WEB APPLICATION SECURITY SCANNER EVALUATION CRITERIA

VERSION 1.0

Web Application Security Consortium

TABLE OF CONTENTS

INTRODUCTION

Web Application Security Scanners are automated tools to test web applications for common security problems such as Cross-Site Scripting, SQL Injection, Directory Traversal, insecure configurations, and remote command execution vulnerabilities. These tools crawl a web application and locate application layer vulnerabilities and weaknesses, either by manipulating HTTP messages or by inspecting them for suspicious attributes.

A large number of web application scanning tools are available, both commercial and open source. Effective use of these tools is an important part of a thorough web application security assessment, and regular security scans are required to comply with security requirements such as section 6.6 of the Payment Card Industry Data Security Standard (PCI-DSS).

The Web Application Security Scanner Evaluation Criteria (WASSEC) is a set of guidelines to evaluate web application scanners on their ability to effectively test web applications and identify vulnerabilities. It covers areas such as crawling, parsing, session handling, testing, and reporting.

The goal of the WASSEC is to create a vendor-neutral document to help guide web application security professionals during web application scanner evaluations. This document provides a comprehensive list of features that should be considered when conducting a web application security scanner evaluation. Different users will place varying levels of importance on each feature, and the WASSEC provides the user with the flexibility to take this comprehensive list of potential scanner 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 web application security scanners must provide in order to be considered a "complete" scanner, and evaluating specific products and providing the results of such an evaluation is outside the scope of the WASSEC 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 500-269, "Software Assurance Tools: Web Application Security Scanner Functional Specification Version 1.0", contains minimal requirements for mandatory and optional web application scanner features. This document can be found at https://samate.nist.gov.

CONTRIBUTORS

This document is the result of a team effort. The following people have contributed their time and expertise to this project:

ANURAG AGARWAL (WHITEHAT SECURITY)

VIJAY AGARWAL (FOUNDSTONE)

ROBERT AUGER (WASC)

EMILIO CASBAS (S21SEC)

LEONARDO CAVALLARI (NSRAV)

MATTHIEU ESTRADE (BEE WARE)

ROMAIN GAUCHER (CIGITAL, INC.)

JEREMIAH GROSSMAN (WHITEHAT SECURITY)

ROBERT HANSEN (SECTHEORY)

AMIT KLEIN

CHAD LODER (RAPID7)

KEN PFEIL (WESTLB AG)

TYLER REGULY (NCIRCLE NETWORK SECURITY)

IVAN RISTIC (BREACH SECURITY)

ORY SEGAL (IBM)

SHEERAJ SHAH (BLUEINFY SOLUTIONS PVT. LTD.)

CHRIS SHIFLETT (OMNITI)

BRIAN SHURA (APPSEC CONSULTING) [PROJECT LEADER]

TOM STRIPLING (SECURITY PS)

CHRIS SULLO (HEWLETT-PACKARD)

EVALUATION CRITERIA

1. PROTOCOL SUPPORT

In order to test web applications, a scanner must support all communication protocols that are commonly used by web applications and intermediary network devices. The underlying communication protocol used by web applications is the Hypertext Transfer Protocol (HTTP).

1.1. **Transport Support:** A web application scanner should support the following protocols:

1.1.1. **HTTP 1.1**: This is the current version of HTTP and is the version that is most widely used by HTTP clients and servers.

1.1.2. **HTTP 1.0:** This version of HTTP is still widely used, especially by proxy servers, and therefore should be supported by a web application scanner.

1.1.3. **SSL/TLS:** Web applications that handle sensitive information employ Transport Layer Security to encrypt the data in transit.

1.1.4. **HTTP Keep-Alive:** For performance reasons, a web application scanner should keep all connections open with the web server and re-use these connections for multiple HTTP requests unless instructed not to by the web server.

1.1.5. **HTTP Compression:** Some web servers compress content before sending it to the client in order to improve performance. Web application scanners should support HTTP compression for optimized performance and in order to properly parse compressed content encountered during a scan.

1.1.6. **HTTP User Agent Configuration:** The ability of a web application scanner to mimic a specific browser can be very important. Many web applications present different content depending on which browser the client is using. A scanner should provide the user with the ability to configure a specific user agent string that is to be used during the course of the scan.

