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: Durva Kadam Student Roll No: 23

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:

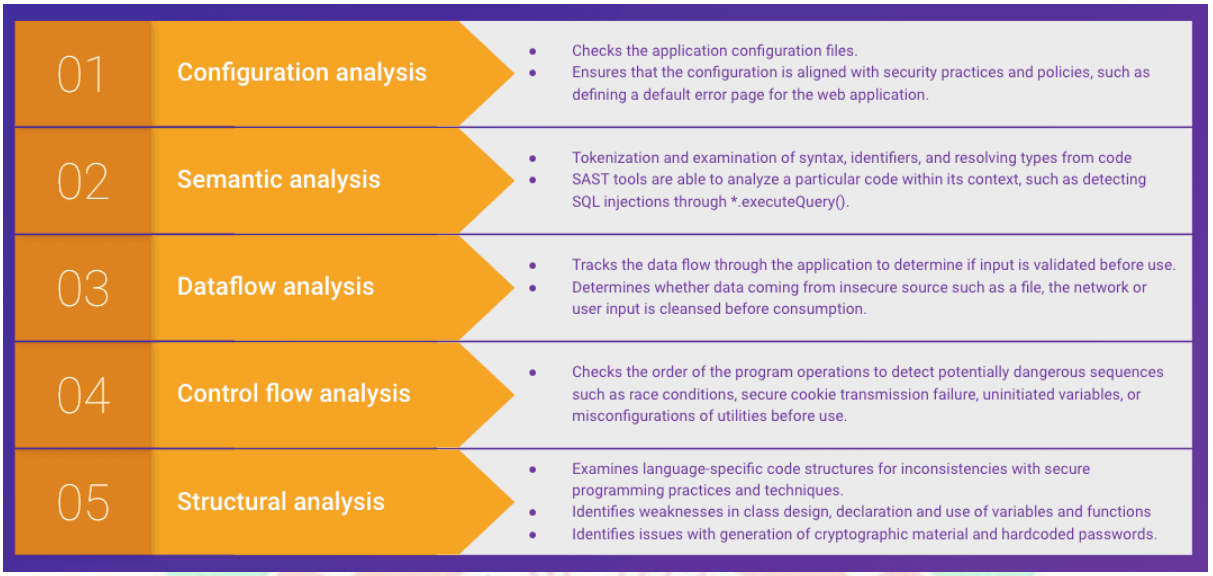
What is Static Application Security Testing (SAST)?

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.



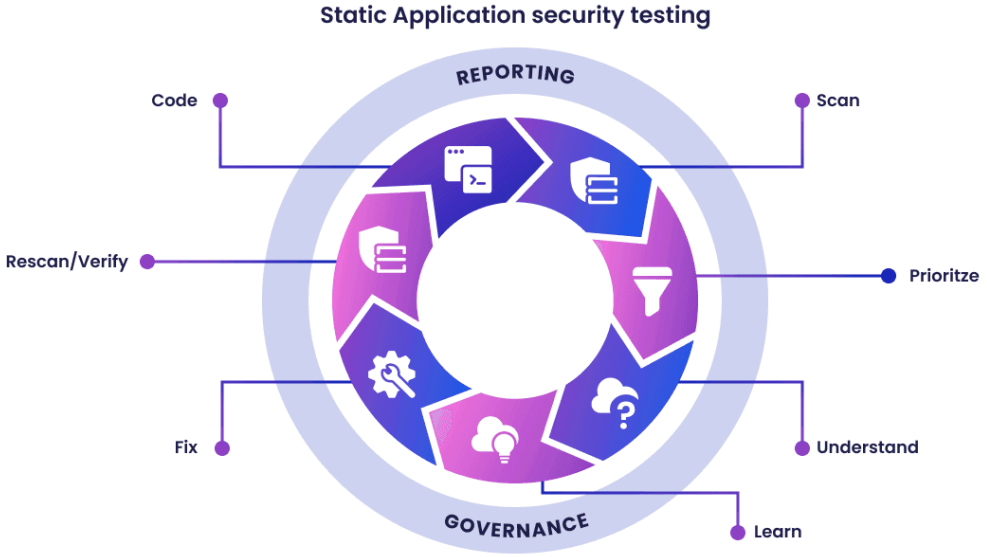
What vulnerabilities can SAST tools find?

