WEB APPLICATION SECURITY CONSORTIUM

STATIC ANALYSIS TECHNOLOGIES EVALUATION CRITERIA

(SATEC)

Version 1.0

2013

Table of Contents

Introduction

Static code analysis is the analysis of software source or binary code. It aims at automating code analysis to find as many common software security weaknesses as possible. There are several open source and commercial static code analysis tools and services available in the market for organizations to choose from.

Static code analysis is rapidly becoming an essential part of most software organizations' application security assurance program. Mainly because of their ability to analyze large amounts of source code in considerably shorter amount of time than a human could, uncover potential weaknesses, in addition to the ability to automate security knowledge and workflows.

The goal of the SATEC project is to create a vendor-neutral set of criteria to help guide application security professionals during the process of acquiring a static code analysis technology that is intended to be used during source-code driven security programs. This document provides a comprehensive list of criteria that should be considered during the evaluation process. Different users will place varying levels of importance on each feature, and the SATEC project provides the user with the flexibility to take this comprehensive list of potential criteria, narrow it down to a shorter list which contains the most important or most relevant set of criteria, assign weights to each criterion, 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 analysis vendors must provide in order to be considered a "complete" solution. In addition, evaluating specific products and providing the results of such an evaluation is outside the scope of the SATEC project. Instead, this project provides criteria and documentation to enable anyone to evaluate static code analysis tools and services and choose the product that best fits their needs. NIST Special Publication 500-283, "Source Code Security Analysis Tools 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..

Introduction (Contd.)

Target Audience

The target audience of this document is the technical staff of software organizations who are looking to automate parts of their application security assurance programs using one or more static code analysis technology, as well as application security professionals who are responsible for performing application security reviews. The document will take into consideration those who would be evaluating the technology and those who would actually be using it.

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 or services for security testing. The vendor-neutral criteria defined in this document are selected using a consensus-driven review process comprised of volunteer subject matter experts. Every organization is unique and has a unique software development environment, this document aims to help organizations achieve their application security goals through acquiring the most suitable tool for their own unique environment.
The document will strictly stay away from evaluating or rating vendors. However, it will focus on the most important aspects of static code analysis technologies that would help the target audience identify the best technology for their environment and development needs.

Contributors

- Aaron Weaver (Pearson Education)
- Abraham Kang (HP Fortify)
- Alec Shcherbakov (AsTech Consulting)
- Alen Zukich (Klocwork)
- Arthur Hicken (Parasoft)
- Amit Finegold (Checkmarx)
- Benoit Guerette (NorthSec)
- Chris Eng (Veracode)
- Chris Wysopal (Veracode)
- Dan Cornell (Denim Group)
- Daniel Medianero (Buguroo Offensive Security)
- Dinis Cruz (SecurityInnovation)
- Gamze Yurttutan
- Herman Stevens
- Janos Drencsan
- James McGovern (HP)
- Joe Hemler (Gotham Digital Science)
- Jojo Maalouf (Hydro Ottawa)
- Laurent Levi (Checkmarx)
- Mushtaq Ahmed (Emirates Airlines)
- Ory Segal (Akamai)
- Philippe Arteau
- Sherif Koussa (Software Secured) [Project Leader]
- Srikanth Ramu (University of British Columbia)
- Romain Gaucher (Coverity)
- Sneha Phadke (eBay)
- Wagner Elias (Conviso)

Contact

Participation in the Web Application Security Scanner Evaluation Criteria project is open to all.
If you have any questions about the evaluation criteria, please contact Sherif Koussa
(sherif dot koussa at gmail dot com)

Criteria

1. Deployment:

Static code analysis technologies often represent a significant investment by software organizations looking to automate parts of their application security assurance programs. Not only do these technologies represent a monetary investment, but they demand time and effort by staff members to setup, operate, and maintain them. In addition, staff members are required to check and act upon the results generated by the technology. Understanding the ideal deployment environment will maximize the derived value, help the organization uncover more potential security flaws and could avoid unplanned hardware purchase cost. The following factors are essential to understanding the technology's capabilities and hence ensuring its proper utilization.

1.1 Deployment Model:

Vendors deliver static code analysis technologies through one or both of the following models:

» **Desktop Technologies:** the vendor delivers the software as a package to their users, the package is installed locally inside the organization's premises on one or more machines.