1.2. **Proxy Support:** It is not uncommon for a scanner to not have direct access to a website. In this situation the scanner should allow the configuration of a proxy. It should be noted that, while sometimes necessary, scanning a web application through proxy can result in false negatives and should be avoided if possible. The following proxy capabilities should be supported by a scanner:

1.2.1. **HTTP 1.0 proxy**

1.2.2. **HTTP 1.1 proxy**

1.2.3. **Socks 4 proxy**

1.2.4. **Socks 5 proxy**

1.2.5. **PAC File Support:** In some cases a scanner will need to follow special rules to determine which URLs to request through a proxy and which URLs to request directly. A scanner should support the use of a proxy auto-configuration (PAC) file in order to properly handle these situations.

2. AUTHENTICATION

This section covers support for standard or widely deployed authentication methods that web application scanners should support in order to effectively test applications that require authentication.

2.1. **Authentication Schemes:** A web application security scanner should support the following authentication schemes:

2.1.1. **Basic**

2.1.2. **Digest**

2.1.3. **HTTP Negotiate (NTLM and Kerberos)**

2.1.4. **HTML Form-based**

2.1.4.1. **Automated -** The user supplies the scanner with a valid user id and password while configuring the scan.

2.1.4.2. **Scripted -** The scanner allows the user to script or record the login process prior to performing the scan.

2.1.4.3. **Non-Automated** - The scanner supports user intervention while the scan is being performed - for example, to solve a CAPTCHA or

enter a value from a hardware token for a web application that has implemented two-factor authentication.

2.1.5. **Single Sign On** (e.g. Cas, OpenID, AWS, AuthSub)

2.1.6. **Client SSL Certificates**

2.1.7. **Custom implementations** within the extensible HTTP authentication framework.

3. SESSION MANAGEMENT

During a web application security scan, it is crucial that scanners will maintain a "living" and valid session with the application at all times. A valid application session is mainly required for:

Web crawling - In order to achieve the best possible coverage of the application, the scanner needs to be in a valid living session that will allow it to discover all possible web elements (e.g. parameters, cookies, forms, links, etc.).

Test phase - Most application-layer security test cases require that the HTTP request sent by the scanner will be considered by the application as being "in-session". For example, testing an HTML form that requests bank account details, for SQL Injection. If the scanner sends an HTTP request which does not have valid session tokens, the application will simply disregard this request and will probably redirect the scanner to the application's login page.

In order to maintain a valid session, the scanner needs to cope with all sorts of session management mechanisms.

3.1. **Session Management Capabilities:** Scanners should support the following criteria:

3.1.1. Understand that the application is asking it to **start a new session**, using a certain type of token as a method of uniquely identifying this session

3.1.2. Perform a **session token refresh**, when instructed to do so by the application.

3.1.3. Detect that a currently held session was invalidated by the application (**session expired**).

3.1.4. Know how to start a new session and **reacquire session tokens**, in case the current session has expired.

3.2. **Session Management Token Type Support:** The most common types of session management tokens, which should be supported by a web application scanner, are:

3.2.1. **HTTP Cookies (RFC 2965):** HTTP cookies are probably the most commonly used type of web application session tokens. Web application scanners should be able to imitate the proper behavior of a web browser, such as accepting a request to set a new cookie and following a cookie value refresh instruction.

3.2.2. **HTTP Parameters:** Oftentimes, web applications use an HTTP parameter to track a web session. There are several ways in which an application can instruct the web browser to use parameters as session tokens, for example, by embedding them in subsequent HTML links, or as a hidden HTML form parameter. Web application scanners should be able to use HTTP parameters as session tokens.

3.2.3. **HTTP URL Path:** Some web applications embed session tokens inside the Path part of the URL, for example: `http://example.com/app/{SESSION_TOKEN}/dir/file.aspx`
In such cases, a web application scanner should be able to cope with such a token format. It is important to note that if the scanner is performing some sort of an application structure analysis during a scan, for example, in order to detect web directories, it will not treat such a session token as an actual part of the application structure, as this might generate irrelevant test cases.

3.3. **Session Token Detection Configuration:** Web application scanners should allow users to define the following session token configurations:

3.3.1. **Automatic Session Token Detection and Value Refresh:** The scanner will attempt to detect session tokens on its own and will decide which tokens should be automatically tracked / refreshed during the scan.

