# **Leveraging Trivy for Comprehensive SBOM Analysis: Vulnerability, License, and Secret Scanning with VEX Integration**

## **1. Introduction**

Trivy, an open-source security scanner developed by Aqua Security, has emerged as a versatile and comprehensive tool for identifying security issues across various artifacts, including container images, filesystems, Git repositories, and, critically, Software Bill of Materials (SBOMs).1 Its capabilities extend beyond vulnerability detection to include license compliance scanning and secret detection, making it a valuable asset in modern DevSecOps practices.

This report provides an expert-level guide to utilizing Trivy for a multi-faceted security analysis workflow. It focuses on three key areas: performing vulnerability scans on SBOMs with Vulnerability Exploitability eXchange (VEX) filtering, conducting license scans on SBOMs, and executing secret scans directly on repositories. The objective is to outline a robust workflow, detailing the sequence of Trivy commands, considerations for VEX document creation and updates, and best practices for storing and managing the generated security artifacts (SBOMs, VEX documents, and scan reports) to ensure traceability and support compliance objectives. A structured approach to these processes is paramount for effectively managing software supply chain security.

## **2. Understanding the Core Components: SBOM and VEX**

A foundational understanding of SBOMs and VEX is crucial before delving into specific Trivy commands and workflows. These two components are central to modern software supply chain security and vulnerability management.

### **2.1. The Role of Software Bill of Materials (SBOM)**

A Software Bill of Materials (SBOM) is a formal, machine-readable inventory that lists all software components, libraries, and dependencies included in a software artifact.3 Its significance in the current software landscape cannot be overstated, as it provides essential transparency into the software supply chain. This transparency is fundamental for effective vulnerability management, enabling organizations to quickly identify if they are affected by newly disclosed vulnerabilities in their dependencies.4 Furthermore, SBOMs play a critical role in license compliance by detailing the licenses associated with each component, helping organizations avoid legal and compliance risks.4

Commonly adopted SBOM formats include CycloneDX, developed by the Open Worldwide Application Security Project (OWASP), and the Software Package Data Exchange (SPDX), which is an ISO/IEC standard.4 GitHub, for instance, provides a native SBOM export feature for repositories, generating SBOMs in the SPDX JSON format. These exports typically include direct and transitive dependencies (for supported package ecosystems), component versions, and associated license information.3 The completeness of such SBOMs relies on the accuracy of manifest files (e.g., package.json, pom.xml) and any dependencies submitted via GitHub's dependency submission API.3

### **2.2. Vulnerability Exploitability eXchange (VEX)**

While SBOMs list components and their known vulnerabilities, not all vulnerabilities present in a component are necessarily exploitable in the context of a specific product. The Vulnerability Exploitability eXchange (VEX) addresses this by providing a machine-readable advisory format that communicates the exploitability status of vulnerabilities within a particular product or component.9 VEX acts as a complementary metadata layer to SBOMs, allowing software producers to assert whether a product is affected by a specific vulnerability and why.9

The primary purpose of VEX is to contextualize the findings from an SBOM vulnerability scan, thereby reducing the noise from false positives and enabling security teams to prioritize remediation efforts on vulnerabilities that pose a genuine risk.9 Key VEX statuses, as defined by CISA and adopted by formats like OpenVEX, include 12:

* **not\_affected**: The product is not impacted by the vulnerability.
* **affected**: The product is impacted, and actions are recommended.
* **fixed**: The product version contains a fix for the vulnerability.
* **under\_investigation**: The impact of the vulnerability on the product is currently being assessed.

For a status of not\_affected, a justification is required. Common justifications include 15:

| **Justification** | **Description** |
| --- | --- |
| component\_not\_present | The affected component is not present in the product. |
| vulnerable\_code\_not\_present | The component is present, but the specific vulnerable code within that component is not included or used. |
| vulnerable\_code\_not\_in\_execute\_path | The vulnerable code is present but is not reachable or cannot be executed in the product's runtime context. |
| vulnerable\_code\_cannot\_be\_controlled\_by\_adversary | The vulnerable code is present and reachable, but an attacker cannot control the inputs to trigger the exploit. |
| inline\_mitigations\_already\_exist | The product has other security controls or mitigations in place that prevent the vulnerability's exploitation. |

Trivy supports several VEX formats, including OpenVEX, CSAF (Common Security Advisory Framework), and CycloneDX VEX (when used with a CycloneDX SBOM as an independent BOM).17 This flexibility allows organizations to choose the VEX format that best fits their tooling and communication needs.

## **3. Trivy Command-Line Usage for Specified Scans**

Trivy offers a rich set of command-line options to perform the requested scans. Understanding the correct syntax is key to integrating these scans effectively into a security workflow.

### **3.1. Vulnerability Scan of an SBOM with VEX Filtering**

Trivy can scan an SBOM for vulnerabilities and apply VEX data to filter the results, providing a more accurate picture of actual risks.

