# **Analyzing GitHub SPDX SBOMs with Trivy and VEX**

## **1. Introduction**

### **1.1. Context**

The security of the software supply chain has become a paramount concern in modern software development. Organizations increasingly rely on third-party and open-source components, introducing potential risks if these dependencies are not properly managed and vetted. Software Bill of Materials (SBOMs) have emerged as foundational artifacts for achieving transparency into software composition, providing a detailed inventory of components and dependencies.1 This transparency is crucial for identifying vulnerabilities, ensuring license compliance, and managing overall software risk.3 However, simply listing components is insufficient; effective security requires analyzing this inventory for known vulnerabilities and understanding their actual impact. Vulnerability Exploitability eXchange (VEX) provides a standardized mechanism to communicate whether a known vulnerability in a component is actually exploitable within the context of a specific product, thereby adding crucial context to raw vulnerability data and enabling more focused remediation efforts.5

### **1.2. Objective**

This report provides an expert analysis of leveraging the Trivy security scanner to analyze SBOMs exported from GitHub repositories. The focus is on establishing practical workflows for vulnerability assessment (enhanced with VEX), license compliance analysis (including considerations for commercial use restrictions), and evaluating the feasibility of secret detection using SBOMs. It aims to clarify the capabilities and limitations of using Trivy with GitHub's standard SPDX-formatted SBOMs.

### **1.3. Key Questions Addressed**

This analysis directly addresses the core questions surrounding the integration of GitHub SBOMs and Trivy:

* What is an effective and straightforward method for analyzing a GitHub-exported SBOM using Trivy?
* Can Trivy ingest GitHub's SPDX SBOM format?
* How can Trivy perform vulnerability assessments on the SBOM, and how does VEX integration work, particularly with SPDX SBOMs?
* What are Trivy's capabilities regarding dependency analysis based on the SBOM?
* How does Trivy handle license scanning from an SBOM, specifically concerning open-source licenses and potential commercial use restrictions?
* Is it possible to use Trivy to detect secrets by scanning an SBOM?

### **1.4. Methodology**

The findings and recommendations presented in this report are based on the synthesis of information from official documentation for GitHub, Trivy, SPDX, and VEX, supplemented by technical articles, community discussions, and established best practices in software supply chain security and vulnerability management.

## **2. Understanding GitHub SBOM Exports (SPDX Format)**

### **2.1. Generation and Format**

GitHub facilitates software supply chain transparency by allowing users to export the dependency graph of a repository as an SBOM.1 This export functionality generates the SBOM in the Software Package Data Exchange (SPDX) format, a widely recognized, industry-standard specification maintained by the Linux Foundation and published as ISO/IEC 5962:2021.7 Exports can be triggered either through the GitHub user interface or programmatically via the GitHub REST API.1 The adoption of SPDX is significant because it promotes interoperability, allowing the SBOM generated by GitHub to be consumed by various downstream analysis tools that adhere to the standard.2

### **2.2. Content and Scope**

The SBOM exported from GitHub provides a formal, machine-readable inventory of a project's dependencies and associated metadata.1 Key data points included are:

* **Component Identification**: Designation assigned to a unit of software (Component Name), Version String, Unique Identifier (e.g., Package URL - PURL), Supplier Name.1 PURLs are particularly important for enabling vulnerability lookups across databases.10
* **Dependency Information**: The SBOM lists the ecosystems and packages the repository depends on (direct dependencies).1 For many common package ecosystems, GitHub's dependency graph also identifies transitive dependencies (dependencies of dependencies) and includes their relationship paths within the SBOM export.1 Supported ecosystems for transitive dependency information include Cargo, Composer, Go modules, Maven, npm, NuGet, pip (with lock files), pnpm, Python Poetry, RubyGems, Swift Package Manager, and Yarn.1
* **License Information**: Details regarding the declared licenses of each dependency are included in the SPDX export.1 This aligns with SPDX's origins in license tracking.8
* **Other Metadata**: Information such as copyright details, author of the SBOM (GitHub), and timestamp of generation are also typically present, satisfying minimum requirements outlined by standards bodies like NTIA.1

### **2.3. Data Source and Limitations**

It is crucial to understand that the GitHub-exported SBOM is derived directly from the repository's dependency graph.1 The dependency graph, in turn, is constructed by parsing manifest files (e.g., package.json, pom.xml, requirements.txt) and lock files (e.g., package-lock.json, Cargo.lock) found within the repository, supplemented by any dependencies submitted via the Dependency Submission API.1

This reliance on the dependency graph introduces potential limitations:

* **Accuracy**: The accuracy and completeness of the SBOM are contingent upon the correctness of the manifest and lock files within the repository.1 Errors or omissions in these files will directly translate into inaccuracies in the generated SBOM. For example, if a developer manually vendors a library without declaring it in a manifest, it may not appear in the SBOM.
* **Completeness**: While GitHub supports transitive dependency tracking for many ecosystems, this support is not universal.1 For ecosystems where transitive dependencies are not fully resolved by the dependency graph, the resulting SBOM will be incomplete.
* **Scope**: The GitHub SBOM export explicitly focuses on the *dependencies* of the repository. It does *not* include information about *dependents*—other repositories or packages that rely on the repository being analyzed.1

### **2.4. Implications for Analysis**

The nature of GitHub's SBOM export carries significant implications for subsequent analysis. While providing a standardized SPDX document is a valuable starting point, it represents the *detected* state of dependencies based on the dependency graph, not necessarily the absolute ground truth of every component involved in the software's build or runtime.1 Users must consider the potential for incompleteness stemming from inaccurate manifest files or limitations in transitive dependency resolution for certain ecosystems.1 This means that relying solely on the GitHub-exported SBOM without considering these factors might lead to an underestimation of the actual attack surface or license compliance scope.