3.3.2. **Manual Session Token Configuration:** The user will define what denotes a session token, based on HTTP parameters, cookies, or any other type of configuration that is relevant (e.g. parse parts of the

response and extract certain data from it, which serves as the session token value).

3.4. **Session Token Refresh Policy:** Session tokens are sometimes refreshed by the web application during a scanning session. The scanner's session configuration should enable the user to define when, or during which phase of the scan, the session tokens should be refreshed. The following configuration options should be provided by the scanner:

3.4.1. **Fixed Session Token Value:** When a session token is marked to use a fixed value, this value will never change during the scan.

3.4.2. **Login Process Provided Token Value:** Once the web application scanner logs into the application it will extract token values, which were issued as part of the login process, and will use them until it detects that the session was invalidated.

3.4.3. **Dynamic Token Value:** The scanner will always use the most recent session token value, as supplied by the application at all times. This means that if during the crawling or testing phase of the scan, a new value is detected, the scanner will stop and refresh all subsequent HTTP requests, with the most recent value. This mode is extremely important for applications that make use of "nonces", which dynamically change all the time.

4. CRAWLING

Crawling is the term used to describe the action taken by a program as it browses from page to page on a website. The crawler will visit a starting page and parse the provided links, crawling to those pages. This process will continue until some user-defined criteria are reached or the process is completed and there are no more links to crawl. Crawling is essential to a web application security scan - it ensures that the scanner is aware of all linked pages that exist on the website. Crawlers should be as configurable as possible, allowing the user to define a large number of criteria to ensure a thorough and efficient crawl.

4.1. **Web Crawler Configuration:** With regards to crawling, a web application scanner should:

4.1.1. Provide the user with the option to **define a starting URL**. While '/' is commonly seen as the home URL of a website, this is not always the case.

4.1.2. Provide the user with the option to **define additional hostnames (or IP addresses)** in a list or a range. A website may consist of several domains or subdomains (e.g. www.example.com, members.example.com, faq.example.com, etc.). It is important that the user have the option of auditing these in a single scan.

4.1.3. Provide the user with the option to **define exclusions for**:

4.1.3.1. **Specific hostnames (or IPs)**

4.1.3.2. **Specific URLs or URL patterns (regular expressions)**

4.1.3.3. **Specific file extensions**

4.1.3.4. **Specific parameters**

4.1.4. Provide the user with the ability to **limit redundant requests**. Large sites with catalogs or photo galleries will often have numerous redundant pages. The capability to tune a crawler to limit the requests to these redundant pages should exist.

4.1.5. Provide the user with the option of **supporting concurrent sessions**. The capability should exist to enable support for multiple sessions (logins) or limit crawling to a single session.

4.1.6. Provide the user with the ability to **specify a request delay**. Network limitations may require that requests be throttled, while a high bandwidth network may not require this. By setting a delay between requests, network traffic can be limited accordingly.

4.1.7. Provide the user with the option to **define a maximum crawl depth**. The 'depth' of a website is a measurement of how many clicks it takes to reach a certain page from the starting page. If you clicked three links, then you are at a depth of three. The user may want to limit their scan to pages that are two or three levels deep.

4.1.8. Provide the user with a method of **training the crawler**. The process of training a crawler involves accessing a browser (internal or external to the scanning solution) and manually browsing a website. As the website

is browsed, valid pages and form inputs are recorded for the scanner to later use.

4.2. **Web Crawler Functionality:** During operation, a crawler should:

4.2.1. **Identify newly discovered hostnames**: Websites commonly include links to other sites. It is important to be able to identify when this is occurring so that the user can increase the scope of the crawl if necessary.

4.2.2. **Support automated form submission**: Forms that accept varying data types are common on websites. Quite often additional pages with new links exist behind these forms and the crawler should be able to access these additional pages.

4.2.3. **Detect error pages and custom 404 responses**: The crawler should know when it has accessed an error page or found a custom 'Page Not Found' (404) response.

4.2.4. **Redirect support:** The crawler should have the capability to:

4.2.4.1. **Follow HTTP redirects**: For example, HTTP status codes 301, 302, 303, and 307

4.2.4.2. **Follow Meta Refresh redirects**: For example:

```
<meta http-equiv="refresh" content="1; URL=http://abc.site">
```

4.2.4.3. **Follow JavaScript redirects:** For example:

```
<script type="text/javascript">
window.location = "http://www.example.com";
</script>
```

4.2.5. **Identify and accept cookies:** During any visit to a website, many cookies may potentially be set. The crawler should recognize these cookies, store them, and pass them back to the web server while crawling.

