Practical DevSecOps

Comprehensive Guide to SAST Implementation

A Step-by-Step Guide To Implement Static Application Security Testing (SAST)

practical-devsecops.com

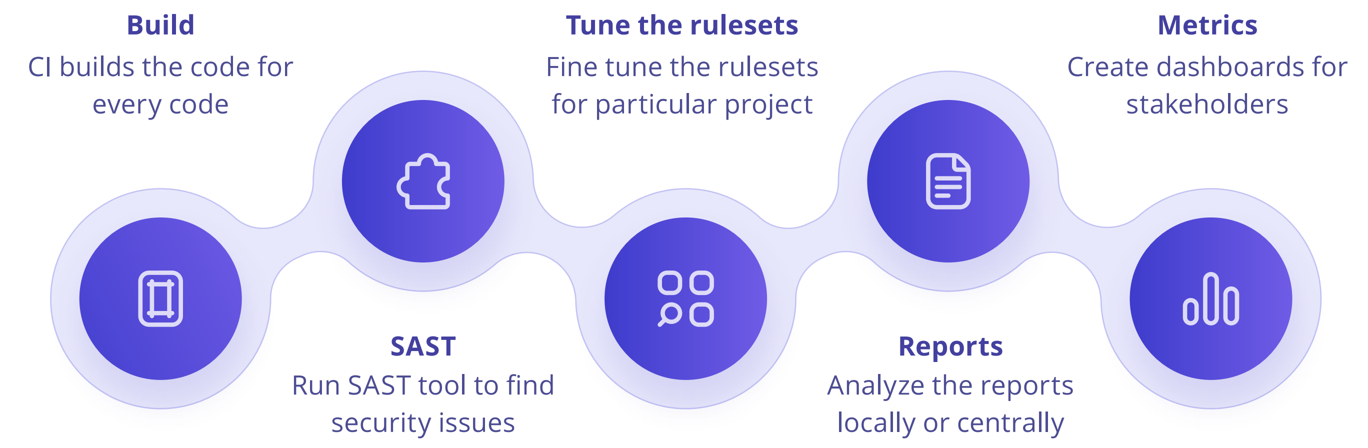
CONTENTS

CHAPTER 1

Introduction to SAST and How It Works

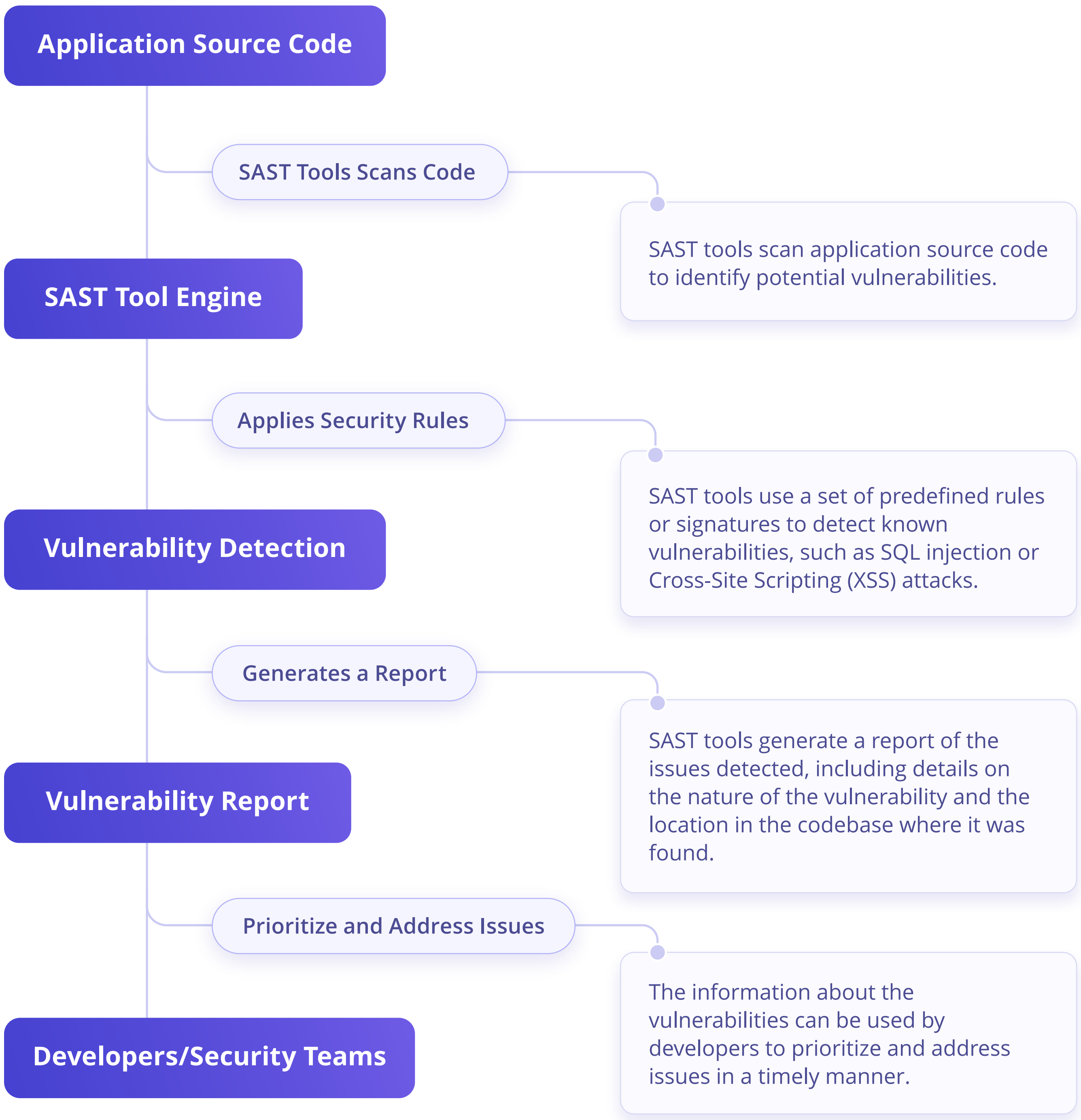
As software development becomes increasingly complex and fast-paced, ensuring the security of your code is more important than ever. Static Application Security Testing (SAST) is one of the key tools for achieving this goal.

SAST is a type of security testing that analyzes application source code and identifies potential vulnerabilities. Unlike Dynamic Application Security Testing (DAST), which tests applications while they are running, SAST focuses on identifying issues within the code itself. This approach allows developers to detect and address vulnerabilities early in the development process, reducing the likelihood of security issues arising in production.


Build
CI builds the code for every code
Tune the rulesets
Fine tune the rulesets for particular project
Metrics
Create dashboards for stakeholders
SAST
Run SAST tool to find security issues
Reports
Analyze the reports locally or centrally

SAST - Workflow

How Does SAST Work?



SAST tools scan application source code to identify potential vulnerabilities.

SAST tools use a set of predefined rules or signatures to detect known vulnerabilities, such as SQL injection or Cross-Site Scripting (XSS) attacks.

SAST tools generate a report of the issues detected, including details on the nature of the vulnerability and the location in the codebase where it was found.

The information about the vulnerabilities can be used by developers to prioritize and address issues in a timely manner.

NOTE

SAST can also identify coding practices that are not necessarily vulnerabilities but may lead to potential security issues.

CHAPTER 2

Understanding Your Codebase

Before implementing a SAST workflow, it is essential to have a good understanding of the codebase.

Step 1: Assessing the Scope of the Codebase

Identify all the applications and modules that need to be scanned. Ensure all relevant code is included in the scan to identify all potential vulnerabilities.

Step 2: Identify Key Components and Libraries in the Codebase

Identify third-party libraries and frameworks that the application relies on.

