# **Comprehensive Analysis of SBOM, HBOM, and SCA Generation Tools: A Solution Architecture and Implementation Framework**

**1. Executive Summary:**

The security of modern software supply chains has become a paramount concern, necessitating robust strategies for identifying and managing software and hardware components. This report details a research effort focused on identifying open-source GitHub projects capable of generating Software Bills of Materials (SBOMs), Hardware Bills of Materials (HBOMs), and performing Software Composition Analysis (SCA). The analysis reveals a landscape of specialized tools with varying capabilities and platform support. Addressing the need for comprehensive coverage across diverse technologies, including IoT, cloud, Docker, Windows, and Linux, requires a multi-faceted approach. This report proposes a solution architecture that strategically combines several of these open-source projects to achieve broad SBOM, HBOM, and SCA coverage. Furthermore, it includes a questionnaire designed to scope the effort for implementing such a solution and a framework for assessing the Level of Effort (LOE) involved.

**2. Introduction:**

The increasing complexity of modern software development, characterized by the extensive use of third-party components and diverse deployment environments, has amplified the risks associated with software supply chains. Malicious actors are increasingly targeting these interconnected ecosystems, making proactive security measures essential.1 Recognizing this evolving threat landscape, the ability to generate comprehensive inventories of software and hardware components, coupled with the capacity to analyze these components for vulnerabilities and licensing implications, has become critical.

An SBOM serves as a formal, machine-readable inventory of software components within an application, providing transparency into open-source usage and helping to expose supply chain vulnerabilities.3 The significance of SBOMs is underscored by initiatives like the U.S. executive order promoting their use.4 However, the adoption and effective utilization of SBOMs still face challenges, including the maturity of available tools and the lack of industry consensus on what an SBOM should contain.4 While the focus has often been on software, the concept extends to hardware with the HBOM, which is particularly relevant in the context of IoT and embedded systems where the provenance and security of hardware components are equally important.5

Complementary to BOM generation is Software Composition Analysis (SCA), a cybersecurity process focused on identifying and managing open-source components within software applications.1 By scanning project dependencies, SCA tools detect known vulnerabilities, license compliance issues, and outdated libraries, enabling developers to mitigate risks throughout the software development lifecycle.1 Modern SCA tools are evolving to offer advanced features such as risk-based vulnerability prioritization and malware detection, moving beyond basic dependency scanning to provide more actionable intelligence.6

This research aims to address the critical need for comprehensive security analysis by identifying active open-source GitHub projects that facilitate SBOM and HBOM generation, as well as SCA. The objective is to evaluate their functionalities, the range of technologies they support, and their potential to be integrated into a holistic solution. Ultimately, this report seeks to provide a well-defined architecture and practical guidance for organizations looking to enhance their software supply chain security posture.

**3. Research Findings: Analysis of GitHub Projects:**

To identify relevant open-source projects, GitHub was searched using terms such as "SBOM generation," "HBOM generation," "Software Composition Analysis," and "SCA." The search was focused on projects that have demonstrated activity within the last six months, indicating ongoing maintenance and relevance. The following table summarizes the key projects identified and their core characteristics.