4.2.6. **Support AJAX applications:** At a minimum, a crawler should be able to automatically submit XmlHTTPRequests that are found during the crawling process.

5. PARSING

In order to thoroughly scan a web application for security problems, a web application scanner must first map out the web application's structure and functionality. The mapping process is done by the web crawler component, which makes use of different types of content parsers to extract information from web content. This information may include URLs, HTML forms, HTML form parameters, HTML comments, and so forth.

5.1. **Web Content Types:** A scanner should be capable of parsing the following content types to extract information about the application's structure and functionality:

5.1.1. **HTML:** HTML is the most elementary building block of the World Wide Web. Web application scanners must be able to fully parse and understand HTML content.

5.1.2. **JavaScript:** JavaScript is probably the most common web client-side scripting language in use today. Web application scanners should be able to parse JavaScript content to extract static and dynamic URLs. This includes in-line JavaScript embedded within "script" tags on HTML pages, script in standalone JavaScript files (usually referred to from a "script" tag with a "src" attribute), and JavaScript embedded within HTML tags in event handler attributes, such as an "img" tag with an "onClick" attribute.

5.1.3. **VBScript:** some web applications make use of VBScript as the language of choice for client-side scripting. Although less popular than JavaScript (mainly due to the fact that it is currently not supported in many browsers other than Microsoft's Internet Explorer), VBScript content parsing should be supported by web application scanners.

5.1.4. **XML:** XML content may appear in several scenarios when scanning web applications - for example, SOAP web services messages, Web Service Description Language (WSDL), XML Data Islands (XML Data embedded into an HTML page), WebDAV requests, XHTML, and so forth. Web application scanners should be able to parse XML content.

5.1.5. **Plaintext:** Some web applications may store information in plaintext files. Web application scanners should be able to parse plaintext files

and to extract relevant information. A common example of a plaintext file that exists in most web applications and contains application structure information is the Robots.txt file.

5.1.6. **ActiveX Objects:** Web applications may encapsulate client-side logic as ActiveX controls, which are downloaded and executed by the web browser. Web application scanners should be able to extract application information from such controls.

5.1.7. **Java Applets:** Similar to ActiveX objects, web developers may choose to encapsulate client-side logic as Java Applets. Applets are small applications that are delivered as Java bytecode and executed on the client-side by the JVM (Java Virtual Machine). Web application scanners should be able to extract application information from Java Applets.

5.1.8. **Flash:** Adobe Flash is a very popular client-side platform for delivering multimedia content to web browsers. Flash is often used to create interactive web applications with heavy animations and graphics, and was also adapted recently to support building RIAs (Rich Internet Applications). Web application scanners should support extraction of content and information from applications that make use of Flash.

5.1.9. **CSS:** Cascading style sheets (CSS) is a language widely used in web applications to describe the presentation of HTML documents. Web application scanners should support the extraction of URLs from both inline CSS and external CSS files.

5.2. **Character Encoding Support:** Web applications often include content encoded in forms other than ASCII. A web application scanner should be able to parse and understand content encoded in the following encoding types:

5.2.1. **ISO-8859-1:** Defined in RFC 2616 as the default encoding for HTTP content

5.2.2. **UTF-7:** 7-bit Unicode Transformation Format

5.2.3. **UTF-8:** 8-bit UCS/Unicode Transformation Format

5.2.4. **UTF-16:** 16-bit Unicode Transformation Format

5.3. **Parser Tolerance:** Oftentimes, web content may not conform to appropriate standards, either by mistake or due to various types of problems, ranging

from bad developer habits to communication problems. In such cases, it is crucial that content parsers will be able to cope with partial or non well-formed content and still be able to extract the relevant information from the application responses. It is important that web applications be at least as robust in their HTML parsing as web browsers are.

5.4. **Parser Customization:** Since web technologies and standards evolve rapidly, web applications may include links and other types of information which is encapsulated in various ways not covered in section 4.1 of this document. In order to support web applications that make use of such technologies and standards, it is recommended that web application scanners' parsers will allow user customization for link and content extraction. For example, the following non-standard URL contains parameter names and values, which are concatenated by the string '::'. Each parameter name and value pair use the string '^^' to denote the equal sign: `http://www.some.site/appEntry?p1^^v1::p2^^v2...pN^^vN`

In this scenario, the parser should be customizable to understand this non-standard URL format and properly parse the parameter names and values.