» **Software-as-a-Service (SaaS))Technologies:** Users submit their applications' source code or binaries to the vendor, where they get scanned and the final results are delivered back to users.

This document will refer to Desktop-based static code analysis technologies as "tools" and will refer to SaaS-based static code analysis technologies as "services". The document could use the term "technology" to reference both desktop-based tools and SaaS-based services.

1.2 Tool Installation Support:

A static code tool should provide the following :
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 the tool and its subsystems.

» **SaaS Based Services:** since there is no download or installation typically involved in using a SaaS based services, the vendor should be able to provide the following:

- Clear instructions on how to get started using the service.
- Estimated turn-around time from the time at which the code is submitted until the time when the results are received.
- What measures are being taken to keep the submitted code or binaries as well as to the reports confidential.

Criteria (Contd.)

1.3 Deployment Architecture:

Vendors provide various 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 tool's capabilities of accelerating the scanning speed (e.g. ability to multi-chain machines, ability to take advantage of multi-threaded/multi-core environments, etc)
- The ability of the tool to scale in order to handle scanning more applications if needed.

1.4 Setup and Runtime Dependencies:

The vendor should be able to state whether the tool or the service uses a compilation based analysis or source code based analysis.

- **Compilation based analysis:** where the tool or the service first compiles the code together with all dependencies, or whether the binaries are just analyzed directly. Either ways, the tools or the service requires all the application's dependencies to be available before conducting the scan, therefore the application is scanned as close to the production environment as possible.
- **Source code based analysis:** does not require software dependencies to be available for the scan to run. This could allow for quicker scans since the dependencies are not required at scan time.
- **Dynamic based analysis:** where the tool or the service analyzes data collected from real-time (or simulated) application or code execution.

2. Technology Support:

Most organizations use more than one programming language internally 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 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 technologies available today support more than one programming language. However, an organization looking to use a static code analysis tool or service 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 vendor's supported list of programming languages and versions. Vendors provide several levels of support for the same language, understanding what level of support the vendor provides for each programming language, and their versions, in addition to the frameworks used and their versions, is key to understanding the coverage and depth provide by the technology for each language. One way of understanding the level of support for a particular language is to inspect the tool's signatures (AKA Rules or Checkers) for that language.

2.2 Programming Environment 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. Understanding the relationship between the application and the frameworks/libraries is key in order to detect vulnerabilities resulting from the application's usage of the framework or the library, and the following in particular:

- The ability of the tool or service to understand how frameworks (e.g. Struts, Spring, Rails, etc) work and analyze the data based on actual data flow, and follow tainted data between the application and the framework.
- The ability of the tool or service to connect client side (e.g. hidden field, etc) and server side traces (e.g. parameters).
- Identify whether the application is using the framework in an insecure manner.
- Identify well-known vulnerabilities as defined by the Common Vulnerabilities and Exposures (CVE)

Criteria (Contd.)

2.3 Technology Configuration Support:

Several tweaks provided by the tool could potentially uncover serious weaknesses.

- **Configuration Files Redefinition:** Extending the tool's knowledge to treat files with non-standard configuration files extensions (e.g. *.ini, *.properties, *.xml, etc) as configurations files.
- **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 by mapping the non-standard extension to a standard one.

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. For SaaS based services, the vendor should be able to indicate whether there are APIs to initiate the scan remotely, this becomes a desirable feature for scenarios involving large number of applications.

3.2 IDE integration support:

The vendor should be able to enumerate which IDEs (and versions) are being supported by the technology being evaluated, as well as what scanning using the IDE will incorporate (e.g. what exactly will get scanned). For example, does an Eclipse plugin scan JavaScript files and configuration files, or does it only scan Java and JSP files.

3.3 Build systems support:

Organizations usually utilize static analysis technologies differently. Some organizations scan the code as part of their daily build system. The vendor should be able to enumerate the build systems supported and their versions (Ant, Make, Maven, etc). In addition, the vendor should be able to describe what gets scanned exactly in this context.

3.4 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 tools 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 currently 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:** authoring custom signature are used to "educate" the tool of the existence of a custom cleansing module, custom tainted data sources and sinks as well as a way to enforce certain programming styles by developing custom signatures for these styles.
- **Rule Packaging:** Ability to package rules and to run selective sets of rules.
- **Selective Scanning:** Ability to run (re-run) a scan for one type of issue.
- **Unit Tests:** Ability to create unit tests to validate new or edited rules.
- **Training:** the vendor should state whether writing new signatures require extra training.