However, GitHub's choice of the SPDX format is a major advantage.1 As an open, widely adopted standard 2, SPDX ensures that the generated SBOM can be readily ingested and processed by a broad ecosystem of security and analysis tools, including Trivy.11 This standardization simplifies the initial step of integrating SBOM analysis into security workflows.

## **3. Leveraging Trivy for SBOM Analysis**

### **3.1. Trivy Overview**

Trivy is a popular open-source, comprehensive security scanner developed by Aqua Security. Its capabilities extend beyond vulnerability scanning to include detection of misconfigurations, secrets (in specific contexts), and software licenses.13

### **3.2. Ingesting GitHub SPDX SBOMs**

Trivy is explicitly designed to analyze SBOMs. It provides the sbom subcommand specifically for this purpose, allowing users to point Trivy at an SBOM file for analysis.11 The basic command structure is $ trivy sbom /path/to/sbom\_file.11

Crucially for this workflow, Trivy supports the ingestion of SBOMs in both SPDX tag-value (--format spdx) and SPDX JSON (--format spdx-json) formats.11 Since GitHub exports SBOMs in SPDX 1, Trivy can directly consume these files. Trivy automatically detects the input format, simplifying the command usage.11

### **3.3. Vulnerability Scanning Workflow**

Vulnerability scanning is the default operation when using the trivy sbom command.11 The process involves Trivy parsing the component information (package names, versions, identifiers like PURLs) from the provided SPDX SBOM.1 It then queries its vulnerability intelligence sources (which include NVD, OSV, GitHub Advisories, etc., when scanning directly 17) to identify known vulnerabilities associated with the detected components and versions.

The results are typically presented in a table format, listing the vulnerable package, vulnerability ID (e.g., CVE), severity, installed version, fixed version (if available), and a brief description.12

### **3.4. License Scanning Workflow**

Trivy can also perform license analysis on the components listed in an SBOM. This capability is not enabled by default for the sbom subcommand but can be activated using the --scanners flag:

* --scanners license: Performs only license scanning.
* --scanners vuln,license: Performs both vulnerability and license scanning.11

When invoked, Trivy extracts license information embedded within the SPDX SBOM. SPDX provides standard fields for this, such as licenseConcluded (the license determined to apply) and licenseDeclared (the license stated by the component author).11 Trivy analyzes this extracted license information and classifies each license according to the Google License Classification framework.20

This classification categorizes licenses into types like Forbidden, Restricted, Reciprocal, Notice, Permissive, Unencumbered, and Unknown. Trivy maps these classifications to severity levels, providing a quick risk assessment 20:

| **Classification** | **Severity** | **Implication** |
| --- | --- | --- |
| Forbidden | CRITICAL | High risk, likely incompatible with commercial use |
| Restricted | HIGH | Significant obligations or restrictions |
| Reciprocal | MEDIUM | Often requires source code sharing (Copyleft) |
| Notice | LOW | Typically requires attribution |
| Permissive | LOW | Minimal restrictions |
| Unencumbered | LOW | Public domain or equivalent |
| Unknown | UNKNOWN | License not recognized, requires manual review |

*20*

The classifications "Forbidden" (Critical) and "Restricted" (High) serve as strong indicators of potential conflicts with commercial use or distribution, or the presence of significant obligations (like strong copyleft requirements).21 However, it is essential to understand that Trivy reports the *classification* based on the identified license's type, not a detailed analysis of the *specific legal terms* within the license text.21 Therefore, while Trivy's license scan provides valuable signals for risk management and compliance checks, it does not replace the need for legal counsel to interpret specific license obligations and their impact on commercial usage.

### **3.5. Secret Scanning Limitations**

The user query asks about secret detection using Trivy on an SBOM. However, **Trivy cannot perform secret scanning by analyzing an SBOM file.** Trivy's secret scanning functionality is designed to analyze the *content* of files within container images, filesystems, or Git repositories.23 It searches for patterns matching known secret formats (API keys, passwords, tokens) within actual code, configuration files, or other artifacts.23

SBOMs, by their nature, provide metadata about software components—names, versions, licenses, dependencies—they do not contain the raw source code or configuration file content where secrets typically reside.10 Consequently, the trivy sbom command, which operates solely on the SBOM data, lacks the necessary input (file content) to perform secret detection.14 Secret scanning must be conducted as a separate step in the security workflow, targeting the source code repository or built artifacts directly using commands like trivy repo, trivy fs, or trivy image.23

### **3.6. Accuracy Concerns: The "Custom Properties" Issue**

Trivy's documentation consistently includes a significant caveat regarding SBOM scanning: "Passing SBOMs generated by tool other than Trivy may result in inaccurate detection because Trivy relies on custom properties in SBOM for accurate scanning".11

This warning has direct implications for scanning GitHub-generated SPDX SBOMs, as these SBOMs will not contain Trivy's specific custom properties. The potential impact of scanning such an "external" SBOM includes 11:

* **Incomplete Component Detection**: Trivy might fail to recognize or parse all components listed if it expects specific non-standard fields or structures.
* **Inaccurate Version Identification**: Vulnerability matching relies heavily on precise version identification. If version information is not presented in the way Trivy expects (potentially relying on custom fields), matching accuracy could suffer.
* **Missed Licenses**: Similarly, if Trivy's license detection leverages custom properties beyond standard SPDX fields, licenses might be missed or misidentified.

