Web Application Firewall Documentation for Developers and Testers of Applications on the Web DMZ

US COURTS NATIONAL WEB DMZ INFRASTUCTURE

Prepared by:

Office of Information Technology

Information Technology Security Office

(OIT-ITSO)

Effective Date:

Revisions:

Version 1.0	September 22, 2011	Initial Draft.
Version 1.1	October 20, 2011	Revisions and addition of diagrams
Version 1.2	October 25, 2011	Revisions and addition of diagrams
Version 1.3	December 5, 2011	Revisions
Version 1.4	March 26, 2012	Revisions and updates

Table of Contents

1.0 Audience 1

2.0 Scope 1

3.0 Introduction 1

4.0 Role of Web Application Firewalls 2

4.1 Threats that WAFs May Counter 2

4.2 Web Application Firewall Operation Model 3

5.0 Function of the Web Application Firewall in the National Web DMZ Infrastructure 4

5.1 Deployment Architecture 5

6.0 Imperva SecureSphere Web Application Firewall 7

6.3 SecureSphere Application Behavioral Modeling 9

6.4 SecureSphere Web Application Profiles 9

6.5 SecureSphere Security Policy Types 15

6.6 Web Error Page Responses 18

6.7 URL Rewrites and URL Redirection 19

6.8 Name-based Virtual Hosting 19

6.9 Session Tracking 20

6.10 Blocking of Attacks 20

7.0 Web Profile Creation and Tuning Procedures 21

8.0 References 24

Appendix A: Predefined Security Policies 25

A1: Stream Signature 25

A2: HTTP Protocol Validation 25

A3: HTTP Protocol Signatures 26

A4: Web Service Custom 27

A5: Web Service Correlated Validation 28

A6: Web Profile Policy 28

A7: Security Policy Rule Parameters 29

Appendix B: Web Server Configuration 30

List of Figures

Figure 1: Example of An Inline Kernel Reverse Proxy Configuration. 5
Figure 3: SecureSphere OSI Protection Model. 7
Figure 4: SecureSphere Security Engine Operations. 8
Figure 6: SecureSphere WAF application Web Profile Format. 11
Figure 7: Web Profile Creation and Tuning Process Flow. 21

List of Tables

Table 1: WASC Threat Classification of Attacks 2
Table 2: WAF Usage Comparison. 4
Table 3: Policy Type Descriptions. 16
Table 4: Web Profile Policy Rules. 18
Table 5: Stream Signature 25
Table 6: HTTP Protocol Validation 26
Table 7: HTTP Protocol Signatures 26
Table 8: Web Service Correlated Validation 28
Table 9: Web Profile Policy 28
Table 10: Security Policy Rule Parameters 29

1.0 Audience

This document provides information to application owners, web application testers and web application developers for applications deployed on the National Web DMZ Infrastructure.

2.0 Scope

This document provides application owners, developers and testers with useful information to facilitate a better understating of the operations and functionality of the Web Application Firewall (WAF) security control component of the National Web DMZ Infrastructure; and the Information Technology Security Office (ITSO) procedures associated with the deployment of web applications to enhance the secure deployment of Judiciary national applications.

3.0 Introduction

Traditional firewalls (network firewall), while serving an important function, do not address certain issues as they relate to data security. Network firewalls provide comprehensive network security, and in some cases basic application awareness, however, they lack the ability to understand or protect the application or its data.

A Web Application Firewall (WAF) is *"an intermediary device, sitting between a web-client and a web server, analyzing OSI Layer-7 messages for violations in the programmed security policy. A web application firewall is used as a security device protecting the web server from attack*" (http://www.webappsec.org/glossary.html#WebApplicationFirewall)

WAFs provide protection above and beyond what network firewalls and intrusion detections systems are capable of.

4.0 Role of Web Application Firewalls

Web Application Firewalls (WAFs) are designed to protect against attacks caused by un-remediated web application coding vulnerabilities. WAFs are but one of many technical countermeasures that might be used to mitigate residual application risks associated with these applications. WAFs are typically used when vulnerability management processes do not provide adequate assurance that code vulnerabilities have been identified or corrected. WAFs do not eliminate the need for a secure software development process. WAFs also serve to protect against application level threats which are undetectable by traditional means of protection such as network firewalls and intrusion prevention systems. The following section outlines some of these threats.

4.1 Threats that WAFs May Counter

The table below provides the accepted industry-wide Web Application Security Consortium (WASC) web application security threat taxonomy. Hyperlinks contained in the table below provide technical details and examples of the common attack surface. WAFs can provide some protection against these threats by enforcing a particular type of security model that denies all but the normal and expected URL sequences.

Abuse of Functionality	Brute Force	Buffer Overflow
Content Spoofing Cross-Site Request Forgery	Credential/Session Prediction Denial of Service	Cross-Site Scripting Fingerprinting
Format String	HTTP Response Smuggling	HTTP Response Splitting
HTTP Request Smuggling	HTTP Request Splitting	Integer Overflows
LDAP Injection OS Commanding	Mail Command Injection Path Traversal	Null Byte Injection Predictable Resource Location
Remote File Inclusion (RFI)	Routing Detour	Session Fixation
SOAP Array Abuse	SSI Injection	SQL Injection
URL Redirector Abuse XML External Entities	XPath Injection XML Entity Expansion	XML Attribute Blowup XML Injection
XQuery Injection		

Table 1: WASC Threat Classification of Attacks

4.2 Web Application Firewall Operation Model

WAFs operate based on either, or a combination, of two operation models namely Positive Security Model and Negative Security Model.

1. Positive Security Model (PSM)

   Enforcing a security model that denies all but the normal and expected Universal Resource Locator (URL) sequences is known as a “positive" security model or a "whitelist" security model. A positive security model denies all transactions by default, but uses rules to allow only those transactions that are known to be valid or safe. While secure, a positive security model can be more difficult to maintain if the web application changes frequently. In addition, composition of rules that only allow a white list of acceptable URL sequences can be difficult without a detailed understanding of each application.

   WAF products attempt to reduce the burden of rule development by providing automatic learning modes where “normal” patterns are statistically learned resulting in the “automatic” production of rules or what is termed the "web application profile". Such learning modes are good but a deep understanding of what and how the WAF determines as “normal” is necessary. Application developers and operations personnel should review the produced rules in detail to determine the appropriateness of the rules for the application and its suspected vulnerabilities.

2. Negative Security Model (NSM)

   In contrast to the positive security model is the "negative" security model or "blacklist" security model where all known malicious URL requests and payload patterns matching a defined policy set is denied. The WAF knows what URL requests constitutes an attack and permits all other requests to go through to the protected web application. In order to determine malicious requests, signatures which form rule sets are required. Signatures match against known malicious patterns. As data is passed through negative security policy, it is evaluated against individual signatures. If there is a match, the data is rejected; otherwise it is allowed to pass through. The negative security policy does not take into account the functionality of the protected application and denies requests to the application once that request violates any enabled signature. Other caveats of this approach are that this model is only as good as the signatures, NSM is highly dependent on signature updates and is not very accurate. On the other hand, the advantages of NSM are the simple and straightforward deployment, out-of-the-box protection and non-requirement for customization.

5.0 Function of the Web Application Firewall in the National Web DMZ Infrastructure

As a security control on the National Web DMZ Infrastructure, the Imperva WAF serves the following purposes:

1. Provides defense-in-depth: An approach employed as part of the National Web DMZ Infrastructure design is to have layers of protection to provide a comprehensive defense system against potential attacks. The WAF provides protection at the application layer against attacks targeted against vulnerabilities in the application logic and code.
2. Acts as a compensating control for the lack of visibility into HTTP data encrypted over SSL/TLS: Most applications on the Web DMZ, due to user authentication and content privacy requirements, require encrypted logins and sessions. At present, the intrusion detection and prevention mechanisms deployed on the infrastructure do not decrypt traffic in order to inspect packets. The WAF provides SSL/TLS termination and decryption which facilitates content inspection performed by the WAF.
3. Acts as a secondary layer of protection for web applications: The primary layer of protection for web applications is secure code practices. The WAF is not a substitute for developing secure code for web applications.
4. *Provide detection of application abnormalities: broken links, web pages.*

The following is a summary of what the WAF provides and does not provide from a usage standpoint:

What the WAF Does	**What the WAF Does Not Do**
1. Model legitimate Web application usage. 2. Alert or block access requests that: a. Deviate from normal application and data usage b. Attempt to exploit known and unknown vulnerabilities c. Originate from malicious sources d. Violate corporate policies e. Are part of a sophisticated multi-stage attack *3. Update Web defenses with research-driven intelligence on current threats.* 4. Virtually patch application vulnerabilities through integration with Web application vulnerability scanners, reducing the window of exposure and impact of ad-hoc application fixes.	1. Provide anti-virus inspection of uploaded files or content. 2. Provide web application load balancing. 3. Perform web server health status monitoring. 4. Make coffee ☺.