**Table 1: GitHub Projects for SBOM, HBOM, and SCA**

| **Name** | **Purpose** | **Functionality** | **Last Update Date** | **Supported Platforms/Codebases** |
| --- | --- | --- | --- | --- |
| bomctl | Format-agnostic SBOM tooling | Supports complex operations on multiple SBOM files (fetch, import, alias, export, merge) in various formats (SPDX, CycloneDX). | January 8, 2025 | Linux, Windows, macOS (Go) |
| sbom-utility | API platform for BOM validation, analysis, and editing | Validates SPDX and CycloneDX (including HBOM) against schemas, supports querying, trimming, patching, and diffing. | November 7, 2024 | Unix/Linux, Windows (Go) |
| microsoft/sbom-tool | SPDX 2.2 SBOM generation | Uses Component Detection libraries and ClearlyDefined API for component and license information; CI/CD integration. | February 4, 2025 | Windows, Linux, macOS, Docker,.NET (C#) |
| scanoss/sbom-workbench | GUI for SPDX-Lite SBOM generation | Scans source code using SCANOSS API, generates SPDX-Lite SBOMs, supports advanced settings. | (Requires further investigation) | Cross-platform (Node.js) |
| OWASP DependencyCheck | Vulnerability analysis of dependencies | Identifies publicly disclosed vulnerabilities by checking for CPE identifiers and linking to CVE entries. | February 16, 2025 | Wide range via build tool plugins (.NET, GoLang, Elixir, npm, Ruby, Java) |
| ScanCode-Toolkit | License, copyright, and dependency detection | Detects licenses using database comparison, supports numerous package and build manifest formats. | March 6, 2025 | Windows, macOS, Linux (Python) |
| OpenSCA-cli | Open-source SCA for dependencies, vulnerabilities, and licenses | Parses configuration files of various package managers, supports multiple report formats (JSON, XML, SPDX, CycloneDX). | March 28, 2025 | Wide range (Go, Python) |
| Retire.js | JavaScript library vulnerability detection and SBOM generation | Scans web and Node.js apps for vulnerable JavaScript libraries, generates CycloneDX SBOMs. | (Activity within 6 months) | JavaScript-focused (command line, browser extensions) |
| bundler-audit | Ruby dependency vulnerability check | Checks Gemfile.lock for vulnerable gem versions and insecure gem sources. | August 22, 2024 | Ruby (requires Bundler) |
| Dependency-Track | Intelligent Component Analysis platform | Consumes and produces CycloneDX SBOMs, tracks component usage, identifies vulnerabilities and license risks, integrates with vulnerability intelligence sources. | March 12, 2025 | Ecosystem agnostic (Java) |

**3.1. SBOM Generation Tools:**

bomctl is designed to be a versatile tool for managing SBOMs regardless of their format.7 It focuses on bridging the gap between SBOM generation and analysis by enabling complex operations on multiple SBOM files. Its support for the NTIA minimum fields, as well as other fields supported by the protobom library, makes it a valuable asset for organizations aiming for compliance and comprehensive SBOM management.7 The ability to fetch SBOMs from various sources like HTTPS, OCI, Git, GitHub, and GitLab, along with its import and export functionalities in formats like SPDX and CycloneDX, highlights its role in integrating different parts of the software supply chain.7 The project's roadmap includes features for generating diffs between SBOMs, further emphasizing its utility in tracking changes and understanding the evolution of software components.7

The sbom-utility project offers an API platform for interacting with BOMs, supporting both SPDX and CycloneDX formats, including variants like HBOM.5 Its core function is to validate BOMs against their declared schemas, ensuring data integrity and adherence to standards.5 Beyond validation, it provides capabilities for analyzing and editing BOM data, such as trimming unnecessary information, applying patches, and even performing experimental diff operations.5 The utility's powerful query command allows for the extraction and filtering of specific information from BOM documents, proving useful for targeted analysis and reporting.5

Microsoft's sbom-tool provides a dedicated solution for generating SPDX 2.2 compliant SBOMs.8 It leverages Microsoft's Component Detection libraries to identify software components and the ClearlyDefined API to gather crucial license information.8 Its integration capabilities with CI/CD pipelines through GitHub Actions and Azure DevOps make it a practical choice for automating SBOM generation as part of the software development process.8 The tool's support for multiple platforms, including Windows, Linux, macOS, and Docker, alongside its availability as a.NET tool, ensures broad applicability, particularly within.NET-centric environments.8

scanoss/sbom-workbench offers a graphical interface to facilitate the scanning and auditing of source code for license compliance, ultimately generating SPDX-Lite SBOMs.9 By utilizing the SCANOSS API, it simplifies the process of identifying open-source components within a codebase.9 While it produces SPDX-Lite, which might have limitations compared to the full SPDX or CycloneDX standards, its user-friendly GUI could be beneficial for manual audits and for users who prefer a visual approach to SBOM generation and license analysis.9 The tool also includes features for advanced settings and local cryptography detection, indicating a focus on providing granular control over the scanning process.9

**3.2. HBOM Generation Tools:**

While the search query specifically included HBOM generation, the research predominantly highlighted tools focused on SBOMs. However, sbom-utility explicitly states its support for all CycloneDX BOM variants, including HBOM.5 This suggests that for generating HBOMs, particularly in the CycloneDX format, sbom-utility is a viable open-source option. Other snippets related to HBOM in the initial search 10 did not directly identify active open-source tools dedicated to HBOM generation within the last six months. This indicates that while the need for HBOMs is growing, the open-source tooling in this specific area might be less mature compared to SBOMs.