Unfortunately, the Trivy documentation does not specify *what* these custom properties are, making it difficult to predict the exact degree of inaccuracy or to manually adapt external SBOMs.11 Users should be aware that while Trivy *can* scan GitHub's SPDX SBOMs, the results for both vulnerabilities and licenses might be less accurate or complete compared to scanning an SBOM generated by Trivy itself or scanning the source artifact directly.

### **3.7. Implications for Analysis and Workflow Design**

The capabilities and limitations of Trivy when scanning GitHub SBOMs lead to several important considerations for workflow design:

* **Ease vs. Accuracy Trade-off**: The simplest workflow involves directly feeding the GitHub-exported SPDX SBOM into trivy sbom.1 This is convenient and leverages readily available artifacts. However, this convenience comes at the cost of potential inaccuracies due to both the inherent limitations of the GitHub dependency graph data (potential incompleteness) and Trivy's preference for its own custom properties in SBOMs.11 Achieving higher accuracy might require generating the SBOM directly from the source code or container image using Trivy itself, adding a step to the workflow but potentially yielding more reliable results.
* **License Scanning as a Risk Indicator**: Trivy's license classification provides a valuable first-pass risk assessment, effectively flagging licenses often associated with commercial use restrictions (e.g., 'Forbidden', 'Restricted') or significant obligations (e.g., 'Reciprocal').21 This helps prioritize components needing further scrutiny. However, because Trivy doesn't parse the full license text or provide legal interpretation 21, these classifications must be treated as signals prompting deeper investigation and consultation with legal experts, not as definitive compliance checks.
* **Secret Scanning as a Separate Workflow**: It's clear that secret scanning is fundamentally different from SBOM-based vulnerability and license analysis. It requires direct content analysis.23 Therefore, any workflow aiming to detect secrets must include a separate step using Trivy (or another tool) to scan the actual source code repository or built artifacts, independent of the SBOM analysis.26 Attempting to achieve secret detection via trivy sbom is not feasible.

To clarify Trivy's capabilities when processing an SBOM, the following table summarizes the different scanner types:

**Table 1: Trivy Scanner Capabilities on SBOM Input**

| **Scanner Type** | **Supported via trivy sbom?** | **Required Flags** | **Key Output/Findings** |
| --- | --- | --- | --- |
| Vulnerability | Yes (Default) | *None* | Known vulnerabilities (CVEs, etc.) in listed components |
| License | Yes | --scanners license or --scanners vuln,license | License IDs, Classifications (Risk Severity) |
| Secret | **No** | N/A | N/A (Requires direct file/image/repo scan) |
| Misconfiguration | **No** | N/A | N/A (Requires direct config file/IaC/image scan) |

*11*

## **4. Integrating VEX with Trivy and GitHub SBOMs**

### **4.1. Overview of VEX**

Vulnerability Exploitability eXchange (VEX) is a crucial standard designed to augment SBOMs by providing context about the exploitability of known vulnerabilities.5 It acts as a form of security advisory, typically machine-readable, that communicates whether a specific product is actually affected by a vulnerability present in one of its components.6 The primary goal of VEX is to reduce the noise generated by vulnerability scanners by filtering out findings that are not exploitable in a particular context, thus allowing teams to prioritize remediation efforts on genuine risks.6

VEX documents typically contain statements linking a product, a vulnerability (e.g., a CVE ID), and a status. Common statuses include 5:

* affected: The product is impacted by the vulnerability. An action statement (e.g., patch instructions) is usually required.5
* not\_affected: The product is not impacted by the vulnerability, even though the vulnerable component is present. A justification is required.5
* fixed: The vulnerability was present but has been remediated in the specified product version.
* under\_investigation: The product's status regarding the vulnerability is currently being analyzed.

For the not\_affected status, VEX defines several standard justifications to explain *why* the product isn't impacted, such as 34:

* component\_not\_present: The vulnerable component is not included in the product (less common when VEX accompanies an SBOM listing the component).
* vulnerable\_code\_not\_present: A specific vulnerable part of the component is not included (e.g., tree-shaking removed it).
* vulnerable\_code\_not\_in\_execute\_path: The vulnerable code exists but is never called or executed in the product's specific implementation.
* vulnerable\_code\_cannot\_be\_controlled\_by\_adversary: The vulnerable code is used, but requires privileges or conditions unattainable by an attacker.
* inline\_mitigations\_already\_exist: The product has its own built-in protections that negate the vulnerability.