Table 2: WAF Usage Comparison.

5.1 Deployment Architecture

The WAF is deployed in an inline kernel reverse proxy as shown in the diagram below.

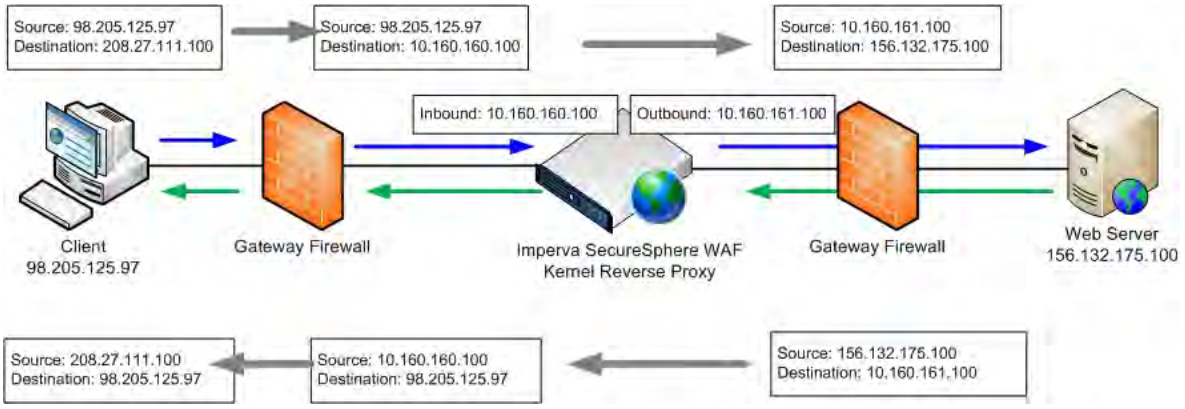

Figure 1: Example of An Inline Kernel Reverse Proxy Configuration.

Traffic flows as described.

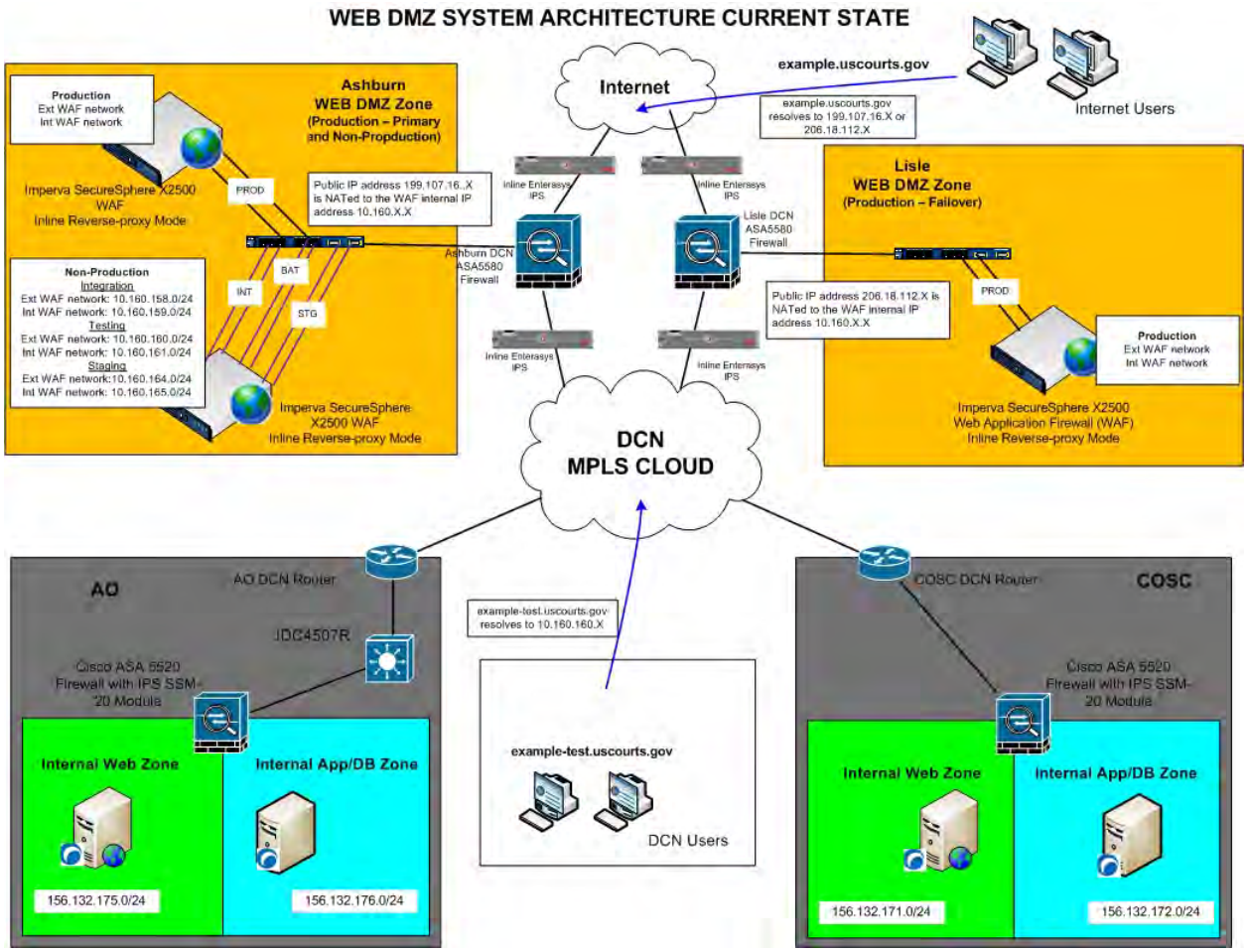

Figure 2: National Web DMZ System Architecture.

Implications

As a result of this design, by default, the source IP address seen in host web server logs is always that of the WAF and not that of the initiating client and as such configuration is required on web servers to log the IP address of actual clients. Please see Appendix B for details.

6.0 Imperva SecureSphere Web Application Firewall

This section will discuss in detail the Imperva SecureSphere WAF and its operation as it relates to modeling application usage and providing protection.

6.1 SecureSphere OSI Protection

The SecureSphere system's protection operates in layers that approximate to the OSI 7-layer model. The firewall corresponds to OSI layers 2 thru 4, Protocol Validation and Application Layer Signatures approximate to OSI layer 7, etc. as shown in the figure below. However, several of SecureSphere's advanced protection processes (such as Profile Evaluation, Web/DB Correlation, and Correlated Attack Detection) operate on the behavior of the application and thus represent an effective layer 8 - not defined in the OSI model.

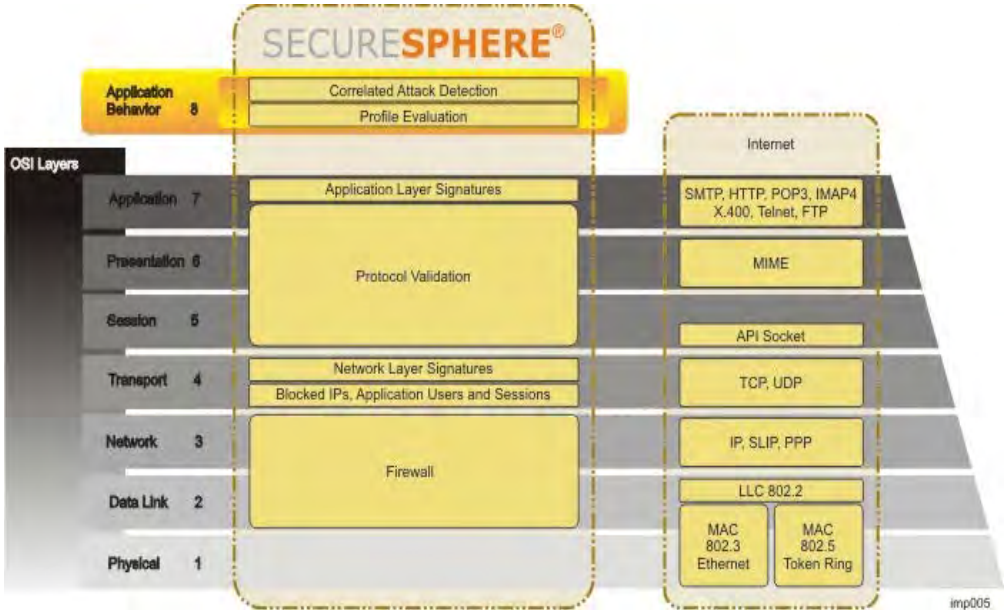

Figure 3: SecureSphere OSI Protection Model.

