Contact Information

Document Revision: SGOS 6.7.x—05/2019-D

Copyright © 2018 Symantec Corp. All rights reserved. Symantec, the Symantec Logo, the Checkmark Logo, Blue Coat, and the Blue Coat logo are trademarks or registered trademarks of Symantec Corp. or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners. This document is provided for informational purposes only and is not intended as advertising. All warranties relating to the information in this document, either express or implied, are disclaimed to the maximum extent allowed by law. The information in this document is subject to change without notice.

THE DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. SYMANTEC CORPORATION SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE. SYMANTEC CORPORATION PRODUCTS, TECHNICAL SERVICES, AND ANY OTHER TECHNICAL DATA REFERENCED IN THIS DOCUMENT ARE SUBJECT TO U.S. EXPORT CONTROL AND SANCTIONS LAWS, REGULATIONS AND REQUIREMENTS, AND MAY BE SUBJECT TO EXPORT OR IMPORT REGULATIONS IN OTHER COUNTRIES. YOU AGREE TO COMPLY STRICTLY WITH THESE LAWS, REGULATIONS AND REQUIREMENTS, AND ACKNOWLEDGE THAT YOU HAVE THE RESPONSIBILITY TO OBTAIN ANY LICENSES, PERMITS OR OTHER APPROVALS THAT MAY BE REQUIRED IN ORDER TO EXPORT, RE-EXPORT, TRANSFER IN COUNTRY OR IMPORT AFTER DELIVERY TO YOU.

Americas:Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

Rest of the World:Symantec Limited
Ballycoolin Business Park
Blanchardstown, Dublin 15, Ireland
Preface

About This Guide .................................................................................................................................. 1
About WAF ........................................................................................................................................... 1
Documentation Resources ..................................................................................................................... 1

Chapter 1: Deployment Strategy for Production Environments

Step 1: Categorize Applications by State .............................................................................................. 3
Step 2: Categorize Applications by Size ............................................................................................... 3
  Determine Legacy Application Size with ProxySG Reports ............................................................. 4
Step 3: Define an Outline of WAF Security Controls to Enable ........................................................... 4
  Recommended Outline of Security Controls .................................................................................. 4
  Security Controls Categorized by Probability of Generating False-Positives .............................. 6
Step 4: Deploy the WAF ....................................................................................................................... 7
  Small Legacy Applications: Deploy the WAF ............................................................................... 7
  Large Legacy Applications: Deploy the WAF ............................................................................. 12
  New Applications: Deploy the WAF .......................................................................................... 15

Chapter 2: Protect Servers From Attacks

Solution: Write Policy to Protect Servers From Attacks ..................................................................... 17
Before You Begin ................................................................................................................................ 18
Add WAF Policy ................................................................................................................................ 18
Set up Access Logging ......................................................................................................................... 19
Analyze and Refine Policy .................................................................................................................... 20
  Check Access Logs ........................................................................................................................ 21
  Perform a Policy Trace ................................................................................................................... 22
  Create Policy Exemptions ............................................................................................................. 23
  Set Block Action in Policy ............................................................................................................. 23

Chapter 3: Configure and Review the WAF Access Log

Appendix A: WAF Policy Reference

  effective_date=.............................................................................................................................. 32
  http.csrf.authentication_link().................................................................................................... 33
  http.csrf.detection() ..................................................................................................................... 35
  http.csrf.token.insert() ................................................................................................................ 36
  http.csrf.token.name() .................................................................................................................. 37
  http.request.body.data_type() ..................................................................................................... 38
  http.request.body.inspection_size() ............................................................................................ 38
http.request.detection.bypass_cache_hit() ................................................................. 39
http.request.detection.exception() ............................................................................. 41
http.request.detection.other() .................................................................................... 42
http.request.detection.result.application_protection_set=................................................. 44
http.request.detection.result.validation= ................................................................. 44
http.request.log_details[header,body]() ..................................................................... 45
http.request.normalization.default() ........................................................................... 46

Appendix B: Support for HTTP Attributes
Preface

About This Guide
This guide describes features in SGOS 6.7.x that help you protect web applications from attacks.

About WAF
Today, more and more companies rely on web-based applications. While the "webification" of these applications brings many benefits, it also poses new challenges—the need to secure content, improve the user experience, and reduce the complexity associated with administrative overhead and performance.

The Web Application Firewall (WAF) solves these challenges. The Symantec® WAF solution can:
- Protect your web servers
- Accelerate web content
- Simplify operation

Documentation Resources
This document refers to SGOS product documentation. When noted in this document, consult the following version 6.7.x documents at MySymantec:
- SGOS Administration Guide
- Content Policy Language Reference
- Multi-Tenant Policy Deployment Guide
- Reverse Proxy Deployment WebGuide

These and all other current SGOS documentation are located at:
https://support.symantec.com/content/unifiedweb/en_US/Documentation.html?prodRefKey=1145522
In the Release field, select SGOS 6.7.x.
Chapter 1: Deployment Strategy for Production Environments

Retrofitting a Web Application Firewall (WAF) on top of an existing web application infrastructure can be challenging; for example, an operations team handling an infrastructure with many diverse web applications might find the added log data from a WAF overwhelming.

This chapter details the strategy for deploying the Symantec WAF in an enterprise environment, at scale, and directly in production. Use this strategy to minimize the volume of log data to review and to show quantifiable improvement to project stakeholders.

Deployment Strategy Overview

To deploy the WAF in production, do the following:

- “Step 1: Categorize Applications by State” on page 3
- “Step 2: Categorize Applications by Size” on page 3
- “Step 3: Define an Outline of WAF Security Controls to Enable” on page 4
- “Step 4: Deploy the WAF” on page 7

Step 1: Categorize Applications by State

The following table contains guidelines for determining your application’s state and an overview of how they are deployed:

<table>
<thead>
<tr>
<th>New Applications</th>
<th>Legacy Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>❖ Have not had user traffic flowing through them</td>
<td>❖ Have had user traffic flowing through them</td>
</tr>
<tr>
<td>❖ When deployed, are immediately set to Block mode</td>
<td>❖ When deployed, are initially set to Monitor mode and transitioned to Block mode, after troubleshooting false positives</td>
</tr>
<tr>
<td>❖ When deployed, have their false positives troubleshooted before the application goes live to avoid negatively impacting customers</td>
<td></td>
</tr>
</tbody>
</table>

Step 2: Categorize Applications by Size

The following table contains guidelines for determining your application’s size:

<table>
<thead>
<tr>
<th>Small Applications</th>
<th>Large Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>❖ Do not have rich user flows; most content is static</td>
<td>❖ Have rich user flows; most content is dynamic</td>
</tr>
<tr>
<td>❖ Have low traffic volume</td>
<td>❖ Have high traffic volume</td>
</tr>
</tbody>
</table>
**Determine Legacy Application Size with ProxySG Reports**

Run a report on the number of requests each application handles in a day to help determine the size of the application:

---

**Note:** These steps are only applicable for legacy applications that already have traffic flowing through a ProxySG infrastructure.

