SAST vs SCA vs DAST

---

## 🧱 1. Static Application Security Testing (SAST)

➡️ **Phase:** *Code / Build (Pre-deployment)*
➡️ **Approach:** "**White-box testing**" – analyzes the source code without executing it.

---

### 🔍 Purpose

SAST examines your **source code, bytecode, or binaries** for security vulnerabilities before the application runs.
It helps developers **"shift security left"**, finding issues early in the development lifecycle.

---

### 🧠 How It Works

- Scans **static source code** line by line.
- Matches patterns against **known insecure coding practices** (e.g., SQL injection, XSS, insecure cryptography).
- Integrates directly into **IDE or CI/CD pipeline** so developers can fix issues instantly.

---

### 🧩 Common Issues Detected

- Hardcoded passwords or API keys

- SQL Injection
- Cross-Site Scripting (XSS)
- Buffer overflows
- Insecure input validation
- Use of weak or deprecated cryptography

---

### 🧰 Popular SAST Tools

| Tool | Description |
|---|---|
| **SonarQube** | Open-source code quality & security scanner integrated with CI tools. |
| **Checkmarx** | Enterprise-grade static code analysis with policy enforcement. |
| **Snyk Code** | Cloud-native developer-first tool with real-time feedback. |
| **Fortify Static Code Analyzer** | Deep static analysis for enterprise environments. |
| **Semgrep** | Lightweight open-source static analysis using rule-based scanning. |

---

### ⚙️ Best Practices

- Run SAST at **every code commit or pull request**.

- Automate scans via **Jenkins, GitHub Actions, or GitLab CI**.
- Define **severity thresholds** (block build on critical issues).
- Educate developers on **secure coding guidelines**.

---

✅ **Outcome**

- Detects **vulnerabilities early** in the SDLC.
- Reduces cost and effort for later-stage fixes.
- Improves **code security hygiene** and **developer awareness**.

---

📦 **2. Software Composition Analysis (SCA)**

➡️ **Phase:** *Build (Dependency & library validation)*
➡️ **Approach:** Analyzes **open-source components** used in your app.

---

🔍 **Purpose**

Modern apps rely heavily on **third-party libraries** (npm, pip, Maven, etc.).
SCA identifies **known vulnerabilities**, **outdated versions**, and **license risks** in these components.

---

🧠 **How It Works**

- Scans package.json, pom.xml, requirements.txt, or similar dependency manifests.
- Compares package versions against **CVEs (Common Vulnerabilities and Exposures)** databases like NVD.
- Flags outdated or vulnerable dependencies and suggests secure versions.
- Monitors for **license compliance issues** (e.g., GPL vs MIT).

---

### 🧩 Common Issues Detected

- Use of vulnerable open-source libraries (e.g., Log4j vulnerability).
- Unpatched dependencies.
- License violations (restrictive licenses in commercial software).
- Transitive dependency risks (nested libraries).

---

### 🧰 Popular SCA Tools

| Tool | Description |
|---|---|
| **Snyk Open Source** | Scans dependencies in real time; integrates with GitHub, GitLab, IDEs. |
| **OWASP Dependency-Check** | Free and open-source SCA tool that checks for CVEs. |
| **WhiteSource / Mend** | Enterprise-grade component risk management. |

| | |
|---|---|
| **JFrog Xray** | Integrates with Artifactory for binary and dependency scanning. |
| **Anchore** | Focuses on container and image dependency vulnerabilities. |

---

### ⚙️ Best Practices

- Include SCA as part of **CI/CD build pipeline**.
- Automate **vulnerability alerting** via Slack or email.
- Continuously **update dependencies** (apply patch management).
- Review **license policies** before using open-source libraries.

---

### ✅ Outcome

- Ensures **supply chain security**.
- Prevents exploitation through **known vulnerabilities**.
- Enables **legal compliance** with licensing.

---

### 🌐 3. Dynamic Application Security Testing (DAST)

➡️ **Phase:** *Test / Deploy (Runtime testing)*
➡️ **Approach:** "**Black-box testing**" – tests a running application externally, like a hacker would.

---

### 🔍 Purpose

DAST tests **live applications** by simulating real-world attacks.
It looks for vulnerabilities that appear **only at runtime** — such as broken authentication, input validation, and logic flaws.

---

### 🧠 How It Works

- Scans **web app endpoints, APIs, or URLs** while the app is running.
- Sends **malicious payloads** to detect exploitable behaviors.
- Observes responses to identify weaknesses like injection or insecure redirects.

---

### 🧩 Common Issues Detected

- Cross-Site Scripting (XSS)
- SQL Injection
- Cross-Site Request Forgery (CSRF)
- Authentication & session management flaws
- Server misconfigurations
- Broken access control

---

## 🧰 Popular DAST Tools

| Tool | Description |
|---|---|
| **OWASP ZAP** | Free, open-source scanner for web apps; integrates with CI/CD. |
| **Burp Suite** | Professional-grade web vulnerability scanner (manual + automated). |
| **Netsparker** | Commercial DAST with advanced automation. |
| **Acunetix** | Automated web security testing for APIs and web apps. |
| **AppScan (IBM)** | Enterprise-level DAST integrated with CI/CD and reporting. |

---

## ⚙️ Best Practices

- Run DAST scans in **staging environments** (never on production).
- Automate post-deployment scans in CI/CD.
- Combine results with **SAST and SCA** for better context.
- Use authenticated scans for deeper coverage (login sessions).

---

## ✅ Outcome

- Detects **runtime vulnerabilities** that SAST can't find.
- Provides real-world attack simulation.

- Enhances **defense-in-depth** by verifying production readiness.

---

### 🧩 Summary Table

| Testing Type | Stage | Type | Focus | Common Tools |
|---|---|---|---|---|
| **SAST** | Code / Build | Static | Source code vulnerabilities | SonarQube, Checkmarx, Fortify |
| **SCA** | Build | Composition | Vulnerable dependencies, licensing | Snyk, OWASP Dependency-Check, JFrog Xray |
| **DAST** | Test / Deploy | Dynamic | Runtime / behavioral vulnerabilities | OWASP ZAP, Burp Suite, Acunetix |

---

### 🔒 Combined Value in DevSecOps

| Integration | Purpose |
|---|---|
| **SAST + SCA** | Secure your code and third-party dependencies early in CI. |
| **DAST + SAST** | Cover both static (code) and dynamic (runtime) weaknesses. |
| **SAST + SCA + DAST** | Full-spectrum AppSec coverage across SDLC — from code to runtime. |

---