5.5. **Extraction of Dynamic Content (Client-Side Logic Execution):** Due to the dynamic nature of client-side scripting languages, it is often not enough to be able to statically parse scripting languages such as JavaScript, VBScript, or even Flash applications in order to detect links and other relevant application information. In such cases, web application scanners should be able to emulate user interaction with client-side logic in order to dynamically extract this information.

6. TESTING

Testing an application for vulnerabilities is the core functionality of a web application security scanner. This section lists the types of vulnerabilities that a web application scanner should be capable of detecting, as well as the testing-related configuration and customization options that a scanner should provide.

6.1. **Testing Configuration:** It is sometimes important to prevent the web application scanner from testing a given part of a web application. A scanner should provide the ability to reduce its visibility of the web application based on different criteria.

6.1.1. **Host names or IPs**: The tester should be able to specify specific host names or IP addresses to be ignored by the web application scanner during the test phase. Note that the scanner might be able to crawl and parse these host names and IP addresses for reasons such as logging in and data gathering, but should not test them.

6.1.2. **URL patterns:** Using patterns (e.g., regular expression, etc.), the tester should be able to specify which URLs should not be tested by the web application scanner.

6.1.3. **File extensions:** The user should have the ability to specify which extensions to exclude from testing.

6.1.4. **Parameters**: The user should have the ability to specify specific input parameters to exclude from testing.

6.1.5. **Cookies:** It is sometimes easier to specify which part of the tested web application the scanner should ignore based on cookie specific values. The tester should then be able to specify one or many cookie values that would prevent the scanner from testing the web page.

6.1.6. **HTTP headers:** The user should have the ability to configure the scanner to exclude certain pages from testing based on specific HTTP response headers.

6.2. **Testing Capabilities:** A web application scanner is looking for two major types of security problems - vulnerabilities and architectural weaknesses. The following list of problems to test was mostly extracted from the WASC Threat Classification v2.0.

