

Web Application Security Minimum Protection Criteria

Evaluation Criteria for Testing Application Security Products

Introduction

Goal of Evaluation Criteria	2
What Kind of Products Can Qualify	2
Protecting the Underlying Platform and Protocols	2
Protecting Customized Application Code	2
Protecting Applications in a Real-World Environment	3
Testing Results	3

Specific Security Requirements

Defenses That Do Not Require Session Awareness	4
- Requirement #1: Preventing Command Execution Attacks	5
- Requirement #2: Enforcing Strict Controls on Application Inputs	6
Defenses That Require Session Awareness	7
- Requirement #3: Preventing Cookie Tampering	8
- Requirement #4: Preventing Form Field Tampering	9
- Requirement #5: Preventing URL and Parameter Tampering	10

Goal of Evaluation Criteria

The need for application security has grown dramatically in recent months. Web applications, and the business data they link to, must be protected from the myriad of application-layer attacks that threaten to disable business-critical processes. Unfortunately, while the awareness of application security has increased, so has confusion among enterprise customers as to how to address this critical problem. In the absence of clear standardized test results to guide their application security buying decisions, customers must wade through a litany of vendor claims that are confusing at best, and potentially even misleading.

This multi-vendor group has established the following set of criteria which define the minimum protection capabilities customers should expect when evaluating any application security product.

What Kind of Products Can Qualify

This evaluation criteria is not intended to favor any particular type or form factor of security product. Qualifying products may be of any deployment form factor: network or host-based, software or hardware. Qualifying products may implement any attack protection architecture, including dropping attack traffic, blocking the effect of an attack and blocking the attacker. The only condition is that products must be capable of actually preventing application threats, not just detecting them.

Protecting the Underlying Platform and Protocols

It is assumed that any legitimate application security device will also provide baseline protection for the underlying web infrastructure and traffic protocols. This functionality should include:

- **Web Infrastructure Protection**: Product should be able to block known attacks against the underlying web application infrastructure, including packaged applications, the web and application server software (Windows Server, IIS, Apache, WebLogic, WebSphere, etc) and the server operating system (Linux, Solaris, Windows, etc).
- **Malformed HTTP Request Protection**: Product should be able to recognize deviations from the established protocol definitions for web traffic.

Because this basic level of protection is assumed to be common to all commercial application security products, it is not included as part of this evaluation criteria.

Protecting Customized Application Code

A modern application security solution must provide the ability to accurately block attempts to manipulate the business logic of the application itself. Customized application code is highly dynamic and varies from application-to-application. More importantly, the majority of transactional business applications incorporate links from application logic directly into back-end databases containing sensitive business information. If the application code is compromised, critical business data can be easily accessed by hackers, often leaving no trace that they were ever there in the first place. With these facts in mind, this evaluation criteria focuses primarily on the ability to protect customized application code in real-world environments.

Protecting Applications in a Real-World Environment

When evaluating application security products it is essential to test a variety of scenarios which model real-world enterprise usage. The ability to recognize a specific signature of a well-known PeopleSoft cross-site scripting vulnerability, for example, does not adequately predict a security product's ability to

protect the myriad of highly dynamic applications common in Fortune 1000 enterprise networks under real-world conditions. Examples of overall testing methodology to determine success in this area include the following:

- a) Security product must be able to protect customized web applications running on any standard back-end platform. When evaluating whether products meet this criteria, testers will be free to use any custom-built web application of their choice running on any commercial web server, application server or operating system platform.
- b) Security product must be able to automatically recognize and normalize any of the character encoding schemes used by hackers to subvert security detection, including URL encoding. When evaluating whether products meet this criteria, testers may use any of these encoding methods to disguise their attack attempts.
- c) Security product must be able to detect attacks encapsulated within SSL encrypted traffic. When evaluating whether products meet this criteria, testers will be free to run attacks against applications that require end-to-end SSL encryption between the browser client and the backend server. While the security product may intercept and decrypt traffic for inspection purposes, encrypted traffic must remain encrypted at all times when traveling outside the security device.
- d) Security product must be able to protect dynamic, customized applications without requiring security administrators to have extensive knowledge of each protected application in order to configure security policies. While legitimate products will vary in ease-of-use, testers may, at their discretion, chose to fail any product that requires an unrealistic level of pre-configuration, such as manually entering long lists of URLs, parameters, cookies and application-specific restrictions in order to enable protection.
- e) Security product must be able to distinguish between legitimate user requests and illegitimate attack attempts. When evaluating whether products meet this criteria, testers may browse the application at will with protection enabled to ensure that the chosen security settings do not disrupt normal application usage due to a preponderance of security false positives.

Testing Results

The products tested will either pass or fail based on the requirements listed below. Vendors whose products successfully pass will be publicly listed at

http://www.icsalabs.com/services/evaluatedproducts_new.shtml.

Vendors whose products fail will be provided a privately report detailing the requirements that were not satisfied.

