WEB APPLICATION SECURITY CONSORTIUM

STATIC ANALYSIS TOOLS EVALUATION CRITERIA

 Version 1.0 2013
Table of Contents

1. List of Static Code Analysis Tools
2. Download An Evaluation Worksheet
3. Table of Contents
4. Introduction
5. Target Audience
6. Scope
7. Contributors
8. Project Leader
9. Criteria:
   i. 1. Platform Support
   ii. 2. Technology Support
   iii. 3. Scan, Command and Control Support
   iv. 4. Product Signature Update
   v. 5. Reporting Capabilities
   vi. 6. Triage and Remediation Support
   vii. 7. Enterprise Level Support
10. Index A: Static Code Analysis Preparation Cheat Sheet
11. Index B: References
Static Code Analysis is the analysis of software code without actually executing the binaries resulting from this code. Static code analysis aims at automating code analysis to find as many common quality and security software issues as possible. There are several open source and commercial tools available in the market for organizations to choose from.

Static code analysis tools are rapidly becoming an essential part of every software organization’s application security assurance program. Mainly because of the tools’ ability to analyze large amounts of source code in considerably shorter amount of time than a human could.

The goal of the SATEC project is to create a vendor-neutral document to help guide application security professionals during source-code driven security assessments. This document provides a comprehensive list of features that should be considered when conducting a security code review. Different users will place varying levels of importance on each feature, and the SATEC provides the user with the flexibility to take this comprehensive list of potential tool features, narrow it down to a shorter list of features that are important to the user, assign weights to each feature, and conduct a formal evaluation to determine which scanning solution best meets the user’s needs.

The aim of this document is not to define a list of requirements that all static code analyzers must provide in order to be considered a “complete” analyzer, and evaluating specific products and providing the results of such an evaluation is outside the scope of the SATEC project. Instead, this project provides the tools and documentation to enable anyone to evaluate web application security scanners and choose the product that best fits their needs. NIST Special Publication 500-283, “Source Code Security Analysis Tool Functional Specification Version 1.1”, contains minimal functional specifications for static code analysis tools. This document can be found at [http://samate.nist.gov/index.php/Source_Code_Security_Analysis.html](http://samate.nist.gov/index.php/Source_Code_Security_Analysis.html).
Target Audience
The target audience of this document is the technical staff of software organizations who are looking to automate parts of their source code driven security testing using one or more static code analysis tool. The document will take into consideration those who would be evaluating the tool and those who would actually be using the tool.

Scope
The purpose of this document is to develop a set of criteria that should be taken into consideration while evaluating Static Code Analysis tools for security testing. Every software organization is unique in their environment. The goal is to help organizations achieve better application security in their own unique environment, the document will strictly stay away from evaluating or rating tools. However, it will aim to draw attention to the most important aspects of static analysis tools that would help the target audience identified above to choose the best tool for their environment and development needs.

Contributors

- Aaron Weaver
- Abraham Kang
- Alec Shcherbakov
- Alen Zukich
- Benoit Guerette
- Daniel Medianero
- Gamze Yurttutan
- Henri Salo
- Herman Stevens
- Janos Drencsan
- James McGovern
- Jojo Maalouf
- Mushtaq Ahmed
- Ory Segal
- Philipe Arteau
- Ramu Srikanth
- Romain Gaucher
- Sneha Phadke
- Wagner Elias

Project Leader
Sherif Koussa (sherif dot koussa at gmail dot com)
1. Platform Support:
Static code analysis tools often represent a significant investment by software organizations looking to automate parts of their software security testing processes. Not only do these tools represent a monetary investment, but they demand time and effort by staff members to setup, operate, and maintain the tool. In addition, staff members are required to check and act upon the results generated by the tool. Understanding the ideal deployment environment for the tool will maximize the return on investment, help the organization uncover potential security flaws and will avoid unplanned hardware purchase cost. The following factors are essential to understanding the tool's capabilities and hence ensuring proper utilization of the tool which will reflect positively on tool utilization and ensuring maximum return on investment (ROI).

1.1 Tool Installation Support:
A static code analysis tool should provide the following as a minimum:
- **Installation manual**: specific instructions on installing the tool and its subsystems if any (e.g. IDE plugins) including minimum hardware and software requirements
- **Operations manual**: specific and clear instructions on how to configure and operate that tool and its subsystems.
- **Weaknesses Taxonomy**: this includes all the weaknesses that the tool could potentially find and an explanation of each of these weaknesses. This is potentially helpful to provide a quick overview of the tool’s capabilities.