6.2.1. **Authentication**

6.2.1.1. Brute Force

6.2.1.1.1. Lack of Account Lockout

6.2.1.1.2. Different Login Failure Message For Valid vs. Invalid User names

6.2.1.2. Insufficient Authentication

6.2.1.3. Weak Password Recovery Validation

6.2.1.4. Lack of SSL On Login Pages

6.2.1.5. Auto-complete Not Disabled On Password Parameters

6.2.2. **Authorization**

6.2.2.1. Credential/Session Prediction

6.2.2.1.1. Sequential Session Token

6.2.2.1.2. Non-Random Session Token

6.2.2.2. Insufficient Authorization

6.2.2.2.1. Ability To Forcefully Browse To "Logged-In" URL Without Logging In

6.2.2.2.2. Ability To Forcefully Browse To High-Privilege URL While Logged In Under Low-Privilege Account

6.2.2.2.3. HTTP Verb Tampering

6.2.2.3. Insufficient Session Expiration

6.2.2.4. Session Fixation

6.2.2.4.1. Failure To Generate New Session ID After Login

6.2.2.4.2. Permissive Session Management

6.2.2.5. Session Weaknesses

6.2.2.5.1. Session Token Passed In URL

6.2.2.5.2. Session Cookie Not Set With Secure Attribute

6.2.2.5.3. Session Cookie Not Set With HTTPOnly Attribute

6.2.2.5.4. Session Cookie Not Sufficiently Random

6.2.2.5.5. Site Does Not Force SSL Connection

6.2.2.5.6. Site Uses SSL But References Insecure Objects

6.2.2.5.7. Site Supports Weak SSL Ciphers

6.2.3. **Client-side Attacks**

6.2.3.1. Content Spoofing

6.2.3.2. Cross-Site Scripting

6.2.3.2.1. Reflected Cross-Site Scripting

6.2.3.2.2. Persistent Cross-Site Scripting

6.2.3.2.3. DOM-Based Cross-Site Scripting

6.2.3.3. Cross-Frame Scripting

6.2.3.4. HTML Injection

6.2.3.5. Cross-Site Request Forgery

6.2.3.6. Flash-Related Attacks

6.2.3.6.1. Cross-Site Flashing

6.2.3.6.2. Cross-Site Scripting Through Flash

6.2.3.6.3. Phishing/URL Redirection Through Flash

6.2.3.6.4. Open Cross-Domain Policy

6.2.4. **Command Execution**

6.2.4.1. Format String Attack

6.2.4.2. LDAP Injection

6.2.4.3. OS Command Injection

6.2.4.4. SQL Injection

6.2.4.4.1. Blind SQL Injection

6.2.4.5. SSI Injection

6.2.4.6. XPath Injection

6.2.4.7. HTTP Header Injection / Response Splitting

6.2.4.8. Remote File Includes

6.2.4.9. Local File Includes

6.2.4.10. Potential Malicious File Uploads

6.2.5. **Information Disclosure**

6.2.5.1. Directory Indexing

6.2.5.2. Information Leakage

6.2.5.2.1. Sensitive Information In Code Comments

6.2.5.2.2. Detailed Application Error Messages

6.2.5.2.3. Backup Files (home.old, home.bak, etc.)

6.2.5.2.4. Include File Source Code Disclosure

6.2.5.3. Path Traversal

6.2.5.4. Predictable Resource Location

6.2.5.5. Insecure HTTP Methods Enabled

6.2.5.6. WebDAV Enabled

6.2.5.7. Default Web Server Files

6.2.5.8. Testing and Diagnostics Pages (test.asp, phpinfo.htm, etc.)

6.2.5.9. Front Page Extensions Enabled

6.2.5.10. Internal IP Address Disclosure

6.3. **Testing Customization:** Even if they use the same basic technologies and communication protocols, web applications contain specific vulnerabilities that require very precise payloads to be enabled. A web application scanner should:

6.3.1. **Allow the user to modify existing tests**: It should be possible to either modify the attack steps (payload or algorithm) and/or the detection criteria.

6.3.2. **Allow the user to create new customized tests**: The web application scanner might have a specific API to allow the creation of new tests, or simply allow the user to add new tests based on templates, regular expressions, etc.

6.4. **Test Policy:** Most web application scanners have a large number of built-in tests, but in many situations, only a subset of these tests will be needed. A web application scanner should allow the user to **create personalized test policies** that specify which tests to include in a scan.

7. COMMAND AND CONTROL

The Command and Control capabilities of web application scanners can have a significant influence on usability and therefore are an important aspect to consider when conducting an evaluation. The types of Command and Control features most valued by an end user will vary based on the user's situation - some of the following features will be important to a large enterprise with many users and web applications to scan, but will not necessarily apply to a small company with a single user looking for an effective, low-cost scanning solution.

7.1. **Scan Control Capabilities:** The following control capabilities should be considered during a web application scanner evaluation:

7.1.1. **Ability to schedule scans:** Often the owners of web applications will only allow the applications to be scanned during specified time windows. Therefore, it is important that a web application scanner provide the user with the ability to schedule scan. This scheduling should include a **start time** as well as **maximum scan duration**, after which the scan will be paused if it has not yet completed.

7.1.2. **Ability to pause and resume scans**: A scanner should provide the operator with the ability to pause a scan and then resume the scan later from the point at which it was paused.

7.1.3. **Ability to view real-time status of running scans**: A scanner should provide the operator with a way of viewing the real-time status of running scans. This status could include information such as which tests currently run and scan completion percentage.

7.1.4. **Ability to define re-usable scan configuration templates**: Often a significant amount of time and effort is involved in optimally configuring a web application scanner for a particular application. A scanner should provide the user with the ability to save a scan's configuration so that it can be re-used for later scans.

7.1.5. **Ability to run multiple scans simultaneously**: Organizations that have many web applications to scan will find that the ability to run multiple scans at the same time is a desirable feature.

7.1.6. **Ability to support multiple users:** For organizations that plan to roll out a web application scanning solution to many users, the scanner's ability to support multiple users should be considered. For example, some scanning solutions will need to be installed and run on each user's workstation. Others provide a centralized web-based management interface that allows multiple users to configure and schedule scans and view scan results.

7.1.7. **Ability to support remote/distributed scanning**: The ability to support remote or distributed scanning is very important for decentralized environments where web applications are located behind firewalls or must be accessed over VPNs or slow links To effectively scan web applications in these environments, a web application scanner should provide the ability to deploy scanning "agents" inside an organization's network, which can be securely controlled by a remote operator.