The base command for scanning an SBOM is trivy sbom <SBOM\_PATH>.18 Trivy automatically detects the input SBOM format, supporting CycloneDX (JSON), SPDX (both tag-value and JSON formats), and CycloneDX-type attestations.18 It's important to note that for CycloneDX, Trivy currently supports JSON format, not XML, for SBOM input.18

To enable VEX filtering, the --vex option is used. This option can accept a local VEX file path (e.g., --vex path/to/vex.json), a URL to a VEX document in a VEX repository, or the special keyword sbom-ref to discover VEX documents referenced within the SBOM itself (for CycloneDX SBOMs with exploitability-statement external references).9 Trivy supports OpenVEX, CSAF, and CycloneDX VEX files (the latter requires the input SBOM also to be in CycloneDX format).17 When a VEX document is provided, Trivy filters the detected vulnerabilities based on the status (e.g., not\_affected, fixed) asserted in the VEX statements.17

An example command to scan an SBOM with VEX filtering, outputting results in JSON format, would be:

trivy sbom path/to/your/sbom.json --vex path/to/your/project.openvex.json --format json --output vuln\_report.json

This command scans sbom.json, applies filtering based on project.openvex.json, and saves the JSON output to vuln\_report.json.20

Trivy offers various output formats using the --format <FORMAT> flag, including table (default), json, sarif (Static Analysis Results Interchange Format, suitable for GitHub code scanning), and template (for custom output using Go templates).26 The --output <FILE\_PATH> flag directs the output to a specified file.26 To view vulnerabilities that were suppressed by VEX or other filtering mechanisms, the --show-suppressed flag can be used.28

### **3.2. License Scan of an SBOM**

Trivy can also perform license scanning on an SBOM to identify and classify the licenses of the software components.

The command to perform a license scan on an SBOM is trivy sbom <SBOM\_PATH> --scanners license.20 This command explicitly tells Trivy to activate only the license scanner. Supported SBOM input formats include CycloneDX (JSON) and SPDX (JSON).20

The output of a license scan identifies the licenses associated with each component and classifies them according to categories such as "Forbidden," "Restricted," "Reciprocal," "Notice," "Permissive," "Unencumbered," and "Unknown".29 This classification is based on Google's license classification framework.29 Each classification is typically mapped to a severity level (e.g., "Forbidden" licenses are often marked as "CRITICAL").29

Regarding commercial use restrictions, Trivy's reporting is indirect. It flags licenses based on their classification (e.g., a "Forbidden" or "Restricted" license likely has terms incompatible with typical commercial use), but it does not provide direct summaries of specific license terms or obligations.30 Users need to consult the actual license texts for detailed restrictions.

An example command for an SBOM license scan outputting to a table format would be:

trivy sbom path/to/your/sbom.json --scanners license --format table --output license\_report.txt

.29

While Trivy also supports an extended license scanning mode (--license-full) for direct filesystem or image scans to find licenses in source files and documents, this option is not directly applicable when the input is solely an SBOM.29

### **3.3. Secret Scan of a Repository**

Trivy is capable of scanning Git repositories or local filesystems for inadvertently exposed secrets, such as API keys, passwords, and tokens.

The command for scanning a repository is trivy repo <REPO\_URL\_OR\_PATH> or, for a local filesystem, trivy fs <PATH>.34 Secret scanning is enabled by default for these scan types but can also be explicitly invoked using --scanners secret.34 It is crucial to understand that SBOMs are **not** direct inputs for secret scanning; this process requires access to the file contents within the repository or filesystem.18 An SBOM primarily lists components and metadata, not the content of every file where secrets might reside.39

The output from a secret scan includes details of any detected secrets, their category (e.g., AWS key, GitHub token), severity, the file path where the secret was found, and the specific line number.34

An example command to scan a remote GitHub repository for secrets and output the results in JSON format is:

trivy repo https://github.com/your/repo --format json --output secrets\_report.json

.35

Trivy's secret scanning behavior can be customized using a trivy-secret.yaml configuration file to define custom detection rules or allow specific patterns. This file can be specified using the --secret-config <PATH> option.34

## **4. Workflow for SBOM, VEX, and Scan Execution**

A systematic workflow is essential for effectively utilizing Trivy's capabilities for SBOM analysis, VEX management, and scanning. This section outlines the recommended steps, from obtaining an SBOM to managing the generated artifacts.

### **4.1. Prerequisite: Obtaining an SBOM for a GitHub Repository**

Before any scanning can occur, an SBOM for the target GitHub repository must be available. There are several methods to obtain or generate one:

1. **GitHub's Native SBOM Export:** GitHub allows users to export an SBOM for a repository directly from the dependency graph UI or via its REST API. The exported format is SPDX JSON and includes details about direct and, where supported by the ecosystem, transitive dependencies, along with their versions and licenses.3
2. **Generation using Trivy:** Trivy itself can generate an SBOM from a Git repository. The command would be trivy repo <REPO\_URL\_OR\_PATH> --format <spdx-json|cyclonedx> --output repo\_sbom.json. This ensures the SBOM is in a format Trivy understands well, potentially including custom properties that enhance its scanning accuracy.19
3. **Other Third-Party Tools:** Tools like Syft (from Anchore) are also widely used for generating SBOMs from various sources, including container images and filesystems, which can then be used as input for Trivy.40

The choice of method may depend on existing tooling, desired SBOM format, and the need for specific metadata that one tool might provide over another.

### **4.2. VEX Document Creation and Management**

VEX documents are crucial for contextualizing vulnerability data from SBOMs. Their creation and lifecycle management require careful consideration.