6.2 SecureSphere Security Engine

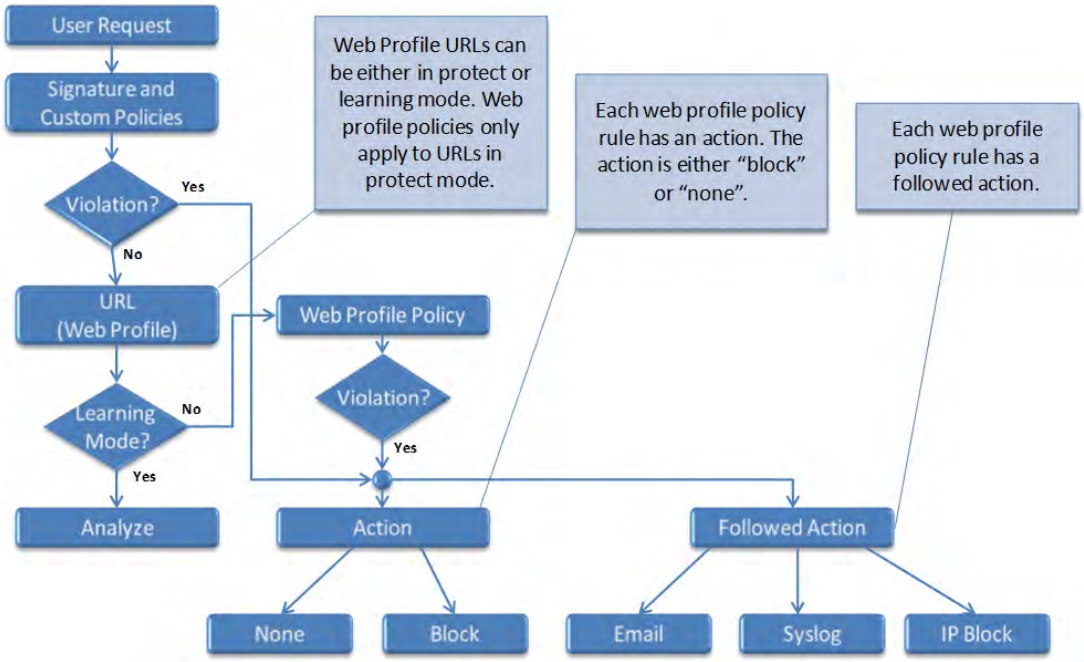

Figure 4: SecureSphere Security Engine Operations.

During the time the profile is being created, the application is protected from generic and known attacks by the SecureSphere ADC content (and user-defined policies) and SecureSphere does not learn behavior which violates existing policies. For example, SecureSphere does not learn a client request which includes abnormal HTTP behavior or which SecureSphere identifies as containing a worm.

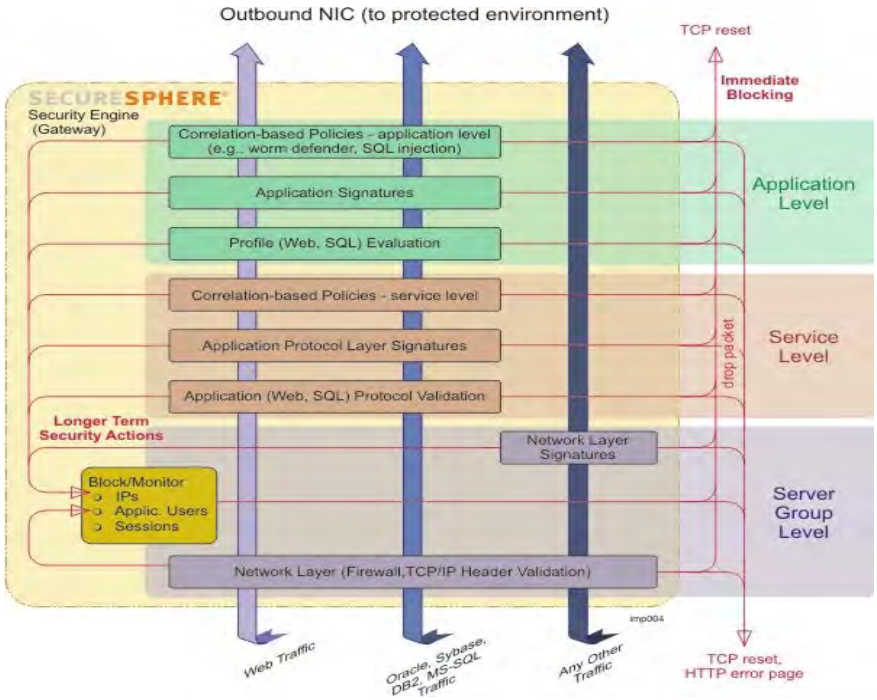

Figure 5: SecureSphere Security Engine Protection

6.3 SecureSphere Application Behavioral Modeling

The Imperva SecureSphere system provides protection against such threats that require a higher level of protection, at what it calls the application behavioral layer. SecureSphere's Dynamic Profiling technology automatically examines web application traffic to create a comprehensive profile of their structure and behavior. Dynamic profiling overcomes the biggest drawback of other application security and compliance solutions – manual creation and maintenance of audit controls and security policies. Valid application changes are automatically recognized and incorporated into the profile over time, ensuring that SecureSphere detects potentially malicious exceptional activity. The automatically generated profile can be manually tuned and controlled at any time.

SecureSphere begins creating an application's profile the first time it sees traffic for the application. Over time, SecureSphere sees more traffic and refines the profile accordingly. Eventually SecureSphere builds an accurate model of the application and begins to enforce the profile. All of this happens automatically, though at every stage the profile can be tuned manually to improve its accuracy.

6.4 SecureSphere Web Application Profiles

A Web profile is a model of a Web application, specifying how the application's URLs and cookies are normally accessed. The profile is built up over time automatically by SecureSphere as it learns the application, and then refined by the administrator. Once the learning period is over, SecureSphere starts protecting the application by identifying profile deviations, that is, HTTP requests which do not conform to the profile, and taking the action specified in the policy associated with the profile. Profiles represent the "whitelist" security model, what is allowed, in contrast to SecureSphere Application Defense Center (ADC) signatures, for example, which represent the "blacklist" model, what is not allowed.

A web application profile is built around URLs and cookies, and SecureSphere begins creating the application's profile the first time it sees traffic for the application. A SecureSphere Web Application profile consists of the following components:

1. URLs List: SecureSphere learns the application URLs based on the traffic to the application, collecting the following information:
    - URLs, their HTTP methods, parameters and their attributes
    - cookies
    - host names
    - login URLs
    - correlation information
2. URL Patterns
3. Cookies List
4. Learned Hosts
5. Web Application User Tracking

Each application has an associated "web profile" policy, which defines what SecureSphere does when a profile deviation is detected. In addition to the profile policy, an application can have any number of other policies associated with it.

6.4.1 Web Profile URL Modes

During the time the profile is being created, the application is protected from generic and known attacks by the SecureSphere ADC content (and user-defined policies) and SecureSphere does not learn behavior which violates existing policies. For example, SecureSphere does not learn a client request which includes abnormal HTTP behavior or which SecureSphere identifies as containing a worm. Each URL can be in one of the following modes:

- Learning mode: SecureSphere is not enforcing the profile policy for this URL, because it is still learning how the URL is accessed.
- Protect mode: SecureSphere is enforcing the profile policy.