---

1. In Management Center, click **Reports > Reporter**.
2. Under **User Behavior**, click **Sites**. This report filters traffic per site, and displays the number of requests and page views for the selected time window.
3. Click **24h** to view the traffic for the last 24 hours for the appropriate sites.
4. Review the report to determine the size of the application.

For more information, see "View a Reporter Report" in the *Management Center 2.1 Configuration & Management Guide*.

**Step 3: Define an Outline of WAF Security Controls to Enable**

Plan which security controls to enable by considering your site’s environment and requirements, and then creating an outline of the minimum security controls your network requires to be enabled.

Not all controls are applicable for every site or some have a higher chance of generating false positives. For information on security controls’ probability of generating false positives, see "Security Controls Categorized by Probability of Generating False-Positives" on page 6.

After establishing this outline, you can enable additional controls in your security profiles.

**Recommended Outline of Security Controls**

The following table lists the recommended initial settings for the security controls in your security profile. These settings are recommended for all states and sizes of applications.

<table>
<thead>
<tr>
<th>Section</th>
<th>Control</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Validation</td>
<td>Protocol Compliance</td>
<td>Disable</td>
</tr>
<tr>
<td></td>
<td>JSON Validation</td>
<td>Enable</td>
</tr>
<tr>
<td></td>
<td>Request Body Data Type</td>
<td>Enable</td>
</tr>
<tr>
<td></td>
<td>Null Byte Detection</td>
<td>Enable</td>
</tr>
<tr>
<td></td>
<td>Invalid Form Data</td>
<td>Enable</td>
</tr>
<tr>
<td></td>
<td>Multiple Headers</td>
<td>Enable</td>
</tr>
<tr>
<td></td>
<td>Parameter Pollution</td>
<td>Disable</td>
</tr>
<tr>
<td></td>
<td>Parameter Pollution Separator</td>
<td>Disable</td>
</tr>
</tbody>
</table>
### Table 1–2

<table>
<thead>
<tr>
<th>Section</th>
<th>Control</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>Restricted File Upload Types</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Request Normalization</td>
<td>Path Normalization</td>
<td>Enable</td>
</tr>
<tr>
<td>Header Normalization</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Parameter Normalization</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Cookie Normalization</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Multiple Encoding</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Invalid Encoding</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Blacklist</td>
<td>Enable Blacklist</td>
<td>Enable</td>
</tr>
<tr>
<td>Analytics Filter</td>
<td>Enable Analytics Filter</td>
<td>Enable</td>
</tr>
<tr>
<td>Security Engines</td>
<td>HTML Injection</td>
<td>Enable</td>
</tr>
<tr>
<td>Command Injection</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows: Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Linux: Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OSX: Disable</td>
<td></td>
</tr>
<tr>
<td>Code Injection</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>SQL Injection</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Cross Site Scripting</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>XML Validation</td>
<td>XML External Entity</td>
<td>Disable</td>
</tr>
<tr>
<td>Include Reference Detection</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Invalid XML Action</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Expand CDATA Sections</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Request Security</td>
<td>Block HTML Tag Injection</td>
<td>Enable</td>
</tr>
<tr>
<td>Block Security Scanners</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Request Smuggling</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Buffer Overflows</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Aggressive Header Injection</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Block Insecure SSL Ciphers</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Response Security</td>
<td>Force &quot;secure&quot; and &quot;HttpOnly&quot;</td>
<td>Disable</td>
</tr>
<tr>
<td></td>
<td>Rewrite &quot;Server&quot; Response Header</td>
<td>Enable</td>
</tr>
</tbody>
</table>
Security Controls Categorized by Probability of Generating False-Positives

Not all WAF security controls generate the same amount of false-positive data. This table lists the false-positive probability for common WAF security controls.

### Table 1–2

<table>
<thead>
<tr>
<th>Section</th>
<th>Control</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Application Fingerprint Protection</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>HTTP Public Key Pinning</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>HTTP Strict Transport Security</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>X-XSS-Protection</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>X-Content-Type-Options</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Clickjacking: X-Frame-Options</td>
<td>Disable</td>
<td></td>
</tr>
<tr>
<td>Enable Response Error Codes Cloaking</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Optimizations</td>
<td>POST Body Processing</td>
<td>Enable</td>
</tr>
<tr>
<td>Cache Control</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>Logging</td>
<td>Log Header</td>
<td>On Monitor &amp; Block</td>
</tr>
<tr>
<td></td>
<td>Log Body</td>
<td>On Monitor &amp; Block</td>
</tr>
<tr>
<td>Cross-Site Request Forgery</td>
<td>Enable CSRF Protection</td>
<td>Disable</td>
</tr>
</tbody>
</table>

### Table 1–3

<table>
<thead>
<tr>
<th>Security Control</th>
<th>Likelihood of False Positive</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid Encoding</td>
<td>Low</td>
<td>Unlikely to occur for most traffic.</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>Low</td>
<td>Unlikely to occur in valid traffic.</td>
</tr>
<tr>
<td>JSON Validation</td>
<td>Low</td>
<td>Unlikely for web applications with well-formed JSON</td>
</tr>
<tr>
<td>Invalid Form Data</td>
<td>Low</td>
<td>Unlikely for web applications with correct forms</td>
</tr>
<tr>
<td>Null Byte Detection</td>
<td>Low</td>
<td>Unlikely to occur for most traffic.</td>
</tr>
<tr>
<td>Multiple Headers</td>
<td>Low</td>
<td>Unlikely for most web frameworks.</td>
</tr>
</tbody>
</table>
Step 4: Deploy the WAF

Deploy the WAF for your application using the appropriate strategy:

- For small, legacy applications, see "Small Legacy Applications: Deploy the WAF" on page 7.
- For large, legacy applications, see "Large Legacy Applications: Deploy the WAF" on page 12.
- For small or large, new applications, see "New Applications: Deploy the WAF" on page 15.

