St. Francis Institute of Technology

(An Autonomous Institution)

AICTE Approved | Affiliated to University of Mumbai
A+ Grade by NAAC: CMPN, EXTC, INFT NBA Accredited: ISO 9001:2015 Certified

Department of Information Technology

A.Y. 2025-2026
Class: BE-IT A/B, Semester: VII

Subject: Secure Application Development Lab

Student Name: Charis Fernandes Student Roll No: 26

Experiment – 4: Study of different SAST (Static Application Security Testing) tools

Aim: To study and exercise on different SAST (Static Application Security Testing) tools

Objective: After performing the experiment, the students will be able to –

- To get familiar with the features provided by the open source SAST tools on GitHub

Lab objective mapped: ITL703.2: To understand the methodologies and standards for developing secure code

Prerequisite: Basic knowledge Information Security, software enginerring

Requirements: Personal Computer, Windows operating system browser, Internet Connection etc.

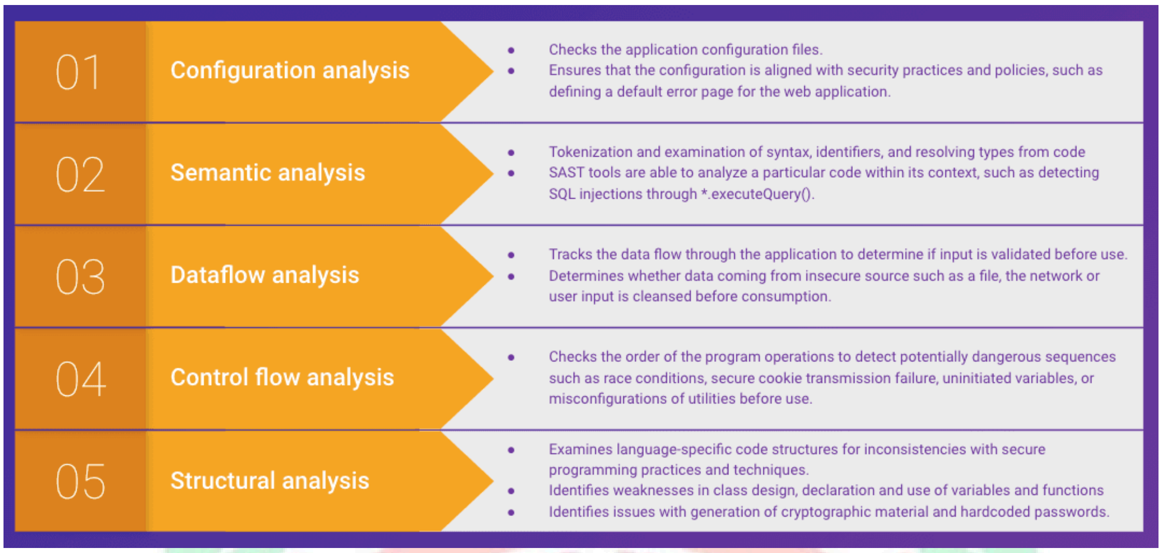
Pre-Experiment Theory:

What is Static Application Security Testing (SAST)?

SAST is a vulnerability scanning technique that focuses on source code, byte code, or assembly code. The scanner can run early in your CI pipeline or even as an IDE plugin while coding. SAST tools monitor your code, ensure protection from security issues such as saving a password in clear text or sending data over an unencrypted connection.

How do SAST tools work?

SAST is a technique used to evaluate source code without actually executing it. It involves examining the program's structure and syntax to identify potential issues and errors, such as coding mistakes, security vulnerabilities, and performance bottlenecks. The process involves parsing the source code, building an abstract syntax tree, and applying various analysis techniques to detect issues. By providing early feedback on potential issues in the code, SAST can help improve software quality and reduce the likelihood of errors and security vulnerabilities.