Tools for VEX Creation:

Several tools can assist in creating VEX documents in various formats supported by Trivy:

* **vexctl**: This command-line tool from the OpenVEX project is designed for creating, merging, and attesting OpenVEX documents. A typical creation command is vexctl create --product "pkg:generic/your-product@1.0" --vuln "CVE-2023-12345" --status "not\_affected" --justification "vulnerable\_code\_not\_in\_execute\_path" -o my\_product.openvex.json.15
* **CSAF Tools**: For CSAF (Common Security Advisory Framework) VEX documents, tools like Secvisogram (a CSAF editor) and various validator libraries are available. Trivy's support for CSAF VEX makes these tools relevant.17
* **CycloneDX VEX**: If using CycloneDX SBOMs, VEX information can be embedded or provided in a separate CycloneDX VEX document. Trivy consumes the latter. These can be created manually adhering to the CycloneDX specification or with tools that output this format.17

Determining VEX Justifications:

The justification for a not\_affected status is a critical part of a VEX statement. Determining the correct justification often involves a combination of automated analysis and manual review:

* **component\_not\_present** or **vulnerable\_code\_not\_present**: This can often be verified through build analysis, examining the final artifact, or by confirming that a specific module of a library was not included.16 For example, a base container image might include many packages, but later layers could remove some, or a library might be included but only a subset of its non-vulnerable functions are actually compiled into the product.16
* **vulnerable\_code\_not\_in\_execute\_path**: This justification implies that although the vulnerable code is present, it is never called during the application's runtime. Ascertaining this may require Static Application Security Testing (SAST) tools to perform call graph analysis or data flow analysis.10 Some commercial tools like Endor Labs claim to automate parts of this reachability analysis.10 However, for complex applications, manual review by developers familiar with the codebase is often necessary to confirm that no execution path to the vulnerable code exists under any condition.16
* **vulnerable\_code\_cannot\_be\_controlled\_by\_adversary**: This means the vulnerable code might be reachable, but an attacker lacks the means to provide the specific input or control needed to trigger the vulnerability. This usually requires a deep understanding of the application's input validation, data handling, and operational context.16 For instance, if a vulnerability requires specific network ports to be open, but those ports are disabled by default and cannot be enabled by the user, this justification might apply.16
* **inline\_mitigations\_already\_exist**: This applies when the product has other security controls (e.g., input sanitization further upstream, sandboxing, firewalls blocking access to a vulnerable port) that effectively neutralize the vulnerability.16

The process of determining these justifications, especially vulnerable\_code\_not\_in\_execute\_path and vulnerable\_code\_cannot\_be\_controlled\_by\_adversary, often involves significant manual effort and expertise. While SAST tools can aid in identifying unreachable code, the final assertion in a VEX document carries weight and implies a thorough investigation.

VEX Document Lifecycle and Updates:

VEX documents are not static. They must be treated as living documents that evolve with the software and the threat landscape.11

* **Updates**: As vulnerabilities are patched, software components change, new vulnerabilities are discovered, or further analysis refines the understanding of exploitability, VEX documents must be updated. Statuses like under\_investigation should eventually resolve to a more definitive state.9
* **Merging**: For OpenVEX, vexctl merge can be used to combine multiple VEX statements for the same product, effectively creating a new version of the VEX document that reflects the latest status. vexctl processes statements chronologically to determine the current effective status.43
* **Version Control**: Storing VEX documents in a version control system like Git is highly recommended. This provides an auditable history of changes and allows for collaborative review and management.52
* **Automation**: GitHub Actions can be used to automate parts of the VEX lifecycle. For example, the openvex/generate-vex action can generate OpenVEX documents from predefined templates during a release process.53 Custom scripts using vexctl can also be integrated into CI/CD pipelines for merging or attesting VEX documents.

### **4.3. Recommended Scan Order and Commands**

Assuming an SBOM for a GitHub repository is already available (e.g., repo\_sbom.spdx.json) and a corresponding OpenVEX document exists (e.g., project.openvex.json), a logical sequence of Trivy scans would be:

1. Vulnerability Scan of SBOM with VEX Filtering:  
   This is typically the first scan performed on the SBOM.  
   Command:  
   Bash  
   trivy sbom repo\_sbom.spdx.json --vex project.openvex.json --format sarif --output vuln\_report.sarif --severity HIGH,CRITICAL  
     
   This command scans the SPDX SBOM, applies VEX filtering from project.openvex.json, filters for HIGH and CRITICAL severity vulnerabilities, and outputs the results in SARIF format, suitable for ingestion into security dashboards or GitHub code scanning.
2. License Scan of SBOM:  
   This scan identifies licenses of components listed in the SBOM.  
   Command:  
   Bash  
   trivy sbom repo\_sbom.spdx.json --scanners license --format json --output license\_report.json  
     
   This command performs a license scan on the same SBOM and outputs the results in JSON format for further analysis or record-keeping.
3. Secret Scan of the Repository:  
   This scan is performed directly on the repository's codebase, not the SBOM.  
   Command (run from the root of the checked-out repository):  
   Bash  
   trivy repo. --scanners secret --format json --output secret\_report.json  
     
   This command scans the current directory (assumed to be the repository root) for secrets and outputs the findings in JSON format.

This order prioritizes vulnerability assessment with contextualization from VEX, followed by license compliance checks, and finally, direct code scanning for secrets.

### **4.4. Storing and Managing Artifacts**

Proper storage and management of the generated security artifacts are crucial for traceability, compliance, and ongoing security monitoring.