Defenses That Do Not Require Session Awareness

The Problem of Improper Input Validation

Most web applications rely on inputs from HTTP requests to determine how to respond. This makes it easy for attackers to enter inputs the application is not expecting. The goal of these attacks is typically to gain control of servers, obtain unauthorized access to sensitive data linked to these applications, or break the trust relationship between the application and application users.

Stateless Defenses Against Improper Input

Requirements 1 and 2 deal with input validation defenses that do not require the security product to maintain specific knowledge of the individual user session state. Requirement #1 addresses the ability to prevent the injection of dangerous commands that should never appear in any legitimate user request. Requirement #2 addresses the ability to enforce strict limits on application inputs to ensure that all user requests match the expected format and data type.

Requirement #1 - Preventing Command Execution Attacks

Description

The security product must detect and block application inputs containing malicious executable commands. Detecting these attacks is relatively straightforward for most applications as there is typically no legitimate use case whereby an application client would legally submit executable content to a web application.

Real World Enterprise Exposure

Web applications which do not properly validate client inputs are at risk from a wide range of simple command execution attacks. Common attacks include SQL Injection, LDAP Injection and OS Commanding. If accepted by the application, these commands can assume system privileges which should only be granted trusted application code.

For example, a hacker could inject a structured query language (SQL) command into an application via an HTML-based form field. If undetected, the application could then construct a database query, using the hacker-supplied command rather than a legitimate user input (e.g. an account number). If successful, the hacker could gain unauthorized access to confidential database information using the web application as a conduit. A possible consequence could be a hacker who extracts all customer account numbers from a back-end database.

Protection Test Methodology

Examples of simple test scenarios that would validate a security product's ability to provide this method of protection include the following:

- The tester submits the following executable commands into a web application form field the contents of which are used by the application in the construction of a SQL query.

Test input Scenario:

Test input A: 99'OR x=x

(this represents an attempt to read out the contents of the affected database table)

Test input B: 99'UNION SELECT NULL, Name, ID, NULL, FROM SysObjects WHERE XType='U'

(this represents an attempt to retrieve the contents of the Name and ID fields from the SysObjects table of the database)

- The tester appends “;cat%20%2fetc%2fpasswd” to a URL in an attempt to obtain the “/etc/passwd” file from a vulnerable underlying Linux system, providing unauthorized access to login account information.

Success Measurement

The security product must be able to prevent attempts to inject executable commands into application inputs.

Requirement #2 – Enforcing Strict Controls on Application Inputs

Description

The security product must be able to enforce controls on application inputs. Controlling the format and type of data submitted to a web application is an effective means of protecting the web application from attack. URLs and form fields serve as common inputs to web applications and constitute popular attack vectors.

Real World Enterprise Exposure

The failure of a security product to enforce application input controls may permit application inputs to be used as attack vectors. Common threats include cross-site scripting exploits, buffer overflows on input fields, and others. A successful cross-site scripting attack, for example, could result in an attacker stealing the on-line identity of a legitimate application user and using these misappropriated privileges to purchase products or transfer account funds.

Protection Test Methodology

To validate this protection method, the tester would define and enforce a security policy that limits the acceptable data types for client-supplied form field values and URL parameters. Examples of simple test scenarios that would validate a security product's ability to provide this method of protection include the following:

- The tester sets a security policy that limits client-supplied inputs for a specific application form field to nine numeric digits. The tester then submits a value within this form field that violates this restriction.
- The tester sets a security policy that limits the query string in a client URL to nine numeric digits. The tester then submits a query string that violates this restriction.

Success Measurement

The security product must be able to prevent attempts to insert illegal data types into application inputs.

Defenses That Require Session Awareness

The Problem of State in Web Applications

In the world of applications, session state refers to the process of tracking the progress of a user's interaction with the application itself. In the world of web applications, this translates to managing the information that is used to identify a particular client browser (and associated user) to the application, and to manage that individual's usage of the application. A simple example of session-dependent information is whether or not an eCommerce customer has paid for the contents of his or her shopping cart.

Because the HTTP protocol is stateless, the task of managing the state of communications between the web server and the browser client is left up to the application itself. As a result, there is no accepted standard for how a given web application defines and tracks session-dependent information. This is a choice left up to the developer. In most cases, management of application state in a web application is handled either through, cookies, form fields or URL parameters.

Typically, the dynamic nature of session-dependent information makes it impossible to write signatures to detect such attacks. Consequently, a genuine application security product must provide session awareness to detect and block these attacks.

The Implications of Attacks on Session-Dependent Information

When it comes to web applications, controlling session-dependent information often means controlling the web application itself. The ability to modify this state information allows an attacker threaten the application logic itself. The implications would include that ability to modify a business rule (such as the price of an item in an online store, to cite a simple example), disrupt user sessions in progress, display user credentials, and even assume administrative privileges in some cases.

Requirements 3-5 address the baseline criteria for protecting applications from attempts to manipulate session-dependent elements.