Small Legacy Applications: Deploy the WAF

This chapter details how to deploy WAF directly in production for small applications. In this strategy, for each small application, you will:

1. "Create and Configure Policy Objects to Monitor and Block Traffic" on page 8
2. "Review WAF Events Data" on page 10
3. "Add Exemptions for False Positives" on page 11
4. "Transition a Small Legacy Application to Block Mode" on page 11

**Note:** Symantec recommends selecting one small application to begin with. For subsequent iterations, you can move groups of small applications through this strategy simultaneously.

The following image depicts the completed strategy for small applications.

CREATE AND CONFIGURE POLICY OBJECTS TO MONITOR AND BLOCK TRAFFIC

Create tenants, a Tenant Determination File, a WAF security profile, and WAF Application objects, which are necessary for deploying policy for your application.

One tenant and WAF Application object are set to monitor traffic, and the other tenant and WAF Application object to block traffic. Initially, you configure the appliance to monitor traffic and, after reviewing the logs and resolving false positives, transition the appliance to Block mode.

To create and configure the necessary policy objects:

1. In Management Center, create a tenant for small applications in monitor mode. See "Manage Tenants" in the Management Center 2.1 Configuration & Management Guide.
2. Create a tenant for small applications in block mode. See "Manage Tenants" in the Management Center 2.1 Configuration & Management Guide.
3. Create a new or edit an existing Tenant Determination File and add a rule to route the small application’s traffic to the Monitor mode tenant.

**Note:** Only one Tenant Determination File can be installed per ProxySG appliance. If no previous Tenant Determination File has been created, create a one. If a Tenant Determination File already exists, update the existing one.

See "Specify Tenant Determination Rules" in the Management Center 2.1 Configuration & Management Guide.

4. Deploy the Tenant Determination File to the appropriate ProxySG appliances:
a. In Management Center, click **Configuration > Policy** and open the Tenant Determination file.
b. Click the **Targets** tab and click **Add Targets**.
c. Select the appropriate targets.
d. Click **Next**.
e. Ensure the **Deployment Type** is **Landlord Slot**.
f. Click **Finish**.

5. Create a WAF security profile using either your own WAF security target or Symantec’s recommended security target. See "Recommended Outline of Security Controls" on page 4 and "Configure WAF Security Rules" in the Management Center 2.1 Configuration & Management Guide.

6. Create a WAF Application object for Monitor mode:
   a. In Management Center, click **Configuration > Policy**.
   b. Click **Add Policy**.
   c. Type a **Policy name**.
   d. For the **Policy type**, select **WAF Application**.
   e. For the **Tenant**, select the tenant you created for small applications in monitor mode.
   f. Click **Next**.
   g. Select the **Environment**.
   h. Click **Finish**.
   i. Under **WAF Application Settings**, select the WAF security profile you created.
   j. Click **OK**.
   k. For the **Block/Monitor Override**, select **Change all WAF controls to** and set it to Monitor.
   l. Click **Save**, type a change description, and click **Save**.
   m. Navigate to the **Targets** tab and add targets.

7. Create a WAF Application object for Block mode:
   a. In Management Center, click **Configuration > Policy**.
   b. Click **Add Policy**.
   c. Type a **Policy name**.
   d. For the **Policy type**, select **WAF Application**.
   e. For the **Tenant**, select the tenant you created for small applications in block mode.
   f. Click **Next**.
   g. Select the **Environment**.
h. Click **Finish**.

i. Under **WAF Application Settings**, select the WAF security profile you created.

j. Click **OK**.

k. For the **Block/ Monitor Override**, select **Change all WAF controls to** and set it to **Block**.

l. Click **Save**, type a change description, and click **Save**.

m. Navigate to the **Targets** tab and add targets.

8. Deploy the WAF Application object. See "Install Policy" in the *Management Center 2.1 Configuration & Management Guide*.

9. (Optional) If you are using a Management Center job to deploy policy, update the job with the new WAF application object, and run the job. See "Edit a Job" and "Monitor Jobs" in the *Management Center 2.1 Configuration & Management Guide*.

In this initial Monitor mode configuration, the ProxySG appliance routes traffic to the Tenant Determination File, which then routes traffic to the Monitor mode tenant. This tenant contains and deploys the WAF application policy for "Small Applications in Monitor Mode".

**Review WAF Events Data**

When enough traffic has flowed through the ProxySG appliance, review the WAF events to ensure your configuration is correct. The amount of traffic that constitutes as "enough" depends on your environment. Typically, traffic for a 24-hour period during a normal business day is a reasonable amount.

**Note:** If automation or functional regression testing is available to you, consider leveraging it during this evaluation stage of the deployment.

To review the WAF events:

1. In Management Center, create a new report with the following parameters:
   
   a. Grouping Type: Two Level
   b. Group By: Site
   c. Then By: Attack Family
   d. Chart: Bar
   e. Time Frame: 24h
   f. Columns Display: Requests

   See "Create a Custom Reporter Report" in the *Management Center 2.1 Configuration & Management Guide*.


3. Review the log details for the Monitor mode site.
1. For the Monitor mode site that you want to review the user data for, expand the report menu to show the per-Attack Family data.

2. To view the Full Log Details report, right-click on each of the Attack Family links. The only link you do not need to click is the "No Attack Family" link.

3. For each Attack Family, review each entry in the Full Log Details report that the WAF flagged.

4. Any requests from non-malicious clients that the WAF flagged are false positives. Create exemptions for these clients in your security profile. See "Add Exemptions for False Positives" on page 11. For obvious attack traffic that the ProxySG appliance has detected, no exemptions are required as they are valid detections.

**Add Exemptions for False Positives**

A false positive is a valid client request that was incorrectly flagged by the WAF and, if the WAF application was set to Block mode, the request would have been blocked. To ensure legitimate traffic is not blocked, add exemptions to your security profile. These exemptions instruct the WAF to ignore these requests and similar ones so that they are not blocked when you switch the application from Monitor mode to Block mode.

**Before You Begin**

Ensure you have reviewed the Full Log details report. You require the details in the report to determine which applications and clients had false positives logged against them and require exemptions. See "Review WAF Events Data" on page 10.

To add exemptions:

1. In Management Center, in the appropriate security profile, create a new exemption. See "Manage WAF Security Policy" in the Management Center 2.1 Configuration & Management Guide.