**3.3. Software Composition Analysis (SCA) Tools:**

OWASP DependencyCheck is a well-established open-source SCA tool focused on identifying publicly disclosed vulnerabilities in a project's dependencies.17 It operates by checking if a dependency has a Common Platform Enumeration (CPE) identifier and then generates reports linking to associated Common Vulnerabilities and Exposures (CVE) entries.18 Its strength lies in its broad support for various platforms and codebases through plugins for popular build tools like Maven, Gradle, and Ant, as well as direct analysis capabilities for languages such as.NET, GoLang, Elixir, npm/pnpm/yarn, and Ruby.18 This wide-ranging support makes it a valuable tool for organizations with diverse technology stacks.

ScanCode-Toolkit distinguishes itself with its highly accurate license detection engine, which performs a full comparison between a database of license texts and the scanned code, rather than relying solely on pattern matching.19 It can detect licenses, copyrights, package manifests, and direct dependencies in both source code and binary files.19 Its extensive support for numerous package and build manifest formats, including those for Alpine, BUCK, Android, Autotools, Bazel, JavaScript, Java, Rust, and many others, makes it a comprehensive tool for understanding the composition and licensing of software projects.19

OpenSCA-cli aims to provide a comprehensive open-source solution for software supply chain security by detecting open-source dependencies, vulnerabilities, and ensuring license compliance.20 It achieves this by parsing configuration files of various package managers, supporting a wide array of languages including Java, JavaScript, PHP, Ruby, Golang, Rust, Erlang, and Python.20 The tool offers flexibility in reporting by supporting multiple formats such as JSON, XML, HTML, SPDX, CycloneDX, and SARIF.20 This broad language support and the ability to output in standard formats make OpenSCA-cli a strong contender for organizations seeking a unified SCA solution.

Retire.js is specifically designed to detect the use of JavaScript libraries with known vulnerabilities.22 It can be utilized in various ways, including as a command-line scanner, a Grunt or Gulp plugin, and even as browser extensions for Chrome and Firefox.22 Notably, Retire.js can also generate SBOMs in the CycloneDX format.22 Given the prevalence of JavaScript in modern web and Node.js applications, Retire.js provides a focused and valuable capability for identifying and mitigating vulnerabilities within this specific ecosystem.

bundler-audit focuses on providing patch-level verification for Ruby projects that utilize Bundler for dependency management.24 It checks a project's Gemfile.lock for vulnerable gem versions and also identifies insecure gem sources that use http:// or git://.24 A key advantage of bundler-audit is that it can perform these checks without requiring a network connection.24 For organizations with Ruby-based applications, bundler-audit offers a targeted and efficient way to ensure the security of their gem dependencies.

Dependency-Track stands out as an intelligent Component Analysis platform that leverages the capabilities of SBOMs to identify and reduce risk in the software supply chain.25 It consumes and produces CycloneDX SBOMs and supports a wide range of platforms and codebases, including applications, libraries, frameworks, operating systems, containers, firmware, hardware, and services.25 It integrates with multiple sources of vulnerability intelligence, such as the National Vulnerability Database (NVD) and GitHub Advisories, to provide comprehensive risk assessment.27 Dependency-Track's ability to track component usage across all applications within an organization's portfolio, coupled with its robust policy engine and notification capabilities, makes it a powerful central platform for managing software supply chain security.27

**3.4. GitHub's Built-in Security Features:**

GitHub offers several built-in security features that are relevant to SBOM, HBOM, and SCA. GitHub Advanced Security integrates SAST, SCA, and secret scanning directly into the platform.28 The Dependency Graph provides a summary of the manifest and lock files within a repository and can export dependencies as an SBOM in the SPDX format.3 Dependabot alerts users when their code depends on packages with known vulnerabilities and can even create pull requests to automatically update to secure versions.30 Code Scanning, utilizing GitHub's CodeQL analysis engine or third-party tools, can be configured to scan code for security vulnerabilities and coding errors.31 These built-in features offer a convenient and often seamless way to incorporate security analysis into the development workflow for projects hosted on GitHub.