01	Configuration analysis	• Checks the application configuration files. • Ensures that the configuration is aligned with security practices and policies, such as defining a default error page for the web application.
02	Semantic analysis	• Tokenization and examination of syntax, identifiers, and resolving types from code • SAST tools are able to analyze a particular code within its context, such as detecting SQL injections through *.executeQuery().
03	Dataflow analysis	• Tracks the data flow through the application to determine if input is validated before use. • Determines whether data coming from insecure source such as a file, the network or user input is cleansed before consumption.
04	Control flow analysis	• Checks the order of the program operations to detect potentially dangerous sequences such as race conditions, secure cookie transmission failure, uninitiated variables, or misconfigurations of utilities before use.
05	Structural analysis	• Examines language-specific code structures for inconsistencies with secure programming practices and techniques. • Identifies weaknesses in class design, declaration and use of variables and functions • Identifies issues with generation of cryptographic material and hardcoded passwords.

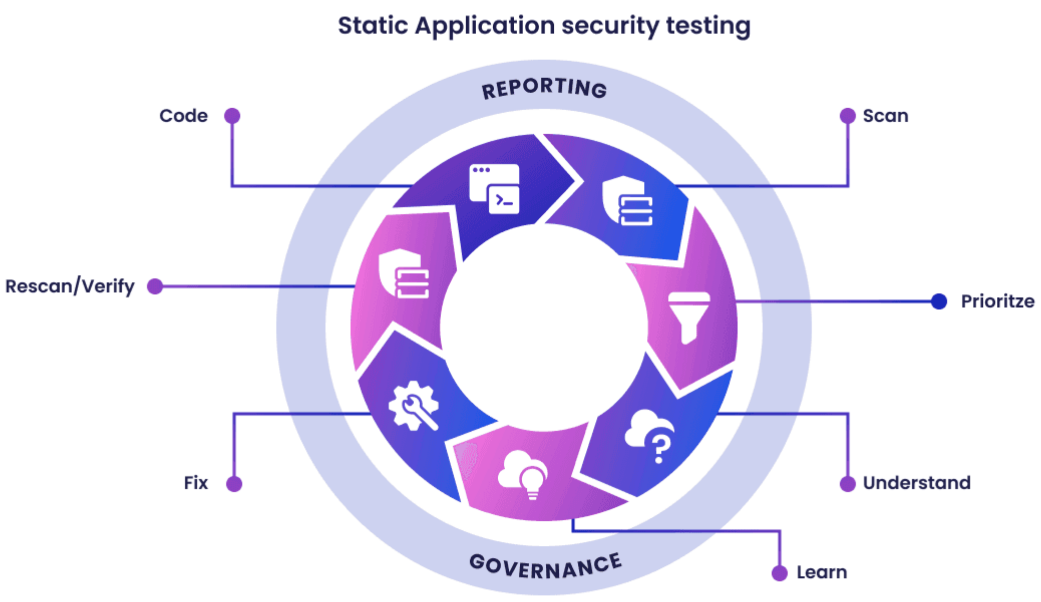
What vulnerabilities can SAST tools find?

SAST tools detect a range of security incidents and vulnerabilities in source code, including:

- Dataflow issues
- Semantic errors
- Misconfigured settings
- Control flow problems
- Structural flaws
- Memory issues

7 stages of a security application testing scan

The seven stages of a SAST scan are:



1. **Scan** your code as it's being written.
2. **Prioritize** and triage based on the severity and impact of the vulnerabilities found
3. **Understand** the nature of the vulnerabilities found by reviewing scan data and assessing the associated risk level.
4. **Learn** from the scan findings to prevent similar vulnerabilities in the future. This includes improving code quality, adopting secure coding practices, and implementing developer security training.
5. **Fix** vulnerabilities found in the scan by patching the code or implementing other remediation measures.
6. **Rescan** to verify that the fix has worked.
7. **Continue** coding while integrating security into the development process to prevent vulnerabilities from being introduced in future code.

Procedure:
Select any five open source SAST tools from GitHub and discuss them in detail.

Post-Experimental Exercise

Questions:
- How to find the right SAST tool to secure the software development lifecycle (SDLC)?
- Compare SAST with DAST

Conclusion:
- Write what was performed in the experiment.
- Write the significance of the topic studied in the experiment.

References

- https://snyk.io/learn/application-security/static-application-security-testing/#how

- https://github.com/analysis-tools-dev/static-analysis

Procedure: An In-Depth Study of Five Open Source SAST Tools

---

1. SonarQube

Description

SonarQube is a leading open-source **platform** for continuous inspection of code quality and security. It's not just a single tool but a centralized server that analyzes code, tracks metrics over time, and serves as a single source of truth for the health of a codebase.