### **4.2. Trivy's VEX Support**

Trivy incorporates support for VEX, allowing it to consume VEX documents and filter its vulnerability scan results accordingly.31 When a VEX document indicates that a specific vulnerability is not\_affected or fixed for a component within the scanned target, Trivy will typically suppress that vulnerability finding from its main report.31

Trivy supports multiple VEX formats 31:

* CycloneDX VEX (JSON format, requires input SBOM to also be CycloneDX)
* OpenVEX (JSON format)
* CSAF (Common Security Advisory Framework) (JSON format)

A critical point for the user's scenario involves format compatibility. **When scanning an SPDX SBOM (such as one exported from GitHub), Trivy requires the accompanying VEX document to be in either OpenVEX or CSAF format**.31 Trivy *cannot* use a CycloneDX VEX document to filter results if the input SBOM is SPDX; CycloneDX VEX is only supported when the input SBOM itself is also in CycloneDX format.31

To provide the VEX document during a scan, the --vex flag is used, followed by the path to the VEX file (e.g., $ trivy sbom --vex /path/to/vex.openvex.json /path/to/sbom.spdx.json).31 Trivy also supports dynamically retrieving VEX documents referenced within a CycloneDX SBOM using --vex sbom-ref, but this is less relevant when starting with an external SPDX SBOM.42

The compatibility requirements are summarized below:

**Table 2: Trivy VEX Format Support & Requirements**

| **VEX Format** | **Required Input SBOM Format(s)** | **Trivy Support (--vex flag)** | **Key Considerations** |
| --- | --- | --- | --- |
| CycloneDX VEX | CycloneDX only | Yes | Cannot be used with SPDX SBOMs (like GitHub's) |
| OpenVEX | CycloneDX or SPDX | Yes | **Suitable for use with GitHub SPDX SBOMs** |
| CSAF | CycloneDX or SPDX | Yes | **Suitable for use with GitHub SPDX SBOMs** |

*31*

### **4.3. Workflow: GitHub SPDX SBOM + Trivy + OpenVEX/CSAF**

Based on the compatibility requirements, the practical workflow for using Trivy with a GitHub SPDX SBOM and VEX is as follows:

1. **Export SPDX SBOM from GitHub**: Obtain the sbom.spdx.json (or tag-value) file for the target repository.1
2. **Perform Initial Vulnerability Scan (Optional but Recommended)**: Run Trivy on the SBOM without VEX (trivy sbom /path/to/sbom.spdx.json) to get an initial list of vulnerabilities.
3. **Create VEX Document (OpenVEX or CSAF)**: This is often the most challenging step. Based on the initial scan results and further analysis (see section 4.4), create a VEX document in either OpenVEX or CSAF format. This document will list vulnerabilities identified in step 2 and assign a status (affected, not\_affected, fixed, under\_investigation) and justification (for not\_affected) for each one in the context of the specific product.5 Tools like vexctl can help structure OpenVEX documents.43
4. **Scan with Trivy and VEX**: Execute Trivy again, this time providing both the SPDX SBOM and the created VEX document: $ trivy sbom --vex /path/to/vex.(openvex|csaf).json /path/to/sbom.spdx.json.
5. **Analyze Filtered Results**: Review Trivy's output. Vulnerabilities marked as not\_affected or fixed in the VEX file should now be suppressed, allowing focus on the remaining affected or under\_investigation issues.31

### **4.4. VEX Creation and Justification Challenges**

Generating accurate VEX statements, particularly the justifications for not\_affected status, is non-trivial.34 Determining justifications like vulnerable\_code\_not\_in\_execute\_path often requires analysis beyond what standard SBOM generation or vulnerability scanning tools provide.34

Methods and tools to assist VEX creation include:

* **Manual Analysis**: Code review to determine if vulnerable functions are called.
* **Static Application Security Testing (SAST)**: Some advanced SAST tools can perform call graph analysis to determine code reachability, providing evidence for vulnerable\_code\_not\_in\_execute\_path justifications.34
* **VEX Generation Tools**: Tools like vexctl help create syntactically correct OpenVEX documents based on user input.43 GitHub Actions like openvex/generate-vex can automate parts of the generation from templates.46
* **Commercial Platforms**: Platforms like Sonatype SBOM Manager 35, Endor Labs 34, Mend 33, or Anchore Enterprise 48 may offer features to streamline VEX creation, sometimes incorporating automated reachability analysis.34
* **Guidance**: CISA provides use cases and guidance on minimum data elements for VEX documents.5

VEX documents are not static. As software evolves, vulnerabilities are patched, or analysis provides new insights, VEX documents must be updated to reflect the current state. This requires establishing processes for VEX document lifecycle management.6

### **4.5. Implications of Using VEX**

Integrating VEX significantly enhances the utility of SBOM vulnerability scanning, but introduces trade-offs:

* **Increased Actionability vs. Added Complexity**: VEX dramatically improves the signal-to-noise ratio of vulnerability reports, allowing teams to focus on real threats.6 However, the process of generating accurate VEX statements, especially those requiring reachability analysis, adds considerable complexity and effort compared to simply scanning an SBOM.34 It often involves manual investigation or integrating additional, potentially sophisticated tools like SAST.34 This makes the overall workflow less "easy," even though the final output is more valuable.
* **Criticality of Format Interoperability**: The requirement to use OpenVEX or CSAF VEX with SPDX SBOMs in Trivy underscores the importance of understanding and managing format compatibility.31 Teams cannot simply generate a VEX document in any format; the choice is constrained by the input SBOM format (SPDX from GitHub) and the capabilities of the consuming tool (Trivy). This necessitates careful planning and tool selection to ensure the VEX data can actually be utilized in the intended workflow.