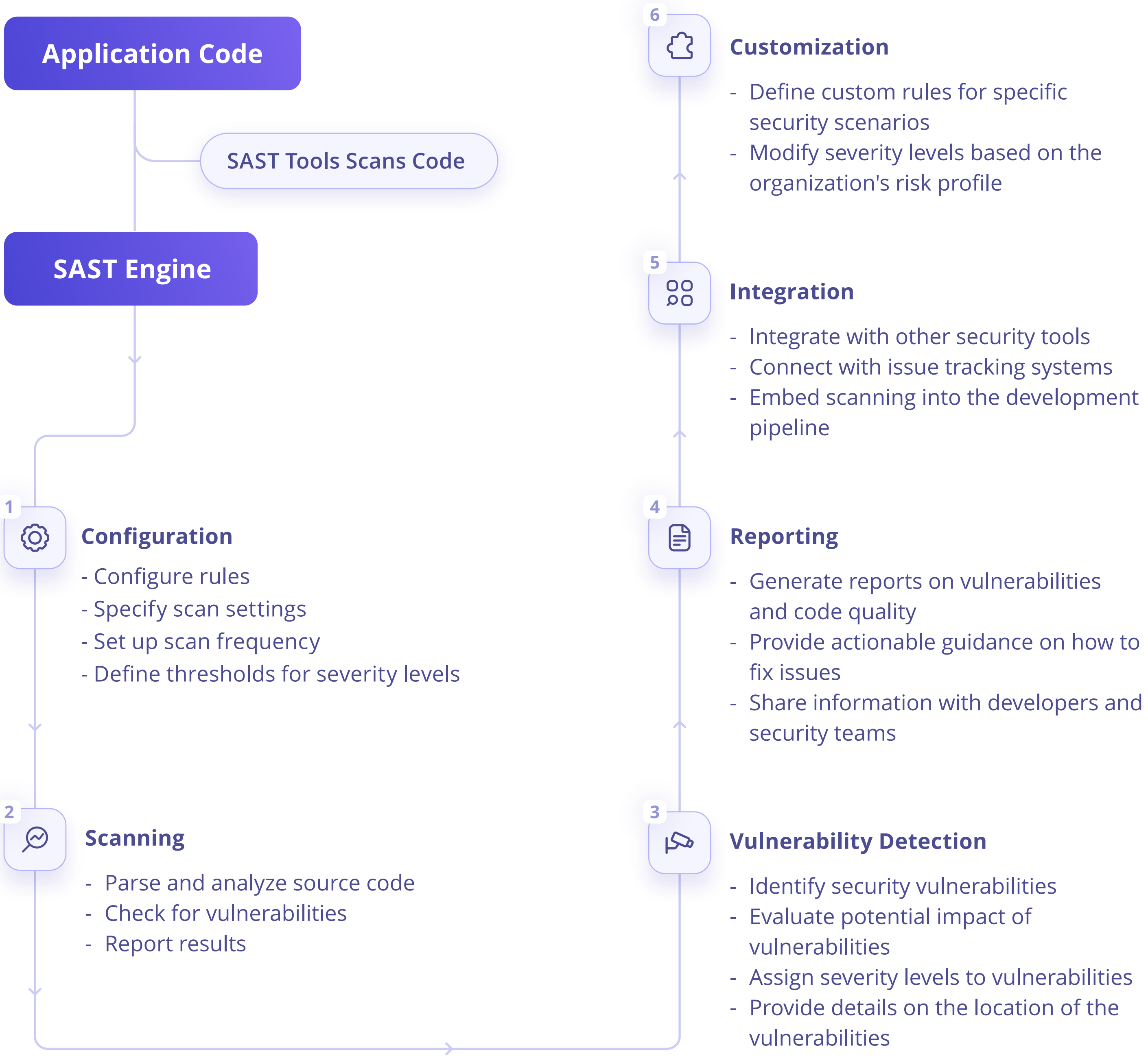
"By assessing the scope of the codebase, and identifying key components and libraries, it is possible to understand the codebase better and implement an effective SAST workflow.

CHAPTER 3

Setting Up Your SAST Environment

Before implementing SAST, setting up an appropriate environment for running scans is essential.