* **SBOMs:**
  + **Version Control:** SBOMs should ideally be version-controlled alongside the codebase they represent. This allows tracking changes in dependencies with each software version.56
  + **Artifact Repositories:** Store SBOMs in an artifact repository (e.g., JFrog Artifactory, Sonatype Nexus) or attach them to build artifacts in CI/CD systems. GitHub Releases can also be used to associate SBOMs with specific software versions.9
  + **Dedicated Platforms:** Tools like Dependency-Track are designed to ingest and manage SBOMs, providing a centralized inventory and continuous monitoring capabilities.57
* **VEX Documents:**
  + **Version Control (Git):** Git is the ideal system for storing and versioning VEX documents due to its diffing capabilities and support for collaborative workflows (e.g., pull requests for VEX updates).52
  + **Co-location vs. Central Repository:** VEX documents can be co-located with the project's source code (e.g., in a .vex/ directory as suggested by some OpenVEX practices 60) or stored in a dedicated central VEX repository.
  + **Trivy VEX Repository Support:** Trivy can consume VEX documents from VEX repositories that adhere to the VEX Repository Specification. This allows organizations to manage VEX data centrally and have Trivy automatically fetch relevant documents during scans.28
* **Scan Reports:**
  + **CI/CD Artifact Storage:** Most CI/CD systems provide mechanisms to store build artifacts, including scan reports.64
  + **Security Dashboards/Platforms:** Upload scan reports (especially in standard formats like SARIF) to security dashboards or vulnerability management platforms. For example, SARIF reports can be uploaded to the GitHub Security tab to display code scanning alerts.27 Dependency-Track can also ingest vulnerability reports.65
  + **Traceability:** Ensure reports are linked to specific code commits, build numbers, and SBOM/VEX versions to maintain clear traceability.

Effective artifact management ensures that security information is accessible, auditable, and can be used to track remediation progress and demonstrate compliance over time.

## **5. CI/CD Integration with GitHub Actions**

Integrating these Trivy scans into a CI/CD pipeline, such as GitHub Actions, automates the security analysis process, providing rapid feedback to developers.

### **5.1. Workflow Setup**

A typical GitHub Actions workflow for these scans would trigger on events like push to main or release branches, pull\_request targeting these branches, or workflow\_dispatch for manual runs.64 The first step in the job is usually actions/checkout@v4 to get the repository code.64 Trivy itself can be installed manually or, more commonly, used via the aquasecurity/trivy-action.

### **5.2. Using aquasecurity/trivy-action**

The aquasecurity/trivy-action simplifies running Trivy scans in GitHub Actions.

* **SBOM Vulnerability Scan Example:**  
  YAML  
  name: Trivy Scans  
  on:  
   push:  
   branches: [ main ]  
   pull\_request:  
   branches: [ main ]  
  jobs:  
   build\_and\_scan:  
   runs-on: ubuntu-latest  
   steps:  
   - name: Checkout code  
   uses: actions/checkout@v4  
    
   # Assume SBOM (e.g., repo\_sbom.spdx.json) is generated or available here  
   # Assume VEX (e.g., project.openvex.json) is available here  
    
   - name: Trivy SBOM Vulnerability Scan with VEX  
   uses: aquasecurity/trivy-action@0.28.0 # Specify a stable version  
   with:  
   scan-type: 'sbom'  
   input: 'path/to/your/repo\_sbom.spdx.json' # Path to your SBOM file  
   format: 'sarif'  
   output: 'trivy-vuln-results.sarif'  
   severity: 'HIGH,CRITICAL'  
   # VEX file input (check trivy-action documentation for the exact input name)  
   # Option 1: Direct input if supported by trivy-action  
   # vex-path: 'path/to/your/project.openvex.json'   
   env:  
   # Option 2: Environment variable if trivy-action/Trivy CLI picks it up  
   TRIVY\_VEX: 'path/to/your/project.openvex.json'   
    
   - name: Upload SARIF file  
   uses: github/codeql-action/upload-sarif@v3  
   with:  
   sarif\_file: 'trivy-vuln-results.sarif'  
    
   - name: Trivy License Scan from SBOM  
   uses: aquasecurity/trivy-action@0.28.0  
   with:  
   scan-type: 'sbom'  
   input: 'path/to/your/repo\_sbom.spdx.json'  
   scanners: 'license'  
   format: 'json'  
   output: 'license\_report.json'  
    
   - name: Upload License Report  
   uses: actions/upload-artifact@v4  
   with:  
   name: license-report  
   path: license\_report.json  
    
   - name: Trivy Secret Scan of Repository  
   uses: aquasecurity/trivy-action@0.28.0  
   with:  
   scan-type: 'repo' # Or 'fs' to scan the checked-out filesystem  
   image-ref: '.' # Indicates current directory for 'fs' scan-type  
   scanners: 'secret'  
   format: 'json'  
   output: 'secret\_report.json'  
   # github-pat: ${{ secrets.GITHUB\_TOKEN }} # May be needed for 'repo' scan type on private repos  
    
   - name: Upload Secret Report  
   uses: actions/upload-artifact@v4  
   with:  
   name: secret-report  
   path: secret\_report.json  
  This example demonstrates scanning an existing SBOM for vulnerabilities with VEX, performing a license scan on the same SBOM, and scanning the repository for secrets. The trivy-action documentation should be consulted for the most current method of passing a VEX file; if a direct input like vex-path is not available, setting an environment variable such as TRIVY\_VEX might be the way Trivy CLI options are passed through.66 For VEX repositories, the runner environment must have access to the repository.yaml configuration file, or it needs to be configured dynamically within the CI job.28

### **5.3. Incorporating vexctl Commands**

If VEX documents are managed within a Git repository (either the same one or a separate, accessible one), vexctl commands can be integrated into the workflow.