## **5. Specific Analysis Areas: Deeper Dive**

### **5.1. Dependency Analysis**

The GitHub-exported SPDX SBOM serves as the primary input for dependency analysis in this workflow.1 It provides Trivy with the list of components (direct and, where available, transitive) and their versions.1 Trivy uses this inventory to query vulnerability databases and check license information.11 The effectiveness of this analysis is directly tied to the quality of the input SBOM. As discussed (Section 2.3, 2.4), inaccuracies or incompleteness in the GitHub dependency graph due to parsing errors or limited ecosystem support will propagate into Trivy's analysis, potentially leading to missed vulnerabilities or licenses.1

### **5.2. License Compliance Analysis**

Trivy's license analysis, triggered via --scanners license, leverages the license information present in the SPDX SBOM.11 Its core contribution is the classification of identified licenses (e.g., MIT, Apache-2.0, GPL-3.0) into risk categories based on the Google License Classification framework.21

This classification is particularly useful for flagging licenses that often carry implications for commercial use or impose strong obligations:

* **Forbidden/Critical**: Licenses in this category (e.g., AGPL) often have terms incompatible with proprietary commercial distribution or require extensive source code disclosure, posing significant compliance risks.20
* **Restricted/High**: These licenses might have patent clauses, specific attribution requirements, or other conditions that need careful review in a commercial context.20
* **Reciprocal/Medium**: This typically includes "copyleft" licenses (e.g., GPL, MPL) that require derivative works or modifications to be licensed under similar terms, impacting proprietary code integration.20

While Trivy effectively signals these potential risks based on license type, it's crucial to reiterate its limitation: **Trivy does not interpret the specific legal text of the licenses**.21 It identifies "GPL-3.0" and classifies it as "Reciprocal" (Medium risk), but it doesn't tell the user the exact obligations imposed by GPL-3.0 regarding distribution, modification, or commercial use.4 Therefore, Trivy's output should trigger a legal review process, not replace it. Tools like Dependency-Track offer more advanced policy engines where organizations can define granular rules based on license groups (e.g., "Forbidden Commercial Use Licenses") for more automated compliance checks.53

### **5.3. Secret Detection (Reiteration of Limitation)**

As established previously (Section 3.5), analyzing an SBOM is not a viable method for detecting secrets like API keys or passwords accidentally committed to source code.10 SBOMs lack the necessary file content for secret scanning tools to analyze.26 Trivy's sbom subcommand exclusively processes the SBOM metadata for vulnerability and license information.11 Therefore, secret detection must be performed as a separate, parallel process within the CI/CD pipeline, using Trivy's repo, fs, or image scanning modes directly on the codebase or built artifacts.23

## **6. Alternative Tools and Approaches**

While Trivy offers a versatile solution, the software composition analysis (SCA) and SBOM ecosystem includes several other notable open-source and commercial tools, each with different strengths.54

* **Grype (Anchore)**: Often used in conjunction with Anchore's Syft SBOM generator, Grype is a dedicated vulnerability scanner optimized for speed when scanning SBOMs.57 It supports SPDX and CycloneDX SBOM inputs and can consume OpenVEX documents for filtering results (--vex flag).40 It reports license information found in SBOMs, but details on specific commercial restriction reporting are limited in available documentation.40 Grype does not appear to perform secret scanning on SBOMs.40 The Syft+Grype combination represents a popular, focused open-source workflow for SBOM generation and vulnerability scanning.55
* **Dependency-Track (OWASP)**: Positioned as an SBOM analysis *platform*, Dependency-Track excels at continuous monitoring and portfolio-level risk management.17 It ingests SBOMs (primarily CycloneDX, though SPDX import might be possible via API or conversion 18) and CycloneDX VEX documents.17 Its key strength lies in its robust policy engine, allowing granular control over security, operational, and license compliance rules (including defining allowed/forbidden license groups).17 It aggregates vulnerability data from multiple sources.18 It does not perform secret scanning.62
* **Snyk**: Snyk provides integrated tooling (CLI, API, UI) for vulnerability scanning and license compliance management, often tightly integrated into developer workflows.19 Snyk CLI can generate SBOMs (SPDX, CycloneDX) including license data 19 and test existing SBOMs (SPDX JSON, CycloneDX JSON) for vulnerabilities.64 Details on how Snyk consumes external VEX documents during SBOM testing are not clearly specified in the reviewed documentation.65 Snyk does not appear to perform secret scanning *via* its SBOM testing command.66 Snyk also offers a separate tool, parlay, for enriching existing SBOMs with data like licenses or Snyk vulnerability information.19
* **Other Commercial Tools**: Numerous commercial platforms offer advanced SCA and SBOM management features. Examples include FOSSA (strong focus on license compliance) 54, Mend.io (formerly WhiteSource) 56, Anchore Enterprise (platform built around Syft/Grype) 56, Sonatype (Nexus Lifecycle, SBOM Manager with VEX support) 35, and Endor Labs (focus on dependency selection and reachability analysis for VEX).34 These often provide enterprise features like advanced policy management, reporting, and integrations.

The choice between these tools often depends on specific organizational priorities. Trivy offers broad scanning capabilities (vuln, misconfig, license, secret) within a single, easy-to-deploy binary. Grype+Syft provide a focused, performant open-source path for SBOM generation and vulnerability scanning. Dependency-Track is ideal for centralized, continuous monitoring and policy enforcement across a portfolio. Snyk emphasizes developer workflow integration. Commercial tools typically offer enhanced features, support, and potentially more sophisticated analysis (like automated reachability for VEX) at a cost.