**4. Proposed Solution Architecture for Comprehensive Security Analysis:**

To achieve comprehensive SBOM, SAST, and SCA coverage across the diverse technologies outlined in the user's query, a layered solution architecture is proposed, leveraging the strengths of the identified open-source GitHub projects.

**4.1. Components and their Roles:**

* **SBOM Generation Layer:**
  + **microsoft/sbom-tool:** To be utilized for generating SPDX SBOMs, particularly for.NET-based components and systems where the SPDX format is preferred or mandated. Its CI/CD integration will facilitate automated SBOM creation within Microsoft-centric development pipelines.
  + **sbom-utility:** To serve as a central tool for validating SBOMs generated by other tools, ensuring adherence to standards. Its capability to support HBOM as a CycloneDX variant makes it crucial for analyzing IoT devices and hardware components. It can also be used for converting SBOMs between SPDX and CycloneDX formats if needed for downstream processing.
  + **Retire.js:** To be employed specifically for JavaScript-heavy projects to generate CycloneDX SBOMs, providing detailed insights into the software bill of materials for this prevalent technology.
  + **GitHub's Dependency Graph:** To provide a baseline SPDX SBOM for repositories hosted on GitHub, offering a quick and integrated view of software dependencies.
* **SCA and Vulnerability Analysis Layer:**
  + **OWASP DependencyCheck:** To perform broad vulnerability scanning across a wide range of programming languages and build systems, complementing the more specialized tools. Its extensive CVE database integration will help identify known security flaws in dependencies.
  + **ScanCode-Toolkit:** To conduct in-depth license compliance analysis, ensuring that the use of open-source components adheres to organizational policies and legal requirements. Its detailed license detection capabilities will provide clarity on the licensing landscape of project dependencies.
  + **OpenSCA-cli:** To offer a combined solution for both vulnerability and license scanning, supporting many modern programming languages and providing output in standard formats like SPDX and CycloneDX. This tool can act as a versatile scanner across different project types.
  + **Retire.js:** To provide focused vulnerability detection specifically for JavaScript dependencies, leveraging its specialized knowledge of the JavaScript ecosystem.
  + **bundler-audit:** To ensure the security of Ruby projects by specifically checking for vulnerabilities in gem dependencies managed by Bundler.
  + **GitHub's Dependabot:** To provide continuous, automated vulnerability alerts and facilitate remediation through pull requests directly within the GitHub workflow, ensuring proactive management of dependency vulnerabilities.
* **SAST Layer:**
  + **GitHub's Code Scanning:** To be utilized with CodeQL for static analysis of the application's source code, identifying potential security vulnerabilities and coding errors before they are deployed.
* **Centralized Management and Analysis Layer:**
  + **Dependency-Track:** To serve as the central platform for aggregating and analyzing the outputs from all the other tools. It can import and store SBOMs in CycloneDX format (and potentially SPDX after conversion), consolidate vulnerability information from the various SCA tools, and provide a unified dashboard for risk management and mitigation. Its policy engine and reporting capabilities will offer a comprehensive view of the organization's software supply chain security posture.

**4.2. Data Flow and Interactions:**

The proposed architecture envisions an integration of these tools within the software development lifecycle, ideally within a CI/CD pipeline. Source code and build manifests would be the primary inputs for the SBOM generation and SCA tools. For instance, upon code commit or build initiation, tools like microsoft/sbom-tool, Retire.js, and GitHub's Dependency Graph could generate SBOMs. These SBOMs, along with the codebase, would then be analyzed by SCA tools like OWASP DependencyCheck, ScanCode-Toolkit, OpenSCA-cli, Retire.js, and bundler-audit. The findings from these analyses, along with the generated SBOMs (preferably in CycloneDX format), would be ingested into Dependency-Track. GitHub's Code Scanning would analyze the source code and report findings as alerts within the GitHub platform, which could potentially be integrated with Dependency-Track for a unified view. Dependabot would continuously monitor dependencies in GitHub repositories and raise alerts and pull requests as needed. sbom-utility could be used at various stages to validate SBOMs and potentially convert them to the preferred format for Dependency-Track.