Function

Its core function is to perform automatic reviews of code by using static analysis to detect bugs, "code smells" (confusing or hard-to-maintain code), and security vulnerabilities. It integrates into the CI/CD pipeline to provide continuous feedback, ensuring that quality and security standards are maintained with every code change.

Language

SonarQube is a **polyglot** platform, offering extensive support for over 20 programming languages, including **Java**, **C#**, **JavaScript**, **TypeScript**, **Python**, **PHP**, **Go**, **Swift**, and **Kotlin**.

Features

- **Quality Gates**: Enforces a set of conditions (e.g., no new critical vulnerabilities, 80% code coverage) that must be met before code can be released.
- **Security Analysis**: Includes rule sets to detect common vulnerabilities like SQL injection, cross-site scripting (XSS), and other risks listed in the OWASP Top 10.
- **IDE Integration**: The **SonarLint** plugin provides real-time feedback directly in the developer's IDE (like VS Code or IntelliJ IDEA), catching issues as code is written.

- **Comprehensive Dashboards**: Offers detailed visualizations and reports on code quality, security, and maintainability metrics over time.

Advantages

- **All-in-One Platform**: Combines analysis for bugs, code smells, and security in a single tool, providing a holistic view of code health.
- **Extensive Language Support**: Its ability to analyze many different languages makes it ideal for organizations with diverse technology stacks.
- **Enforces Standards**: Quality Gates provide a powerful, automated way to enforce quality and security standards across all projects.

Disadvantages

- **Complexity**: As a full platform with a server and database, it can be more complex to set up and maintain compared to a simple command-line tool.
- **Resource Intensive**: Running a dedicated SonarQube server can require significant system resources, especially for large codebases or many projects.

---

2. Bandit

Description

Bandit is a focused SAST tool specifically designed to find common security issues in **Python** code. It is a project of the Python Code Quality Authority (PyCQA).

Function

Bandit works by parsing Python source code files to build an **Abstract Syntax Tree (AST)** for each one. It then runs a series of specialized plugins against the nodes of the AST to identify code patterns that are indicative of security vulnerabilities. This approach allows for a deeper understanding of the code's structure compared to simple text-based searching.

Language

Bandit is exclusively for **Python**.

Features

- **Security-Focused Plugins**: Comes with a large set of built-in tests that check for known Python security issues, such as unsafe deserialization, weak cryptography, and command injection risks.
- **Confidence and Severity Reporting**: Each finding is rated with a severity level (Low, Medium, High) and a confidence level (Low, Medium, High) to help developers prioritize fixes.
- **Configuration and Baselines**: Scans can be configured to ignore certain tests, and results can be baselined to only report new findings in a pull request.

- **Flexible Output**: Reports can be generated in multiple formats, including JSON, HTML, and CSV, for easy integration with other systems.

Advantages

- **Highly Specialized**: Its exclusive focus on Python security makes it very effective at finding vulnerabilities specific to the language and its common libraries.
- **Effective Prioritization**: The use of severity and confidence levels helps teams focus on the most critical and certain vulnerabilities first.
- **CI/CD Friendly**: It's lightweight, fast, and easily integrated into automated build and test pipelines.

Disadvantages

- **Single-Language Support**: It only works for Python, making it unsuitable for projects that use multiple programming languages.
- **Narrow Focus**: It is strictly a security tool and will not report on general code quality, style, or performance issues.

---

3. ESLint

Description

ESLint is a highly popular and powerful open-source linter for **JavaScript** and **TypeScript**. It's designed to be extremely pluggable and configurable, allowing teams to enforce a wide variety of coding standards and best practices.

Function

ESLint statically analyzes code to find problematic patterns. While its core function is to maintain code quality and style consistency, its extensible nature allows it to be configured as a potent SAST tool by adding security-specific plugins that introduce rules for detecting common JavaScript vulnerabilities.

Language

It primarily supports **JavaScript** and **TypeScript**.

Features

- **Pluggable Architecture**: Its functionality can be easily extended with plugins. For security, the **eslint-plugin-security** is commonly used to add rules that detect vulnerabilities.
- **Customizable Rules**: Every rule can be enabled, disabled, or configured through a central .eslintrc file, giving teams complete control over the analysis.
- **Automatic Fixes**: For many issues it finds, ESLint can automatically rewrite the code to conform to the configured rules, saving significant developer time.

