Special Publication 500-270

# Source Code Security Analysis Tool Test Plan

Draft 1 for public comment of Version 1.0

**Michael Koo**
**Romain Gaucher**
**Vadim Okun**
**Information Technology Laboratory (ITL)**
**Software Diagnostics and Conformance Testing Division**
**9 March 2007**

NIST
**National Institute of Standards and Technology**
Technology Administration, U.S. Department of Commerce

# Abstract

This document provides a set of metrics, including test suites and methods, to determine how well a particular source code security analysis tool conforms to the requirements specified in *Source Code Security Analysis Tool Functional Specification Version 1.0* [SCA]. Each programming language has a corresponding set of test suites. The test suites are intended to be used by tool developers and tool users alike to increase their level of confidence in product quality. Each test suite consists of test cases. Each test case contains test description, weakness contained in the test case, expected result and test code. The detailed information of the test case, such as start parameters, procedures for executing a test file and test file itself can be retrieved from the SAMATE Reference Dataset (SRD) http://samate.nist.gov/SRD/.

As this document evolves, new versions will be posted to the web site at [http://samate.nist.gov/index.php/Source_Code_Security_Analysis](http://samate.nist.gov/index.php/Source_Code_Security_Analysis).

# Table of Contents

# 1 Introduction

There is a critical need in society to ensure that software assurance tools produce accurate, repeatable and objective results. The Software Assurance Metrics and Tool Evaluation (SAMATE) project at the National Institute of Standards and Technology (NIST) is working to establish a methodology for testing software assurance tools by developing functional specifications, test procedures, test criteria, and test suites. The results provide the information necessary for toolmakers to improve tools, for users to make informed choices about acquiring and using software assurance tools, and for interested parties to understand the tools' capabilities. This project is further described at http://samate.nist.gov/.

SAMATE is a joint project of the Department of Homeland Security and the NIST Information Technology Laboratory (ITL). Since all documents are posted on the web for public review and comment, the entire computer software assurance community has the opportunity to participate in the development of the specifications and test methods.

For this document, a Source Code Security Analyzer examines source code to detect and report weaknesses that can lead to security vulnerabilities. Other static analysis tools, for examples tools that scan bytecode or binary code or examine web sites, are not covered.

# 2 Purpose

This document, along with files in the SRD [SRD], provides a means to test the functions of a tool based on the requirements in *Source Code Security Analysis Tool Functional Specification Version 1.0* [SCA].

The test plan is generic in that it can be applied to any programming language. Each language will have its own specific set of test suites.

The test methodology described in this document focuses only on the requirements of in [SCA]. Testing the performance robustness, scalability, usability, etc. is outside of the scope of this document.

# 3 Test Methodology

For the reader's convenience, this section repeats the requirements in [SCA]. This section also describes the approach to measure the tool under test.

## 3.1 Requirements from [SCA]

### 3.1.1 Requirements for Mandatory Features

A source code security analysis tool must be able to achieve the following six mandatory tasks:

- **SCA-RM-1:** Identify all of the code weaknesses listed in Appendix A.
- **SCA-RM-2:** Generate a text analysis of the code weaknesses that it identifies.
- **SCA-RM-3:** Identify the weakness with a name semantically equivalent to those in Appendix A.
- **SCA-RM-4:** Specify the location of a weakness by providing the directory path, file name and line number.
- **SCA-RM-5:** Identify any weaknesses within the relevant the coding complexities listed in Appendix B
- **SCA-RM-6:** Have an acceptably low false-positive rate.

### 3.1.2 Requirements for Optional Features

If the tool supports the applicable optional feature, then the requirement for that feature applies:

- **SCA-RO-1:** Produce an XML-formatted report.
- **SCA-RO-2:** Not identify a weakness instance, which has been suppressed.
- **SCA-RO-3:** Use the CWE name of the weakness it identifies.

## 3.2 Measurement of Fulfillment of Requirements

Briefly, testing takes three steps:

1. Prepare – install tool, choose appropriate test suites
2. Run tool on test cases in test suites – determine if results are as expected
3. Summarize results

To prepare to run the tests, the tool must be installed, of course. The tester should establish what names the tool uses corresponding with the errors in SCA, Appendix A. This is one part of satisfying SCA-RM-3, the names that the tool uses mean the same thing as the names used in the test. For automated checking, a correspondence should be made between the two sets of names. If the tool uses CWE names, the correspondence, and SCA-RO-3, are trivial. SCA-RM-3 is still important: it is conceivable that a tool reports a weakness at the right place, but refers to it by some entirely inappropriate name.