7.2. **Control Interfaces Provided:** Web application scanners can provide a variety of interfaces for the user to control the scanner:

7.2.1. **Client Application with GUI:** Some scanners are designed to be installed on each user's workstation and provide a graphical user interface for configuring and controlling scans.

7.2.2. **Command Line Interface**: Some scanners provide a command line interface, with command options and/or configuration files used to input the scan settings.

7.2.3. **Web-Based Interface**: Some scanning solutions provide a web-based command and control interface. This is especially common in systems that support multiple users.

7.3. **Extensibility and Interoperability:** The ability to extend the functionality a web application scanner and integrate it with an organization's defect tracking systems can be important considerations for some users. The following are extensibility and interoperability features that can be provided by a web application scanning solution:

7.3.1. **Scan API**: Advanced users of web application scanners will sometimes find it useful to write their own code for controlling the scanner, executing custom tests, extracting data for custom reports, etc. For these types of users, the existence of an Application Programming Interface (API) should be considered. A scan API should be well-documented and supported. Ideally, example code which shows how to use the API should be available.

7.3.2. **Ability to integrate with common bug-tracking systems**: Organizations often use defect tracking systems to track the status of web application defects. A web application scanner should provide the ability to send vulnerability information to bug-tracking systems.

8. REPORTING

In order for scanning results to be viewed outside of the tool's interface, web application scanners should be able to generate reports of each scan. Because reports are often used by different groups within an organization, scanners should provide the ability to customize the format and information included in their reports.

8.1. **Types of Reports:** Although the specific reporting options may vary, scanners should provide the following types of reports:

8.1.1. **Executive Summary:** An executive summary provides a concise picture of the scan results. This report allows a reader to able to determine high-level results at a glance.

8.1.2. **Technical Detail Report:** Scanners must be able to provide all technical information required for readers to reproduce the identified issues. This should include:

8.1.2.1. The ability to include **full request and response data**

8.1.2.2. The ability to include a **list of all hosts and URLs** included in the scan

8.1.3. **Delta Report:** Scanners should provide the ability to compare results from two or more scans and show differences or trends over time.

8.1.4. **Compliance Report:** 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:

8.1.4.1. OWASP Top 10

8.1.4.2. WASC Threat Classification

8.1.4.3. SANS Top 20

8.1.4.4. Sarbanes-Oxley (SOX)

8.1.4.5. Payment Card Industry Data Security Standard (PCI DSS)

8.1.4.6. Health Insurance Portability and Accountability Act (HIPAA)

8.1.4.7. Gramm-Leach-Bliley Act (GLBA)

8.1.4.8. NIST 800-53

8.1.4.9. Federal Information Security Management Act (FISMA)

8.1.4.10. Personal Information Protection and Electronic Documents Act (PIPEDA)

8.1.4.11. Basel II

8.2. **Advisories For Each Unique Vulnerability Type:** Scanners should have the capability to produce advisories for each unique vulnerability type they identify. These advisories should include the following information:

8.2.1. **Vulnerability description**

8.2.2. **CVE or CWE ID**

8.2.3. **Severity level**

8.2.4. **CVSS version 2 Score**

8.2.5. **Remediation guidance**

8.2.6. **Remediation code example(s)**

8.3. **Report Customization:** Scanners should provide the ability to customize their reports, including the following:

8.3.1. **Ability to add custom notes to vulnerabilities**, which will be included in any generated reports.

8.3.2. **Ability to mark vulnerabilities as false positives** and remove them from the report. Note that it should be possible for the scanner to list and/or report on all of the "excluded" vulnerability findings. Ideally, the scanner should document who marked the vulnerability as a false positive, when it was marked, and the justification.

8.3.3. **Ability to adjust the risk level of vulnerabilities**, including:

8.3.3.1. Base, Temporal, and Environmental metrics that affect CVSS score

8.3.3.2. Severity level or other risk quantifiers

8.3.4. **Ability to identify and report vulnerabilities according to their content location**, which could be URL, Portlet Name, Page Title, or User Defined (e.g. a regular expression pattern in the response).

8.3.5. **Ability to include customizations** such as addition of a company logo or modification of report footer and header.