What does it entail to setup a SAST environment?



1 **Configuration**

- Configure rules
- Specify scan settings
- Set up scan frequency
- Define thresholds for severity levels

2 **Scanning**

- Parse and analyze source code
- Check for vulnerabilities
- Report results

3 **Vulnerability Detection**

- Identify security vulnerabilities
- Evaluate potential impact of vulnerabilities
- Assign severity levels to vulnerabilities
- Provide details on the location of the vulnerabilities

4 **Reporting**

- Generate reports on vulnerabilities and code quality
- Provide actionable guidance on how to fix issues
- Share information with developers and security teams

5 **Integration**

- Integrate with other security tools
- Connect with issue tracking systems
- Embed scanning into the development pipeline

6 **Customization**

- Define custom rules for specific security scenarios
- Modify severity levels based on the organization's risk profile

Step 1: Define the Scope of the SAST Environment

Defining the scope of the SAST program is essential to tailor it to an organization's needs and ensure effective vulnerability identification.

- Identify the applications and codebases to be scanned.
- Determine the frequency of scans.

Step 2: Selecting Your SAST Tool

Select the appropriate SAST tool for your organization's needs and consider the following factors:

- SAST tool's support for scanning the programming languages in the codebase.
- The level of expertise required to use the tool.
- Choose a tool that fits your organization's needs and budget.
- Research and compare available SAST tools.
- If there is time, a Proof of Concept with your shortlisted SAST tools on your codebase would reveal the actual effectiveness of a tool.
- There are plenty of other considerations in selecting a SAST tool, some of which are discussed in Chapter 4, "Pairing SAST Tools With CI/CD Pipelines"

Certified DevSecOps Professional (CDP)

Here are some of the most widely used SAST tools

Tool Name	Category	Description
SonarQube	Quality & Security	Open-source platform for continuous code quality and security analysis of Java, C++, C#, Python, and more.
Checkmarx CxSAST	Enterprise	A comprehensive and scalable SAST solution that covers multiple programming languages and offers various integration options with development environments.
Fortify Static Code Analyzer	Enterprise	Offers powerful SAST capabilities for identifying vulnerabilities in a wide range of programming languages, including Java, .NET, and C/C++.
Veracode Static Analysis	Enterprise	Delivers SAST in the cloud, offering in-depth and precise coverage of code security weaknesses while remaining easy to use and accessible.
CodeSonar	Advanced Analysis	Offers advanced capabilities in both SAST and DAST that enable users to quickly find and eliminate vulnerabilities in their code.
ESLint	JavaScript	Open-source SAST tool that can be easily integrated with JavaScript development environments, providing automatic error reporting and suggestions for fixing security issues.
Infer	Mobile	Open-source tool that analyzes Objective-C and Swift code for iOS applications, focusing on detecting memory leaks, null pointers, and other common vulnerabilities.

Step 3: Installing and Configuring Your SAST Tool

Installing and configuring a SAST tool for your environment may involve installing additional plugins or dependencies to ensure the tool can run effectively, and efficiently.

- Install the SAST tool on appropriate machines.
- Configure the tool to match your organization's development environment.

Step 4: Integrate With Your Build Process

Integrate the SAST tool with your build process to automate scanning.

SAST tools can be configured to run on Continuous Integration builds, nightly builds, scheduled builds, and continuous delivery pipelines as well. Choosing the point of integration in the build system for a SAST tool depends on the size of the codebase, the time the SAST tool takes for a scan, the rule sets that are present in the SAST tool, and many other factors that are directly proportional to the efficiency and effectiveness of a scan.

Also, set up notifications and alerts for scan results ranging from reports being sent to email addresses, chat systems, collaboration systems, and other places where developers, and operations flock together.

Step 5: Configuring Your Scan Settings

Before running your first SAST scan, it is essential to configure your scan settings appropriately, which includes:

- Defining the scope of the scan, for example, the project or the folder that needs scanning.
- Setting up authentication for accessing the codebase.
- Defining any custom rules or signatures you want the tool to use.
- Define scan policies to ensure scans are consistent and thorough.
- Optionally, you can also determine how to handle false positives and negatives.

