573d2df0d1e3a4e0f9edc668"
        },
        {
          "alg": "SHA-256",
          "content": "87363a4c94eaabeefd8b930cb059f66b64c9f7d632862f23de3012da7660047b"
        }
      ]
    }
  ]
}
```

Fig. 4. Sample CycloneDX format of SBoM in JSON

CycloneDX can document component types including applications, containers, libraries, files, firmware, frameworks, and operating systems. CycloneDX supported data formats are XML, JSON, and Protobuf. An example of CycloneDX SBoM in JSON format is shown in attached picture.

- The **Standard for software identification (SWID)** was developed by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). SWID is an XML file containing a list of software components and their licenses, patch statuses, and installation bundles.

Important to note is that an SBoM does not necessarily inform you directly if vulnerabilities exist in the components. You'll typically need to use a tool like SCA or take the time to check each component manually. Using an SCA tool is much more efficient as it gets connected with different database (open-source) to get the vulnerabilities.

If your application is using a vulnerable component, it's not necessarily your fault. The vulnerable component may be present because a library that your code is using directly has a dependency on another library. This is called a transitive dependency. Transitive dependencies are pulled in automatically by build systems, aka package managers.

USING THE VULNERABILITY EXPLOITABILITY EXCHANGE (VEX) WITH AN SBOM

VEX, which stands for Vulnerability Exploitability eXchange, is what the US National Telecommunications and Information Administration (NTIA) describes as a “companion artifact” to an SBoM [8]. It is a form of asecurity advisory that indicates whether a product or products are affected by a known vulnerability or vulnerabilities.

While the VEX concept was developed to fill a particular need regarding use of software bills of materials (SBoMs), VEX is not limited to use with SBoMs or necessarily expected to be included in the SBoM itself.

The primary use cases for VEX are to provide users (e.g., operators, developers, and services providers) additional information on whether a product is impacted by a specific vulnerability in an included component and, if affected, whether there are actions recommended to remediate.

To reduce effort spent by users investigating non- exploitable vulnerabilities that don't affect a software product, suppliers can issue a VEX. A VEX is an assertion about the status of a vulnerability in specific products. The status can be:

- Not affected – No remediation is required regarding this vulnerability.
- Affected – Actions are recommended to remediate or address this vulnerability.
- Fixed – Represents that these product versions contain a fix for the vulnerability.
- Under Investigation – It is not yet known whether these product versions are affected by the vulnerability. An update will be provided in a later release

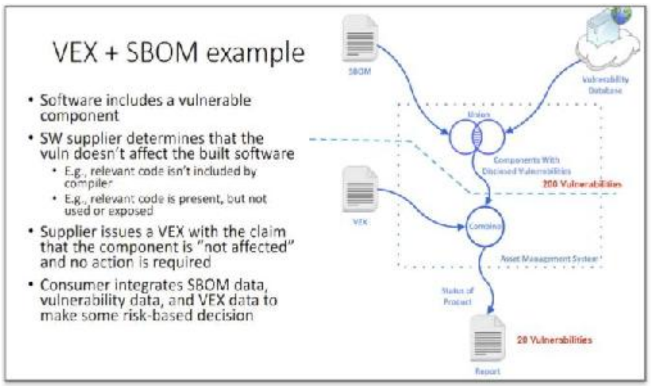
To understand the solution VEX provides, it's important to first understand the problem.

This is best explained with a sample scenario:

Consider a company using software product in their mission critical system. They need to determine if that product exposes the company to any security risks. To do this, the company's team (security or individual team) goes to the software vendor and requests the product's SBoM. With the SBoM in hand, the company can check all the software components and perform an assessment of the vulnerabilities associated with each of these components.

Imagine that the assessment resulted in the discovery of 200 vulnerabilities in the product. Of these vulnerabilities, some are marked as critical in the National Vulnerability Database (NVD). Because of it, the company rushes back to the OEM demanding answers.