1.2 Scalability Support:
Vendors provide various tool deployment options. Clear description of the different deployment options must be provided by the vendor to better utilize the tool within an organization. In addition, the vendor must specify the optimal operating conditions. At a minimum the vendor should be able to provide:
- The type of deployment: server-side vs client-side as this might require permissions change or incur extra hardware purchase.
- Ability to run simultaneous scans at the same time.
- The tools capabilities of accelerating the scanning speed (e.g. ability to multi-chain machines, ability to take advantage of multi-threaded/multi-core environments, etc)
2. Technology Support:
Most organizations leverage more than one programming language within their applications portfolio. In addition, more software frameworks are becoming mature enough for development teams to leverage and use across the board as well as a score of 3rd party libraries, technologies, libraries which are used both on the server and client side. Once these technologies, frameworks and libraries are integrated into an application, they become part of it and the application inherits any vulnerability within these components.

2.1 Standard Languages Support:
Most of the tools support more than one programming language. However, an organization looking to acquire a static code analysis tool should make an inventory of all the programming languages, and their versions, used within the organizations as well as third party applications that will be scanned as well. After shortlisting all the programming languages and their versions, an organization should compare the list against the tool’s supported list of programming languages and versions.

2.2 Frameworks Support:
Once an application is built on a top of a framework, the application inherits any vulnerability in that framework. In addition, depending on how the application leverages a framework or a library, it can add new attack vectors. It is very important for the tool to be able to be able to trace tainted data through the framework as well as the custom modules built on top of it. At large, frameworks and libraries can be classified to three types:

» **Server-side Frameworks**: which are the frameworks/libraries that reside on the server, e.g. Spring, Struts, Rails, .NET etc.

» **Mobile Frameworks**: which are the frameworks that are used on mobile devices, e.g. Android, iOS, Windows Mobile etc.

» **Client-side Frameworks**: which are the frameworks/libraries that reside on browsers, e.g. JQuery, Prototype, etc.

The tool should understand the relationship between the application and the frameworks/libraries and detect vulnerabilities resulting from the application’s usage of the framework or the library. and the following in particular:

» identify whether the application is using the framework in a insecure manner.

» The tool would be able to follow tainted data between the application and the framework.

» The tool’s ability to identify security misconfiguration issues in the framework/library.

» Any well-known vulnerability as identified by the [Common Weakness Enumeration (CWE)](https://cwe.mitre.org/) (CWE)
2.3 Industry Standards Aided Analysis:
The tool should be able to provide analysis that is tailored towards one of the industry standard weaknesses classification, e.g. OWASP Top 10, CWE/SANS Top 25, WASC Threat Classification, etc. This becomes a desirable feature for many reasons. For example, an organization that just started its application security program, a full standard scan might prove overwhelming, especially with an extensive portfolio of applications. Focusing on a specific industry standard in this case would be a good place to start for that particular organization.

2.4 Technology Configuration Support:
Several tweaks provided by the tool could potentially uncover serious weaknesses.

1. Configuration Files Redefinition: most tools provide support for standard configuration files (e.g. web.xml, web.config, etc). However, some applications tend to extend configurations to other file types (e.g. *.ini, *.properties, *.xml, etc). It is a desirable and a beneficial feature to configure the tool to treat a non-standard extension as a configuration file.

2. Extension to Language Mapping: the ability to extend the scope of the tool to include non-standard source code file extensions. For example, JSPF are JSP fragment files that should be treated just like JSP files. Also, HTC files are HTML fragment files that should be treated just like HTML files. PCK files are Oracle’s package files which include PL/SQL script. While a tool does not necessarily have to understand every non-standard extension, it should include a feature to extend its understanding to these extensions.
3. Scan, Command and Control Support:
The scan, command and control of static code analysis tools has a significant influence on the user’s ability to configure, customize and integrate the tool into the organization’s Software Development Lifecycle (SDLC). In addition, it affects both the speed and effectiveness of processing findings and remediating them.

3.1 Command line support:
The user should be able to perform scans using the command line which is a desirable feature for many reasons, e.g. avoiding unnecessary IDE licensing, build system integration, custom build script integration, etc.

3.2 IDE integration support:
The tool should provide plugins for the IDE(s) used by the organizations’ developers. Plugins provide great value to auditors and developers alike in the triage and remediation stages. IDE Plugins combine the ability to scan and view the results inside the IDE, giving more flexibility to view the code surrounding a certain finding. This increases the accuracy and speed of assessing findings, which reflects back on minimizing the time needed to process a scan’s results.

3.3 Customization:
The tool usually comes with a set of signatures (AKA as rules or checkers), this set is usually followed by the tool to uncover the different weaknesses in the source code. Static code analysis should offer a way to extend these signatures in order to customize the tool’s capabilities of detecting new weaknesses, alter the way the tool detect weaknesses or stop the tool from detecting a specific pattern. The tool should allow users to:

- **Add/delete/modify core signatures:** Core signatures come bundled with the tool by default. False positives is one of the inherit flaws in static code analysis tools in general. One way to minimize this problem is to optimize the tool’s core signatures, e.g. mark a certain source as safe input.