**Note:** When selecting WAF engines, properties, and rules to be ignored, scope exemptions as narrowly as possible. For example, if a specific URL is problematic, only turn off the specific request part for the WAF engine that flagged the URL. Turning off a part and not the entire engine ensures that the engine still runs on all other parts for that URL, and on all parts of all other URLs.

2. To install the updated configuration on the ProxySG targets, save the security profile and redeploy the associated WAF application:
   a. In the policy editor, click Save.
   b. Click the Targets tab and click Install to Target.

3. Repeat these steps for any other false positives.

**Transition a Small Legacy Application to Block Mode**

After refining your policy, transition the application to Block mode. To transition the appliance, update the Landlord to route the application’s traffic to the Block mode tenant instead of to the Monitor tenant.
Note: Both the Monitor mode and Block mode WAF applications reference the same security profile. This reference ensures that when you transition your the ProxySG appliance to Block mode, the appliance uses any exemptions that you added when you refined your policy.

To transition the application to Block mode:

1. In Management Center, click Configuration > Policy and open the Tenant Determination file.
2. Select the Monitor mode tenant and click Edit.
3. In the Tenant field, click the edit icon. The Select Tenant window displays.
4. Select the Block mode tenant.
5. Click OK.
6. Click Save.
7. To redeploy the Tenant Determination (Landlord) file, click the Targets tab and click Install to Target.

For further information, see "Manage WAF Security Policy" in the Management Center 2.1 Configuration & Management Guide.

Small Applications: Next Steps

If you have additional small applications to deploy, you can deploy multiple small applications simultaneously. Select one or more small applications and repeat the deployment strategy, or deploy applications of other states and sizes. To return to the start of the deployment types, see "Step 4: Deploy the WAF" on page 7.

Large Legacy Applications: Deploy the WAF

Large web applications are complex and handle substantial amounts of traffic. Because of these factors, they require a unique deployment strategy.

To enable WAF controls for each large application without impacting other web applications:

1. "Create Unique Security Profiles for Large Applications" on page 13
2. "Create and Configure Policy Objects to Monitor Traffic" on page 13
3. "Review and Refine Policy" on page 15
4. "Transition the Security Control to Block Mode" on page 15

Note: Symantec recommends selecting one large application to begin with. On later iterations, you can move groups of large applications through this strategy simultaneously.
Create Unique Security Profiles for Large Applications

Create a unique security profile for each large application. Having a profile for each allows you to tailor the security settings; for example, if you have a web application that runs on Linux, in the security profile for that application, you can enable the Linux sub-option for the Command Injection engine. In this scenario, enabling that sub-option optimizes performance and reduces the number of false positives.

For each large application’s security profile, initially all security controls are disabled except for one, which is set to Monitor mode. For the one control in Monitor mode, review the data events for false positives, create exemptions, and transition it to Block mode. Repeat the process for the next security control.

To create a security profile for your large application:

2. Disable all security controls. To disable them, view all of the controls and ensure each checkbox for each security control is unchecked.
3. Set one security control to Monitor mode:
   a. Click Security Engines.
   b. For the Command Injection control, select the operating system that your application uses and set the action to Monitor.
   c. Click Save.


Create and Configure Policy Objects to Monitor Traffic

Create a tenant, a Tenant Determination File, and a WAF Application object, which are necessary for deploying policy for your application.

To create objects for your WAF application:

1. Create a tenant for your large applications. See "Manage Tenants" in the Management Center 2.1 Configuration & Management Guide.

   Large applications require only one tenant because you create a unique security profile for each application. In this unique profile, you can toggle security controls on and off, and from monitor to block.

2. Create a new or edit an existing Tenant Determination File and add a rule to route the application’s traffic to the tenant for your large application.
Note: Only one Landlord can be installed per ProxySG appliance. If no previous Landlord has been created, create a Landlord object. If a Landlord already exists, update the existing one.

See "Specify Tenant Determination Rules" in the Management Center 2.1 Configuration & Management Guide.

3. Create a WAF Application object for your application:
   a. In Management Center, click Configuration > Policy.
   b. Click Add Policy.
   c. Type a Policy name.
   d. For the Policy type, select WAF Application.
   e. For the Tenant, select the tenant you created for your large applications.
   f. Click Next.
   g. Select the Environment.
   h. Click Finish.
   i. Under WAF Application Settings, select the WAF security profile you created for this large application.
   j. Click OK.
   k. Ensure the Block/Monitor Override control is disabled.
   l. Click Save, type a change description, and click Save.
   m. Navigate to the Targets tab and add targets.

See "Configure WAF Application Objects" in the Management Center 2.1 Configuration & Management Guide.

4. Deploy the WAF application. See "Install Policy" in the Management Center 2.1 Configuration & Management Guide.

5. Update the Tenant Determination File to use the appropriate ProxySG targets:
   a. Click Configuration > Policy and open the Tenant Determination File.
   b. Click the Targets tab and click Add Targets.
   c. Select the appropriate targets.
   d. Click Next.
   e. Ensure the Deployment Type is Landlord Slot.
   f. Click Finish.

6. (Optional) If you are using a Management Center job to deploy policy, update the job with the new WAF application object, and run the job. See "Edit a Job" and "Monitor Jobs" in the Management Center 2.1 Configuration & Management Guide.
Review and Refine Policy

After a reasonable amount of traffic has passed through the ProxySG appliance, review the WAF events and add exemptions where necessary. See "Review WAF Events Data" on page 10 and "Add Exemptions for False Positives" on page 11.

Transition the Security Control to Block Mode

To transition the security control to Block mode:

1. Click Configuration > Shared Objects and click on the security profile for your large application.
2. Transition the control to Block mode:

   Note: In these steps, the Security Engine control for Command Injection is used as an example for the first control to be transitioned from Monitor mode to Block mode. Depending on the security target you created for your application, the controls you choose to enable might differ.

   a. Click Security Engines.
   b. For the Command Injection control, set the action to Block.
   c. Click Save.
3. Redeploy the associated WAF application. See "Install Policy" in the Management Center 2.1 Configuration & Management Guide.

Large Applications: Next Steps

With one security control transitioned to Block mode, proceed to the next security control and repeat the process.

If you have small applications and large applications to deploy a WAF for, you can deploy large applications in parallel with the small applications. You can deploy them at the same time because each large application has its own WAF application, tenant, and security profile. Making changes to any of these large application objects does not affect small application objects, which have their own security profile. Similarly, you can on-board multiple large applications at the same time. To return to the start of the deployment types, see "Step 4: Deploy the WAF" on page 7.