**4.3. Coverage Considerations:**

The combination of these tools is designed to provide broad coverage across the user's specified technologies. For IoT devices, sbom-utility's HBOM support is critical. Cloud applications can be analyzed using the various SCA tools supporting languages like Java, Python, and JavaScript, which are commonly used in cloud environments. Docker containers can be assessed by tools that analyze container images (though not explicitly detailed for these specific tools, many SCA tools have container scanning capabilities that would need to be investigated further). Windows and Linux platforms are supported by several tools, including microsoft/sbom-tool, sbom-utility, OWASP DependencyCheck, and ScanCode-Toolkit. Popular programming languages such as Java, Python, JavaScript, C#, Go, C/C++, and Ruby are covered by one or more of the selected tools.

**4.4. Conceptual Diagram:**

\

*(Textual Description of the Diagram: The diagram illustrates a layered architecture. The first layer, "SBOM Generation," includes components like 'Source Code & Build Manifests' feeding into 'microsoft/sbom-tool' (outputting SPDX), 'sbom-utility' (supporting SPDX & CycloneDX/HBOM), 'Retire.js' (outputting CycloneDX), and 'GitHub Dependency Graph' (outputting SPDX). The output SBOMs then flow to the second layer, "SCA and Vulnerability Analysis," which includes 'OWASP DependencyCheck', 'ScanCode-Toolkit', 'OpenSCA-cli', 'Retire.js', 'bundler-audit', and 'GitHub Dependabot' (interacting directly with GitHub Repositories). The third layer, "SAST," shows 'Source Code' being analyzed by 'GitHub Code Scanning'. Finally, the SBOMs and findings from the SCA and SAST layers are ingested into the fourth layer, "Centralized Management and Analysis," represented by 'Dependency-Track', which provides a unified view of risks.)*

**5. Scoping Questionnaire for Implementation:**

To effectively implement a comprehensive security analysis solution tailored to specific needs, the following questionnaire is designed to gather essential information:

* **Product and System Types:**
  + What types of products or systems will be analyzed (e.g., IoT devices, cloud applications, desktop software, mobile apps)?
  + Are there specific categories or classifications of products within your organization?
* **Platforms and Operating Systems:**
  + What platforms and operating systems are involved (e.g., Linux distributions, Windows versions, specific IoT platforms like embedded Linux, RTOS)?
  + Are there specific hardware architectures to consider for HBOM?
* **Programming Languages and Technologies:**
  + What programming languages are primarily used in your codebases (e.g., Java, Python, JavaScript, C#, Go, C/C++, Ruby)?
  + What package managers and build tools are used (e.g., Maven, Gradle, npm, pip, NuGet, Go modules, Cargo, Bundler)?
  + Are Docker or other container technologies used?
  + Are there any specific frameworks or libraries that are heavily relied upon?
* **SBOM and HBOM Requirements:**
  + Are there specific requirements for SBOM formats (e.g., SPDX, CycloneDX)?
  + Are there any specific standards or guidelines to adhere to for SBOM or HBOM content (e.g., NTIA minimum elements)?
  + Is the generation of HBOM a critical requirement?
* **Analysis Scope and Frequency:**
  + What level of detail is required for the analysis (e.g., direct dependencies only, transitive dependencies)?
  + How frequently should the analysis be performed (e.g., daily, weekly, with each code change, upon release)?
  + Are there specific triggers for initiating the analysis process?
* **Existing Security Tools and Processes:**
  + What security tools or processes are currently in place for software development and supply chain security?
  + Are there existing CI/CD pipelines that the solution needs to integrate with?
  + What is the current level of maturity in terms of security practices?
* **Compliance and Regulatory Requirements:**
  + Are there any specific compliance or regulatory requirements that need to be met (e.g., GDPR, HIPAA, industry-specific regulations)?
  + Do these requirements have specific implications for SBOM, SCA, or SAST?