* **Setup vexctl**:  
  YAML  
  - name: Setup vexctl  
   run: |  
   # Example: Install Go if not present, then install vexctl  
   # sudo apt-get update && sudo apt-get install -y golang-go  
   go install github.com/openvex/vexctl@latest  
   echo "$(go env GOPATH)/bin" >> $GITHUB\_PATH  
    
  Alternatively, a dedicated setup-vexctl GitHub Action might be available.55
* **Merge VEX Updates**:  
  YAML  
  - name: Merge VEX Updates  
   run: |  
   # Ensure project.openvex.json and any update.vex.json are present  
   # Example: vexctl merge --product "your-purl" project.openvex.json update.vex.json --out merged.vex.json  
   # This merged.vex.json would then be used by the Trivy scan step  
   # Further steps would be needed to commit merged.vex.json back to the repository if desired  
    
  The openvex/generate-vex GitHub Action is suitable for generating VEX documents from predefined templates, typically stored in a .openvex directory initialized by vexctl generate --init.53 This action is more about initial generation or regeneration from templates rather than complex updates to existing, manually curated VEX statements.

### **5.4. Storing Artifacts Produced by the Workflow**

All generated artifacts (SBOMs, VEX documents used/updated, scan reports) should be archived for traceability.

* Use actions/upload-artifact@v4 to store these files within the GitHub Actions run.64  
  YAML  
  - name: Upload All Scan Reports  
   uses: actions/upload-artifact@v4  
   with:  
   name: security-scan-artifacts  
   path: |  
   trivy-vuln-results.sarif  
   license\_report.json  
   secret\_report.json  
   path/to/your/repo\_sbom.spdx.json  
   path/to/your/project.openvex.json   
   # Add other relevant artifacts
* For formal releases, consider attaching these artifacts to GitHub Releases.
* If using a platform like Dependency-Track, the workflow could include a step to upload SBOMs and VEX documents via its API.57

### **5.5. Deeper Considerations for CI/CD Automation**

Automating SBOM and VEX workflows in CI/CD introduces several considerations that go beyond simple command execution.

A critical aspect is **managing the VEX "source of truth."** If VEX documents are updated based on CI/CD scan results (e.g., a new vulnerability is found, and a VEX statement needs to be created or updated), a clear, controlled process is essential. If this update is manual and happens outside the pipeline, subsequent CI runs might continue to report the vulnerability until the authoritative VEX source is updated and accessible to the pipeline. Conversely, if the pipeline attempts to automatically update VEX documents (e.g., using vexctl merge and then committing back to Git), this process must be carefully governed to prevent erroneous or unvetted VEX statements from being integrated. This implies the need for a robust Git workflow for VEX documents themselves, potentially involving feature branches for VEX changes, pull requests, and reviews by the security team before merging into the main VEX source.52 The CI pipeline should always fetch the latest *approved* VEX data before initiating scans.

Another significant challenge is the **automation of complex VEX justifications within a standard CI pipeline**. Justifications like vulnerable\_code\_not\_in\_execute\_path often necessitate sophisticated static analysis capabilities, akin to SAST, to perform call graph or data flow analysis.10 Standard CI runners may not be equipped with these specialized tools, or the analysis might be too resource-intensive or time-consuming for a typical CI cycle. Consequently, for these intricate justifications, VEX documents will likely be managed and updated primarily through an offline or semi-automated process involving security analysts and dedicated tools. The CI pipeline would then *consume* these pre-vetted VEX documents. Actions like openvex/generate-vex are more geared towards generating VEX from "golden" data or templates rather than performing dynamic, deep code analysis for such justifications.53

Finally, it's important to **balance automation with necessary human oversight**. While automation is a key goal of DevSecOps, the VEX process, in particular, often requires human judgment. A tool can identify a vulnerability, and may even assist in determining code reachability. However, the decision to formally assert a status like not\_affected with a specific justification, or to declare a product affected and outline a remediation plan, is fundamentally a risk management decision that often requires human expertise and sign-off.9 The CI/CD workflow should facilitate this by producing clear, actionable reports for human review and by having well-defined integration points where human-generated or human-approved VEX data is injected into the automated scanning process. The pipeline can automate the *application* of VEX data, but not always its *authoritative creation* for complex scenarios.

## **6. Advanced Considerations and Best Practices**

Beyond the basic commands and workflow, several advanced considerations and best practices can significantly improve the effectiveness and accuracy of this security scanning strategy.

### **6.1. Addressing PURL Accuracy for Effective VEX Matching**

The effectiveness of VEX filtering hinges on the precise matching of Package URLs (PURLs) between the SBOM and the VEX document. Discrepancies in how PURLs are generated by the SBOM tool versus how they are specified in the VEX document can lead to VEX statements not being correctly applied, thereby failing to suppress irrelevant vulnerabilities.25 This challenge is particularly pronounced for commercial software components, where standardized PURL generation can be less consistent than for open-source packages.68 An issue noted in Dependency-Track, for example, indicated that VEX might sometimes be applied only by CVE ID if the PURL or component matching is inconsistent, potentially leading to overly broad or incorrect application of VEX statements.69

**Best Practice**: When authoring VEX statements, meticulously ensure that the product and subcomponent PURLs precisely match those present in the SBOM being scanned. If possible, extract PURLs directly from the generated SBOM to use in VEX documents. Be mindful of Trivy's PURL matching rules (e.g., matching without version or qualifiers) and their implications for how VEX statements are applied.17 Using overly generic PURLs in VEX can lead to unintended suppression of vulnerabilities.