New Applications: Deploy the WAF

For new applications, do the following:

1. "Add a Security Gate to Your Organization’s Processes" on page 16
2. "Categorize the New Application’s Size" on page 16
3. "Set New Applications to Block Mode" on page 16
4. "Review and Refine Policy" on page 16
Add a Security Gate to Your Organization’s Processes

Add an operations security gate to the application development and release process in your organization. Doing so will include the network/security operations teams in the new application release process. Including these teams allows them to assess the new web application and prepare for the security-related WAF changes. Adding WAF considerations early in the process allows you to maximize the integration testing of the new web application with the WAF.

Categorize the New Application’s Size

Categorize the anticipated size of the application. Determine whether to treat the new application as small (using a small-application security profile, WAF application, and tenant that is shared among all small applications) or as a large application (using a unique security profile, WAF application, and tenant for each large application). See "Step 1: Categorize Applications by State" on page 3.

Set New Applications to Block Mode

For new applications, immediately set them to Block mode instead of monitoring them first. Follow either the strategy for small applications or the strategy for large applications, but modify the steps so that you start your application in Block mode.

For the small application strategy, see "Step 4: Deploy the WAF" on page 7.
For the large application strategy, see "Large Legacy Applications: Deploy the WAF" on page 12.

Review and Refine Policy

After a reasonable amount of traffic has passed through the ProxySG appliance, review the WAF events and add exemptions where necessary. See "Review WAF Events Data" on page 10 and "Add Exemptions for False Positives" on page 11.
Chapter 2: Protect Servers From Attacks

More and more businesses are moving to web-based applications. The move brings many benefits, but also poses significant risks because many web applications contain security vulnerabilities. SQL Injection and Cross-Site Scripting (XSS) attacks are among the most common security risks to web applications.

Symantec’s WAF solution includes advanced detection engines designed to combat these and other threats as well as the ability to handle false positives. In addition to using pattern matching, the ProxySG appliance can perform attack detection:

- Based on *attack families*, which are types of common attacks.
- Using *content nature detection engines*, understand the nature of the content, enabling the WAF to distinguish attacks from innocuous traffic.
- Using *blacklists*, which discover well-known attack patterns quickly and efficiently.
- Using *Analytics Filter*, which detects attack characteristics and triggers intelligently based on the sum of the anomalies. This technology is based on attack types matching with weights and thresholds.

Integrate WAF With Other Symantec Solutions

You can use other Symantec solutions to optimize your WAF deployment:

- Multi-tenant policy - Create security profiles with different levels of protection and define different WAF policies for each. For example, you might create less restrictive policies for internal URLs and policies with stronger protection for other requests.
  
  Refer to the *Multi-Tenant Policy Deployment Guide* for details.

- Symantec Management Center - Use Management Center (1.5.x and later) to construct WAF policy. In contrast to writing ProxySG policy using CPL, you can use Management Center’s user-friendly wizards to define tenants, create WAF policies, and deploy the policies to target appliances.

  Refer to the *Management Center Configuration & Management Guide* for details.

*Note:* To use these solutions with WAF, you require a Web Application Protection subscription bundle; refer to the *Multi-Tenant Policy Deployment Guide* and *Management Center Configuration & Management Guide* for information on additional requirements.

Solution: Write Policy to Protect Servers From Attacks

To take advantage of WAF features in SGOS 6.7.x and protect your back-end servers against attacks, complete the tasks in the following table.
Table 2–1  WAF Solution Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Go to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Meet prerequisites.</td>
<td>&quot;Before You Begin&quot;</td>
</tr>
<tr>
<td>2. Add CPL to your local or central policy file.</td>
<td>&quot;Add WAF Policy&quot;</td>
</tr>
<tr>
<td>3. Set up access logging.</td>
<td>&quot;Set up Access Logging&quot;</td>
</tr>
<tr>
<td>4. Analyze and refine your policy.</td>
<td>&quot;Analyze and Refine Policy&quot;</td>
</tr>
</tbody>
</table>

Before You Begin

Before using WAF features in SGOS 6.7.x in your reverse proxy deployment, you must:

- Set up your test environment for WAF evaluation.
- Obtain a Web Application Protection subscription or verify that your existing one is valid.
- Install required versions of SGOS and the Application Protection database.
- (If applicable) Licenses for Management Center and multi-tenancy.

Add WAF Policy

Configure WAF features by adding CPL to your local or central policy file. For more information on CPL, refer to the Content Policy Language Reference.

The following example includes:

- A definition set of engines and the label for them.
- Rules for detecting and monitoring several types of attacks.
- Rules for detecting and monitoring several request validation properties, such as invalid encoding and invalid JSON.
- A virtual patch, that is, policy designed to prevent malicious users from exploiting a known security vulnerability (in this example, an Apache Tomcat vulnerability).

Example

```
;; Conditions
;; =========
define application_protection_set AllEnginesAllParts
  engine=xss
  engine=Injection.sql
  engine=Injection.code
  engine=Injection.command
  engine=Injection.html
  engine=Reference.directory_traversal
  engine=blacklist
  engine=analytics_filter
end
```
define condition tomcat_loop_attack
    request.header.Content-Type.length=4000..
end

;; Normalization
;;==================
<proxy>
http.request.normalization.default(auto)

;; Evasion Detection
;;==================
<proxy>
http.request.body.data_type(auto) \\ http.request.detection.other.null_byte(monitor) \\ http.request.detection.other.invalid_encoding(monitor) \\ http.request.detection.other.invalid_form_data(monitor) \\ http.request.detection.other.invalid_json(monitor) \\ http.request.detection.other.multiple_encoding(monitor) \\ http.request.detection.other.multiple_header(monitor) \\ http.request.body.inspection_size(65536) \\ http.request.detection.other.threshold_exceeded(monitor)

;; WAF Engines
;;=============
<proxy>
http.request.detection.AllEnginesAllParts(monitor)

;; Virtual Patch
;;==============
<proxy> condition=tomcat_loop_attack
    force_exception(invalid_request, \\ "Transaction ID: $(x-bluecoat-transaction-uuid)"
    log_rewrite.x-bluecoat-waf-attack-family("Tomcat loop attack")

Set up Access Logging

You can inspect the logs to check for issues in the WAF configuration after you run test traffic through the appliance.

1. If access logging is not enabled already, enable it.
   Select Configuration > Access Logging > General. Select Enable Access Logging and then click Apply to save your changes.