6.4.2 Protect Mode

For URLs in Protect mode, SecureSphere inspects client requests and server responses, detecting violations and enforcing the profile policy. The profile can be tuned by modifying the parameters of a URL in Protect mode. For example, WAF administrators can remove false positives or make updates to attribute settings on a URL's parameters. All of a URL's parameters, except for correlation data, can be manually modified.

Developers and Testers: For production application instances, all web profile URLs will always be in protect mode so as to enforce the web profile policy and detect violations. For non-production web application instances , the web application web profile URLs will be in protect mode only during the web profile tuning phase. This facilitates the detection of possible false-positives and the tuning of the web profile.

6.4.3 Learning Mode

The SecureSphere WAF builds web profiles for each defined web application. At first, each web profile URL is in learning mode, during which time SecureSphere builds a list of its parameters based on client requests and application/server responses. While in Learning mode, SecureSphere does not enforce the profile policy for the URL. In addition to URLs which SecureSphere process learns automatically, URLs can manually be added to the profile. To avoid learning redundant URLs, SecureSphere can be configured to group URLs based on patterns. The Web Profile policy creates a list of URLs (with and without parameters). For URLs without parameters it monitors the HTTP Methods and for URLs with parameters it monitors the HTTP methods along with the parameter arguments and values. The web profiler stores the names of the arguments with their respective types, minimum and maximum lengths, and whether they are required and/or read only.

In summary, The URL profiles include the following information:

- A list of URLs used by this server group
- HTTP methods used by each URL
- A list of parameters included in each URL
- A set of attributes for each parameter:
    - Value type
    - Minimum length
    - Maximum length
    - Whether or not it is required or optional
    - Whether or not it is a read-only parameter
    - Whether or not it is a parameter prefix

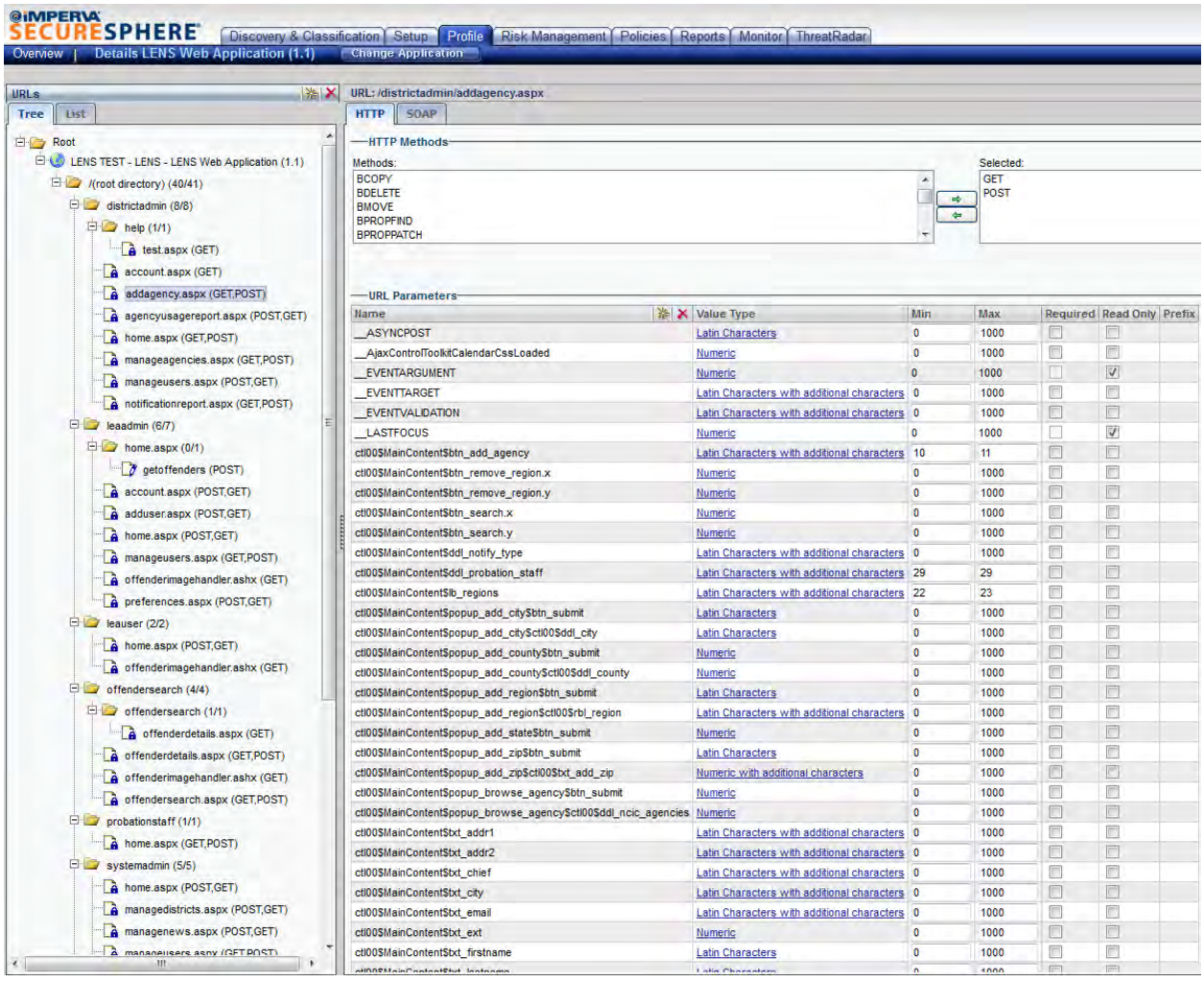

Figure 6: SecureSphere WAF application Web Profile Format.

When web profiling is enabled any detected URL is in learning mode until it is determined that enough has been learned about the URL and moves the URL is moved to "protect" mode.

SecureSphere determines when to move a URL to protect mode based on number of occurrences and/or elapsed time. A URL can be switched to protect in two ways:

1. SecureSphere learned it well enough and decided that it converged.
2. Enough time has passed for SecureSphere to decide to switch it to protect anyway. This decision is time triggered, i.e. from time to time we check the existing URL's and move the old ones to protect.

The default settings for moving URLs to protect (there is a separation between URLs that have parameters in them and URLs with no parameters) are as follows:

URLs with no parameters will behave as follows:

A. When 3 hours passed since SecureSphere first saw the URL, the number of occurrences will be checked. If it has at least 50 occurrences it will move to protect.
B. When 72 hours passed since SecureSphere first saw the URL, the number of occurrences will be checked. If it has at least 7 occurrences it will move to protect.
C. When 96 hours passed since SecureSphere first saw the URL, the number of occurrences will be checked. If it has less than 10 occurrences it will move to protect.
D. When 240 hours passed since SecureSphere first saw the URL the URL will be moved to protect no matter how many occurrences it had.

URLs with parameters will behave as follows:

E. When 96 hours passed since SecureSphere first saw the URL, the number of occurrences will be checked. If it has less than 10 occurrences it will move to protect.
F. When 240 hours passed since SecureSphere first saw the URL, the URL will be moved to protect no matter how many occurrences it had.

** If a URL with parameters has more than 100,000 occurrences it will move to protect no matter when SecureSphere first saw it.

The default setting for duration required to switch all URLs to protect mode can be changed from the management web interface. A URL can be manually moved to protect mode and vice-versa. A URL or web directory can also be “locked”. This prevents any further learning or manual modification of learned methods or parameters for that URL, and at the root or sub-directory level, any other new URLs from being learnt and added to the web profile.

When a URL is moved to protect mode, it begins triggering events for activity seen to a protect-enabled URL that does not match the built web profile. SecureSphere web application behavior profiling implements incremental learning. As such even when URLs have been moved to protect mode, the profile can be updated when a false-positive event occurs by manually clicking on the “Add to profile” button.

Testers: The quality of completeness of the learned web profile is a function of the comprehensiveness of the data set observed by SecureSphere during learning mode i.e. it should be ensured that:

- *All possible URLs and their parameters are hit.*
- *All possible language character sets are hit.*
- *All possible character sets for individual parameters are hit.*
- *All possible minimum and maximum parameter values are hit.*

Application functional testers should endeavor to perform tests that are comprehensive. A comprehensive functional test will include requests to all URLs and all URL parameters within the application, use the maximum possible parameter length and all possible allowed character values as supported by the application. Automated functional testing using scripts or testing software is encouraged as this provides the advantage of manageability as the tested application goes through version update iterations.

