# 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: **Keith Fernandes** Student Roll No: **25**

## 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:**

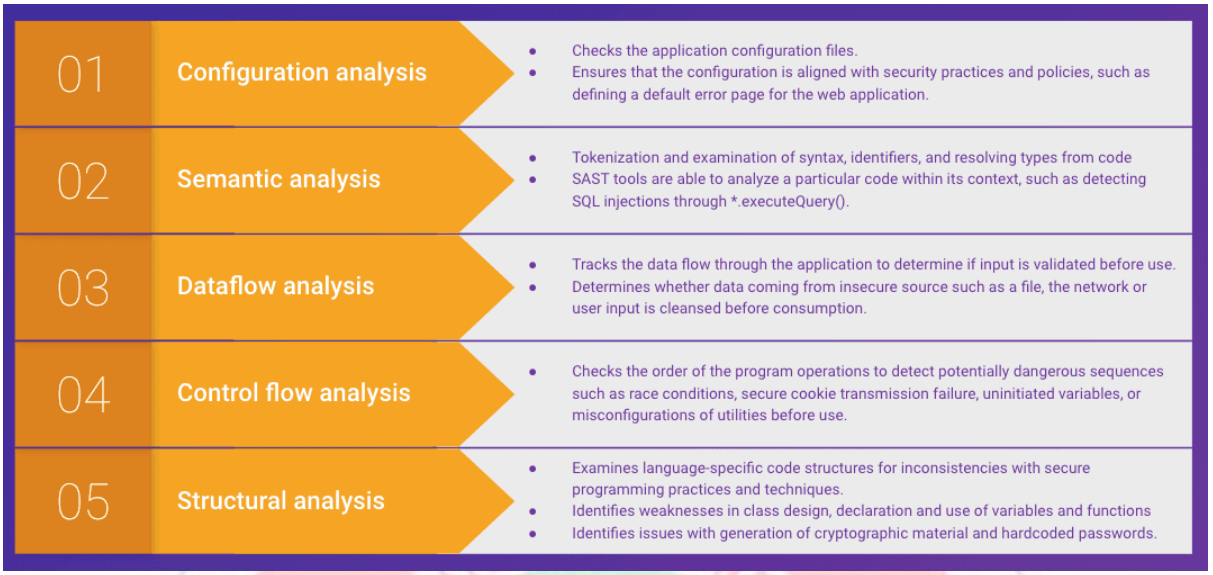
**What is Static Application Security Testing (SAST)?**

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.



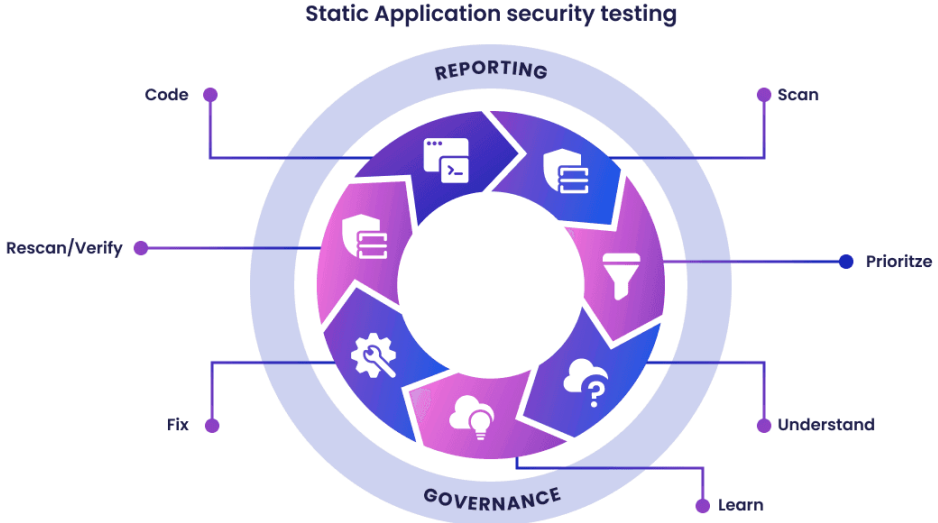

**What vulnerabilities can SAST tools find?**

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.

**1. Semgrep**

- **Overview:** Lightweight static analysis tool that uses rules (similar to grep but syntax-aware). It can scan code for security vulnerabilities, code quality issues, and enforce coding standards.
- **Key Features:**
  - Works on 30+ programming languages.
  - Rule-based matching with YAML configuration.
  - Integrates with CI/CD pipelines (GitHub Actions, GitLab CI).
  - Pre-built security rules available from the Semgrep registry.
- **Use Case:** Detects hardcoded secrets, SQL injection patterns, unsafe function calls, etc.
- **Why Important:** Very fast, customizable, and beginner-friendly for secure coding enforcement.

**2. SonarQube Community Edition**

- **Overview:** Popular code quality and security analysis platform. It performs SAST by analyzing source code and detecting bugs, code smells, and security vulnerabilities.
- **Key Features:**
  - Supports 20+ languages (Java, C, C++, Python, JS, etc.).
  - OWASP Top 10 and CWE vulnerability detection.
  - Rich dashboard and reports for developers and managers.
  - Integration with CI/CD and IDEs (like IntelliJ, VS Code).
- **Use Case:** Helps organizations monitor overall security + maintainability.
- **Why Important:** Widely adopted in enterprises for continuous inspection of code security.

**3. Bandit (for Python)**

- **Overview:** Python-specific security linter that scans code for common security issues.

- **Key Features:**
  - Detects insecure usage of `eval`, hardcoded passwords, weak cryptography, etc.
  - Generates reports in multiple formats (JSON, text, HTML).
  - Easy to integrate into GitHub Actions or pre-commit hooks.
- **Use Case:** Ideal for Python web apps (Django, Flask) to catch vulnerabilities early.
- **Why Important:** Lightweight but powerful tool for Python security compliance.

**4. Cppcheck (for C/C++)**

- **Overview:** A static analysis tool for C/C++ that focuses on detecting undefined behavior and security issues.
- **Key Features:**
  - Detects memory leaks, buffer overflows, null pointer dereference.
  - Provides MISRA C (safety standard) compliance checking.
  - Works offline with no dependencies.
- **Use Case:** Embedded systems and critical applications where C/C++ vulnerabilities can lead to serious issues.
- **Why Important:** Essential for detecting low-level coding flaws that can become critical security vulnerabilities.

**5. FindSecBugs (extension of SpotBugs for Java)**

- **Overview:** Security-focused plugin for SpotBugs (Java static analysis).
- **Key Features:**
  - Identifies 130+ security vulnerabilities in Java code.
  - Detects SQL injection, XXE, command injection, LDAP injection, etc.
  - Supports major Java frameworks (Spring, Struts, Android).
- **Use Case:** Perfect for Java enterprise apps where web security flaws are common.
- **Why Important:** Specialized for Java security, aligned with OWASP standards.

**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/anal__ysis-tools-dev/static-analysis