## **7. Conclusion and Recommendations**

### **7.1. Summary of Workflow**

The analysis indicates that a practical workflow for using Trivy to analyze GitHub-exported SPDX SBOMs, incorporating VEX and addressing the user's specific concerns, involves these steps:

1. **Export SPDX SBOM**: Obtain the SPDX SBOM (JSON or tag-value) from the GitHub repository UI or API.
2. **Generate VEX (Optional but Recommended)**: Perform an initial vulnerability assessment (e.g., using trivy sbom <sbom\_file>). Based on this and further analysis (potentially including manual review or SAST for reachability), create a VEX document in **OpenVEX** or **CSAF** format detailing the status and justification for vulnerabilities. Tools like vexctl can aid creation.
3. **Vulnerability Scan with VEX**: Run Trivy providing both the SPDX SBOM and the OpenVEX/CSAF VEX file: $ trivy sbom --vex <vex\_file> <sbom\_file>. Analyze the filtered vulnerability results.
4. **License Scan**: Run Trivy specifically for license analysis: $ trivy sbom --scanners license <sbom\_file>. Review the classified licenses, paying attention to high-risk categories (Forbidden, Restricted, Reciprocal) as indicators for further legal review regarding commercial use and obligations.
5. **Secret Scan (Separate Process)**: Implement a separate step in the CI/CD pipeline to scan the source code repository or built artifact directly for secrets using Trivy (e.g., $ trivy repo <repo\_url> or $ trivy image <image\_name>). **Do not attempt secret scanning via the SBOM.**

### **7.2. Addressing "Best or Easiest"**

The "easiest" approach is to directly scan the GitHub SPDX SBOM with Trivy (trivy sbom <sbom\_file>) for vulnerabilities (default) and licenses (--scanners license). This leverages readily available artifacts and Trivy's direct support for SPDX.

However, achieving the "best" results—meaning the most accurate and actionable findings—requires addressing several complexities:

* **Accuracy**: The potential inaccuracies stemming from GitHub's dependency graph limitations and Trivy's preference for its own SBOM properties mean the "easy" path might not be the most reliable.
* **Actionability**: Raw vulnerability lists can be noisy. Incorporating VEX significantly improves actionability but adds the non-trivial step of VEX document creation and management.
* **Completeness**: Secret scanning is entirely outside the scope of SBOM analysis and requires a separate process.

Therefore, a balance must be struck. The recommended workflow above represents a practical compromise, enhancing the "easy" path with VEX for better actionability while acknowledging accuracy caveats and the necessity of separate secret scanning.

### **7.3. Key Recommendations**

Based on this analysis, the following recommendations are provided:

1. **Verify Trivy Accuracy on External SBOMs**: Be cognizant of the potential for reduced accuracy when Trivy scans SBOMs generated by external tools like GitHub.11 For critical applications, consider periodically validating Trivy's findings by comparing them against results from scanning the source artifact directly with Trivy or using Trivy to generate the SBOM itself.
2. **Implement VEX for Prioritization**: Strongly recommend adopting VEX to filter vulnerability noise and focus remediation efforts.6 Use **OpenVEX** or **CSAF** formats when working with GitHub's SPDX SBOMs and Trivy.31 Start by documenting basic statuses (affected, not\_affected, fixed) and incrementally add justifications, particularly for not\_affected. Leverage tools like vexctl 43 or investigate platforms with integrated VEX generation capabilities. Acknowledge that justifying not\_affected based on reachability may require manual analysis or SAST integration.34
3. **Understand License Reporting Scope**: Utilize Trivy's license classifications (Forbidden, Restricted, etc.) as valuable risk indicators for potential commercial use conflicts or compliance issues.21 However, recognize that this is not a substitute for legal analysis. Use the identified license (e.g., GPL-3.0) as input for a formal legal review to understand specific obligations.21
4. **Implement Separate Secret Scanning**: Ensure that secret detection is handled as a distinct CI/CD pipeline step, scanning the actual source code repository or built artifacts using tools like Trivy (trivy repo, trivy fs, trivy image).23 Do not rely on SBOM analysis for finding secrets.
5. **Evaluate Alternative Tools**: Depending on specific needs and existing infrastructure, evaluate alternative tools. Consider Grype+Syft for a focused open-source vulnerability workflow 58, Dependency-Track for centralized portfolio management and policy enforcement 17, Snyk for deep developer workflow integration 19, or commercial tools for enterprise features and support.56
6. **Automate the Workflow**: Integrate the chosen steps—SBOM export/generation, VEX creation/update (where feasible), Trivy scanning (vuln, license), and separate secret scanning—into automated CI/CD pipelines to ensure continuous analysis and timely feedback.48 Store SBOMs and VEX documents alongside build artifacts for traceability and compliance.71

### **7.4. Final Thoughts**

Securing the software supply chain through SBOM analysis is an evolving discipline. Tools like GitHub SBOM exports and Trivy provide powerful capabilities, but understanding their interactions, limitations, and the nuances of standards like SPDX and VEX is essential for effective implementation. Achieving robust security requires a combination of automated tooling, awareness of potential inaccuracies, integration of contextual information like VEX, and a commitment to continuous monitoring and process improvement.