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]Cppcheck

Cppcheck is an open-source static analysis tool designed specifically for C and C++ code. Here are some key features and details about Cppcheck:

Key Features:

1. **Static Code Analysis**: Cppcheck analyzes source code without executing it, identifying potential bugs, coding errors, and security vulnerabilities.
2. **Comprehensive Error Detection**: It can detect various issues, including:
    - Memory leaks
    - Buffer overflows
    - Uninitialized variables
    - Type mismatches
    - Invalid type casting
    - Misuse of standard libraries
3. **Customization**: Users can configure checks and define rules based on their specific coding standards or project requirements.
4. **Cross-Platform**: Cppcheck runs on multiple operating systems, including Windows, Linux, and macOS.
5. **Integration**: It can be easily integrated into IDEs and build systems, allowing for seamless use within existing development workflows.
6. **Output Formats**: Cppcheck provides output in various formats, including XML, which can be useful for automated tools and reports.
7. **Lightweight**: The tool is designed to be lightweight and efficient, with low resource consumption.

Usage:

- **Installation**: Cppcheck can be downloaded from its official website or installed via package managers for various operating systems.
- **Running the Tool**: Users can run Cppcheck from the command line or through a graphical user interface. It allows you to specify the directories or files to analyze.

Community and Support:

Cppcheck has an active community that contributes to its development and maintenance. Users can find documentation, tutorials, and support through forums and its GitHub repository.

Conclusion:

Cppcheck is a valuable tool for developers working with C and C++, helping to ensure code quality and security through static analysis. By integrating Cppcheck into the development process, teams can identify issues early, leading to more secure and reliable software.

2]Holistic: Static Application Security Testing Tool

Overview: Holistic is a static application security testing (SAST) tool designed to enhance code quality and security throughout the software development lifecycle. It provides developers with real-time feedback and integrates seamlessly into existing workflows, making it easier to identify and remediate vulnerabilities.

Key Features:

1. **Comprehensive Code Analysis**:Scans source code to identify vulnerabilities, coding errors, and potential security risks.Supports various programming languages.
2. **User-Friendly Interface**:Intuitive design that simplifies navigation and understanding of scan results.
3. **Real-Time Feedback**: Integrates into development environments to provide immediate feedback as developers write code. Helps catch issues early in the development process.
4. **Customizable Rulesets**: Allows users to define and customize rules based on specific security requirements and coding standards.
5. **Integrations**: Supports integration with popular development tools and CI/CD pipelines, facilitating automated security testing.
6. **Collaboration Features**: Enables team collaboration by allowing sharing of findings and discussions about vulnerabilities and remediation strategies.
7. **Reporting and Metrics**: Offers detailed reports and metrics on code quality and security posture, helping teams track improvements over time.

Benefits for Database Security:

1. **Vulnerability Detection**:Identifies potential SQL injection vulnerabilities and improper data handling that could lead to data breaches.
2. **Code Quality Improvement**: Analyzes queries and data access patterns, encouraging developers to write secure and efficient database interactions.
3. **Best Practices Enforcement**: Enforces coding standards for database interactions, such as parameterized queries, mitigating risks associated with SQL injection.
4. **Integration with CI/CD**: Automates security checks on database-related code as part of the development workflow, enhancing overall security.
5. **Detailed Reporting**: Provides comprehensive reports that highlight vulnerabilities related to database access, allowing teams to prioritize remediation efforts.
6. **Collaboration Features**: Promotes team collaboration to address database security vulnerabilities, improving overall security posture.

Conclusion:

Holistic is a powerful SAST tool that enhances application security and quality by integrating security practices directly into the development process. By focusing on database-related vulnerabilities and promoting secure coding practices, Holistic helps teams protect sensitive data and maintain the integrity of their database systems.

3]Checkstyle: Code Quality Tool

Overview: Checkstyle is an open-source static code analysis tool designed for Java that helps developers write code that adheres to a specified coding standard. It enforces coding conventions, promotes best practices, and improves the overall quality of Java code.

Key Features:

1. **Coding Standard Enforcement**:Checkstyle allows teams to define coding standards and ensures that the code adheres to those standards, helping maintain consistency across projects.
2. **Customizable Configuration**: Users can create custom configuration files to tailor the checks to specific project requirements, including naming conventions, formatting, and complexity metrics.
3. **Integration with Build Tools**: Checkstyle integrates seamlessly with popular build tools like Maven, Gradle,and Ant, allowing for automated checks during the build process.
4. **IDE Support**: Plugins are available for popular Integrated Development Environments (IDEs) like Eclipse, IntelliJ IDEA, and NetBeans, providing real-time feedback as developers write code.
5. **Reporting**: Checkstyle generates detailed reports in various formats (XML, HTML, etc.) that highlight violations of coding standards, making it easier for teams to track issues and improvements.
6. **Code Metrics**: In addition to style checks, Checkstyle can provide metrics related to code complexity, length, and other factors that can affect maintainability.