Each computer language has three corresponding test suites. Test suite SCA-TS-1-*name*, where *name* is the name of the language, has programs with weaknesses. For instance, SCA-TS-1-CPP is for the C++ language and SCA-TS-1-Java is for Java. It tests features that are described in requirements SCA-RM-1 through SCA-RM-5. It can also test requirements SCA-RO-1 and SCA-RO-3. Test suite SCA-TS-2-*name* tests requirement SCA-RM-6, false alarms or false positives. Test suite SCA-TS-3-*name* tests requirement SCA-RO-2, suppressing warnings.

For each language to be tested, download the appropriate test suites SCA-TS-1 and TS-2. If SCA-RO-2, warning suppression, is to be tested, download the appropriate test suites TS-3, too.

Since there may be several hundred individual test results to check, we encourage users to write a harness for their own needs. The “More Downloads” section of the SRD will shortly have sample scripts. All the tests could be run, and results saved, then the determination made if the results are as expected. Alternatively the determination could be made as each test is run. The advantage of making the determination after all runs is that the determination can be rerun if need be without rerunning the tool on all the test cases.

Note that the results of running the tool on test suite SCA-TS-1 are used for many requirements. The results of SCA-TS-2 and SCA-TS-3 are used for one requirement each.

### 3.2.1 Mandatory Features

- To determine if SCA-RM-1 is met from the results of SCA-TS-1 …

  To determine if SCA-RM-2 is met from the results of SCA-TS-1 …

  To determine if SCA-RM-3 is met from the results of SCA-TS-1 …

  To determine if SCA-RM-4 is met from the results of SCA-TS-1 …

  To determine if SCA-RM-5 is met from the results of SCA-TS-1 …

  For each test case, the tool under testing is expected to generate a report that identifies the weakness with a name semantically equivalent to those in Appendix A of [SCA] and its location (e.g. path name and line number(s) of weakness).

- To determine what to report for SCA-RM-6 from the results of SCA-TS-2 …

  The number of errors reported divided by number of test cases will be the ratio of false positive.

### 3.2.2 Optional Features

- To test requirement SCA-RO-1, if XML-formatted report is not a default for the tool under testing, turn on that feature and then run test suite SCA-TS-1 for the appropriate language. The tool under test should generate report as described in 3.2.1 in XML-format.

- To test requirement SCA-RO-2, run test suite SCA-TS-3 first to prove the tool can identify the weaknesses. Then run SAC-TS-3 again after the weaknesses described in Appendix A are suppressed in the tool under test. The tool should generate no report on any weakness that is suppressed.
- To test requirement SCA-RO-3, run test suite SCA-TS-1 for the appropriate language. For each test case, the tool under testing is expected to generate a report that identifies the weakness with correct CWE name.

# 4 Test Suites

A test suite is a collection of test cases explicitly selected for a special purpose. Each test case is an atomic program that ensures a specific functionality required by SCA can be performed by the tool under testing. Test suites and their test cases are stored in SRD [SRD]. Each SRD test case entry provides test file, description of weakness, CWE classification, type of code complexity, location of the weakness and other test case metadata.

## *4.1 Test Suites for the C Language*

### 4.1.1 Test Suite SCA-TS-1-C (SRD Test Suite 45)

This test suite will cover source code weaknesses in C listed in Appendix A of *Source code security analysis tool Functional Specification* [SCA].

| **Source Code Weakness** | **CWE ID** | **Code Complexity** | **SRD Test case ID** | **Remark** |
|---|---|---|---|---|
| Basic XSS | 80 | Basic | 1794 | |
| | | Scope | 1781 | |
| | | Address alias level | 1919 | |
| | | Container | 1921 | |
| | | Loop complexity | 1792 | |
| Resource Injection | 99 | Basic | 1897 | |
| | | Scope | 1901 | |
| | | Address alias level | 1895 | |
| | | Container | 1899 | |
| OS Command Injection | 78 | Basic | 11 | Test function 'system()'. |
| | | Basic | 1780 | Test function 'execlp()'. |
| | | Scope | 1885 | |
| | | Local control flow | 1881 | |

| | | | | |
|---|---|---|---|---|
| | | Loop structure | 1883 | |
| SQL Injection | 89 | Basic | 1796 | |
| | | Array index complexity | 1798 | |
| | | Scope | 1800 | |
| Stack Overflow | 121 | Basic | 1486 | |
| | | Array index complexity | 1544 | |
| | | Scope | 1548 | |
| | | Basic | 1563 | Test function gets() |
| | | Basic | 1565 | Test function fgets() |
| | | Array index complexity | 1751 | |
| | | Array length/limit complexity | 1905 | |
| | | Index alias level | 1907 | |
| | | Loop Structure | 1909 | |
| Heap Overflow | 122 | Basic | 15 | |
| | | Scope | 1612 | |
| | | Array address complexity | 1843 | |
| | | Array index complexity | 1845 | |
| | | Memory location | 1847 | |
| Format string vulnerability | 134 | Basic | 10 | |
| | | Address alias level | 92 | |
| | | Scope | 93 | |
| | | Container | 1831 | |
| | | Local control flow | 1833 | |
| Improper Null Termination | 170 | Basic | 1849 | |
| | | Taint | 1857 | |
| | | Buffer address type | 1852 | |
| | | Container | 1854 | |
| | | Address alias level | 1850 | |
| Heap Inspection | 244 | Basic | 1737 | |
| Often Misused: String Management | 251 | Basic | 1865 | |
| | | Taint | 1873 | |
| | | Scope | 1871 | |
| | | Address alias level | 1867 | |
| | | Container | 1869 | |
| Hard-coded | 259 | Basic | 1810 | |