### **6.2. Impact of Trivy's Custom SBOM Properties on External SBOMs**

Trivy's documentation indicates that scanning SBOMs generated by tools other than Trivy *may result in inaccurate detection* because Trivy relies on unspecified custom properties within the SBOM for optimal scanning accuracy.18 The exact nature of these custom properties is not publicly detailed.20

This reliance could potentially affect both vulnerability and license detection accuracy when using an SBOM generated by an external tool, such as GitHub's native SBOM export feature.20

**Best Practice/Mitigation**: If achieving the highest possible accuracy with Trivy is the primary concern, consider generating the primary SBOM *using Trivy itself* (e.g., trivy fs. --format spdx-json -o trivy\_sbom.json). This Trivy-generated SBOM can then be used for subsequent trivy sbom vulnerability and license scans. If using an externally generated SBOM is unavoidable, be aware of this potential for discrepancies and consider cross-verifying critical findings with other tools or manual checks.

### **6.3. Alternative Vulnerability Suppression (When VEX is Overkill or Not Ready)**

While VEX provides a standardized and context-rich way to manage vulnerability exploitability, there are simpler methods for suppressing vulnerabilities in Trivy when a full VEX document is not yet available or deemed too complex for a particular scenario:

* **.trivyignore file**: This is a straightforward method to ignore vulnerabilities by listing their CVE IDs (or other vulnerability IDs recognized by Trivy). Each ID should be on a new line. Optional expiration dates can be added to an ignore rule (e.g., CVE-2021-XXXXX exp:2024-12-31).75 Trivy also supports a YAML-based ignore file, .trivyignore.yaml, which offers more structured ignore rules, potentially including ignoring by path, though this is often marked as experimental.80
* **Rego Policies for Filtering**: Trivy allows the use of custom Rego policies for more sophisticated filtering of scan results via the --ignore-policy <path\_to\_rego\_file> flag.80 These policies can inspect various fields of a DetectedVulnerability object, such as VulnerabilityID, PkgName, Severity, and CweIDs, enabling complex suppression logic.85 This method offers more flexibility than .trivyignore but lacks the standardization of VEX for communicating exploitability status externally. It's important to note that Rego policies for filtering vulnerability scan results operate on the output of scanners like vuln, while trivy conf uses Rego for misconfiguration policy checks against configuration files.18

### **6.4. Regularly Updating VEX Documents**

The vulnerability landscape, software dependencies, and the product's own codebase are constantly changing. Therefore, VEX documents cannot be static, "set and forget" artifacts.11 A VEX statement that is accurate today might become outdated tomorrow.

**Best Practice**: Establish a formal process for the periodic review and update of all VEX statements. This is especially critical for vulnerabilities marked as under\_investigation, which should be resolved to a definitive status in a timely manner. Reviews should also be triggered when the software undergoes significant changes, dependencies are updated, or new vulnerability intelligence becomes available.9

### **6.5. Advanced Implications and Nuances**

Delving deeper into the use of VEX and SBOMs reveals further strategic considerations:

The **trustworthiness of VEX documents** is paramount. Since VEX statements are assertions made by an author, their value is directly proportional to the credibility of that author and the thoroughness of their analysis. A VEX document stating not\_affected can suppress critical vulnerability alerts; if this assertion is incorrect or based on superficial analysis, it can lead to a dangerous false sense of security.16 Organizations consuming VEX data, particularly from third-party suppliers, must establish policies regarding which VEX sources they trust. For internally generated VEX, a robust, auditable creation and approval process is essential. Cryptographically signing VEX documents (e.g., using vexctl attest --sign 43) can help verify the authorship and integrity of the document but does not inherently guarantee the correctness of the justifications within.

The **granularity of VEX statements**—whether they apply to an entire product or specific subcomponents—is another critical factor. Applying VEX at an inappropriate level of granularity can result in either over-suppression (hiding real risks) or under-suppression (failing to filter out irrelevant alerts). A VEX statement can apply to a whole product or to distinct subcomponents, typically identified by their PURLs.15 If a VEX document declares an entire product not\_affected by a CVE, it might inadvertently suppress alerts for that CVE in *all* its internal components, even if the not\_affected status truly applies only to a specific usage of one component. Conversely, if a subcomponent is genuinely not\_affected, but the VEX statement only addresses the parent product without specific subcomponent detail, alerts might still surface for that subcomponent if the scanning tool cannot infer the inherited status. Trivy's support for OpenVEX subcomponents helps address this by allowing more precise scoping of VEX assertions.87 This underscores the importance of accurate PURL definition for both products and their subcomponents in VEX statements, which in turn relies on the accuracy of PURLs in the SBOM itself. The objective should always be to make VEX assertions as precise as possible.

Finally, while the primary operational use case discussed for VEX is the filtering of scanner results, it's important to recognize that VEX documents serve a broader strategic purpose as a **communication tool**. VEX is, in essence, a form of security advisory.9 Proactively providing VEX documents alongside SBOMs to customers, partners, and regulatory bodies can preemptively address their concerns about vulnerabilities, thereby reducing the support burden and fostering trust.9 This is particularly valuable in regulated industries and for fulfilling contractual obligations, such as those related to U.S. federal government software procurement.5 Therefore, the VEX management workflow should encompass not only internal scan filtering but also mechanisms for the external publication and consumption of VEX data. This broader perspective might influence choices regarding VEX formats (CSAF is often preferred for formal advisories) and distribution channels.