- **IDE Integration**: It integrates seamlessly into almost all modern code editors, providing developers with real-time feedback as they type.

Advantages

- **Extremely Flexible**: Its pluggable and configurable nature allows it to be tailored to the exact needs of any JavaScript project.
- **Excellent Developer Experience**: Real-time feedback inside the code editor helps catch and fix issues at the earliest possible stage.
- **Strong Community**: Has a massive community that creates and maintains a vast ecosystem of plugins and configurations.

Disadvantages

- **Requires Security Configuration**: It is not a dedicated security tool out of the box. A team must actively install and configure security-specific plugins to use it for SAST.
- **Potential for Noise**: Without careful configuration, it can generate a large number of style-related warnings that may distract from critical security issues.

---

4. Checkstyle

Description

Checkstyle is a classic static analysis tool used to help programmers write **Java** code that adheres to a specific coding standard. It automates the process of checking code against a defined set of rules.

Function

Its primary function is to enforce coding conventions and best practices. While not a dedicated security scanner, it serves as a **preventative** SAST tool by allowing teams to prohibit insecure coding patterns, restrict the use of dangerous APIs, and enforce a level of code quality that makes security vulnerabilities less likely to occur.

Language

Checkstyle is exclusively for **Java**.

Features

- **Rich Set of Checks**: Comes with a large library of modules to check everything from code layout and naming conventions to code complexity and usage of specific language features.
- **High Configurability**: Uses a detailed XML configuration file that gives teams fine-grained control over which rules are enforced.

- **Build Tool Integration**: Integrates easily with standard Java build tools like **Maven** and **Gradle**, allowing checks to be run automatically as part of the build process.
- **Custom Rule Creation**: Teams can write their own custom checks in Java to enforce unique, organization-specific rules.

Advantages

- **Enforces Code Quality**: By enforcing a strict coding standard, it improves code readability and maintainability, which indirectly enhances security.
- **Prevents Bad Practices**: Can be configured to prevent developers from using known insecure functions or patterns, acting as a preventative control.
- **Mature and Stable**: It's a long-standing tool in the Java ecosystem with a proven track record.

Disadvantages

- **Not a Security Scanner**: It is not designed to find complex security vulnerabilities like injection flaws. It primarily checks code against a set of stylistic and structural rules.
- **Single-Language Support**: It only works for Java.

---

5. Brakeman

Description

Brakeman is a fast and powerful open-source security scanner designed specifically for **Ruby on Rails** applications. It's highly regarded in the Ruby community for its accuracy and ease of use.

Function

Brakeman functions by statically analyzing the entire codebase of a Rails application—including models, views, and controllers. Because it has a deep, built-in understanding of the Rails framework's conventions and architecture, it can effectively trace data flow and identify security vulnerabilities with high accuracy and minimal false positives.

Language

It is built for **Ruby**, with a specific focus on the **Ruby on Rails** framework.

Features

- **Framework-Aware**: Its knowledge of Rails allows it to detect a wide range of framework-specific vulnerabilities, including SQL injection, XSS, command injection, and insecure redirects.
- **Zero Configuration**: It works out of the box with most Rails applications, requiring no complex setup to get started.

- **Fast Scans**: It can scan large applications in just a few minutes, making it ideal for running in a CI/CD pipeline on every commit.
- **Confidence Levels**: Assigns a confidence level (High, Medium, Weak) to each warning, helping developers triage and prioritize the most important issues.

Advantages

- **High Accuracy for Rails**: Its specialization in Ruby on Rails leads to more accurate findings and fewer false positives compared to generic SAST tools.
- **Fast and Easy to Use**: Its speed and zero-configuration nature make it incredibly easy to add to any Rails project's workflow.
- **Actionable Reports**: Generates clear reports that pinpoint the exact location of a vulnerability, making remediation straightforward.

Disadvantages

- **Highly Specialized**: Its greatest strength is also its biggest limitation. It only works for Ruby on Rails applications and cannot be used for other Ruby projects or different languages.
- **Limited Scope**: It focuses on framework-level vulnerabilities and may not detect security issues in custom application logic that falls outside of standard Rails patterns.