Step 6: Setting Up Your SAST Server

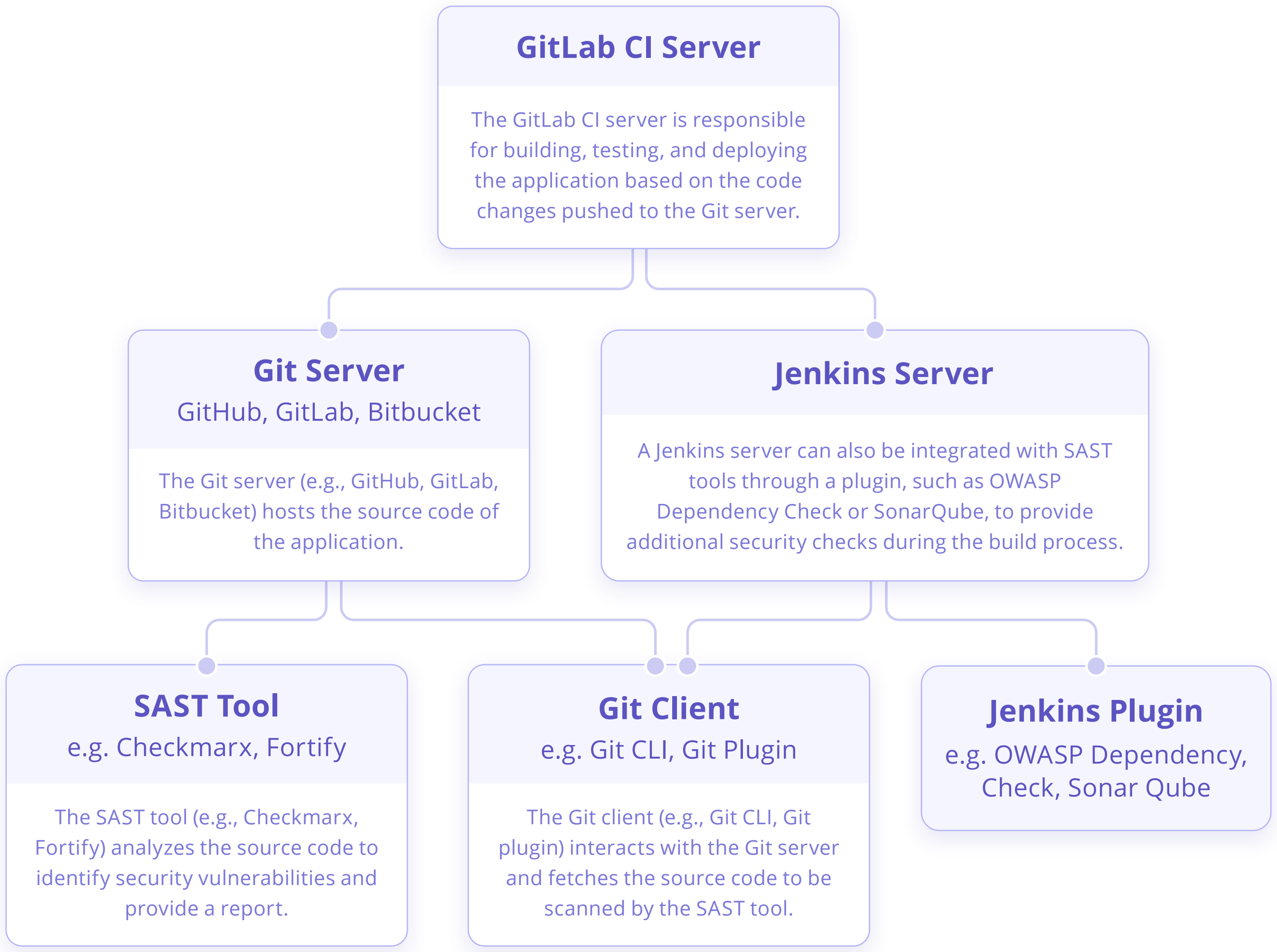
To ensure that your SAST scans can run efficiently and effectively, setting up a dedicated SAST server is important.