## **7. Conclusion**

Implementing a structured workflow that leverages Trivy for comprehensive Software Bill of Materials (SBOM) analysis—encompassing vulnerability scanning with VEX contextualization, license compliance checks, and repository secret detection—significantly enhances an organization's software supply chain security posture. This approach facilitates proactive risk management by providing early and continuous feedback on potential security weaknesses and compliance issues. The integration of VEX, in particular, allows teams to cut through the noise of vulnerability alerts, focusing resources on issues that pose a genuine threat in their specific product context, thereby reducing alert fatigue and improving remediation efficiency. Diligent artifact management, including versioning and secure storage of SBOMs, VEX documents, and scan reports, ensures traceability and supports audit and compliance mandates.

The commands and workflow outlined in this report provide a solid foundation. However, the DevSecOps landscape, vulnerability intelligence feeds, and security tooling are in a state of constant evolution. Organizations must commit to a cycle of continuous improvement, regularly reviewing and refining their scanning processes, VEX justification criteria, and tool configurations to maintain their effectiveness against emerging threats and evolving standards.

While automation through tools like Trivy and CI/CD integration is a powerful enabler, the human element remains indispensable. Expert judgment is critical for analyzing complex vulnerabilities, making informed risk-based decisions, and authoring accurate, well-justified VEX statements. A successful software security program effectively blends the strengths of automated tooling with the nuanced understanding and critical thinking of skilled security professionals. By adopting such a balanced and adaptive approach, organizations can build more resilient software and navigate the complexities of the modern software supply chain with greater confidence.