2. Set the main access log to use the bcreporterwarp_v1 format. This format includes new fields for WAF. See "Configure and Review the WAF Access Log" on page 25 for descriptions of the fields.
Note: Alternatively, you can:

- Create a new log based on the bcreporterwarp_v1 access log format. Assign the new log format to the main log (see below).
- Use policy to enable logging to the new format.

Select Configuration > Access Logging > Logs > General Settings. For Log, select main. For Log Format, select bcreporterwarp_v1. You receive a message stating that changing the log format might cause different fields to be logged. Click OK to close the message.

Click Apply to save your changes. The console might display another message about changing the log format; click Close to dismiss the message.

After installing policy and completing access log configuration, run traffic through the appliance.

Analyze and Refine Policy

After installing an initial version of WAF policy on the ProxySG appliance ("Add WAF Policy" on page 18), you can analyze the results of the traffic to determine what attacks have been detected.

There is a chance that the detection engines have flagged a legitimate request as an attack. For example, if a blog post includes an example of a cross-site scripting (XSS) attack, the appliance interprets the example as an actual attack and blocks the post. This might be undesirable behavior and considered a false positive. Address this and other kinds of false positives with the following workflow.

Table 2–2 Analyze and Refine Policy

<table>
<thead>
<tr>
<th>Step</th>
<th>Overview</th>
<th>Reference (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Check access logs to determine which rules or engines you must update to address false positives, false negatives, and other wanted behavior.</td>
<td>See &quot;Check Access Logs&quot; on page 21.</td>
</tr>
<tr>
<td>2</td>
<td>(Optional) Perform a trace to troubleshoot policy.</td>
<td>See &quot;Perform a Policy Trace&quot; on page 22.</td>
</tr>
<tr>
<td>3</td>
<td>(If using Management Center) Save virtual patches as CPL fragments in Management Center; thus, you can reuse the virtual patches in larger policies, including tenant-specific policy.</td>
<td>Refer to the Management Center Configuration &amp; Management Guide.</td>
</tr>
</tbody>
</table>
Check Access Logs

Check the access logs. To learn more about access logs, refer to the *SGOS Administration Guide*.

The following are examples of a log format and output:

**Sample Log Format and Output**

The following is an example of a log format that includes some WAF fields:

```
date c-ip x-bluecoat-transaction-uuid cs-method cs-host x-bluecoat-waf-attack-family x-bluecoat-waf-block-details x-bluecoat-waf-monitor-details
```

The following is an example of output from the sample log format:

---

**Table 2–2 Analyze and Refine Policy**

<table>
<thead>
<tr>
<th>Step</th>
<th>Overview</th>
<th>Reference (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Based on your analysis of the access logs, create policy exemptions to eliminate false positives and other unwanted behavior. As a best practice, make sure that <code>block</code> and <code>monitor</code> actions precede policy exemptions; otherwise, policy that addresses false positives is overridden and requests that you intended to allow might be denied.</td>
<td>See &quot;Create Policy Exemptions&quot; on page 23.</td>
</tr>
<tr>
<td>5</td>
<td>Run traffic through the appliance and confirm through access logs (and optionally, other troubleshooting tasks) that requests match both general rules and exemptions appropriately.</td>
<td>Repeat steps 1 through 3 in this table as often as required.</td>
</tr>
</tbody>
</table>
| 6    | After confirming that false positives no longer occur, consider your next step. You can do any of the following according to your needs:  
  * Update policy actions from `monitor` to `block`. Then, move to a production environment when your WAF policy is stable and you observe no other issues with how the appliance handles traffic.  
  * Continue to test and refine policy, move to production, and then update policy actions to `block`.  
  * Continue to test and refine policy, move to production, and gradually update each engine or policy’s actions to `block`.                                                                                   | Repeat the previous steps as needed.            |
| 7    | Because web applications can change, update, or remove functionality, you might need to adapt your policy over time so that the appliance continues to handle traffic as intended.                                                                                   | Refer to this document and the *Content Policy Language Reference* for general information. |
Additional information on the WAF block and monitor details fields:

If the request triggers a blacklist rule that has CVE details, the `x-bluecoat-waf-block-details` and `x-bluecoat-waf-monitor-details` log fields display the relevant CVE numbers in the format `"cve": ["<cve-number>", ...]`.

When invalid encoding or multiple encoding issues are identified in normalization, the function that caused it is added to the `x-bluecoat-waf-block-details` or `x-bluecoat-waf-monitor-details` log fields as appropriate. Access logs include details for following functions:

- Multiple encoding: base64Decode, cssDecode, htmlEntityDecode, jsDecode, urlDecode, urlDecodeUnicode
- Invalid encoding: utf8toUnicode

WAF Log Data

When a WAF engine detects an attack, the log records the request and detection details in the log details field based on the associated action (Block or Monitor):

- `x-bluecoat-waf-block-details`:
  Block actions stop further processing and contain 0 or 1 entries.

- `x-bluecoat-waf-monitor-details`:
  Monitor actions continue WAF processing and might contain multiple entries depending on how many WAF engines or properties find an issue with the request.

  If WAF engines detect a command injection attack, these fields include the version of the command injection engine used for the detection.

  See Table A–1, "WAF Policy Gestures" on page 29 for information on the block and monitor actions, as well as the version keyword.

**Note:** If the action is Ignore, the request is allowed and not logged.

Refer to `define application_protections_set` in the *Content Policy Language Reference* for more information about these properties.

**Note:** The format of these log fields is JSON. The double quotes are escaped by a double-quote as per the ELFF specification. For more information, see "Configure and Review the WAF Access Log" on page 25.

Perform a Policy Trace

(Optional) Perform a policy trace to determine how the appliance applies policy to specific requests.

To enable policy tracing, select Configuration > Policy > Policy Options. Under Default Policy Tracing, select Trace proxy traffic policy execution and click Apply.
To prevent performance issues, turn off the policy trace before you proceed to the next step.

**Create Policy Exemptions**

To handle false positives, define exemptions (Ignore actions) in your baseline WAF policy.

**Note:** Ensure that block and monitor actions precede ignore actions in policy. For Ignore actions to override Block and Monitor detection behaviors as intended, any ignore policy gestures must follow their block and monitor counterparts.

**Example**

The following scenario is an example of a false positive and how to address it using policy:

1. The blacklist engine reports a false positive for the following URL:
   
   `http://app.bluecoat.com`

2. You check the access log and determine that a specific blacklist rule is causing the false positive. To mitigate false positives, turn off the specific offending rule (rather than disabling the entire blacklist engine).
   