The server where a SAST tools is planned to be installed, should have adequate processing power, and memory to handle the workload of running frequent scans on your codebase.

Step 7: Setting Up Your SAST Pipeline

Finally, to ensure that your SAST workflow runs smoothly, it is important to set up a pipeline that automates running scans and analyzing results.

Setting up a SAST pipeline can involve using tools such as Jenkins or GitLab to automate the scanning process and trigger alerts when potential vulnerabilities are detected.


GitLab CI Server
The GitLab CI server is responsible for building, testing, and deploying the application based on the code changes pushed to the Git server.
Git Server
GitHub, GitLab, Bitbucket
The Git server (e.g., GitHub, GitLab, Bitbucket) hosts the source code of the application.
Jenkins Server
A Jenkins server can also be integrated with SAST tools through a plugin, such as OWASP Dependency Check or SonarQube, to provide additional security checks during the build process.
SAST Tool
e.g. Checkmarx, Fortify
The SAST tool (e.g., Checkmarx, Fortify) analyzes the source code to identify security vulnerabilities and provide a report.
Git Client
e.g. Git CLI, Git Plugin
The Git client (e.g., Git CLI, Git plugin) interacts with the Git server and fetches the source code to be scanned by the SAST tool.
Jenkins Plugin
e.g. OWASP Dependency, Check, Sonar Qube

Overall, such integration helps to ensure that the application code is secure and free of vulnerabilities before it is deployed to production.

Step 8: Train Developers on SAST Usage

- Provide training to developers on how to use the SAST tool.
- Communicate the importance of SAST in the development process.

CHAPTER 4

Pairing SAST Tools With CI/CD Pipelines

Why Pair SAST Tools With CI/CD Pipelines?

The traditional approach leads to costly delays, and vulnerabilities may go undetected. On the other hand, CI/CD pipelines aim to automate the software development process from start to finish, from building to testing and deployment.

By integrating SAST into CI/CD pipelines, developers can identify and remediate security vulnerabilities early in the development process. Pairing SAST tools with CI/CD pipelines saves time and money and ensures that security issues are detected and resolved quickly.

"

Software development teams are under increasing pressure to deliver secure applications quickly. Integrating SAST into the CI/CD (Continuous Integration/Continuous Delivery) pipeline is crucial to achieving this goal.

Choosing the Right SAST Tool for Your CI/CD Pipeline

When choosing a SAST tool for your CI/CD pipeline, it is essential to consider the following:

Integration with CI/CD pipelines: Choose a tool that can easily integrate with your CI/CD pipeline.

False positive rate: SAST tools often generate false positives, which can be time-consuming and costly to investigate. Choose a tool with a low false positive rate.

Ease of use: Choose a tool that is easy to set up and use.

Certified DevSecOps Professional (CDP)

Steps To Integrate SAST Tools Into CI/CD Pipelines

Most SAST tools offer plugins for popular CI/CD platforms like Jenkins, GitLab, and TravisCI.

The following steps outline how to integrate SAST tools into a CI/CD pipeline

Step 1: Install the SAST tool in the CI/CD environment.
Step 2: Configure the SAST tool to scan the codebase during the build process.
Step 3: Specify the SAST tool to fail the build if any critical vulnerabilities are detected.
Step 4: Configure the CI/CD pipeline to generate reports on the vulnerabilities detected.
Step 5: Schedule periodic scans to detect new vulnerabilities as the codebase evolves.

Interpreting the Results

After integrating SAST tools into your CI/CD pipeline, you will receive reports on vulnerabilities detected. It is crucial to have a process for interpreting these results and prioritizing remediation efforts. It is recommended to triage the findings by severity level and prioritize the critical ones.