#### Works cited

1. Exporting a software bill of materials for your repository - GitHub Docs, accessed May 12, 2025, <https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/exporting-a-software-bill-of-materials-for-your-repository>
2. What is a Software Bill of Materials (SBOM)? Best Practices - JFrog, accessed May 12, 2025, <https://jfrog.com/learn/sdlc/sbom/>
3. What is an SBOM? SBOM explained in 5 minutes - Legit Security, accessed May 12, 2025, <https://www.legitsecurity.com/blog/what-is-an-sbom-sbom-explained-in-5-minutes>
4. SCA and SBOM: What's the Difference? - Scribe Security, accessed May 12, 2025, <https://scribesecurity.com/blog/sca-and-sbom-what-is-the-difference/>
5. Vulnerability Exploitability eXchange (VEX) – Use Cases - CISA, accessed May 12, 2025, <https://www.cisa.gov/sites/default/files/2023-01/VEX_Use_Cases_Aprill2022.pdf>
6. Vulnerability Exploitability eXchange (VEX) - Revenera, accessed May 12, 2025, <https://www.revenera.com/software-composition-analysis/glossary/vulnerability-exploitability-exchange-vex>
7. github.com, accessed May 12, 2025, <https://github.com/github/docs/blob/main/content/code-security/supply-chain-security/understanding-your-software-supply-chain/exporting-a-software-bill-of-materials-for-your-repository.md#:~:text=About%20the%20dependency%20graph%20and%20SBOM%20exports&text=You%20can%20export%20the%20current,Via%20the%20%7B%25%20data%20variables.>
8. SBOM Standards and Formats: A Guide - Legit Security, accessed May 12, 2025, <https://www.legitsecurity.com/aspm-knowledge-base/sbom-standards-and-formats>
9. using/docs/using-SPDX-to-comply-with-industry-guidance.md at main - GitHub, accessed May 12, 2025, <https://github.com/spdx/using/blob/main/docs/using-SPDX-to-comply-with-industry-guidance.md>
10. SBOM from the security perspective - Cossack Labs, accessed May 12, 2025, <https://www.cossacklabs.com/blog/sbom-from-the-security-perspective/>
11. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/dev/docs/target/sbom/>
12. SBOM - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/target/sbom/>
13. Trivy SBOM - secureCodeBox, accessed May 12, 2025, <https://www.securecodebox.io/docs/scanners/trivy-sbom/>
14. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.43/docs/target/sbom/>
15. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.58/docs/target/sbom/>
16. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.52/docs/target/sbom/>
17. Dependency-Track | Software Bill of Materials (SBOM) Analysis | OWASP, accessed May 12, 2025, <https://dependencytrack.org/>
18. OWASP Dependency-Track, accessed May 12, 2025, <https://owasp.org/www-project-dependency-track/>
19. Include License Info In Your SBOMs with Snyk, accessed May 12, 2025, <https://snyk.io/blog/include-license-info-in-your-sboms/>
20. License Scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.33/docs/licenses/scanning/>
21. License - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/scanner/license/>
22. License scanner · Issue #2303 · aquasecurity/trivy - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy/issues/2303>
23. Secret Scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.52/docs/scanner/secret/>
24. Secret - Trivy, accessed May 12, 2025, <https://trivy.dev/v0.61/docs/scanner/secret/>
25. Secret Scanning - Trivy - Aqua Security, accessed May 12, 2025, <https://aquasecurity.github.io/trivy/v0.27.1/docs/secret/scanning/>
26. Secret - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/scanner/secret/>
27. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.44/docs/target/sbom/>
28. SBOM - Trivy - Aqua Security, accessed May 12, 2025, <https://aquasecurity.github.io/trivy/v0.59/docs/references/configuration/cli/trivy_sbom/>
29. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.40/docs/target/sbom/>
30. SBOM - Trivy, accessed May 12, 2025, <https://trivy.dev/v0.61/docs/target/sbom/>
31. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.50/docs/supply-chain/vex/>
32. Vulnerability Exploitability eXchange (VEX) - CycloneDX, accessed May 12, 2025, <https://cyclonedx.org/capabilities/vex/>
33. Benefits of VEX for SBOMs - Mend.io, accessed May 12, 2025, <https://www.mend.io/blog/benefits-of-vex-for-sboms/>
34. What is VEX and Why Should I Care? | Blog - Endor Labs, accessed May 12, 2025, <https://www.endorlabs.com/learn/what-is-vex-and-why-should-i-care>
35. SBOM VEX Workflow - Sonatype Help, accessed May 12, 2025, <https://help.sonatype.com/en/sbom-vex-workflow.html>
36. Create VEX statements | Documentation - Google Cloud, accessed May 12, 2025, <https://cloud.google.com/artifact-analysis/docs/create-vex>
37. Create an exception using the VEX - Docker Docs, accessed May 12, 2025, <https://docs.docker.com/scout/how-tos/create-exceptions-vex/>
38. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.41/docs/supply-chain/vex/>
39. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.43/docs/supply-chain/vex/>