The vendor's customer support representative, who is being bombarded by many other customers asking similar questions, emails each impacted customer explaining that while the vulnerabilities can theoretically impact the components, their exploitability depends on the way they were compiled by the software development team. The representative goes on to say that testing and code reviews indicate not one of the critical vulnerabilities affects the product and that only 20 of the low-risk vulnerabilities are exploitable. Relieved, the company decides the two low risk vulnerabilities do not require emergency patching and will wait for the next maintenance shutdown to update the software. Unfortunately, a month later several new critical vulnerabilities are discovered in another subcomponent and the company (along with all the other customers) repeats the cycle.


VEX + SBOM example

- Software includes a vulnerable component
- SW supplier determines that the vuln doesn't affect the built software
  - E.g., relevant code isn't included by compiler
  - E.g., relevant code is present, but not used or exposed
- Supplier issues a VEX with the claim that the component is "not affected" and no action is required
- Consumer integrates SBOM data, vulnerability data, and VEX data to make some risk-based decision

Fig. 5. Diagram courtesy of the National Telecommunications and Information Administration (NTIA) [9]

Now, the above scenario could be considered as successful, however time consuming and non-permanent, way of communicating vulnerability risk and managing vulnerabilities.

The solution could be to empower product vendors to discover vulnerabilities within the third-party or open- source dependencies of their products and preemptively assess the exploitability of these vulnerabilities. Once the assessment is complete, the suppliers can export the results into a standardized, machine-readable VEX document that customers can access and easily interpret.

This is important because not all vulnerabilities merit action. In many cases, a vulnerability may exist in a dependent component, but for the specific product, it either has been mitigated by the vendor development team or is inaccessible to attackers.

WHAT IS SOFTWARE COMPOSITION ANALYSIS (SCA) TOOL?

Software composition analysis (SCA) is an automated process that identifies the open-source software in a codebase. This analysis is performed to evaluate security, license compliance, and code quality.

Companies need to be aware of open-source license limitations and obligations. Tracking these obligations manually became too arduous of a task—and it often overlooked code and its accompanying vulnerabilities. An automated solution, SCA, was developed, and from this initial use case, it expanded to analyze code security and quality.

In a modern DevOps or DevSecOps environment, SCA has galvanized the “shift left” paradigm. Earlier and continuous SCA testing has enabled developers and security teams to drive productivity without compromising security and quality.

Using SCA, software development teams can quickly track and analyze any open-source component brought into a project. SCA tools can discover all related components, their supporting libraries, and their direct and indirect dependencies (transitive dependencies).

SCA tools can also detect software licenses, deprecated dependencies, as well as vulnerabilities and potential exploits. The scanning process generates a SBoM, providing a complete inventory of a project’s software assets.

Incorporating SCA tools in your software development workflow correctly is a significant step toward strengthening the security and integrity of the software supply chain by ensuring any borrowed code doesn’t introduce security risks or legal compliance issues into your products.

It can help ensure that any open-source component embedded in applications meets certain standards, to avoid introducing risks that could result in a data breach, compromised intellectual property, or legal disputes.

To accomplish this, SCA tools can identify specific open- source versions, and correlate any associated security vulnerabilities and licensing information. Advanced SCA tools can automate the entire process, from detection and identification of components to vulnerability or license association and remediation of potential risks.

HOW DOES SOFTWARE COMPOSITION ANALYSIS WORK?

SCA tools inspect package managers, manifest files, source code, binary files, container images, and more. The identified open-source component is compiled into a SBoM, which is then compared against a variety of databases, including the National Vulnerability Database (NVD). These databases hold information regarding known and common vulnerabilities. The NVD is a U.S. government repository of vulnerabilities.

SCA tools can also compare SBoMs against other (usually commercial) databases to discover licenses associated with the code and analyze overall code quality (version control, history of contributions, and so on). By comparing the SBoM against a database, security teams can identify critical security and legal vulnerabilities and act quickly to fix them.

WHY IS SOFTWARE COMPOSITION ANALYSIS IMPORTANT?

According to Gartner’s research report on Software Composition Analysis, security came up as a top challenge when working with open source, with 57% of participants rating vulnerabilities as a significant challenge when working with open source.