CHAPTER 5

Creating a Baseline Scan

After setting up your SAST environment, it's time to create a baseline scan. A baseline scan provides a starting point for measuring progress and identifying vulnerabilities in your codebase.

Choose a codebase

Set up the scan

Run the scan

Review the results

CHAPTER 6

Analyzing and Addressing Issues

Analyzing and addressing issues is critical in SAST implementation. Here are the steps to analyze and address issues found in your codebase:

01

Review the results of your SAST scans to identify any vulnerabilities or weaknesses in your codebase.

02

Prioritize the issues based on their severity and potential impact on your codebase, which will help you focus on the most critical issues first.

03

Assign tasks to your development team to address the issues. Be specific about what needs to be done, and guide how to fix the issues.

04

Monitor progress to ensure that issues are being addressed in a timely manner. Use dashboards and reporting to keep track of progress and identify any roadblocks.

05

Verify fixes: Once the issues have been addressed, verify that the fixes have been implemented correctly. Re-run scans to ensure that the vulnerabilities have been resolved.

06

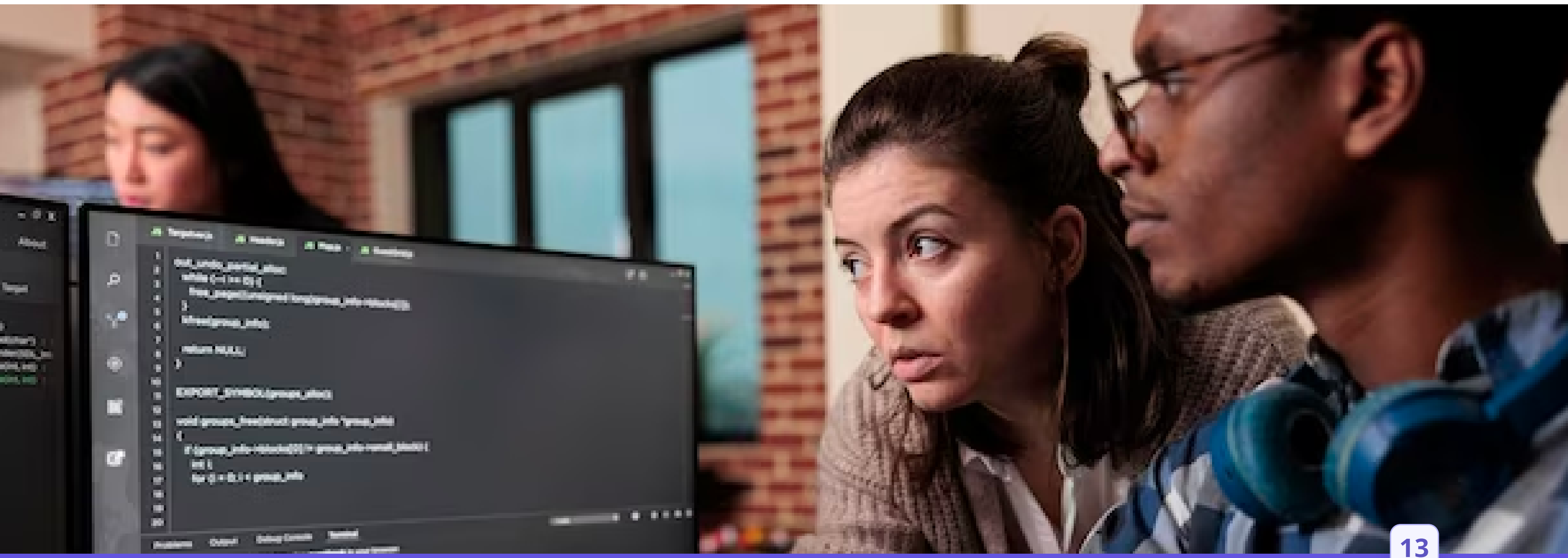
Communicate with stakeholders such as your security team and project managers, to ensure they know the issues and the steps to address them. Provide regular updates to keep them informed of progress.