Requirement #3 – Preventing Cookie Tampering

Description

A cookie is a piece of information that a web server stores in user's browser. Cookies are typically used for a variety of purposes, including setting/managing session state, tracking user demographic and usage information, setting/tracking authorization information and personalizing content. Cookie Tampering refers to the illegitimate modification of this user and state information.

Cookies are passed to the user's browser via the HTTP header. IETF RFC 2109 describes the rules for managing cookie information. Typically cookies can be accessed by directly opening the cookie file on the user's hard drive. Attackers also often access cookie information via a form of cross-site scripting attack (defenses for which are addressed primarily in items 1 and 2).

Real World Enterprise Exposure

The ability to modify a cookie can result in a hacker being able to control session-dependent information. This could allow an attacker to display user credentials, disrupt user sessions in process, assume administrative privileges, or change the business logic of an application. A common consequence of cookie tampering is the theft of private user information from an application that involves employee, partner or customer transactions.

Protection Test Methodology

Following is an example of a simple test scenario that would validate a security product's ability to provide this method of protection:

- The tester uses a proxy to intercept and modify parameters in a session cookie "SessCookie=1892345%2Dcart%3Dunpaid" to "SessCookie=1892345%2Dcart%3Dpaid" to bypass the payment process and obtain free goods and services. Note that this is an overly simple cookie and most have additional parameters and encoded data.

Success Measurement

The security product must be able to prevent attempts to modify application state information stored in cookies.

Requirement #4 – Preventing Form Field Tampering

Description

Web applications use form fields to accept user input for processing, storage, and display within the application and back-end systems. In addition to accepting user input, form fields are also often used for application state management. As a result, they are a popular target of attackers who exploit such applications through the use of client-side proxies.

Form fields are custom tailored to each application, delivered to the client through HTML, and can be displayed by using “View Source” within client browsers. Both the contents of the data accepted by the form field as well as the structure of the form field itself present avenues of attack. When a form is submitted, the current value of each input element within the form is sent to the application as name/value pairs (e.g. “Username/jdoe”). The type of input is controlled by the TYPE parameter, which defaults to text, but can also be set to accept files, passwords, radio buttons, etc. as input. The state of the application is delivered to the user in the configuration of the fields delivered, and the behavior of the application is influenced by both the name/value pairs and associated parameters that are returned to the application.

Real World Enterprise Exposure

By modifying both the construct and the data for a form field, an attacker controls the type and size of data accepted by the application. If the application does not properly validate user input, untrusted input that causes attacks could be delivered through the application and allow unintended modification of session state or application resources.

The ability to modify form fields can result in an attacker being able to control session-dependent information. This could allow the attacker to display user credentials, disrupt user sessions in process, assume administrative privileges, or change the business logic of an application. A common consequence of form field tampering is an attacker assuming the role of an application administrator, then using this authority to further compromise site security.

Protection Test Methodology

Examples of simple test scenarios that would validate a security product’s ability to provide this method of protection include, but are not limited to, the following:

- The tester modifies a hidden form field using a proxy changing “type=user” to “type=admin” to assume administrative privileges for the application and updates the application
- The tester takes a form field parameter and modifies “<INPUT TYPE=text NAME=userid SIZE=30 MAXLENGTH=30>” to “<INPUT TYPE=file NAME=userid SIZE=40000 MAXLENGTH=40000>” in an attempt to upload a malicious file or cause a buffer overflow.

Success Measurement

The product must prevent attempts at modifying application state information stored in form fields.

Requirement #5 – Preventing URL and Parameter Tampering

Description

One of the primary methods of interacting with a web application is through the URL bar in the client's browser. In addition to requesting the functionality of a web application, the manipulation of data in the URL and associated parameters allow for attack. Because these attacks are so simple to execute and require no additional tools, they are often knowingly and unknowingly attempted against web applications. Because there are so many permutations of attack in today's complex and dynamic URL structures, relying on attack signatures to protect web applications from parameter tampering is infeasible.

Real World Enterprise Exposure

The modification of data in URL parameters manipulates both static and dynamic application structures and affects the state and behavior of the web application. The resulting threat is that an attacker could display user credentials, disrupt user sessions in process, assume administrative privileges, or change the business logic of an application.

A common consequence of parameter tampering is the modification of business logic, such as the alteration of pricing or payment information.

Protection Test Methodology

To validate a security product's ability to protect against this avenue of attack, consider the following sample URL. Note that the URL is highly dynamic in nature and that the state of several events within the application are managed through both URL and parameter structure. An example of a simple test scenario that would validate a security product's ability to provide this method of protection includes, but is not limited to, the following:

Sample URL: <http://www.somesite.com/guestlogin/new/login.asp?status=unauth&uid=0>
Parameter: [?status=unauth and uid=0](#)

- The tester tries entering different values for the parameters, such as "status=auth", "uid=0", "uid=99999", "uid=something" in an attempt to generate helpful error messages or gain unauthorized access to the application. Additional parameters or sets of parameters may also be specified, such as "&yes=no".

Success Measurement

The product must prevent attempts at modifying application state information contained in URL parameters.