   ```
   ; specify the engine and rule in a new definition block
   ; associate the ignore action with this definition

   define application_protection_set ignore-mode
   engine=blacklist rule=BL-2000-1
   end

   <proxy>
   http.request.detection.ignore-mode(ignore)
   ```

   For detailed information on creating CPL policy, refer to the *Content Policy Language Reference*.

**Set Block Action in Policy**

**Note:** As "Example" on page 23 shows, block and monitor actions should precede policy exemptions.

After you create exemptions, any false positives that matched previously should now be allowed. To address the traffic that you want to deny, change the monitor action to block in the respective WAF CPL constructs.

Continue to refine policy as required until benign requests and malicious requests are allowed or blocked as appropriate.
Chapter 3: Configure and Review the WAF Access Log

The `bcreporterwarp_v1` access log format allows you to send data about reverse proxy-specific features (such as Geolocation and Application Protection) to Symantec Reporter. This is a reserved format and cannot be edited. The format includes the following access logging fields:

```
date time time-taken c-ip cs-username cs-auth-group x-bluecoat-
transaction-uuid x-exception-id cs(Referer) sc-status s-action cs-
method rs(Content-Type) cs-uri-scheme cs-host cs-uri-port cs-uri-
path cs-uri-query cs-uri-extension cs(User-Agent) s-ip sc-bytes cs-
bytes x-virus-id x-user-x509-subject rs-bytes x-cs-client-effective-ip x-cs-client-
effective-ip-country cs(X-Forwarded-For) rs-service-latency r-ip x-
bluecoat-application-name x-bluecoat-waf-attack-family x-risk-score
x-bluecoat-waf-block-details x-bluecoat-waf-monitor-details x-
bluecoat-request-details-header x-bluecoat-request-details-body x-
bluecoat-waf-scan-info
```

**Note:** Management Center version 1.8.x and later supports Symantec WAF App for Splunk, which aggregates and parses WAF log data, and presents it visually on dashboards. Refer to the ProxySG Web Application Firewall (WAF) App for Splunk Enterprise document and Management Center Configuration & Management Guide for details.

### Table 3–1  `bcreporterwarp_v1` access log fields

<table>
<thead>
<tr>
<th>Access Log Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>date</code></td>
<td>GMT date in YYYY-MM-DD format.</td>
</tr>
<tr>
<td><code>time</code></td>
<td>GMT time in HH:MM:SS format</td>
</tr>
<tr>
<td><code>time-taken</code></td>
<td>Time taken (in milliseconds) to process the request (from the first byte of client request data received by the proxy, to the last byte sent by the proxy to the client, including all of the delays by ICAP, and so on).</td>
</tr>
<tr>
<td><code>c-ip</code></td>
<td>Client IP address.</td>
</tr>
<tr>
<td><code>cs-username</code></td>
<td>Relative username of a client authenticated to the proxy (i.e. not fully distinguished).</td>
</tr>
<tr>
<td><code>cs-auth-group</code></td>
<td>One group that an authenticated user belongs to. If a user belongs to multiple groups, the group logged is determined by the Group Log Order configuration specified in VPM. If Group Log Order is not specified, an arbitrary group is logged. Only groups referenced by policy are considered.</td>
</tr>
</tbody>
</table>
Table 3–1  bcreporterwarp_v1 access log fields

<table>
<thead>
<tr>
<th>Access Log Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-bluecoat-transaction-uuid</td>
<td>Per-request transaction ID that is globally unique across all transaction across multiple ProxySG appliances. Default exception pages include the transaction ID; thus, you can look for the ID in the access log to ascertain if WAF engines correctly detected an attack or if it was a false positive.</td>
</tr>
<tr>
<td>x-exception-id</td>
<td>Identifier of the exception resolved (empty if the transaction has not been terminated).</td>
</tr>
<tr>
<td>cs(Referer)</td>
<td>Request header: Referer.</td>
</tr>
<tr>
<td>sc-status</td>
<td>Protocol status code from appliance to client.</td>
</tr>
<tr>
<td>s-action</td>
<td>What type of action did the appliance take to process this request; possible values include ALLOWED, DENIED, FAILED, SERVER_ERROR.</td>
</tr>
<tr>
<td>cs-method</td>
<td>Request method used from client to appliance.</td>
</tr>
<tr>
<td>rs(Content-Type)</td>
<td>Response header: Content-Type.</td>
</tr>
<tr>
<td>cs-uri-scheme</td>
<td>Scheme from the 'log' URL.</td>
</tr>
<tr>
<td>cs-host</td>
<td>Hostname from the client's request URL. If URL rewrite policies are used, this field's value is derived from the 'log' URL.</td>
</tr>
<tr>
<td>cs-uri-port</td>
<td>Port from the 'log' URL.</td>
</tr>
<tr>
<td>cs-uri-path</td>
<td>Path from the 'log' URL. Does not include query.</td>
</tr>
<tr>
<td>cs-uri-query</td>
<td>Query from the 'log' URL.</td>
</tr>
<tr>
<td>cs-uri-extension</td>
<td>Document extension from the 'log' URL.</td>
</tr>
<tr>
<td>cs(User-Agent)</td>
<td>Request header: User-Agent.</td>
</tr>
<tr>
<td>s-ip</td>
<td>IP address of the appliance on which the client established its connection.</td>
</tr>
<tr>
<td>sc-bytes</td>
<td>Number of bytes sent from appliance to client.</td>
</tr>
<tr>
<td>cs-bytes</td>
<td>Number of bytes sent from client to appliance.</td>
</tr>
<tr>
<td>x-virus-id</td>
<td>Identifier of a virus if one was detected.</td>
</tr>
<tr>
<td>x-cs-client-ip-country</td>
<td>Client country of origin.</td>
</tr>
<tr>
<td>x-user-x509-serial-number</td>
<td>X.509 certificate serial number.</td>
</tr>
<tr>
<td>x-user-x509-subject</td>
<td>X.509 certificate subject.</td>
</tr>
</tbody>
</table>
### Table 3–1  bcreporterwarp_v1 access log fields