#### Works cited

1. Trivy SBOM - secureCodeBox, accessed May 12, 2025, <https://www.securecodebox.io/docs/scanners/trivy-sbom/>
2. The Top 6 Open-Source SBOM Tools - Upwind, accessed May 12, 2025, <https://www.upwind.io/glossary/the-top-6-open-source-sbom-tools>
3. 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>
4. What is a Software Bill of Materials (SBOM)? Best Practices - JFrog, accessed May 12, 2025, <https://jfrog.com/learn/sdlc/sbom/>
5. 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>
6. 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/>
7. SBOM Standards and Formats: A Guide - Legit Security, accessed May 12, 2025, <https://www.legitsecurity.com/aspm-knowledge-base/sbom-standards-and-formats>
8. 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.>
9. Vulnerability Exploitability eXchange (VEX): The Standard Revolutionizing Security Operations - Community.aws, accessed May 12, 2025, <https://community.aws/content/2tIiOyqrZt2ZBt0RxduE2qKl98B>
10. 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>
11. Vulnerability Exploitability eXchange (VEX) - Revenera, accessed May 12, 2025, <https://www.revenera.com/software-composition-analysis/glossary/vulnerability-exploitability-exchange-vex>
12. 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>
13. Benefits of VEX for SBOMs - Mend.io, accessed May 12, 2025, <https://www.mend.io/blog/benefits-of-vex-for-sboms/>
14. spec/OPENVEX-SPEC.md at main - GitHub, accessed May 12, 2025, <https://github.com/openvex/spec/blob/main/OPENVEX-SPEC.md>
15. Create an exception using the VEX - Docker Docs, accessed May 12, 2025, <https://docs.docker.com/scout/how-tos/create-exceptions-vex/>
16. Vulnerability Exploitability eXchange (VEX) - Status Justifications - CISA, accessed May 12, 2025, <https://www.cisa.gov/sites/default/files/publications/VEX_Status_Justification_Jun22.pdf>
17. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.50/docs/supply-chain/vex/>
18. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.43/docs/target/sbom/>
19. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.33/docs/sbom/>
20. SBOM - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/target/sbom/>
21. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/dev/docs/target/sbom/>
22. VEX - Trivy, accessed May 12, 2025, <https://trivy.dev/v0.50/docs/supply-chain/vex/>
23. Overview - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.55/docs/supply-chain/vex/>
24. VEX SBOM Reference - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/supply-chain/vex/sbom-ref/>
25. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.49/docs/supply-chain/vex/>
26. Report Formats - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.17.2/examples/report/>
27. Reporting - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/configuration/reporting/>
28. VEX Repository - Trivy, accessed May 12, 2025, <https://trivy.dev/dev/docs/supply-chain/vex/repo/>
29. License - Trivy, accessed May 12, 2025, <http://trivy.dev/latest/docs/scanner/license/>
30. License - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/scanner/license/>
31. License Scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.33/docs/licenses/scanning/>
32. License scanner · Issue #2303 · aquasecurity/trivy - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy/issues/2303>
33. SBOM - Trivy, accessed May 12, 2025, <https://trivy.dev/v0.61/docs/target/sbom/>
34. Secret Scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.50/docs/scanner/secret/>
35. Code Repository - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.44/docs/target/repository/>
36. Secret - Trivy, accessed May 12, 2025, <https://trivy.dev/v0.61/docs/scanner/secret/>
37. Secret Scanning - Trivy - Aqua Security, accessed May 12, 2025, <https://aquasecurity.github.io/trivy/v0.27.1/docs/secret/scanning/>
38. Secret Scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.52/docs/scanner/secret/>
39. Secret - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/scanner/secret/>
40. View golang: github.com/anchore/grype | OpenText Core SCA - Debricked, accessed May 12, 2025, <https://debricked.com/select/package/golang-github.com%2Fanchore%2Fgrype>
41. 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/>
42. 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>
43. openvex/vexctl: A tool to create, transform and attest VEX metadata - GitHub, accessed May 12, 2025, <https://github.com/openvex/vexctl>
44. CSAF Open Source Tools, accessed May 12, 2025, <https://oasis-open.github.io/csaf-documentation/tools.html>
45. csaf/csaf\_2.0/guidance/faq.md at master · oasis-tcs/csaf · GitHub, accessed May 12, 2025, <https://github.com/oasis-tcs/csaf/blob/master/csaf_2.0/guidance/faq.md>
46. VEX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.43/docs/supply-chain/vex/>
47. 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/>
48. 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>
49. Reachability Analysis with Sonatype CLI, accessed May 12, 2025, <https://help.sonatype.com/en/reachability-analysis-cli.html>
50. Generate VEX documents · MaibornWolff SecObserve · Discussion #1106 - GitHub, accessed May 12, 2025, <https://github.com/MaibornWolff/SecObserve/discussions/1106>
51. SBOM VEX Workflow - Sonatype Help, accessed May 12, 2025, <https://help.sonatype.com/en/sbom-vex-workflow.html>
52. [govulncheck] Generate VEX documents from `govulncheck` output · Issue #116 · kubernetes/sig-security - GitHub, accessed May 12, 2025, <https://github.com/kubernetes/sig-security/issues/116>
53. openvex/generate-vex: OpenVEX GitHub action that ... - GitHub, accessed May 12, 2025, <https://github.com/openvex/generate-vex>
54. openvex/generate-vex - Secure Your GitHub Actions with StepSecurity, accessed May 12, 2025, <https://app.stepsecurity.io/action-advisor/openvex/generate-vex>
55. OpenVEX - GitHub, accessed May 12, 2025, <https://github.com/openvex>
56. 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>
57. 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/>
58. OWASP Dependency-Track, accessed May 12, 2025, <https://owasp.org/www-project-dependency-track/>
59. Dependency-Track | Software Bill of Materials (SBOM) Analysis | OWASP, accessed May 12, 2025, <https://dependencytrack.org/>
60. VEX Repository - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.57/docs/supply-chain/vex/repo/>
61. VEX Repository - Trivy, accessed May 12, 2025, <http://trivy.dev/dev/docs/supply-chain/vex/repo/>
62. VEX Repository - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/supply-chain/vex/repo/>
63. Vulnerability Exploitability Exchange (VEX) - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.56/docs/supply-chain/vex/>
64. aquasecurity/trivy-action: Runs Trivy as GitHub action to ... - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy-action>
65. Dependency-Track - OWASP Developer Guide, accessed May 12, 2025, <https://devguide.owasp.org/05-implementation/02-dependencies/02-dependency-track/>
66. aquasecurity/trivy-action: Runs Trivy as GitHub action to ... - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy-action#inputs>
67. VEX Message | Drupal.org, accessed May 12, 2025, <https://www.drupal.org/project/vex_message/git-instructions>
68. purl needs to identify commercial software - GitHub, accessed May 12, 2025, <https://raw.githubusercontent.com/OWASP/www-project-sbom-forum/main/assets/Purl%20needs%20to%20identify%20commercial%20software%20v3.0.pdf>
69. Export and Import VEX fails to match Vulnerabilities correctly · Issue #3554 · DependencyTrack/dependency-track - GitHub, accessed May 12, 2025, <https://github.com/DependencyTrack/dependency-track/issues/3554>
70. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.58/docs/target/sbom/>
71. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.52/docs/target/sbom/>
72. accessed December 31, 1969, <https://trivy.dev/docs/sbom/detection/>
73. SBOM scanning - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.40/docs/target/sbom/>
74. SPDX - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.33/docs/sbom/spdx/>
75. Filtering - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.44/docs/configuration/filtering/>
76. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.30.4/docs/references/cli/sbom/>
77. SBOM - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.40/docs/references/cli/trivy_sbom/>
78. Issues with .trivyignore working with trivy server integration within Dependency-Track · aquasecurity trivy · Discussion #8351 - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy/discussions/8351>
79. The result of scanning sbom generated by trivy seems wrong #8761 - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy/discussions/8761>
80. Filtering - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/configuration/filtering/>
81. Filtering - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/configuration/filtering/#trivyignore>
82. Filtering - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.47/docs/configuration/filtering/>
83. Filtering - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.41/docs/configuration/filtering/>
84. Vulnerability Filtering - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.22.0/vulnerability/examples/filter/>
85. Filtering - Trivy, accessed May 12, 2025, <https://trivy.dev/latest/docs/configuration/filtering/#by-rego>
86. Custom Policies - Trivy, accessed May 12, 2025, <http://trivy.dev/v0.35/docs/misconfiguration/custom/>
87. trivy/docs/docs/supply-chain/vex/file.md at main · aquasecurity/trivy - GitHub, accessed May 12, 2025, <https://github.com/aquasecurity/trivy/blob/main/docs/docs/supply-chain/vex/file.md>
88. Local VEX Files - Trivy, accessed May 12, 2025, <https://trivy.dev/v0.61/docs/supply-chain/vex/file/>
89. Example CSAF VEX Documents - GitHub, accessed May 12, 2025, <https://github.com/oasis-tcs/csaf/blob/master/csaf_2.0/examples/csaf/csaf_vex/README.md>