CHAPTER 7

Creating Custom Rules

Creating custom rules is a powerful way to extend the functionality of your SAST tool and tailor it to your specific needs. Here are some steps to create custom rules:

1. **Identify areas of your codebase** where you want to enforce additional security measures. This could include areas where you have identified vulnerabilities in the past or areas where you want to enforce additional security best practices.
2. **Define the rule** that you want to enforce. This could be a specific coding practice, such as enforcing the use of parameterized queries, or a broader rule, such as enforcing a specific security policy.
3. **Configure the rule** in your SAST tool. This may involve writing custom code or configuring the tool using the tool's user interface.
4. **Run the scan** and review the results. Identify any areas where the custom rule has been violated.
5. **Address any issues** that the custom rule has identified. This may involve working with your development team to change the codebase.
6. **Monitor the results** of your scans to ensure that the custom rule is being enforced effectively. Use dashboards and reports to track progress and identify areas for improvement.
7. **Continuously improve** your custom rules by analyzing results, identifying areas for improvement, and implementing changes to optimize your scans.



CHAPTER 8

Measuring and Reporting on Success

One of the essential aspects of any security program is measuring its success. Measuring success in SAST implementation requires defining the Key Performance Indicators (KPIs) that will help you determine the program's effectiveness.

Defining KPIs

Here we present some of the KPIs that are essential for measuring success:

Number of vulnerabilities identified:

- Gives an idea of the risk level of your codebase.
- Helps prioritize efforts in addressing issues.

The number of vulnerabilities addressed:

- Determines how effective remediation efforts have been.

Time to identify and address vulnerabilities:

- Measures the efficiency of your SAST workflow.

Reporting Success to Stakeholders

It's important to report on the success of your SAST implementation to stakeholders. Reports can include:

- Metrics on the number of vulnerabilities identified and addressed.
- Time it takes to remediate issues.

Reports can be shared with management, developers, and other stakeholders to demonstrate the value of the SAST program.

CHAPTER 9

Best Practices for SAST

To get the most out of SAST, it's important to follow best practices.

1 **Starting with a small codebase** can help you get familiar with the SAST tool and the workflow.

2 **Prioritizing issues** based on risk level can help you first address the most critical issues.

3 **Involving developers** in the SAST process can help increase buy-in and improve the overall security culture.

4 **Scheduling regular scans** can help ensure that the codebase is always up-to-date with the latest security standards.

5 **Creating custom rules** can help identify issues that are specific to your codebase.

6 **Automating** the SAST workflow can help reduce the time it takes to identify and address vulnerabilities.

7 **Continuously monitoring** the codebase can help identify new issues that may arise.

CHAPTER 10

SAST Strengths and Weaknesses

Static Application Security Testing (SAST) is a valuable tool in a comprehensive security program, but like any security tool, it has its strengths and weaknesses. Understanding the strengths and weaknesses of SAST is crucial in using it effectively in your security program.

S

Strengths

Static Analysis is easy to get started and usually straight forward. Can do both data flow and control flow analysis.

W

Weaknesses

By its nature, static analysis is prone to high levels of false positives and takes considerable time. Can not find runtime and business logic bugs.

O

Opportunities

Fast feedback if set properly. Can be configured to reduce false positives. Support for many languages available.

T

Threats

Many Tools doesn't fit into modern CI/CD pipelines. Lack of handling False positives locally is a big show stopper.

Strengths	Weaknesses
Can analyze code at any stage of development	Can produce false positives and false negatives
Can analyze large codebases quickly	May not detect vulnerabilities that require runtime data
Can identify common vulnerabilities and coding errors	Cannot detect certain types of vulnerabilities, such as design flaws
Can be integrated into development processes for early vulnerability identification	May require specialized knowledge to configure and use effectively
Can help enforce coding standards and best practices	May not be effective at identifying complex vulnerabilities or those that require a deep understanding of the codebase
Can be automated and integrated into CI/CD pipelines	May not be suitable for all types of applications or programming languages

Practical DevSecOps

Become a Certified
DevSecOps Professional

Get Started