40. anchore/grype: A vulnerability scanner for container images ... - GitHub, accessed May 12, 2025, <https://github.com/anchore/grype>
41. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.49/docs/supply-chain/vex/>
42. VEX SBOM Reference - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/supply-chain/vex/sbom-ref/>
43. openvex/vexctl: A tool to create, transform and attest VEX metadata - GitHub, accessed May 12, 2025, <https://github.com/openvex/vexctl>
44. Reachability Analysis: 5 Techniques & 5 Critical Best Practices - Oligo Security, accessed May 12, 2025, <https://www.oligo.security/academy/reachability-analysis-5-techniques-and-5-critical-best-practices>
45. A Step-by-Step Guide to Running a SAST Test - Myrror Security, accessed May 12, 2025, <https://myrror.security/a-step-by-step-guide-to-running-a-sast-test/>
46. OpenVEX GitHub action that generates VEX data, accessed May 12, 2025, <https://github.com/openvex/generate-vex>
47. Demystifying VEX: Simplifying SBOMs with Sonatype SBOM Manager, accessed May 12, 2025, <https://www.sonatype.com/blog/demystifying-vex-simplifying-sboms-with-sonatype-sbom-manager>
48. Vulnerability Exploitability eXchange (VEX): The Standard Revolutionizing Security Operations - Community.aws, accessed May 12, 2025, <https://community.aws/content/2tIiOyqrZt2ZBt0RxduE2qKl98B/vulnerability-exploitability-exchange-vex-the-standard-revolutionizing-security-operations>
49. Reachability Analysis with Sonatype CLI, accessed May 12, 2025, <https://help.sonatype.com/en/reachability-analysis-cli.html>
50. Seven Risk Areas Addressed by SBOMs - Fortress Information Security, accessed May 12, 2025, <https://www.fortressinfosec.com/blog/seven-risk-areas-addressed-by-sboms>
51. What is software composition analysis (SCA)? And how it works - Dynatrace, accessed May 12, 2025, <https://www.dynatrace.com/news/blog/what-is-software-composition-analysis/>
52. What Is Software Composition Analysis (SCA)? Tools and Benefits - Legit Security, accessed May 12, 2025, <https://www.legitsecurity.com/blog/benefits-of-software-composition-analysis>
53. Policy Compliance | Dependency-Track, accessed May 12, 2025, <https://docs.dependencytrack.org/usage/policy-compliance/>
54. The Top 6 Open-Source SBOM Tools - Upwind, accessed May 12, 2025, <https://www.upwind.io/glossary/the-top-6-open-source-sbom-tools>
55. Open-Source Software Composition Analysis Tools Comparison - Planet Crust, accessed May 12, 2025, <https://www.planetcrust.com/open-source-software-composition-analysis-tools-comparison/?utm_campaign=blog>
56. Best SBOM Tools for Developers: Our 2025 Picks - Aikido, accessed May 12, 2025, <https://www.aikido.dev/blog/best-sbom-generation-tools>
57. View golang: github.com/anchore/grype | OpenText Core SCA - Debricked, accessed May 12, 2025, <https://debricked.com/select/package/golang-github.com%2Fanchore%2Fgrype>
58. Open Source Vulnerability Scanning: Tools & How-To Guide - Anchore, accessed May 12, 2025, <https://anchore.com/software-supply-chain-security/open-source-container-vulnerability-scanning-tools/>
59. Open Source Container Security: Anchore - Syft & Grype - Bion Consulting, accessed May 12, 2025, <https://www.bionconsulting.com/blog/open-source-container-security-anchore-syft-grype>
60. Package grype - GitHub, accessed May 12, 2025, <https://github.com/orgs/anchore/packages/container/package/grype>
61. A Practical Approach to SBOM in CI/CD - Dependency-Track - DevSec Blog, accessed May 12, 2025, <https://devsec-blog.com/2024/03/a-practical-approach-to-sbom-in-ci-cd-part-iii-tracking-sboms-with-dependency-track/>
62. Dependency-Track: Introduction, accessed May 12, 2025, <https://docs.dependencytrack.org/>
63. Creating SBOMs with the Snyk CLI, accessed May 12, 2025, <https://snyk.io/blog/creating-sboms-snyk-cli/>
64. SBOM test | Snyk User Docs, accessed May 12, 2025, <https://docs.snyk.io/snyk-cli/commands/sbom-test>
65. Advancing SBOM standards: Snyk and SPDX, accessed May 12, 2025, <https://snyk.io/blog/advancing-sbom-standards-snyk-spdx/>
66. Snyk Documentation | Snyk User Docs, accessed May 12, 2025, <https://docs.snyk.io/>
67. snyk/parlay: Enrich SBOMs with data from third party services - GitHub, accessed May 12, 2025, <https://github.com/snyk/parlay>
68. How to Automate Compliance with FOSSA's SBOM Management Add-on - YouTube, accessed May 12, 2025, <https://www.youtube.com/watch?v=beVn8Vwzc04>
69. Create a Project using a Dependencies SBOM Import - Mend.io Documentation, accessed May 12, 2025, <https://docs.mend.io/platform/latest/create-a-project-using-a-dependencies-sbom-import>
70. SBOM vs. SBOM: Comparing SBOMs from Different Tools and Lifecycle Stages | Blog | Endor Labs, accessed May 12, 2025, <https://www.endorlabs.com/learn/sbom-vs-sbom-comparing-sboms-from-different-tools-and-lifecycle-stages>
71. How to integrate SBOMs into the software development life cycle - Sonatype, accessed May 12, 2025, <https://www.sonatype.com/blog/how-to-integrate-sboms-into-the-software-development-life-cycle>