6.4.4 URL Patterns

URL patterns enable you to reduce the number of URLs in the profile when a large number of URLs can be handled in the same way.

URL patterns describe a group of URLs which SecureSphere does not treat as distinct, different URLs but rather as repeated instances of the same URL. For example, if a site creates a folder for each of its users and all these folders contain files with the same name, the administrator can define a prefix or a suffix. A new URL which matches the pattern is not considered a new URL but another occurrence of the same URL pattern. All URLs which match the pattern are protected in the same way, for example, they share the same HTTP methods.

URL patterns are not learned automatically but are defined manually, and they are always in Protect mode.

Prefix patterns are suitable for folders which contain a large number of files of the same type. For example, if the folder /public/calculators/ contains many files with the same parameters and the same behavior, you can define /public/calculators/ as a URL prefix and any file in the directory which matches the pattern is handled by SecureSphere in the same way.

Suffix patterns are suitable for:

- File types – (for example aspx)
- Specific files – (for example order.asp)
- A file name and part of its path – (for example /public/print.asp matches both /scripts/public/print.asp and /home/public/print.asp)

6.4.5 Dynamic Parameters

Some applications generate dynamic parameters. This has the effect of growing the profile to a very large size. A parameter prefix that matches all dynamic parameters can be added to the profile to consolidate the number of parameters and reduce the profile size. It should be noted that SecureSphere automatically generates parameter prefixes for parameters which start with letters and end with numbers.

Developers: Application developers should provide details on dynamic parameters.

6.4.6 Web Application User Tracking

This feature enables the identification and classification of successful and failed logins for a web application. This enables user logon information association with generated events. Web Application User Tracking (WAUT) learns the Web application's login URLs as part of the process of building the Web application profile. When a Web application user successfully authenticates to the application, SecureSphere associates the Web application user name with an HTTP session and IP address, and tracks the user throughout the duration of the session. Web Application User Tracking supports both form-based and certificate-based user authentication. The former, at present, is more predominant for Judiciary national applications and will be discussed in further detail.

Note: SecureSphere does not authenticate application users, but only tracks successful logins.

6.4.6.1 Form-Based User Authentication

Action URLs are URLs of the forms in which users login to the web application. SecureSphere learns the action URLs and the rules which determine whether the login was successful, based on what the application does after the attempted login.

Login Attempts Decision Rules specify the behavior of Action URL, that is, what SecureSphere has observed happening when the user name field is entered.

The learned Login Result is used as follows:

- If Login Result is Successful, the username is tracked throughout the HTTP session.
- If Login Result is Failed, the username is not tracked through the remainder of the HTTP session.

A policy can be defined to alert on login failures, or on brute force attacks that are characterized by a large number of login failures in a short period of time. (Such a rule has been created for Brute Force Login Attempts and is implemented in production and non-production environments).

- If Login Result is Can't Tell, the username is not tracked through the remainder of the HTTP session.

The login attempt result is determined by either the response code value or the redirect response value. For example, an application with a login URL of /login.asp may return a response code of 302

and redirect to /home.asp if the login is successful and return a response code 200 if unsuccessful. In this case the rule for a successful login can be based on the response code value of 302 or the redirect value of /home.asp, while the rule for a failed login can be based on the response code value of 200.

Although this behavior is learned, it can happen that not enough information is available for SecureSphere to correctly model the behavior. For example, if the web application returns a response code of 200 for both successful and unsuccessful logins, SecureSphere would be unable to distinguish between the two results. As another example, if the field names of the user name or other identifying fields were non-standard (typically in the case of a custom application), then SecureSphere would be unable to know which of the form's fields represents the user name and so on.

Developers and Testers: In this case action URLs for the web application need to be created manually along with login attempts decision rules. For new applications, developers should provide details of login URLs along with login behavior for both successful and unsuccessful login attempts to facilitate this process. Application testers should endeavor to perform login authentication tests for failed and successful login attempts in coordination with ITSO to confirm the functionality and determine the accuracy of created login rules.

6.5 SecureSphere Security Policy Types

A SecureSphere security policy is a set of definitions that characterize security violations and actions that SecureSphere must take in response to them. A security policy is a set of rules that are grouped together based on security feature that they represent. Most of the security policies consist of rules that are building blocks of that policy. Each rule includes 2 main parts, a description of the violation against which the rule protects and a definition of the reaction to that violation.

A summary of security policy types is provided in the tables below:

Policy Type	**Description**
Server Group Level Network Policy Types	
Firewall Policy	Defines the services is used to access the protected servers.
Network Protocol Validation	Validates a proper use of the TCP/IP protocol according to the RFC standard, preventing attacks that are based on TCP/IP protocol vulnerabilities.
Stream Signature	Identifies well known attacks for generic services.
Service Level Web Security Policy Types	
HTTP Protocol Validation	Validates a proper use of the HTTP protocol according to the RFC standard.
Web Service Correlated Validation	Protects against multi-stage web application attacks.
Cookie Signing Validation	When SecureSphere operates in the Kernel Reverse Proxy mode, this policy enables cookie signing and verification.
HTTP Protocol Signatures	Protects against known protocol attacks using signatures.

Web Service Custom	Enables creating user-defined web service level policies based on various combinations of the match criteria.
Application Level Web Security Policy Types	
Web Profile	Alerts on and prevents the application user behavior that deviates from the Application Profile, as defined in SecureSphere.
Web Worm	Provides protection for zero day web worms based on Imperva unique technology.
Web Application Signatures	Protects against known application attacks using signatures.
Web Application Custom	Enables creating of user-defined web application level policies based on various combinations of the match criteria.

Table 3: Policy Type Descriptions.

Security policies have four main configuration option settings: Enabled, Severity, Action, and Followed Action.

Please see Appendix A for details on these security policies and their configured settings.

6.5.1 SecureSphere Web Profile Policy Rules

A Web Profile Policy alerts on and prevents the application user behavior that deviates from the Application Profile, as defined in SecureSphere. A Profile contains application profile definitions learned by SecureSphere. By comparing HTTP requests/responses to the profile, SecureSphere can detect abnormal behavior and prevent malicious activity with pinpoint precision. Each Web application has a single web profile and profile policy associated with it, which is applied only to the profile URLs and cookies in the Protect mode.

The table below lists all and with descriptions on their operation:

Policy Rules	Descriptions
Cookie Injection	SecureSphere learns which cookies should be protected and which should be ignored. A protected cookie is a cookie where SecureSphere can always trace the value that the Web application assigns to it. The value of a protected cookie must remain fixed and should not be altered by the user's browser. SecureSphere will trace and remember all protected cookies assigned by the Web application to each session. If a browser sends to the Web application a protected cookie which was not assigned to it by the Web application, then SecureSphere invokes the Cookie Injection violation.
Cookie Tampering	SecureSphere learns which cookies should be protected and which should be ignored. A protected cookie is a cookie where SecureSphere can always trace the value that the Web application assigns to it. The value of a protected cookie must remain fixed and should not be altered by the user's browser. For protected cookies SecureSphere will trace them and store the values assigned to them by the Web application during the entire user session. If a browser sends a protected