- **Author custom signatures:** This feature is almost invaluable to maximize the tool’s benefits. For example, a custom signature might be needed to “educate” the tool of the existence of a custom cleansing module so to start flagging lines that do not use that module or stop flagging lines that do use it.
3.4 Scan configuration capabilities:
This includes the following capabilities:

» **Ability to schedule scans**: scheduled scans are often a mandatory feature. Scans are often scheduled after nightly builds, some other times they are scheduled when the CPU usage is at its minimum. Therefore, it is important for the user to be able to schedule the scan to run at a particular time.

» **Ability to view real-time status of running scans**: some scans would take hours to finish, it would be beneficial and desirable for a user to be able to see the scan’s progress and the weaknesses found thus far.

» **Ability to save configurations and re-use them as configuration templates**: Often a significant amount of time and effort is involved in optimally configuring a static code analyzer for a particular application. A tool should provide the user with the ability to save a scan’s configuration so that it can be re-used for later scans.

» **Ability to run multiple scans simultaneously**: Organizations that have many applications to scan, will find the ability to run simultaneous scans to be a desirable feature.

» **Ability to support multiple users**: this is important for organizations which are planning to rollout the tool to be used by developers checking their own code. It is also important for organizations which are planning to scan large applications that require more than one security analyst to assess applications concurrently.
3.5 Testing Capabilities:
Scanning an application for weaknesses is the sole most important functionality of the tool. It is essential for the tool to be able to understand, accurately identify and report the following attacks and security weaknesses.

» Abuse of Functionality
» Application Misconfiguration
» Auto-complete Not Disabled on Password Parameters
» Buffer Overflow
» Credential/Session Prediction
» Cross-site Scripting
» Cross-site Request Forgery
» Denial of Service
» Insecure Cryptography
» Format String
» HTTP Response Splitting
» Improper Input Handling
» Improper Output Encoding
» Information Leakage
» Insufficient Authentication
» Insufficient Authorization
» Insufficient Session Expiration
» Integer Overflows
» LDAP Injection
» Mail Command Injection
» Null Byte Injection
» OS Command Injection
» Path Traversal
» Remote File Inclusion
» Session Fixation
» SQL Injection
» URL Redirection Abuse
» XPATH Injection
» XML External Entities
» XML Entity Expansion
» XQuery Injection
4. **Product Signature Update:**
Product signatures (AKA rules or checkers) are what the static code analysis tool use to identify security weaknesses. When making a choice of a static analysis tools, one should take into consideration the following:

4.1 **Frequency of signature update:**
Providing frequent signature update to a static code analysis tool is as important as updating the virus database of an anti-virus program. Hence, it is important to understand the following about a tool’s signature update:

» Frequency of signature update: whether it is periodically, on-demand, or with special subscription, etc.

» Relevance of signatures to evolving threats: Information must be provided by the vendor on how the products signatures maintain their relevance to the newly evolving threats.

4.2 **User signature feedback:**
The tools must provide a way for users to submit feedback on bugs, flawed rules, rule enhancement, etc.
5. Reporting Capabilities:
The ability to communicate clear and actionable analysis results to project stakeholders, is as important as the analysis process itself. A static code analysis tool should provide different ways to represent the analysis results depending on the target audience, for example developers will need as much details as possible in order to be able to effectively fix the weakness in a timely fashion, while upper management might need to focus on a scan’s high level summary, or the risk involved more so than vulnerability details.

5.1 Support for Role-based Reports:
The tool should be able to provide the following types of reports with the ability to mix and match:

» Executive Summary: provides high-level summary of the scan results.

» Technical Detail Reports: provides all the technical information required for developers to understand the issue and effectively remediate it. This should include:
  • Summary of the issue that includes the weakness category.
  • Location of the issue including file name and line of code number.
  • Remediation advice which must be customized per issue and includes code samples in the language of choice.

» Delta Reports: the tool should provide the ability to show the delta between two reports and show the trends overtime.

» Compliance Reports: Scanners should provide a report format that allows organizations to quickly determine whether they are in violation of regulatory requirements or other standards. These reporting capabilities should be considered if certain regulations are important to the organization. The following list provides some potentially applicable standards:
  • OWASP Top 10
  • WASC Threat Classification
  • CWE/SANS Top 25
  • Sarbanes-Oxley (SOX)
  • Payment Card Industry Data Security Standard (PCI DSS)
  • Health Insurance Portability and Accountability Act (HIPAA)
  • Gramm-Leach-Bliley Act (GLBA)
  • NIST 800-53
  • Federal Information Security Management Act (FISMA)
  • Personal Information Protection and Electronic Documents Act (PIPEDA)
  • Basel II
5.2 Report Customization:
The tool should be able to support report customization. At a minimum, the tool should be able to provide the following:

» Ability to include the auditor’s findings notes in the report.
» Ability to mark findings as false positives, and remove them from the report.
» Ability to change the finding’s severity level.
» Ability to suppress findings.
» Ability to change the report’s template to include the organization’s logo, header, footer, report cover, etc.