3.5 Scan configuration capabilities:

This includes the following capabilities:

- **Ability to schedule scans:** Scans are often scheduled after nightly builds, some other times they are scheduled when the CPU usage is at its minimum. Therefore, it might be important for the user to be able to schedule the scan to run at a particular time. For SaaS-based services, the vendor should indicate the allowed window of submitting code or binaries to be scanned.
- **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. For SaaS based service, the vendor should be able to provide accurate estimate of the results delivery.
- **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 analysis tool for a particular application. The 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.
- **Ability to perform incremental scans:** incremental scans proves helpful when scanning large applications multiple times, it could be desirable to scan only the changed portions of the code which will reduce the time needed to assess the results.
- **Ability to re-configure the source code's location:** The ability to re-configure the source code's location after scanning is accomplished. Some times, the scans will be executed on a machine and the results are opened on a different machine.

Criteria (Contd.)

3.6 Testing Capabilities:

Scanning an application for weaknesses is an important functionality of the tool or service. It is essential for the tools and services to be able to understand, accurately identify and report the following attacks and security weaknesses.

- API Abuse
- Application Misconfiguration
- Auto-complete Not Disabled on Password Parameters
- Buffer Overflow
- Command Injection
- Credential/Session Prediction
- Cross-site Scripting
- Denial of Service
- Escalation of Privileges
- Insecure Cryptography
- Format String
- Hardcoded Credentials
- HTTP Response Splitting
- Improper Input Handling
- Improper Output Encoding
- Information Leakage
- Insecure Data Caching
- Insecure File Upload
- Insufficient Account Lockout
- Insufficient Authentication
- Insufficient Authorization
- Insufficient/Insecure Logging
- Insufficient Password Complexity Requirements
- Insufficient Password History Requirements
- Insufficient Session Expiration
- Integer Overflows
- LDAP Injection
- Mail Command Injection
- Null Byte Injection
- Open Redirect Attacks
- OS Command Injection
- Path Traversal
- Race Conditions
- Remote File Inclusion
- Second Order Injection
- Session Fixation
- SQL Injection
- URL Redirection Abuse
- XPATH Injection
- XML External Entities
- XML Entity Expansion
- XML Injection Attacks
- XPATH Injection

3.7 Industry Standards Aided Analysis:

Providing industry-standard-based scanning becomes a desirable feature for many reasons. For example, OWASP Top 10, CWE/SANS Top 25, WASC Threat Classification, DISA/STIG etc provide organizations with starting points to their software security gap analysis and in other cases these classifications become metrics of minimum adherence to security standards. Providing industry standard-aided scanning becomes a desirable feature for many reasons.

4. Product Signature Update:

Product signatures (AKA rules or checkers) are what the static code analysis tools use to identify security weaknesses. When making a choice of a static analysis tool, one should take into consideration the following:

4.1 Frequency of signature update:

Providing frequent signature update to a static code analysis tool ensure the tool's relevance to threat landscape. 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 tool must provide a way for users to submit feedback on bugs, flawed rules, rule enhancement, etc.

5. Triage and Remediation Support:

A crucial factor in a static code analysis tool or service is the support provided in the triage process and the accuracy, effectiveness of the remediation advice. This is vital to the speed in which findings are assessed and remediated by the development team.

5.1 Finding Meta-Data:

Finding's meta-data is the information provided by the tool or service around the finding. Good finding meta-data helps the auditor or the developer to understand the weakness and act upon it quicker. The tool or service should provide the following a long 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 to the application.
- **Location:** the source code file location and the line number of the finding.
- **Data Flow:** the ability to trace tainted data from a source to a sink and vise versa.

5.2 Meta-Data Management:

- The tool or service should provide the ability to mark a finding as false positive.
- Ability to categorize false positives. This enforces careful consideration before marking a finding as false positive, it also allows the opportunity to understand common sources for false positive issues, which could help in optimizing the results.
- Findings marked as false positives should not appear in subsequent scans. This helps to avoid repeating the same effort on subsequent scans.
- The tool or the service should be able to merge\diff scan results. This becomes a desirable feature if\when the application is re-scanned, the tool or service should be able to append results of the second scan to the first one.
- The vendor should be able to indicate whether the tool or the service supports the ability to define policies that incorporate flaw types, severity levels, frequency of scans, and grace periods for remediation.