	cookie to the Web application with a different value than that was assigned by the Web application then SecureSphere invokes the Cookie Tampering violation.
Non- SOAP Access to a SOAP Only URL	When an HTTP request does not include a SOAP message and the URL was profiled as accessed through SOAP, SecureSphere invokes this violation.
Parameter Read Only Violation	SecureSphere learns which parameters are hidden parameters or embedded links whose values are set by the Web server and should not be changed manually by the user. During Protect Mode, SecureSphere traces the values that were set by the Web server and if the user manually altered a value, SecureSphere invokes this violation. The Parameter Read Only Violation must be enabled during learn mode in order for SecureSphere to learn which parameters are read-only.
Parameter Type Violation	For each parameter SecureSphere learns the type of the parameter. For example, SecureSphere can learn that a certain parameter's values consist only of numbers. During protect mode, if a certain parameter value doesn't match the learned types, SecureSphere invokes this violation.
Parameter Value Length Violation	For each parameter SecureSphere learns, using statistical algorithms, the minimum and maximum length of the parameter. During Protect Mode, SecureSphere checks all parameter values against the learned profile. If the parameter length exceeds the learned lengths, SecureSphere invokes this violation.
Required Parameter Not Found	SecureSphere learns the names of all parameters used by each URL. For each parameter, SecureSphere learns whether it's required or not (i.e. must be included or optional). During Protect Mode, if a required parameter is missing, SecureSphere invokes this violation.
Required XML Element Not Found	For each XML-based URL, SecureSphere learns and builds a profile of XML elements that have values in them. For each XML element or attribute, SecureSphere learns whether it's mandatory or optional). During Protect mode, if a mandatory XML element or attribute is missing, SecureSphere invokes this violation.
Reuse of Expired Session's Cookie	This violation is invoked if after receiving a new session identifier the user's browser continues to send protected cookies and protected cookies' values that were assigned to an expired session.
SOAP Access to a Non-SOAP URL	If an HTTP request includes a SOAP message but the URL was not profiled as a SOAP-enabled URL, SecureSphere invokes this violation.
SOAP Element Value Length Violation	For each XML-based URL, SecureSphere learns and builds a profile of XML elements that have values in them. For each value SecureSphere learns, using statistical algorithms, the minimum and maximum length of the value. During Protect mode, SecureSphere checks all XML values against the learned profile. If the value length exceeds the learned lengths, SecureSphere

	invokes this violation.
SOAP Element Value Type Violation	For each XML-based URL, SecureSphere learns and builds a profile of XML elements that have values in them. For each value SecureSphere learns the type of the value. For example, SecureSphere can learn that a certain XML value consists of numbers only. During Protect mode, if a certain XML value doesn't match the learned types, SecureSphere invokes this violation.
Unauthorized Method for Known URL	SecureSphere builds a profile of all allowed URLs. For each allowed URL the profile includes the allowed methods with that URL (e.g. GET, POST, HEAD). SecureSphere invokes this violation if, during Protect Mode, a known URL is sent with an unknown method.
Unauthorized SOAP Action	SecureSphere learns and builds a profile of all allowed SOAP actions for each URL. SecureSphere invokes this violation if the URL is accessed with a SOAP action not listed in its profile.
Unauthorized URL Access	It is possible for the SecureSphere administrator to lock directories in the Web profile (by right-clicking on a directory). SecureSphere invokes this violation when someone tries to access a URL which is not listed in the profile and is part of a locked directory.
Unknown Parameter	SecureSphere learns the names of all parameters used by each URL. During Protect Mode, SecureSphere checks that each URL includes only the learned parameter names. If a URL includes a parameter name which is not part of the profile, SecureSphere invokes this violation.
Unknown SOAP Element	For each XML-based URL, SecureSphere learns and builds a profile of XML elements that have values in them. During Protect mode, SecureSphere checks that each URL includes only the learned XML value names. If a URL includes a value name which is not part of the profile, SecureSphere invokes this violation.

Table 4: Web Profile Policy Rules.

6.6 Web Error Page Responses

6.6.1 Default Error Page

The Default Error Page is a generic error page that is displayed to users when they try to access a forbidden page (i.e. trigger a policy violation). This provides the ability to display a meaningful error page and prevent internal server information data leakage. The Default Error Page response can be set to either of two options: a redirect to a URL page or a custom page with a custom HTTP response code.

- Redirect: The WAF redirects to a specified URL. This could be an error page on the application web site.
- Page: WAF produces custom error pages. (by default, the error page is shown with an automatically generated event ID. If there is no event ID, it means that the WAF was not able to reach the application).

A third option, HTTP Response Code, works with either of the above and is used to configure the response code which is provided to users with the Default Error Page. Response codes must conform to RFC2616.

6.6.2 Web Error Page Policies

Web Error Page Policies are policies that are used to determine what web error page is displayed as the result of a specific set of circumstances. Web Error Page policies consist of rules based on a set of criteria and the Web Error Page to display when these set of circumstances are met.

Web Error Pages (just like the Default Error Page) are displayed to users when they try to access a forbidden page (i.e. trigger a policy violation) and provide a means to supply a meaningful error page and prevent data leakage of internal server information. Web error pages can be customized using html and placeholders, or can alternatively be a redirect which points users to an existing error page. Error pages need to be small enough to fit into a single packet. Placeholders that can be incorporated into the error pages include $(SESSION_ID), $(HOST) and $(EVENT_ID).

Web Error Page Policy rules can be configured based on criteria such as source IP address , host name, response code, policy violation etc. An example of the usefulness of this feature, in the context of applications on the US Courts Web DMZ, is the ability to present a different error page to users on the DCN from users on the Internet.

Application Owners: Application owners should provide requirements for web error page responses.

6.7 URL Rewrites and URL Redirection

URL Rewrites provide content modification for URL headers only and not web page content in server page responses. URL Redirection is used for SSL redirection of client requests from HTTP.

SecureSphere's URL Rewrite feature can be used in two situations:

- To rewrite host name and/or path in Web traffic URLs and headers. In this case the web client is unaware that the traffic is being diverted.
- To redirect Web traffic to another server. In this case, response code 302 is sent to the Web client, which then accesses the new URL in a new request.

The second situation is used for applications that require forcing web clients to use HTTPS when accessing a URL via HTTP.

Application Owners: Application owners should provide requirements for URL Rewrites and URL Redirection.

6.8 Name-based Virtual Hosting

The WAF can listen for applications and create profiles based on hostnames and URL paths. Name-based virtual hosts can be defined at the web application level by restricting monitoring to a defined host

header name. Host to web application mapping is defined to map an application host header name to a web application profile. Rules are then created to forward requests based on the host header name to a protected web application server.

Application Owners: Application owners should provide requirements for application host names.

6.9 Session Tracking

Session tracking enables SecureSphere to track sessions by using token names. Session identifiers are pieces of data used to identify a session. SecureSphere uses session identifiers as an alternative to IP addresses to identify users when monitoring and blocking traffic. They are also used to track session related violations such as read only parameter, cookie protection, web data base correlation, and more. It is important the SecureSphere have the correct sessions to ensure that operations is properly conducted such as profiling of session related elements.

SecureSphere is automatically installed with a list of standard session identifiers for most major applications. In order to get SecureSphere to properly identify all relevant traffic for web applications that use custom identifiers, these identifiers need to be added. These custom session identifiers (token names) need to be added for the WAF to accurately track user sessions.

Developers: Application developers should provide details of custom session identifiers for each application.

6.10 Blocking of Attacks

The SecureSphere® system supports the following blocking methods: TCP Reset and Inline Blocking. As the SecureSphere WAF is deployed in inline mode, inline blocking is used to block for violations to certain policies. Inline blocking means that the gateway will not pass the request to the designated server (i.e., drops the packet).

7.0 Web Profile Creation and Tuning Procedures

A web application profile will be created for each application on the WAF. During the learning phase, the WAF builds a baseline profile of normal application traffic. During the functional testing phase, the WAF enforces this positive security model based on the learned profile. False-positives will be detected and appropriate changes to the WAF application profile and policy will be made before going into production.