**6. Level of Effort Assessment Framework:**

Assessing the Level of Effort (LOE) for implementing and maintaining the proposed security analysis solution requires consideration of several key factors. The initial setup and configuration of each selected tool will involve effort for installation, integration with existing systems (like CI/CD pipelines), and customization of rules and policies. The degree of automation achievable with each tool will significantly impact the ongoing LOE. While tools like GitHub's Dependabot offer a high level of automation for vulnerability alerts and remediation, others might require more manual intervention for initial configuration and triaging of findings.1

The size and complexity of the codebase, including the number of repositories and lines of code, will influence scanning times and the volume of results, thus affecting the effort required for analysis and review. Similarly, the number and types of dependencies will impact scanning duration and the complexity of the SBOMs generated. Dependencies on less common or proprietary components might necessitate manual analysis and investigation.

Manual review of findings is a critical aspect of ensuring the accuracy and relevance of the security analysis. Automated tools can produce false positives, and understanding the context of a vulnerability often requires human expertise. Therefore, the effort involved in security analysts or developers triaging and validating findings must be accounted for. Finally, ongoing maintenance and updates are essential to keep the tools effective. This includes updating vulnerability databases, software versions, and maintaining any integration scripts or workflows. Periodic review and tuning of tool configurations and policies will also be necessary to ensure the solution remains aligned with evolving security requirements.

A phased approach to implementation is recommended, starting with a pilot project to evaluate the selected tools and refine the integration process. Estimating the LOE for each phase, considering the factors outlined above and involving security, development, and operations teams, will provide a more accurate assessment. Metrics such as the number of dependencies, codebase size, and the planned frequency of analysis can be used to quantify the effort. For each tool, the assessment should consider the automation capabilities for tasks like SBOM generation, vulnerability scanning, and reporting, as well as the estimated manual effort for initial setup, rule customization, and triaging findings.

**7. Conclusion and Recommendations:**

This research has identified a set of active open-source GitHub projects that offer valuable capabilities for SBOM and HBOM generation, as well as Software Composition Analysis. The proposed solution architecture, which combines the strengths of tools like microsoft/sbom-tool, sbom-utility, OWASP DependencyCheck, ScanCode-Toolkit, OpenSCA-cli, Retire.js, bundler-audit, Dependency-Track, and GitHub's built-in security features, provides a robust framework for achieving comprehensive security analysis across diverse technologies.

To move forward with implementing this solution, it is recommended that the user:

* Prioritize the selection of specific tools based on the detailed needs and the technology stack, utilizing the provided scoping questionnaire to gather crucial requirements.
* Conduct a pilot implementation with a representative project to evaluate the chosen tools in a real-world scenario and to refine the integration processes.
* Develop a detailed implementation plan and timeline, outlining the steps for tool setup, integration, configuration, and team training.
* Allocate adequate resources for the initial implementation phase, as well as for the ongoing maintenance and operation of the security analysis solution.

By adopting a comprehensive and integrated approach to SBOM, HBOM, and SCA, organizations can significantly enhance their software supply chain security posture, reduce the risk of vulnerabilities, and ensure compliance with relevant regulations. Continuous monitoring and a commitment to ongoing improvement will be essential to maximize the effectiveness of this solution over time.