8.4. **Report Format:** Scanners should provide the capability to generate reports in both human and machine-readable formats, including:

8.4.1. **PDF**

8.4.2. **HTML**

8.4.3. **XML**

8.5. **Vendor Feedback:** Scanners should provide the ability to **automatically report false positives or other types of feedback to the scanner vendor** to help improve the quality of future versions of the product. This information should be encrypted in transit and handled securely by the vendor.

APPENDIX A: ADVICE FOR CONDUCTING A SCANNER EVALUATION

Web application scanners are complex pieces of software that can be a challenge to evaluate. However, conducting an evaluation is a valuable learning experience and will help you identify the tool or tools that best meet your needs. This appendix provides advice for conducting a successful evaluation and covers each phase of the evaluation process.

PREPARATION

Solid preparation is essential for conducting a successful evaluation. During this phase, the following steps should occur:

1. Determine which of the criteria detailed in this document are most important to you as a user. Some features will be essential "must haves", while others will be less important and some will be of no value for your particular situation. A future release of the WASSEC will include an evaluation spreadsheet that will allow you to adjust the weight of each criterion according to your specific needs.
2. There are many non-technical items that could be considered when evaluating a security scanner. The importance of these criteria will vary depending on your environment and the intent of your evaluation. For example, the importance of cost may be different between large and small organizations. If desired, these items may be added to your evaluation spreadsheet and weighted according to your needs. Some items to consider include:
    - Purchase cost
    - Ongoing support cost
    - Additional cost (hardware, etc.)
    - Ease of operation for user (e.g. developer, security analyst, QA analyst)
    - Quality of documentation
    - Support availability (phone, web, user community, etc.)
    - Availability of training
    - Product and/or testing capability update frequency
    - Licensing restrictions
3. Decide which scanners will be in-scope for the evaluation. Keep in mind that conducting a thorough evaluation can be time-consuming, therefore it is best to

use your "must have" criteria to limit your scope to a short list of candidates to evaluate.

4. Obtain the latest version of each scanner you are going to evaluate. For commercial scanners, vendors will normally provide a fully-functional copy of the software for a limited trial period if you are doing a private evaluation.
5. Decide which web applications will be scanned during the evaluation. Some tips:
   - Using a well-known vulnerable application such as WebGoat is unlikely to provide realistic results because the commercial scanners have most likely been optimized for these applications. It is best to choose real-world applications.
   - Select web applications that use a variety of technologies that are representative of your needs. For example, if your company owns a mix of ASP.Net and Java applications, make sure to include both of these technologies in the applications you choose to include in the evaluation.
   - Select web applications that employ a variety of design patterns that are representative of your needs. For example, if you include a blogging website as Application A, a shopping website as Application B, and a site that requires users to log in and provides multiple levels of access as Application C, you will learn much more from your evaluation than if you choose three applications of the same type.
6. Prepare the applications for scanning. It is best to use a non-Production environment, especially if the applications have not been scanned before. If you do choose to scan a Production environment during your evaluation, ensure that all databases used by the application have been backed up and Production Support personnel have been notified of your plans.
7. If you intend to release the results of your evaluation to the public, it is important to record the details of your application setup (for example, application version numbers, web server type and version, database type and version, and operating system) so that others can reproduce your results.

SCANNING

Use the products to scan each of your in-scope web applications. Since most scanners are very configurable, it will often be desirable to run multiple scans against each application using a variety of configurations. For example, you will likely see very different results from a "point and shoot" scan with minimal configuration versus a scan that is configured to log into the web application and manually trained on how to perform all significant transactions.

While configuring and running these scans, evaluate the products based on the criteria you rated as important during the Preparation phase. For example, if Ease of Use is an important criterion in your evaluation, give each product a rating for this criterion.

RESULTS ANALYSIS

For each scanner covered in the evaluation, add up the weighted scores for each section of the WASSEC as well as the scores for any additional criteria you have included, such as cost, ease of use, etc. If you are unsure how to rate certain items, contacting the scanner developer for additional information on specific aspects of the tool can help to fill in the gaps and ensure a thorough and fair evaluation. After completing the evaluation, you will be armed with the information needed to make an informed decision on which scanning solution best meets your needs.

APPENDIX B: LICENSE

CONTACT INFORMATION

If you have any questions, or would like to contribute to future enhancements of the WASSEC project, you can contact the project leader, Brian Shura at bshura73@gmail.com.