| | | | | |
|---|---|---|---|---|
| Password | | | | |
| | | Local control flow | 1839 | |
| | | Loop structure | 1841 | |
| | | Container | 1837 | |
| | | Array Index Complexity | 1835 | |
| Time-of-check Time-of-use race condition | 367 | Basic | 102 | |
| | | Basic | 1806 | |
| | | Local Control Flow | 1808 | |
| Unchecked Error Condition | 391 | Basic | 1928 | |
| Memory leak | 401 | Basic | 1585 | |
| | | Scope | 1588 | |
| Unrestricted Critical Resource Lock | 412 | Basic | 1863 | |
| Double Free | 415 | Basic | 1508 | |
| | | Buffer address type | 99 | |
| | | Loop structure | 1829 | |
| | | Local control flow | 1827 | |
| | | Scope | 1590 | |
| Use After Free | 416 | Basic | 6 | |
| | | Scope | 1917 | |
| | | Address alias level | 1911 | |
| | | Container | 1915 | |
| | | Buffer address type | 1913 | |
| Uninitialized variable | 457 | Data type | 78 | Data type is integer |
| | | Data type | 1482 | Data type is Char * |
| | | Loop Structure | 1757 | |
| Unintentional pointer scaling | 468 | Basic | 1782 | |
| Improper pointer subtraction | 469 | Basic | 1860 | |
| Null Dereference | 476 | Basic | 1760 | |
| | | Address alias level | 1875 | |
| | | Local control flow | 1877 | |
| | | Scope | 1879 | |
| Leftover Debug | 489 | Basic | 1861 | |

| | | | | |
|---|---|---|---|---|
| Code | | | | |

### 4.1.2 Test Suite SCA-TS-2-C (SRD Test Suite 46)

This test suite will be used to examine false positive ratio generated by the tool under testing for C applications.

| Source Code Weakness | CWE ID | Code Complexity | SRD Test case ID | Remark |
|---|---|---|---|---|
| Basic XSS | 80 | Basic | 1795 | |
| | | Scope | 1924 | |
| | | Address alias level | 1920 | |
| | | Container | 1922 | |
| | | Loop complexity | 1793 | |
| Resource Injection | 99 | Basic | 1898 | |
| | | Address alias level | 1896 | |
| | | Container | 1900 | |
| | | Scope | 1902 | |
| OS Command Injection | 78 | Basic | 1931 | Test function 'system()'. |
| | | scope | 1886 | |
| | | Local control flow | 1882 | |
| | | loop structure | 1884 | |
| SQL Injection | 89 | Basic | 1797 | |
| | | Array index complexity | 1799 | |
| | | Scope | 1801 | |
| | | Loop structure | 1930 | |
| Stack Overflow | 121 | Basic | 1547 | |
| | | Array index complexity | 1545 | |
| | | Scope | 1549 | |
| | | Basic | 1566 | |
| | | Array length/limit complexity | 1906 | |
| | | Basic | 1602 | |
| | | Index alias level | 1908 | |
| | | Loop Structure | 1910 | |
| Heap Overflow | 122 | Basic | 1936 | |
| | | Scope | 1615 | |
| | | Array index complexity | 1844 | |
| | | Memory location | 1848 | |

| | | | | |
|---|---|---|---|---|
| | | Array index complexity | 1574 | |
| | | Scope | 1613 | |
| Format string vulnerability | 134 | Scope | 1562 | |
| | | Address alias level | 1560 | |
| | | Local control flow | 1834 | |
| | | Container | 1832 | |
| | | Scope | 1556 | |
| Improper Null Termination | 170 | Basic | 1856 | |
| | | Taint | 1858 | |
| | | Buffer address type | 1853 | |
| | | Container | 1855 | |
| | | Address alias level | 1851 | |
| Often Misused: String Management | 251 | Basic | 1866 | |
| | | Taint | 1874 | |
| | | Scope | 1872 | |
| | | Address alias level | 1868 | |
| | | Container | 1870 | |
| Hard-coded Password | 259 | Local control flow | 1840 | |
| | | Loop structure | 1842 | |
| | | Container | 1838 | |
| | | Array Index Complexity | 1836 | |
| Time-of-check Time-of-use race condition | 367 | Basic | 1892 | |
| | | Local Control Flow | 1894 | |
| Unchecked Error Condition | 391 | Basic | 1929 | |
| Memory leak | 401 | Basic | 1933 | |
| | | Scope | 1586 | |
| | | Address alias level | 1589 | |
| | | Container | 1925 | |
| | | Loop structure | 1926 | |
| Unrestricted Critical Resource Lock | 412 | Basic | 1864 | |
| Double Free | 415 | Basic | 1932 | |