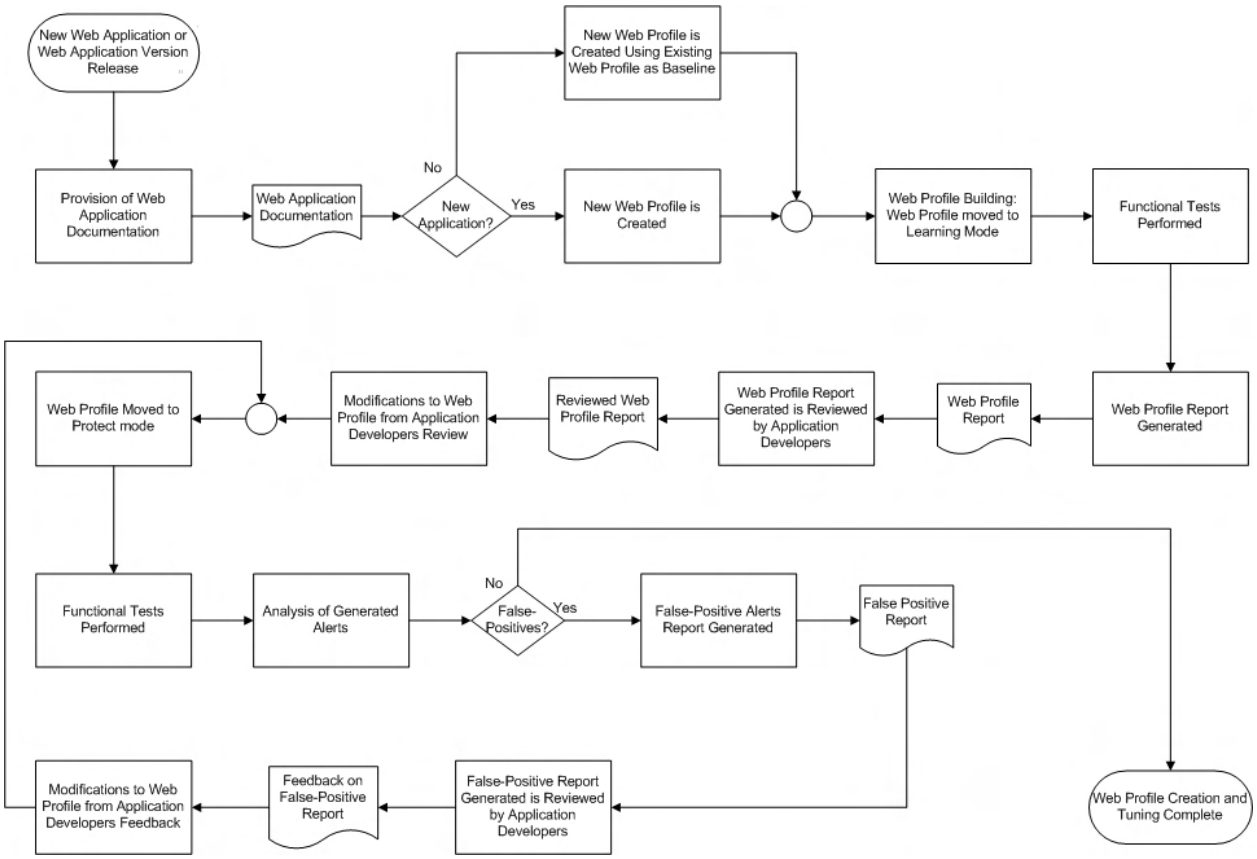

Figure 7: Web Profile Creation and Tuning Process Flow.

Procedures

1. Provision of Web Application Documentation: Web application documentation showing character values and minimum and maximum parameter lengths and application behavior is provided by web application developers.
2. Web Profile Creation: A new web profile is created for every new application. In the case of a version update of an existing application, the existing web profile of the previous version of the application is exported from production and imported for use as starting baseline.
3. Web Profile Building: Web application testers perform functional testing to build profile. During this step, ITSO moves the application web profile (the application's URLs) into protect learning mode for the period of testing. The WAF uses the client requests and the application/server responses from functional tests to build the application profile.
4. Web Profile Tuning: The following steps are performed as part of the web profile tuning process to eliminate out false-positives and achieve an accurate as possible web profile policy for the application.
    a. A web application profile report is generated and exported by ITSO for review by application developers after the creation of web profile during functional tests.
    b. ITSO makes modifications to the application web profile based on feedback from application developers after the review of submitted web profile report and on application documentation.
    c. Functional tests are performed again but this time with the web profile in protect mode.
    d. False-positive reports are generated and provided to web developers for review.
    e. ITSO makes further modifications to the application web profile based on feedback from application developers after the review of submitted false positive and web profile export reports.
    f. Functional tests are preformed again with the web profile in protect mode.
    g. If further changes are made to the application code, application developers are advised to notify ITSO with change details so that the web profile can be updated manually. Application developers or testers will need to test this specific functionality for the application once this changes have been made.

Notification List Creation: Developers are included on a notification list for web profile violations related to their applications. Application owners provide notification mailing list and ITSO adds action to send web profile violation alerts to notification list.

Access to Alerts: Developers are granted access to the SecureSphere web console to view violation alerts related to their applications. ITSO creates account.

8.0 References

1. The Web Application Security Consortium (webappsec.org)
2. Imperva SecureSphere Administration Guide version 8.5
3. Imperva SecureSphere User Guide version 8.5
4. Imperva Evaluation v0.3 - ITSO
5. WAF Evaluation Requirements - ITSO

Appendix A: Predefined Security Policies

Predefined Security Policies on the Imperva SecureSphere with their common configuration settings.

A1: Stream Signature

Policy Rules	Enabled	Severity	Action	Followed Action
Recommended Policy for General Applications – Legacy				
Recommended for Blocking for General Applications – Legacy	Yes	High	Block	Syslog
Recommended for Detection for General Applications - Legacy	Yes	Low	Block	Syslog
Worm and Critical Vulnerabilities for General Applications	Yes	High	Block	Syslog
Recommended Signatures Policy for General Applications				
Recommended for Blocking for General Applications	Yes	High	Block	Syslog
Recommended for Detection for General Applications	Yes	Low	Block	Syslog
Worm and Critical Vulnerabilities for General Applications	Yes	High	Block	Syslog

Table 5: Stream Signature

A2: HTTP Protocol Validation

Policy Rules	Enabled	Severity	Action	Followed Action
Web Protocol Policy				
Abnormally Long Header Line	Yes	Low	None	Syslog
Abnormally Long Request	Yes	Low	Block	Syslog
Double URL Encoding	Yes	Low	None	Syslog
Extremely Long HTTP Request	Yes	High	Block	Syslog
Extremely Long Parameter	Yes	Low	None	Syslog
Illegal Byte Code Character in Header Name	Yes	High	Block	Syslog
Illegal Byte Code Character in Header Value	Yes	Medium	None	Syslog
Illegal Byte Code Character in Method	Yes	High	Block	Syslog
Illegal Byte Code Character in Parameter Name	Yes	High	Block	Syslog
Illegal Byte Code Character in Parameter Value	Yes	Medium	None	Syslog
Illegal Byte Code Character in Query String	Yes	High	Block	Syslog
Illegal Byte Code Character in URL	Yes	High	Block	Syslog
Illegal Chunk Size	Yes	Low	None	Syslog
Illegal Content Length	Yes	High	Block	Syslog
Illegal Content Type	Yes	Low	None	Syslog
Illegal HTTP Version	Yes	High	Block	Syslog
Illegal Host Name	Yes	High	Block	Syslog
Illegal Parameter Encoding	Yes	Low	Block	Syslog
Illegal Response Code	Yes	Low	None	Syslog
Illegal URL Path Encoding	Yes	Low	None	Syslog
Malformed HTTP Header Line	Yes	Low	None	Syslog
Malformed SOAP Message	Yes	Medium	None	Syslog

Malformed URL	Yes	Low	None	Syslog
NULL Character in Header Name	Yes	Low	None	Syslog
NULL Character in Header Value	Yes	Low	None	Syslog
NULL Character in Method	Yes	Low	None	Syslog
NULL Character in Parameter Name	Yes	Low	Block	Syslog
NULL Character in Parameter Value	Yes	Low	None	Syslog
NULL Character in Query String	Yes	Low	None	Syslog
NULL Character in Url	Yes	Low	None	Syslog
Post Request - Missing Content Type	No	Medium	None	Syslog
Redundant HTTP Headers	Yes	Medium	None	Syslog
Redundant UTF-8 Encoding	Yes	High	Block	Syslog
Too Many Cookies in a Request	Yes	Info	None	Syslog
Too Many Headers per Request	Yes	Low	None	Syslog
Too Many Headers per Response	Yes	Medium	None	Syslog
Too Many URL Parameters	Yes	High	Block	Syslog
URL is Above Root Directory	Yes	High	Block	Syslog
Unauthorized Request Content Type	Yes	Medium	None	Syslog
Unknown HTTP Request Method	Yes	High	Block	Syslog

Table 6: HTTP Protocol Validation

A3: HTTP Protocol Signatures

Policy Rules	Enabled	Severity	Action	Followed Action
Recommended Policy for Web Applications – Legacy				
Recommended for Blocking for Web Applications – Legacy	Yes	High	Block	Syslog
Recommended for Detection for Web Applications - Legacy	Yes	Low	Block	Syslog
Worm and Critical Vulnerabilities for Web Applications	Yes	High	Block	Syslog
Recommended Signatures Policy for Web Applications				
Fullwidth/Halfwidth Unicode Encoding on URL Parameter	Yes	No Alert	None	
IIS Code Upload	Yes	No Alert	None	
MSSQL Data Retrieval with Implicit Conversion Errors	Yes	No Alert	None	
Recommended for Blocking for Web Applications	Yes	High	Block	Syslog
Recommended for Detection for Web Applications	Yes	Low	Block	Syslog
Worm and Critical Vulnerabilities for Web Applications	Yes	High	Block	Syslog