#### Works cited

1. What is software composition analysis (SCA)? - GitHub, accessed March 31, 2025, <https://github.com/resources/articles/security/what-is-software-composition-analysis>
2. What Is SCA? Software Composition Analysis Guide - Checkmarx, accessed March 31, 2025, <https://checkmarx.com/glossary/software-composition-analysis-sca/>
3. Exporting a software bill of materials for your repository - GitHub Docs, accessed March 31, 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. SBOM Challenges for Developers: From Analysis of Stack Overflow Questions - arXiv, accessed March 31, 2025, <https://arxiv.org/pdf/2502.03975>
5. CycloneDX/sbom-utility: Utility that provides an API platform ... - GitHub, accessed March 31, 2025, <https://github.com/CycloneDX/sbom-utility>
6. Top 10 Software Composition Analysis (SCA) tools in 2025 - Aikido Security, accessed March 31, 2025, <https://www.aikido.dev/blog/top-10-software-composition-analysis-sca-tools-in-2025>
7. bomctl/bomctl: Format agnostic SBOM tooling - GitHub, accessed March 31, 2025, <https://github.com/bomctl/bomctl>
8. microsoft/sbom-tool: The SBOM tool is a highly scalable ... - GitHub, accessed March 31, 2025, <https://github.com/microsoft/sbom-tool>
9. The SCANOSS SBOM Workbench graphical user interface to scan and audit your source code. - GitHub, accessed March 31, 2025, <https://github.com/scanoss/sbom-workbench>
10. cipher387/osint\_stuff\_tool\_collection: A collection of several hundred online tools for OSINT - GitHub, accessed March 31, 2025, <https://github.com/cipher387/osint_stuff_tool_collection>
11. An empirical analysis of GitHub Copilot - DiVA portal, accessed March 31, 2025, <https://www.diva-portal.org/smash/get/diva2:1775041/FULLTEXT01.pdf>
12. GitHub Codespaces - Hacker News, accessed March 31, 2025, <https://news.ycombinator.com/item?id=23092904>
13. Quarterly Progress Reports To iDigBio Submitted By Active Thematic Collections Networks (TCNs) November 2022, accessed March 31, 2025, <https://www.idigbio.org/wiki/images/3/36/TCN_Updates_2022.11.pdf>
14. CyberMonitor/APT\_CyberCriminal\_Campagin\_Collections: APT & CyberCriminal Campaign Collection - GitHub, accessed March 31, 2025, <https://github.com/CyberMonitor/APT_CyberCriminal_Campagin_Collections>
15. sec-tools/README.md at main - GitHub, accessed March 31, 2025, <https://github.com/anquanscan/sec-tools/blob/main/README.md>
16. DataTurks/hope/docker/mysqlInit.sql at master - GitHub, accessed March 31, 2025, <https://github.com/DataTurks/DataTurks/blob/master/hope/docker/mysqlInit.sql>
17. jeremylong/DependencyCheck: The dependency-check ... - GitHub, accessed March 31, 2025, <https://github.com/jeremylong/DependencyCheck>
18. dependency-check/DependencyCheck: OWASP ... - GitHub, accessed March 31, 2025, <https://github.com/dependency-check/DependencyCheck>
19. aboutcode-org/scancode-toolkit: :mag: ScanCode detects ... - GitHub, accessed March 31, 2025, <https://github.com/aboutcode-org/scancode-toolkit>
20. XmirrorSecurity/OpenSCA-cli: OpenSCA is an open source software supply chain security solution that supports the detection of open source dependencies, vulnerabilities and license compliance with a widely noticed accuracy by the community. - GitHub, accessed March 31, 2025, <https://github.com/XmirrorSecurity/OpenSCA-cli>
21. opensca/opensca-cli - Docker Image - Docker Hub, accessed March 31, 2025, <https://hub.docker.com/r/opensca/opensca-cli>
22. RetireJS/retire.js: scanner detecting the use of JavaScript libraries with known vulnerabilities. Can also generate an SBOM of the libraries it finds. - GitHub, accessed March 31, 2025, <https://github.com/retirejs/retire.js/>
23. Retire.js, accessed March 31, 2025, <https://retirejs.github.io/>
24. rubysec/bundler-audit: Patch-level verification for Bundler - GitHub, accessed March 31, 2025, <https://github.com/rubysec/bundler-audit>
25. DependencyTrack/dependency-track: Dependency-Track is ... - GitHub, accessed March 31, 2025, <https://github.com/DependencyTrack/dependency-track>
26. Dependency-Track - GitHub, accessed March 31, 2025, <https://github.com/DependencyTrack>
27. OWASP Dependency-Track, accessed March 31, 2025, <https://owasp.org/www-project-dependency-track/>
28. Snyk vs GitHub Advanced Security vs Cycode: 3 Key Differences, Pros & Cons, and How to Choose the Best Solution, accessed March 31, 2025, <https://cycode.com/blog/snyk-vs-github-advanced-security-3-key-differences/>
29. Configuring default setup for code scanning - GitHub Docs, accessed March 31, 2025, <https://docs.github.com/code-security/secure-coding/setting-up-code-scanning-for-a-repository>
30. About Dependabot alerts - GitHub Docs, accessed March 31, 2025, <https://docs.github.com/code-security/dependabot/dependabot-alerts/about-dependabot-alerts>
31. About code scanning - GitHub Docs, accessed March 31, 2025, <https://docs.github.com/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning>
32. Introduction to code scanning - GitHub Docs, accessed March 31, 2025, <https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning>