NIST

Special Publication 500-269

Software Assurance Tools:
Web Application Security Scanner
Functional Specification Version 1.0

Paul E. Black

Elizabeth Fong

Vadim Okun

Romain Gaucher

Information Technology Laboratory (ITL)

Software Diagnostics and Conformance Testing Division

21 August, 2007

Abstract:

Software assurance tools are a fundamental resource for providing an assurance argument for today’s software applications throughout the software development lifecycle (SDLC). Software requirements, design models, source code, and executable code are analyzed by tools in order to determine if an application is secure. This document specifies the functional behavior of one class of software assurance tool: the web application security scanner tool. Due to the wide spread use of the World Wide Web and proliferation of web application vulnerabilities, application level web security and assurance requires major attention. This specification defines a minimum capability to help software professionals understand how a tool will meet their software assurance needs.

Keywords:

Homeland security; software assurance; software assurance tools; specification; web application; web application security scanner tool; vulnerability

Errata to this version:

None

Note:

This specification has not had public review. We plan to have public review and comment, then to release a revised version.

Disclaimer:

Any commercial product mentioned is in for information only; it does not imply recommendation or endorsement by NIST nor does it imply that the products mentioned are necessarily the best available for the purpose.

Table of Contents

1 Introduction ... 4

1.1 Purpose ... 4

1.2 Scope ... 4

1.3 Audience ... 5

1.4 Technical Background ... 5

1.5 Glossary of Terms ... 6

2 Functional Requirements ... 8

2.1 High Level View ... 8

2.2 Requirements for Mandatory Features ... 8

2.3 Requirements for Optional Features ... 8

3 References ... 9

Annex A Web Application Vulnerabilities ... 10

Annex B Levels of Defense ... 12

1 Introduction

The NIST Software Assurance Metrics and Tool Evaluation (SAMATE) project provides a measure of confidence in the software tools used in software assurance evaluations. It provides software developers and purchasers with a means of deciding whether the tools in consideration for use should be applied to the purposes required. Through the development of tool functional specifications, test suites and tool metrics, the SAMATE project aims to better quantify the state of the art for all classes of software assurance tools.

1.1 Purpose

This document constitutes a specification[*] for a specific type of software assurance tool, which is referred to here as web application security scanner.

Web application security scanner is an automated program that examines web applications for potential security vulnerabilities [Fong and Okun]. In addition to searching for web application specific vulnerabilities, the tools also look for software coding errors, such as buffer overflows.

This document specifies basic functional requirements for web application security scanner tools used in evaluations of application layer software on the web. Production tools should have capabilities far beyond those indicated here. Many important attributes, like cost and ease of use, are not covered.

The purposes of the web application security scanner tool specification are:

- to define a minimum (mandatory) level of functions that must be available in the tool in order for the purchaser and vendors to qualified the product,
- to produce unambiguous clauses as to what is required in order to conform, and
- to build consensus on tool functions and requirements toward the evaluation of software security assessment tools for behavior and effectiveness.

The functionality described herein may be embedded in a larger tool with more functionality. The functionality could also be covered by several specialized tools.

The functional requirements are the basis for developing test suites to measure the effectiveness of web application security scanner tools. Accompanying documents detail test cases and methods to ascertain to what extent a tool meets these requirements.

1.2 Scope

This specification is limited to software tools that examine software applications, which respond to dynamic web page requests over HyperText Transfer Protocol (HTTP) or other similar protocols. Web applications are designed to allow any user with a web browser and an Internet connection to interact with them in a platform independent way.

This document specifies minimum functionality only. Critical production tools should have capabilities far beyond those indicated here. Many important attributes, like compatibility with integrated development

[*] This specification has not had public review. We plan to have public review and comment, then to release a revised version

environments (IDEs) and ease of use, are not addressed. We focus on the ability of tools to detect vulnerabilities and do not consider other characteristics of the tool, such as supported protocols, supported parsers, supported method, and supported authentication.

Outside of scope are the following types of tools or functionality:

- Tools that scan other artifacts, like requirements, bytecode or binary code.
- Database scanners.
- Functionality that scans web services.
- Other types of system security tools, e.g., firewalls, anti-virus, gateways, routers, switches, intrusion detection/protection systems.
- Functionality that checks infrastructure vulnerabilities, deployment, configuration issues, such as running an obsolete version of web server.

Source code security analysis tools are covered by another specification developed by SAMATE [NIST SP 500-268].

The misuse or proper use of a tool is outside the scope of this specification. The issues and challenges in engineering secure systems and their software are outside the scope of this specification.

It is assumed that, this document, being a specification, only describes what functions should be (or must be) in the tool, and not how to implement these functions. Therefore, techniques, methods and algorithms for implementing certain functions are omitted here.

This specification is not a survey of tool functions and does not discuss variabilities and complexities on how the tool is capable of performing.

1.3 Audience

The target audience for this specification is users and evaluators of web application security scanners. It may also be useful to security professionals with an interest in web security, software assurance researchers, and developers of web application security scanners.

1.4 Technical Background

This section gives some technical background, defines terms we use in this specification, explains how concepts designated by those terms are related, and details some challenges in web application vulnerability assessment for security assurance.

Different authors may use the same term to refer to different concepts. For clarity we give our definitions. To begin, any event, which is a violation of a particular system's explicit (or implicit) security policy, is a *security failure*, or simply, failure. For example, if an unauthorized person gains "root" or "admin" privileges or if unauthorized people can read Social Security numbers through the World Wide Web, security has failed.

A *vulnerability* is a property of system security requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure. (After [NIST SP 800-27]) In our model the source of any failure is a latent vulnerability. If there is a failure, there must have been vulnerability. A vulnerability is the result of one or more *weaknesses* in requirements, design, implementation, or operation.

An *exploit* is a piece of software or technique that takes advantage of a vulnerability to cause a failure. An *attack* is a specific application of an exploit [After AP-Glossary]. In other words, an attack is an action (or sequence of actions) that takes advantage of vulnerability.

A *web application* is a software application executed by a web server, which responds to dynamic web page requests over HTTP [WASC-Glossary]. A web application is comprised of a collection of components, which reside on a web server and interact with databases or other sources of dynamic content. Using the infrastructure of the Internet, web applications allow service providers and clients to share and manipulate information in a platform-independent manner. A web application has a distributed n-tiered architecture. Typically, there is a client (web browser), a web server, an application server (or several application servers), and a persistence (database) server.

Since HTTP is a stateless protocol, web applications use separate mechanisms to maintain application state, or *context*, during a session. A *session* is a series of interactions between user and web application during a single visit to the web site. The context may be maintained via GET or POST session, variables, cookies, and other methods.

A *web application security scanner* is an automated program designed to examine web applications for security vulnerabilities. In using the term vulnerability, we do not attempt to differentiate between the issues that cause critical, costly security failures (e.g., theft of credit card information) and those that do not (e.g., a buffer overflow that causes loss of personal preference information on a non-commercial site).

When a web application identifies vulnerability, it reports one or more attacks. To allow the developers to verify and fix the vulnerability, an attack report must contain relevant information, including script location (e.g., http://foo.com/where/search.pl), input parameters (e.g., file=/etc/passwd), and context. This information is contained within the HTTP headers, so a tool may report the headers. Alternatively, a tool may report this information in a different format. Additionally, the output page can help developer verify the vulnerability.

In order to provide different levels of access to different users, web applications employ various authentication mechanisms. Some functions of a web application may only be accessible to authenticated users; others may produce different output depending on the level of access of the user. A web application security scanner must be able to login, possibly with the help of the user, to an application. Once logged in, the scanner must be able to maintain the logged-in state and recognize when it was logged out.

Since a failure only takes one vulnerability, the requirements have a tone of catching all vulnerabilities. Practical considerations require the *false positive rate* [Fleiss] to be acceptably low for the domain.

In addition, vulnerabilities within one type differ significantly in terms of difficulty of exploiting them and types of attacks that are effective against them. A web application security scanner may be able to find one SQL injection vulnerability, but fail to detect another. The reason is that web application developers implement different defense measures that make attacks more difficult. Annex B presents common defense mechanisms, and then combines them into defense levels of increasing strength.

To save analysis time in later runs, some tools allow the user to *suppress* vulnerability instances so they are not reported again.

Modern web applications use various advanced client-side technologies. Ideally, a web application scanner must understand and support these technologies. At a minimum, it must recognize the presence of a technology that it does not (fully) support and alert the tool user that this may cause its vulnerability assessment to be incomplete.

1.5 Glossary of Terms

This glossary was added to provide context for terms used in this document. Many of the terms were copied from the web security glossary developed by Web Application Security Consortium [WASC-glossary].

Name	Description

(Web) Application Server	A software server, typically using HTTP, which has the ability to execute dynamic web applications. Also known as a middleware, this piece of software is normally installed on or near the web server where it can be called upon.
Authentication	The process of verifying the identity or location of a user, service or application. Authentication is performed using at least one of three mechanisms: “something you have”, “something you know” or “something you are”. The authenticating application may provide different services based on the location, access method, time of day, etc.
Authorization	The determination of what resources a user, service or application has permission to access. Accessible resources can be URL’s, files, directories, servlets, databases, execution paths, etc.
Attack	An action (or sequence of actions) that takes advantage of vulnerability
Exploit	An exploit is a technique or software code (often in the form of scripts) that takes advantage of a vulnerability or security weakness in a piece of target software.
False negative	Failure of a tool to report a weakness, when in fact there is one present in the code.
False positive	Reporting of a vulnerability by a tool, where there is none.
Session	A session is a series of interactions between user and web application during a single visit to the web site.
Security vulnerability	A property of system security requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure.
Source code	A series of statements written in a human-readable computer programming language.
Universal Resource Locator (URL)	A standard way of specifying a location of an object, normally a web page, on the Internet.
Unvalidated input	User-supplied input data that has not been examined or filtered prior to use.
Unfiltered output	Application output that has not been filtered prior to being returned to the client.
Vulnerability	A property of system security requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure.
Web application	A software application executed by a web server which responds to dynamic web page requests over HTTP.
Web application security scanner	An automated program that searches for software security vulnerability within web applications.
Weakness	A defect in a system that may (or may not) lead to a vulnerability.

2 Functional Requirements

In this section we first give a high-level description of the functional requirements for web application security scanner, and then detail the requirements for mandatory and optional features.

2.1 High Level View

A web application security scanner(s) shall be able to (at a minimum):

- Identify a select set of vulnerabilities in a web application.
- For each vulnerability, generate a text report indicating an attack.

2.2 Requirements for Mandatory Features

In order to meet this baseline capability, a web application security scanner(s) must be able to accomplish the tasks described in the mandatory requirements listed below. The tool(s) shall:

WA-RM-1: Identify all of the vulnerabilities listed in Annex A.

WA-RM-2: Report an attack that demonstrates the vulnerability.

WA-RM-3: Specify the attack by providing script location, inputs, and context.

WA-RM-4: Identify the vulnerability with a name semantically equivalent to those in Annex A.

WA-RM-5: Be able to authenticate itself to the application and maintain logged-in state.

WA-RM-6: Have an acceptably low false positive rate.

2.3 Requirements for Optional Features

The following requirements apply to optional tool features. If the tool under test supports the applicable optional feature, then the requirement for that feature applies, and the tool can be tested against it. This means that a specific tool might optionally provide none, some or all of the features described by these requirements. Optionally, the tool(s) shall:

WA-RO-1: Find any vulnerability within the defense mechanisms listed in Annex B.

WA-RO-2: Not identify a vulnerability instance that has been suppressed.

WA-RO-3: Indicate remediation tasks.

WA-RO-4: Produce an eXtended Markup Language (XML) formatted report.

WA-RO-5: Assign a severity rating to the vulnerabilities they identify.

3 References

[AP-Glossary] Sean Barnum, Amit Sethi, *Attack Pattern Glossary*, in Build Security In.

[CWE] Common Weakness Enumeration, The MITRE Corporation, web site http://cve.mitre.org/cwe/index.html#tree

[Fleiss] Fleiss, J. L. (1981). *Statistical Methods for Rates and Proportions*, 2nd ed., John Wiley and Sons, New York, pp 4-8.

[Fong and Okun] Elizabeth Fong and Vadim Okun, *Web Application Scanners: Definitions and Functions*, 40th Annual Hawaii International Conference on System Sciences (HICSS'07), p. 280b, 2007.

[Fong et. al] Elizabeth Fong, Romain Gaucher, Vadim Okun, Paul E. Black and Eric Dalci, *Building a Test Suite for Web Application Scanners*, Hawaii International Conference on System Sciences (HICSS'08), to appear.

[NIST SP 500-268] Paul E. Black, Michael Kass and Michael Koo, *Source Code Security Analysis Tool Functional Specification Version 1.0*, NIST Special Publication 500-268, May 2007.

[NIST SP 800-27] Engineering Principles for Information Technology Security (A Baseline for Achieving Security), NIST SP 800-27, Revision A, June 2004. Available at http://csrc.nist.gov/publications/nistpubs/

[OWASP-TOP-10] OWASP Top 10: The Ten Most Critical Web Application Vulnerabilities, http://www.owasp.org/index.php/Top_10_2007

[PCI-DSS] Payment Card Industry (PCI) Data Security Standard, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf

[WASC-Glossary] Web Application Security Consortium: Glossary, web site http://webappsec.org/projects/glossary/

[WASC-Threat] Web Application Security Consortium: Threat Classification, web site http://www.webappsec.org/projects/threat/

Annex A Web Application Vulnerabilities

The web application vulnerabilities in this table represent a “base set” of vulnerabilities that a web application security scanner must be able to identify if it supports the technology and languages in which the vulnerability exists. Criteria for selection of vulnerabilities include:

- Found in existing applications today.
- Recognized by tools today.
- Likelihood of exploit or attack is medium to high.

In devising the table, we used several taxonomies, including [OWASP-TOP-10], [CWE], and [WASC-Threat]. The table also provides a, possibly incomplete, mapping of the vulnerability names to Open Web Application Security Project (OWASP) Top Ten 2007, Common Weakness Enumeration (CWE), and the Payment Card Industry (PCI) Data Security Standard [PCI-DSS]. The mapping identifies related (not equivalent) terms.

Name	**Description**	**Related terms**	**OWASP Top Ten 2007**	**CWE ID**	**PCI DSS 1.1**
Cross Site Scripting (XSS)	A web application accepts user input (such as client-side scripts and hyperlinks to an attacker’s site) and displays it within its generated web pages without proper validation.	Reflected XSS, persistent (stored) XSS, DOM-based XSS	A1	79	X
SQL Injection	Unvalidated input is used in construction of an SQL statement.	Blind SQL injection	A2	89	X
OS Command Injection	Unvalidated input is used in an argument to a system operation execution function.		A2	78	X
XML Injection	Unvalidated input is inserted into an XML document.	XPath injection, XQuery injection	A2	91	X
HTTP Response Splitting	Unvalidated input is used in construction of HTTP response headers.	CRLF injection	A2	113, 93	X
Malicious File Inclusion	Unvalidated input is used in an argument to file or stream functions.	File inclusion, Remote code execution, Directory traversal	A3	98	

Insecure Direct Object Reference	Unvalidated input is used as a reference to an internal implementation object, such as a file, directory, or database key.	Parameter tampering, Cookie poisoning, Path manipulation	A4	233, 73, 472	X
Cross Site Request Forgery (CSRF)	An application authorizes requests based only on credentials that are automatically submitted by the browser. A CSRF attack forces a logged-in victim's browser to send a request to a vulnerable application, which then performs the chosen action on behalf of the victim, to the benefit of the attacker.	Session riding, One-click attacks, Hostile Linking	A5	352	
Information Leakage	Disclosure of sensitive information or the internal details of the application.	File and directory information leaks, System information leak.	A6	538, 200, 497	X
Improper Error Handling	Error message may display too much information that is useful in exploring a vulnerability.	Error message information leaks, Detailed error handling	A6	388, 209, 390	X
Weak Authentication and Session Management	Lack of proper protection of account credentials and session tokens through their lifecycle.		A7	287	X
Session Fixation	Authenticating a user without invalidating any existing session identifier. This gives an attacker the opportunity to steal authenticated sessions		A7	384	X
Insecure Communication	Transmitting sensitive information (session tokens, credit card numbers or health records) without proper encryption (e.g., SSL).		A9		X
Unrestricted URL Access	Missing or insufficient access control for sensitive URLs and functions.	Predictable resource location, security by obscurity	A10	425	X

Annex B Levels of Defense

A tool must be able to identify vulnerabilities when the application developer implements certain defense measures. This annex presents common defense mechanisms that can be implemented to make various attacks more difficult, and then combines them into defense levels of increasing strength [Fong et. al]. The following table of defense mechanisms is not comprehensive. To be effective, these defense mechanisms must be implemented on the server side.

Defense Mechanism	**Description**	**Example**	**Affected vulnerabilities**
Typecasting	Convert the input string to specific type, such as integer, Boolean, double.	(int)($_GET['var']) transforms "8<script>" into the integer 8	XSS, Injections, Cookie poisoning
Meta-character replacement	Encode characters from a blacklist	"<" is replaced with "&lt;" for HTML documents, quotes replacement… For XSS, replace these characters: ', ", <, >, &, %, #	All technical vulnerabilities
Restrict input range	Restrict the range of integers, the type of an entry (only alphanumeric), length of a string etc.	For HTML injection, use a regular expression such as: [a-zA-Z0-9_\s]+ to restrict the input to alphanumeric characters plus underscore and space. Using data binding for SQL queries (prepared statements), etc.	All technical vulnerabilities
Restricted user management	Use a restricted account for performing data manipulation, SQL queries, etc.	If user is not logged in, use a read-only SQL account that only allows commands like SELECT and EXECUTE; no UPDATE or INSERT allowed.	SQL Injection, OS Command Injection
Use of stronger function	Use a stronger function for performing a secure action	Use SHA-256 instead of SHA-1 or MD5, Salt the passwords, Secure cookies. Using HttpOnly cookies to prevent client-side script to read them.	Weak cryptographic functions, Insecure cookies
Token Validation	Use a unique token to verify the origin of the HTTP request	By adding a unique identifier (token) in the forms or in the HTTP header, the application must be able to verify the origin of the request	Cross-Site Request Forgery
Character encoding handling	Canonicalize resource names and other input that may contain encoded characters.	Sample attacks using encoding: XSS: The UTF-7 encoded string: "+ADw-script+AD4-	XSS, Injections, Cookie poisoning

		alert('XSS')+ADsAPA-/script+AD4-“ should be interpreted as “<script>alert(‘XSS’)</script> SQL Injection: An injection technique consist of replacing the strings to inject by a concatenation of characters	
Hide information	Hide information such as errors, Session ID etc.		All vulnerabilities that can produce an informative output (SQL Injection, XSS, File Inclusion, etc.), Session Management

We can combine the defense mechanisms into *levels of defense* of increasing strength. Each level includes the mechanisms of the previous levels. The following table presents levels of defense for several vulnerability types.

Level of defense	**Description of defense mechanisms used**
Cross-Site Scripting (XSS)	
0	No input filtering. For example, the following URL will result in a pop-up window: http://.../bank/index.php?q=<script>alert(1);</script>
1	Level 0 + Typecasting
2	Level 1 + Meta-character replacement Using the PHP htmlentities function which escapes every HTML character. For example, http://.../bank/index.php?q=<script>alert(1);</script> will insert in the page: &lt;script&gt;alert(1); &lt;/script&gt;
3	Level 2 + Using a special function which looks for possible nested JavaScript.
4	Level 3 + Probing the charset and decoding the inputs.
Cross-Site Request Forgery (CSRF)	
0	No validation
1	Level 0 + Check the HTTP referrer
2	Level 1 + Using a unique token validation
SQL Injection	
0	No filtering of SQL query parameters. Example: http://.../bank/login.php?username=’ OR 1=1;--&password=
1	Level 0 + Hide information Hiding the MySQL errors
2	Level 1 + Typecasting
3	Level 2 + Meta-character replacement. Escaping potential MySQL characters: \x00, \n, \r, \, ', " and \x1a.

4	Level 3 + Restricted user management
5	Level 4 + Using prepared statements or similar methods
File Inclusion	
0	Include input file name concatenated with ‘.inc’
1	Level 0 + Test that file exists (works for remote files)
2	Level 1 + Test that file is in the Apache DOCUMENT_ROOT
3	Level 2 + Meta-character replacement The file name does not contain special characters, such as /etc/…, /…/.., so the file is restricted to a certain directory.