Table 7: HTTP Protocol Signatures

A4: Web Service Custom

Policy Rules	Enabled	Severity	Action	Followed Action
Anti Google Hacking	Yes	High	Block	Syslog
Apache Expect Header XSS	Yes	High	Block	Syslog
Automated Site Reconnaissance/Access	Yes	High	None	Syslog
Automated Vulnerability Scanning	Yes	High	None	Syslog
CVE-2010-2227-Apache-Tomcat-invalid-TE	Yes	Medium	None	Syslog
CVE-2010-3332 ASP Parameter Padding Oracle Brute Force	Yes	High	Block	Syslog
CVE-2010-3332 ASP URL Padding Oracle Brute Force	Yes	High	Block	Syslog
Cross Site Request Forgery	Yes	High	Block	Syslog
Data Leakage - American Express Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - Application Source Code	Yes	High	Block	Syslog
Data Leakage - Developer Comments	Yes	High	Block	Syslog
Data Leakage - Diner's Club / Carte Blanche Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - Discover Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - JCB Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - MasterCard Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - U.S Social Security Number	Yes	High	Block	Syslog
Data Leakage - Visa, Long Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - Visa, Short Credit Card Numbers	Yes	High	Block	Syslog
Data Leakage - enRoute Credit Card Numbers	Yes	High	Block	Syslog
Directory Browsing Detection	Yes	High	Block	Syslog
Directory Traversal (In Cookies/Parameters Value)	Yes	High	Block	Syslog
Directory Traversal (In URL)	Yes	High	Block	Syslog
Directory Traversal (In URL) - Basic Rule	Yes	Medium	None	Syslog
File Download Injection	Yes	High	Block	Syslog
Fullwidth/Halfwidth Unicode Decoding	Yes	High	Block	Syslog
HTTP Response Splitting Vulnerability	Yes	High	Block	Syslog
Hazardous HTTP request methods	Yes	Medium	None	Syslog
IE Discussion Bar- Access to Internal Information	Yes	Low	None	Syslog
ISS Code Upload	Yes	High	Block	Syslog
MSSQL Data Retrieval with Conversion Errors	Yes	High	Block	Syslog
Malformed HTTP Attack (Non compatible HTTP Results Error code)	Yes	High	None	Syslog
OS Command Injection	Yes	High	Block	Syslog
Plain Vanilla Scanner Detection	Yes	High	Block	Syslog
Privacy Violation - Credit Card Number Insertion	Yes	High	Block	Syslog
Privacy Violation - Credit Card Number Insertion by Internal IP Address	Yes	High	Block	Syslog
Privacy Violation - Credit Card Number Insertion by non Internal IP Address	Yes	High	Block	Syslog
Sensitive Error Messages Leakage	Yes	High	Block	Syslog
Suspected parameter tampering - Deprecated	Yes	High	None	Syslog
Suspicious Response Code	No	Medium	None	Syslog
System32 Access	Yes	High	Block	Syslog
ThreatRadar - Anonymous Proxies	No	High	None	Syslog
ThreatRadar - Malicious IPs	No	High	None	Syslog
ThreatRadar - Phishing URLs	No	High	None	Syslog
ThreatRadar - TOR IPs	No	High	None	Syslog

Unsuccessful Directory Browsing	Yes	Low	None	Syslog
WEB MISC Unauthorized File Access	Yes	High	None	Syslog
WEB-FRONTPAGE- Access to Internal Information	Yes	Low	None	Syslog
WEB-FRONTPAGE- External Access to Internal Information	Yes	Low	None	Syslog
WEB-FRONTPAGE-Access to Sensitive Internal Information	Yes	High	Block	Syslog
Webdav Method Detection	Yes	Medium	None	Syslog
eMail Hoarding	Yes	High	Block	Syslog

A5: Web Service Correlated Validation

Policy Rules	Enabled	Severity	Action	Followed Action
Web Correlation Policy				
Cross-site scripting	Yes	High	Block	Syslog
Forceful Browsing	Yes	Medium	None	Syslog
SQL Injection	Yes	High	Block	Syslog
Session Attribute Changes*	Yes	Info	None	Syslog
Too Many of the Same Response Code	Yes	Low	None	Syslog

Table 8: Web Service Correlated Validation

*Blocking is not enabled for Session Attribute Changes as this policy has not been evaluated yet.

A6: Web Profile Policy

Policy Rules	Enabled	Severity	Action	Followed Action
Cookie Injection	Yes	Medium	None	Syslog
Cookie Tampering	Yes	Medium	None	Syslog
Non- SOAP Access to a SOAP Only URL	Yes	Medium	None	Syslog
Parameter Read Only Violation	Yes	Info	None	Syslog
Issue Anomaly For Correlated Parameter Tampering	Yes			Syslog
Issue Anomaly For Requests Without Session	No		None	Syslog
Issue Anomaly For Response Evasion	Yes		None	Syslog
Parameter Type Violation	Yes	Medium	None	Syslog
Parameter Value Length Violation	Yes	Info	None	Syslog
Required Parameter Not Found	Yes	Info	None	Syslog
Required XML Element Not Found	Yes	Info	None	Syslog
Reuse of Expired Session's Cookie	Yes	Info	None	Syslog
SOAP Access to a Non-SOAP URL	Yes	Medium	None	Syslog
SOAP Element Value Length Violation	Yes	Info	None	Syslog
SOAP Element Value Type Violation	Yes	Medium	None	Syslog
Unauthorized Method for Known URL	Yes	Info	None	Syslog
Unauthorized SOAP Action	Yes	High	None	Syslog
Unauthorized URL Access	Yes	High	Block	Syslog
Unknown Parameter	Yes	Info	Block	Syslog
Unknown SOAP Element	Yes	Info	None	Syslog

Table 9: Web Profile Policy

A7: Security Policy Rule Parameters

Enabled	Enables the rule, meaning that this rule participates in protection provided by the policy. If **Enabled** is not selected, the rule is not applied.
Severity	Sets the severity of the rules of the selected policies. The following severity levels can be configured for each rule: • **No Alert:** No alert is generated for the violation by this policy. • **Informative:** Informative violations are not displayed in the dashboard. • **Low** • **Medium** • **High** **Note: When setting the severity to No Alert, followed actions are removed.**
Action	The immediate action that is taken when there is a match on the policy rule: • **None**: SecureSphere takes no action, and only generates violations and alerts, reporting about the security event. • **Block**: SecureSphere blocks the traffic that caused this violation and generates violations and alerts, reporting about the security event. Default: None.
Followed Action	The action that is taken by SecureSphere in addition to the action defined by the **Action** parameter. The additional action is taken from the pre-configured list of actions defined in Action Sets. The Followed Action is taken even when the Action is set to None. Defined action sets include: email, syslog, IP block, user block, session block etc. **Example**: once the violation is blocked, you want SecureSphere to send email to different people in the organization, notifying them about blocking of this specific violation.
Display Web Response Page	Determines whether or not to include the web response page shown to users in alerts for the selected policies. **Note: This option is only available with web based policies.**

Table 10: Security Policy Rule Parameters

Appendix B: Web Server Configuration

The WAF is configured to add the X-Forwarded-For HTTP header field in forwarded requests to the protected web server. The X-Forwarded-For field contains the originating IP address of the client making the request. On web servers, X-Forwarded-For configuration for web server logs is required for the web server to log the actual originating client IP address, rather than that of the WAF.

The following are links to resources that provide details on ways to add the "X-Forwarded-For" field for web server logs for two common web server products:

Microsoft Internet Information Services (IIS):

- http://blogs.iis.net/anilr/archive/2009/03/03/client-ip-not-logged-on-content-server-when-using-arr.aspx
- http://www.iis.net/download/ApplicationRequestRouting

Apache:

- http://devcentral.f5.com/weblogs/macvittie/archive/2008/06/02/3323.aspx
- http://httpd.apache.org/docs/2.3/mod/mod_remoteip.html