5.3 Remediation Support:

- The tool or service should provide accurate and customizable remediation advice.
- Remediation advise should be illustrated with examples written in the same programming language as the finding's.

6. Reporting Capabilities:

The tool or service reporting capability is one of its most visible functionalities to stakeholders. The tool or service should provide different ways to represent the results based on the target audience. For example, developers will need as much details as possible in order to be able to remediate the weakness properly in a timely fashion. However, upper management might need to focus on the report's high level summary, or the risk involved more so than the details of every weakness.

6.1 Support for Role-based Reports:

The tool or service 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 including 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.
  - Flow Details which indicates the tainted data flow from the source to the sink.
- 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

Criteria (Contd.)

6.2 Report Customization:

The tool or service should be able to support report customization. The tool or service 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 report's template to include the organization's logo, header, footer, report cover,etc.

6.3 Report Formats:

The vendor should be able to enumerate the report formats they support (PDF, XML, HTML, etc)

7. Enterprise Level Support:

When making a choice on a static analysis tool or service in the Enterprise, one should take into consideration the ability to integrate the tool or service into various enterprise systems, such as bug tracking, reporting, risk management and data mining.

7.1 Integration into Bug Tracking Systems:

Vendors should be able to enumerate the supported bug tracking applications, in particular the following criteria should be defined by the vendor:

- The way the tool or service integrates into the bug tracking systems, e.g. direct API calls, CSV export, etc
- Whether the tool or service supports manual or automatic bug submission and the criteria used in submitting the bugs.

7.2 Integration into Enterprise Level Risk Management Systems:

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 or service should provide integration into enterprise level risk management systems. The vendor should be able to provide

- The technology used to perform the integration.
- Whether a criteria could be defined to manage integration to enterprise level risk management systems.

7.3 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 generate more findings thus better utilizing training budge for instance.

Projects 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 code analysis tools and services could be used to configure similar projects with additional metadata to detect these patterns. This will build intelligence around them which could help then detect which application components have more security weaknesses and why.

7.4 Licensing Scheme:

Static Code Analysis tools and services varies in their licensing schemes. Usually, the following factors decide on the technology's total cost of ownership.

» **Licensing Scheme Factors:**

- **Metered scan (pay-per-line) license:** licensing fees depend on how many lines of code needs to be scanned.
- **Pay-per-application license:** where a license is issued for a specific application and can't be used for any other applications.
- **Time-based Subscriptions:** one or more applications could be scanned unlimited number of times before the expiration of the license.
- **Per-user licenses:** a user-based license that is usually combined with one or more of the other schemes.
- **Unlimited/perpetual Licenses:** for scanning unlimited applications by unlimited users.
- **Server costs:** for client\server models.

» **Licensing Scheme Enforcement:**

- **License Server:** dedicated server where licenses are stored and can be accessed by users on the network.
- **Local/node-locked:** License is tied to a specific OS type, machine and named user.
- **User locked:** license is tied to a specific username.
- **Floating:** a number of licenses are shared among a larger number of users over time.
- **Trust or contract based:** the licensing scheme mentioned in the contract is assumed to be honoured by the user with no extra enforcement.

Index A

Static Code Analysis Preparation Cheat Sheet:

Taking a decision regarding the best static code analysis tool or service to acquire could be a daunting task. However, preparation for such a task could be very helpful. Every technology is unique so as your corporate environment. The following is a set of information you need to gather which could make the decision much easier to take:

- 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 or service will be integrated into the Software Development Lifecycle
- How will the developers see the scan results
- Budget allocated to the technology 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.

Index B

References

- WASC Threat Classifications (http://projects.webappsec.org/w/page/13246978/Threat%20Classification)
- Web Applications Security Scanner Evaluation Criteria (http://projects.webappsec.org/w/page/13246986/Web%20Application%20Security%20Scanner%20Evaluation%20Criteria)
- NIST Source Code Security Analysis Analyzer 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 Analyzers For Static Code Analysis (http://en.wikipedia.org/wiki/List_of_analyzers_for_static_code_analysis)