Cheat Sheet: 10 key requirements for choosing a Software Composition Analysis (SCA) tool

snyk

1. Developer-focused

If developers aren't using an SCA tool because it slows them down or hampers development, it will simply not be adopted, and hence won't deliver the promised security benefits.
A developer-friendly SCA must:

- Be intuitive and easy to set up and use.
- Integrate with existing development workflows and tools such as Git and IDEs **in a frictionless manner.**
- Be automated and actionable - not only surface issues but also guide developers on the path to remediation with automation and actionable fix advice.

2. Ecosystem support & integrations

An SCA tool must be able to cover the core programming languages and frameworks used to build your applications. Likewise, it should provide integrations across the SDLC. Be sure that the tool provides both an easy integration and one that actually provides results as expected.

Tip! Focus on tools that provide deep support for the core languages used in your development.

3. Dependency analysis

80% of vulnerabilities in open source packages are identified in transitive dependencies! This means that the vast majority of vulnerabilities in your code base are introduced by dependencies you had no idea you were using in the first place. **An SCA tool must be able to accurately interpret all the dependencies in an application to provide practitioners with full visibility.**

4. Accurate & actionable vulnerability detection

The ability of an SCA tool to accurately identify whether an open source package contains vulnerabilities or not depends on the security data it relies on. Be sure the vulnerability data the SCA tool relies on is first and foremost **accurate** but also **comprehensive**, **timely**, and **actionable**.

5. Prioritization

SCA tools will often identify hundreds if not thousands of issues that quickly pile up into backlogs that can easily overwhelm teams. Select an SCA tool that helps you prioritize where best to invest resources for the best impact. Verify the tool can:

- Go beyond CVSS scoring for assessing risk.
- Provide deep application-level context on vulnerabilities. Is a vulnerability reachable? Is it exploitable?
- Automate prioritization across projects and teams.

6. Remediation

Dive deeper into the remediation advice the SCA tool provides around a vulnerability and the workflows it supports to drive actionability. Is there enough information available for understanding where and how to apply a fix? Are automated workflows available?

7. Governance & control

Does the SCA tool provide you with the control you need to control the use of open source in your applications? At the very least, ensure the SCA tool provides policies for defining and automatically enforcing the security and compliance guidelines accepted by your organization.

8. Reporting

Keeping track of the various open source packages being used as well the licences they include is important for various reasons and different business stakeholders. Verify an SCA tool provides you with oversight to track your posture over time and enables you to generate, and share, a BoM report on your open source inventory.

9. Automation & extensibility

The larger you grow, the more challenging it is to perform all the manual operations involved in SCA processes. The ability to automate tasks such as adding new projects and users, or scanning new builds, drives efficiency and also helps reduce friction with existing development workflows. **Verify the SCA tool provides you with a robust API that enables the automation, customization and integration of SCA processes into your existing workflows and systems.**

10. Cloud native applications

Remember, applications today are assembled from containers and other components, extending the original definition of SCA and defining a new set of capabilities needed. **Check if the SCA tool can identify security vulnerabilities in container images and infrastructure as code, as well as help drive remediation.**

See Snyk in Action