| | | | | |
|---|---|---|---|---|
| | | Loop structure | 1830 | |
| | | local control flow | 1828 | |
| | | Scope | 1591 | |
| Use After Free | 416 | Scope | 1918 | |
| | | Address alias level | 1912 | |
| | | Container | 1916 | |
| | | Buffer address type | 1914 | |
| Uninitialized variable | 457 | Basic | 1935 | |
| Unintentional pointer scaling | 468 | Data type | 1927 | |
| Null Dereference | 476 | Basic | 1934 | |
| | | Scope | 1880 | |
| | | Address alias level | 1876 | |
| | | Local control flow | 1878 | |
| Leftover Debug Code | 489 | Basic | 1862 | |

### 4.1.3 Test Suite SCA-TS-3-C (SRD Test Suite 47)

This test suite will examine whether the tool identifies weakness after it has been suppressed for C applications.

| Source Code Weakness | CWE ID | Code Complexity | SRD Test case ID | Remark |
|---|---|---|---|---|
| Basic XSS | 80 | Basic | 1794 | |
| Resource Injection | 99 | Basic | 1897 | |
| OS Command Injection | 78 | Basic | 11 | Test function 'system()'. |
| SQL Injection | 89 | Basic | 1796 | |
| Stack Overflow | 121 | Basic | 1486 | |
| Heap Overflow | 122 | Basic | 15 | |
| Format string vulnerability | 134 | Basic | 10 | |
| Improper Null Termination | 170 | Basic | 1849 | |
| Heap Inspection | 244 | Basic | 1737 | |
| Often Misused: String | 251 | Basic | 1865 | |

| | | | | |
|---|---|---|---|---|
| Management | | | | |
| Hard-coded Password | 259 | Basic | 1810 | |
| Time-of-check Time-of-use race condition | 367 | Basic | 102 | |
| Unchecked Error Condition | 391 | Basic | 1928 | |
| Memory leak | 401 | Basic | 1585 | |
| Unrestricted Critical Resource Lock | 412 | Basic | 1863 | |
| Double Free | 415 | Basic | 1508 | |
| Use After Free | 416 | Basic | 6 | |
| Uninitialized variable | 457 | Data type | 78 | Data type is integer |
| Unintentional pointer scaling | 468 | Basic | 1782 | |
| Improper pointer subtraction | 469 | Basic | 1860 | |
| Null Dereference | 476 | Basic | 1760 | |
| Leftover Debug Code | 489 | Basic | 1861 | |

## *4.2 Test Suites for the C++ Language*

### 4.2.1 Test Suite SCA-TS-1-CPP

This test suite will cover source code weaknesses in C++ listed in Appendix A of *Source code security analysis tool Functional Specification* [SCA]. The test cases of this suite are still under construction.

### 4.2.2 Test Suite SCA-TS-2-CPP

This test suite will be used to examine false positive ratio generated by the tool under testing for C++ applications. The test cases of this suite are still under construction.

### 4.2.3 Test Suite SCA-TS-3-CPP

This test suite will examine whether the tool identifies weakness after it has been suppressed for C++ application. The test cases of this test suite are under construction.

## *4.3 Test Suites for the Java Language*

### 4.3.1 Test Suite SCA-TS-1-JAVA

This test suite will cover source code weaknesses in JAVA listed in Appendix A of *Source code security analysis tool Functional Specification* [SCA]. This test suite is still under development.

### 4.3.2 Test Suite SCA-TS-2-JAVA

This test suite will be used to examine false positive ratio generated by the tool under testing for JAVA applications. The test cases of this suite are still under construction.

### 4.3.3 Test Suite SCA-TS-3-JAVA

This test suite will examine whether the tool identifies weakness after it has been suppressed for JAVA application. The test cases of this test suite are under construction.

# 5 Reference:


[SCA] Source Code Security Analysis Tool Functional Specification Version 1.0
http://samate.nist.gov/docs/SAMATE_source_code_analysis_tool_spec_01_29_07.pdf

[SRD] SAMATE Reference Dataset
http://samate.nist.gov/SRD/
To retrieve test suite, click on "Test Suites" on the tool bar. A list of test suite will display. Select the required test suite.