5.3 Report Formats:
At a minimum the tool should generate reports using the following formats:

» PDF
» XML
» HTML

Criteria (Contd.)
6. Triage and Remediation Support:
A crucial factor in a static code analysis tool is the support provided in the triage process and the accuracy, effectiveness of the remediation advice. This is vital to the speed in which the finding is assessed and remediated by the development team.

6.1 Finding Meta-Data:
Finding meta-data is the information provided by the tool around the finding. Good finding meta-data helps the auditor or the developer understands the weakness and decide whether it is a false positive quicker. The tool should provide the following with each finding:
» Finding Severity: the severity of the finding with a way to change it if required.
» Summary: explanation of the finding and the risk it poses on exploit.
» Location: the source code file location and the line number of the finding.
» Data Flow: the flow of the tainted data between the source and the sink.

6.2 Meta-Data Management:
» The tool should provide the ability to mark a finding as false positive.
» Findings marked as false positives should not appear in subsequent scans.
» The tool should be able to merge\diff scan results.

6.3 Remediation Support:
» The tool should provide accurate and customizable remediation advice.
» Remediation advise should be illustrated with examples written in the same programming language as the finding’s.
7. Enterprise Level Support:
When making a choice on a static analysis tool in the Enterprise, one should take into consideration the ability to integrate the tool into various enterprise systems, such as bug tracking, reporting, risk management and data mining.

7.1 Support for Integration into Bug Trackers:
The tool should provide support for integration into internal bug tracking system so findings can be converted to bugs. Ideally, this would be done automatically by the tool. At a minimum the tool should support manual integration. Alternatively, the tool could support exporting the findings into standard formats (e.g. csv, xml, etc) where they could later be imported into bug tracking systems.

7.2 Data Mining Capabilities Reports:
It is an important goal of any security team to be able to understand the security trends within the organization. To meet this goal, static analysis tools should provide the user with the ability to data mining capabilities, present trends and build intelligence from it.

7.3 Integration into Enterprise Level Risk Management System:
Information security teams and organizations need to present an accurate view of the risk posture of their applications and systems at all times. Hence, the tool should provide integration into enterprise level risk management system.

7.4 Ability to Aggregate Projects:
This pertains to the ability to add meta-data to a new scan. This data could be used to aggregate and classify projects, which could be used to drive intelligence to management. For example, this can help in identifying programming languages that seem to genere more findings thus better utilizing training budge for example. Another example, is to mark certain applications as “External Facing” which triggers the tool to perform a more stringent predefined scan template. Projects in organizations are built using a certain set of technologies and/or frameworks. These can be commercial, open source or built in-house. Certain projects may tend to have more security flaws as compared to others based on a technology or framework used or based on the how the technology/framework is used within a given business context. Static analysis tools could be used to configure similar projects with additional metadata to detect these patterns. This will build intelligence around them that lends to being able to detect which application components have more security weaknesses and why.
7.5 Licensing Scheme:
Static Code Analysis tools varies in their licensing schemes. Usually, the following factors will decide your most effective licensing scheme:

» Number of users having access to the tool
» Type of users having access to the tool
» Number of applications intended to be scanned
» The subscription time span.
» Amount of professional services needed in order to reach maximum tool utilization.
Static Code Analysis Preparation Cheat Sheet:
The key to a successful decision on the correct static code analysis tool to buy is directly proportional to the amount of preparation involved before evaluating any too. Every tool is unique so as your corporate environment, so a choice of tool depends majorly on your preparation. The following is a set of activities to help you prepare before the evaluation is started.

›› A list of the programming languages used in the organization.
›› A list of the frameworks and libraries used in the organization.
›› Who will be tasked to perform the scan
›› How the tool will be integrated into the Software Development Lifecycle
›› How will the developers see the scan results
›› Budget allocated to the tool purchase including the hardware to run the machine (if any)
›› A decision on whether the code (or the binaries) is allowed to be scanned outside the organization.
References:

›› WASC Threat Classifications
   (http://projects.webappsec.org/w/page/13246978/Threat%20Classification)

›› Web Applications Security Scanner Evaluation Criteria

›› NIST Source Code Security Analysis Tool Functional Specifications Version 1.1
   (http://samate.nist.gov/docs/source_code_security_analysis_spec_SP500-268_v1.1.pdf)

›› Static Program Analysis (http://en.wikipedia.org/wiki/Static_program_analysis)

›› List of Tools For Static Code Analysis
   (http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)