Static application security testing (SAST) is a set of technologies designed to analyze application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities. SAST solutions analyze an application from the “inside out” in a non-running state.

There are six simple steps needed to perform SAST efficiently in organizations that have a very large number of applications built with different languages, frameworks, and platforms.

1. **Finalize the tool.** Select a static analysis tool that can perform code reviews of applications written in the programming languages you use. The tool should also be able to comprehend the underlying framework used by your software.
2. **Create the scanning infrastructure, and deploy the tool.** This step involves handling the licensing requirements, setting up access control and authorization, and procuring the resources required (e.g., servers and databases) to deploy the tool.
3. **Customize the tool.** Fine-tune the tool to suit the needs of the organization. For example, you might configure it to reduce false positives or find additional security vulnerabilities by writing new rules or updating existing ones. Integrate the tool into the build environment, create dashboards for tracking scan results, and build custom reports.
4. **Prioritize and onboard applications.** Once the tool is ready, onboard your applications. If you have a large number of applications, prioritize the high-risk applications to scan first. Eventually, all your applications should be on boarded and scanned regularly, with application scans synced with release cycles, daily or monthly builds, or code check-ins.
5. **Analyze scan results.** This step involves triaging the results of the scan to remove false positives. Once the set of issues is finalized, they should be tracked and provided to the deployment teams for proper and timely remediation.
6. **Provide governance and training.** Proper governance ensures that your development teams are employing the scanning tools properly. The software security touchpoints should be present within the SDLC. SAST should be incorporated as part of your application development and deployment process.

Dynamic application security testing (DAST) is a program used by developers to analyze a web application (web app), while in runtime, and identify any security vulnerabilities or weaknesses. Using DAST, a tester examines an application while it's working and attempts to attack it as a hacker would. DAST tools provide beneficial information to developers about how the app behaves, allowing them to identify where a hacker might be able to stage an attack, and eliminate the threat.

DAST is a **black box test**, meaning it is performed from the outside of the application, without a view into the internal source code or app architecture. As a result, the test identifies vulnerabilities by using the same techniques a hacker would and performing attacks on the software. A DAST will employ a fault injection technique, like inputting malware into the software, to uncover threats such as cross-site scripting (XSS) or SQL injection (SQLi).

DAST tools will continuously scan apps during and after development. The DAST scanners crawl through a web app before scanning it. This first step allows the DAST tool to find every exposed input on pages within the app and then test each one.

The tests that are done after the app has been executed are fully automated and allow businesses to immediately identify and resolve any risks before they become serious attacks. Once a vulnerability is discovered, a DAST solution will send an automated alert to the appropriate team of developers so they can remediate it.

There are three ways to use DAST:

Testing early and often in the software development life cycle (SDLC) in collaboration with DevOps -- DAST identifies the problem and DevOps fixes it and in conjunction with other tests as part of a comprehensive approach to web security.

IAST shifts testing left in the SDLC. IAST generally takes place during the test/QA stage of the software development life cycle (SDLC). IAST effectively shifts testing left, so problems are caught earlier in the development cycle, reducing remediation costs and delays. Many tools can be integrated into continuous integration (CI) and continuous development (CD) tools. The latest-generation tools return results as soon as changed code is recompiled and the running app retested, helping developers identify vulnerabilities even earlier in the development process.

Key steps to run IAST effectively:

1. **Deploy DevOps.** IAST requires integration into your CI/CD environment.
2. **Choose your tool.** Select a tool that can perform code reviews of applications written in the programming languages you use and that is compatible with the underlying framework used by your software.
3. **Create the scanning infrastructure and deploy the tool.** Set up access control, authorization, and any integrations required, such as Jira for bug tracking, to deploy the tool.
4. **Customize the tool.** Fine-tune the tool to suit the needs of your organization. Integrate the tool into the build environment, create dashboards for tracking scan results, and build custom reports.
5. **Prioritize and add applications.** Once the tool is ready, add your applications. If you have many applications, prioritize the high-risk web applications to scan first
6. **Analyze scan results.** Triage your scan results to remove false positives. Track and remediate any vulnerability issues as early in the SDLC as possible.
7. **Provide training.** Train your development and security teams on how to use the results from the IAST tool effectively and how to incorporate them into the application development and deployment process.

Runtime Application Self-Protection (RASP) is a technology that runs on a server and kicks in when an application runs. It's designed to detect attacks on an application in real time. When an application begins to run, RASP can protect it from malicious input or behavior by analyzing both the app's behavior and the context of that behavior. By using the app to continuously monitor its own behavior, attacks can be identified and mitigated immediately without human intervention.

When a security event in an app occurs, RASP takes control of the app and addresses the problem. In diagnostic mode, RASP will just sound an alarm that something is amiss. In protection mode, it will try to stop it. For example, it could stop the execution of instructions to a database that appear to be a SQL injection attack.

Other actions RASP could take include terminating a user's session, stopping an application's execution, or alerting the user or security personnel.

Developers can implement RASP in a couple of ways. They can access the technology through function calls included in an app's source code, or they can take a completed app and put it in a wrapper that allows the app to be secured with a single button push. The first approach is more precise because developers can make specific decisions about what they want protected in the app, such as logins, database queries, and administrative functions.

Whichever method is used with RASP, the end result is like bundling a web application firewall with the application's runtime context. That close connection to the app means RASP can be more finely tuned to the app's security needs.