Benefits for Developers:

1. **Improved Code Quality**: By enforcing coding standards, Checkstyle helps prevent common coding errors and improves readability and maintainability.
2. **Consistency Across Teams**: Checkstyle ensures that all team members adhere to the same coding standards, which reduces discrepancies and enhances collaboration.
3. **Early Detection of Issues**: Real-time feedback in IDEs allows developers to catch and correct issues as they code, reducing the time spent on code reviews and bug fixes.
4. **Integration into CI/CD**: Checkstyle can be incorporated into continuous integration and deployment pipelines, automating code quality checks as part of the development process.

Conclusion:

Checkstyle is a valuable tool for Java developers aiming to improve code quality and maintainability. By enforcing coding standards and providing real-time feedback, Checkstyle helps teams produce cleaner, more reliable code while fostering collaboration and adherence to best practices.

4]Liquid Haskell: A Static Verification Tool

Overview: Liquid Haskell is a static program verification tool for Haskell that allows developers to add type annotations and refinement types to their code. It enhances Haskell's type system by enabling more expressive type constraints, helping to catch errors at compile time and improving code reliability.

Key Features:

1. **Refinement Types**: Liquid Haskell introduces refinement types, which allow developers to express more precise types that can capture invariants about data. For example, you can specify that a list is non-empty or that a number is positive.
2. **Static Verification**: The tool statically verifies that the program adheres to the specified types and constraints. This means that many potential runtime errors can be caught during compilation rather than at runtime.
3. **Integration with Haskell**: Liquid Haskell works seamlessly with existing Haskell code, requiring minimal changes to incorporate type annotations and refinements.
4. **Customizable Type Annotations**: Developers can create custom types and constraints tailored to their specific needs, enhancing the expressiveness of Haskell's type system.
5. **Support for Higher-Order Functions** : Liquid Haskell supports higher-order functions and polymorphism, allowing for advanced type constraints that can improve the correctness of complex Haskell programs.

Benefits for Haskell Developers:

1. **Increased Code Reliability**: By enforcing stricter type constraints, Liquid Haskell helps ensure that code behaves as expected, reducing the likelihood of bugs.
2. **Improved Documentation**: Type annotations serve as a form of documentation, making it easier for developers to understand the intended usage and constraints of functions and data types.
3. **Catch Errors Early**: Early detection of type errors reduces the time spent debugging and enhances productivity by providing immediate feedback during the development process.
4. **Better Collaboration**: Clearer type annotations can facilitate better communication among team members about the intended behavior of code components.

Conclusion:

Liquid Haskell is a powerful tool for Haskell developers looking to enhance the safety and reliability of their code through refined type systems and static verification. By incorporating Liquid Haskell into their development workflow.

5]autopep8: Python Code Formatter

Overview: autopep8 is a tool that automatically formats Python code to conform to the PEP 8 style guide. PEP 8 is the official style guide for Python code, promoting readability:

Key Features:

1. **Automatic Formatting**: autopep8 automatically reformats Python code to adhere to PEP 8 standards, saving developers time and effort in manual code styling.
2. **Configurable Options**: Users can customize the behavior of autopep8 through command-line options or configuration files, allowing for flexibility in how code is formatted.
3. **In-place Editing**: The tool can modify files directly, making it easy to integrate into existing workflows without needing to copy and paste formatted code.
4. **Integration with Editors**: autopep8 can be easily integrated into popular text editors and IDEs, providing real-time formatting feedback as developers write code.

Benefits for Python Developers:

1. **Improved Code Consistency**: By enforcing PEP 8 standards, autopep8 helps maintain a consistent coding style across projects and teams.
2. **Enhanced Readability**: Well-formatted code is easier to read and understand, facilitating collaboration and code reviews.
3. **Time Savings**: Automating the formatting process allows developers to focus more on functionality and logic rather than style issues.
4. **Easier Maintenance**: Consistent styling makes it easier for developers to maintain and update codebases, as they can quickly understand the structure and formatting.

Conclusion:

autopep8 is a valuable tool for Python developers aiming to enhance code quality and maintainability by adhering to PEP 8 standards. By automating the formatting process, autopep8 allows developers to produce cleaner, more readable code efficiently, ultimately leading to better collaboration and fewer style-related issues.

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