In addition, over two-thirds of their survey respondents said that they are concerned over the long term viability of open source projects. Next in the list of top challenges were deciding when to seek out commercial support.

Another challenge that the Gartner report highlights is the complex issue of open-source licensing compliance.

Fig. 6. Significant Challenges with Open Source Software

The open-source licenses attached to each, and every open- source component vary from permissive to very restrictive. Organizations need to make sure that they are compliant with the terms and conditions of all the open-source licenses included in their codebase.

CHOOSING AND USING SOFTWARE COMPOSITION ANALYSIS TOOLS

As awareness of the risks of open-source usage grows, a variety of different SCA solutions have been introduced to the market, and vendors continue to develop and innovate their SCA offerings.

- **Comprehensive Vulnerabilities Coverage**

Most SCA tools rely on a database of known open-source vulnerabilities based solely on the National Vulnerability Database (NVD). While the NVD is the largest security vulnerabilities database, it is not the only one out there. There are additional community issue trackers and advisories that publish open-source vulnerabilities that sometimes aren't included in the NVD.

While the detection of security issues is certainly important, it is only the first step in addressing the issue. Some SCA tools also offer remediation support, by offering a recommended version to update, or a patch.

- **Open-Source License Compliance**

SCA tools offer the ability to track and report on the open- source licenses attached to the components, helping organizations remain compliant and avoid open-source licensing risks. Organization should have an open-source policy which can state that which open-source licenses are accepted, which are to be rejected and which ones require to be reviewed by an internal team.

- **Reporting Capabilities and BoM Support**

As per Gartner's report, in the next five years the provision of an up-to-date, comprehensive, and detailed Bill of Materials will become a standard contractual requirement for both buyers and vendors in the next five years. While currently this practice is reserved for regulated industries, the increased reliance on open source and third-party components throughout all the software development industry demands this type of document. This requirement will make SCA tools a must-have for all organizations, as the ability to produce a detailed inventory report covering all aspects of the open-source components included in a project, including licensing and copyright, versioning, and any other relevant information, manually is almost an impossible mission.

- **Developer Support**

Software developers are the ones choosing open-source components, integrating them to the product and updating it in case an issue arises. Therefore, Gartner's recommendation is to choose an open-source management tool that can integrate with development environments, like IDE's and repositories, along with support for remediation processes. Advanced SCA solutions for developers also offer automated workflows that include issue tracking systems and browser support for assessing open-source components even before they are added to the code.

- **Speeding Up Response Time**

Beyond quick detection of vulnerabilities, advanced SCA tools can also accelerate the remediation process by automatically detecting newly published open-source vulnerabilities, including direct and transitive dependencies.

A new capability of prioritizing detected vulnerabilities by analyzing whether a vulnerable method is being used in the application or not. This enables teams to invest their time wisely, remediating the issues that really matter. The right SCA tools can save organizations a lot of the valuable time and money currently spent on slow and human error-prone manual processes.

REFERENCES

[1] https://www.gartner.com/en/newsroom/press-releases/2022-03-07- gartner-identifies-top-security-and-risk-management-trends-for-2022
[2] https://www.ntia.gov/SBOM
[3] Executive Order on Improving the Nation's Cybersecurity | The White House
[4] solarwinds_report_2021.pdf (ny.gov)
[5] https://www.cisecurity.org/log4j-zero-day-vulnerability-response
[6] SBOM at a Glance (ntia.gov)
[7] NVD - Search and Statistics (nist.gov) [8] VEX one-page summary (ntia.gov)
[9] Framing (doc.gov)
[10] https://www.gartner.com/reviews/market/software-composition- analysis-sca
[11] https://linuxfoundation.org/wp- content/uploads/LFResearch_SBOM_Report_final.pdf
[12] https://www.synopsys.com/blogs/software-security/software-bill-of- materials-bom/
[13] https://github.com/spdx/spdx- examples/blob/master/example1/spdx/example1.spdx
[14] https://github.com/sonatype-nexus-community/cyclonedx-sbom- examples/blob/master/package.json
[15] https://www.whitesourcesoftware.com/resources/blog/gartners- software-composition-analysis-report-key-take-aways/