<table>
<thead>
<tr>
<th>Access Log Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs-bytes</td>
<td>Number of bytes sent from upstream host to appliance.</td>
</tr>
<tr>
<td>x-cs-client-effective-ip</td>
<td>In load balancing environments, reports actual client IP address</td>
</tr>
<tr>
<td>x-cs-client-effective-ip-country</td>
<td>In load balancing environments, reports actual client country of origin.</td>
</tr>
<tr>
<td>cs(X-Forwarded-For)</td>
<td>In multi-layer proxy deployments, reports the forwarding proxy.</td>
</tr>
<tr>
<td>rs-service-latency</td>
<td>OCS response time. The time from the start of the OCS connection to when the ProxySG appliance receives the first response byte.</td>
</tr>
<tr>
<td>r-ip</td>
<td>IP address from the outbound server URL.</td>
</tr>
<tr>
<td>x-bluecoat-application-name</td>
<td>Reports the application name.</td>
</tr>
<tr>
<td>x-bluecoat-waf-attack-family</td>
<td>Natural language description of the detected attack family.</td>
</tr>
<tr>
<td>x-risk-score</td>
<td>Risk score.</td>
</tr>
<tr>
<td>x-bluecoat-waf-block-details</td>
<td>Details about the blocked or monitored request, displayed in JSON format consisting of an array of CSV objects:</td>
</tr>
<tr>
<td></td>
<td>Each object is a CSV list of &quot;key&quot;: &quot;value&quot; pairs:</td>
</tr>
<tr>
<td></td>
<td><code>{&quot;key1&quot;: &quot;value1&quot;, &quot;key2&quot;: &quot;value2&quot;,..., &quot;keyN&quot;: &quot;valueN&quot;}</code></td>
</tr>
<tr>
<td>x-bluecoat-waf-monitor-details</td>
<td>See Table A–1, &quot;WAF Policy Gestures&quot; on page 29 for information on the block and monitor actions, as well as the version keyword.</td>
</tr>
<tr>
<td>x-bluecoat-request-details-header</td>
<td>Reports on the contents of the header, body, or both from the request.</td>
</tr>
<tr>
<td>x-bluecoat-request-details-body</td>
<td>These fields are disabled by default; to enable them, use the http.request.log_details() action as follows:</td>
</tr>
<tr>
<td></td>
<td><code>http.request.log_details[header] (yes)</code></td>
</tr>
<tr>
<td></td>
<td><code>http.request.log_details[body] (yes)</code></td>
</tr>
<tr>
<td></td>
<td><code>http.request.log_details[body, header] (yes)</code></td>
</tr>
<tr>
<td></td>
<td>When using these fields in conjunction with the &quot;http.request.detection.result.application_protection_set=&quot;and &quot;http.request.detection.result.validation=&quot; conditions, request header and request body fields can be populated according to WAF detections.</td>
</tr>
</tbody>
</table>
### Table 3–1  bcreporterwarp_v1 access log fields

<table>
<thead>
<tr>
<th>Access Log Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| x-bluecoat-waf-scan-info | Reports whether WAF engines processed the request (WAF_SCANNED) or not (WAF_SCAN_BYPASSED). WAF scanning might be bypassed if:  
  • no WAF policy applied to the request  
  • http.request.detection.bypass_cache_hit() property allowed WAF evaluation to be skipped for performance optimization |
Appendix A: WAF Policy Reference

SGOS 6.7.x includes the following CPL to support WAF.

<table>
<thead>
<tr>
<th>CPL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;define application_protection_set&quot;</td>
<td>Bind a specified label to a set of engines, and then specify the action to take on the defined set.</td>
</tr>
<tr>
<td>&quot;effective_date=&quot;</td>
<td>Control rule selection and usage based on the date that WAF rules were added.</td>
</tr>
<tr>
<td>&quot;http.csrf.authentication_link()&quot;</td>
<td>Specify the authentication details to be included in a Cross-Site Request Forgery (CSRF) token and is used in conjunction with &quot;http.csrf.detection()&quot; and &quot;http.csrf.token.insert()&quot;.</td>
</tr>
<tr>
<td>&quot;http.csrf.detection()&quot;</td>
<td>Validate a CSRF token in client HTTP POST requests, where http.csrf.token.insert is used. The specific type of validation information is determined in another policy gesture such as &quot;http.csrf.authentication_link()&quot;</td>
</tr>
<tr>
<td>&quot;http.csrf.token.insert()&quot;</td>
<td>Method to prevent CSRF attacks on WAF-protected web application servers. CSRF attacks are performed when a malicious site poses as a user who has recently authenticated on a web server, in order to act maliciously on that user’s behalf.</td>
</tr>
<tr>
<td>&quot;http.csrf.token.name()&quot;</td>
<td>(Added in 6.7.4) Set a custom CSRF token name. Use in conjunction with &quot;http.csrf.token.insert()&quot;.</td>
</tr>
<tr>
<td>&quot;http.request.body.data_type()&quot;</td>
<td>Evaluate or emulate HTTP content-type in the request body.</td>
</tr>
<tr>
<td>&quot;http.request.body.inspection_size()&quot;</td>
<td>Specify the maximum number of bytes of an HTTP request body that Web Application Firewall (WAF) content nature detection engines or policy will scan per transaction.</td>
</tr>
<tr>
<td>&quot;http.request.detection.bypass_cache_hit()&quot;</td>
<td>Bypass WAF scanning for requests whose response is served from the object cache on the appliance.</td>
</tr>
<tr>
<td>&quot;http.request.detection.exception()&quot;</td>
<td>Specify a built-in or user-defined exception message to return to the user when a WAF engine or property blocks a request.</td>
</tr>
<tr>
<td>&quot;http.request.detection.other()&quot;</td>
<td>Enables and defines settings for request validation in HTTP requests. The set of validation occurs after the URI path and all names and values are normalized in the query string, cookie, and body in JSON, URL-encoded and Multipart-Form-encoded formats.</td>
</tr>
</tbody>
</table>
Refer to the following sections for descriptions and usage tips.

**define application_protection_set**

Bind a specified label to a set of engines, and then specify the action to take on the defined set.

**Syntax**

```plaintext
define application_protection_set <label>
    engine=<engine_name> keyword=(property1, property2, ..)
    engine=<engine_name> keyword=property
    ..
end
http.request.detection.<label>{block|monitor|ignore}
```

where:

- **label** – Your custom name for the set.
- **engine_name** – The WAF engine.
- **keyword** – Optional specifiers:
  - **language** – One or more languages that the specified engine supports.
  - **host** – One or more operating systems.
  - **part** – The part of the HTTP request that the engine scans. See "Support for HTTP Attributes" on page 49.

### Table A–1 WAF Policy Gestures

<table>
<thead>
<tr>
<th>CPL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;http.request.detection.result.application_protection_set=&quot;</td>
<td>Allows you to define policy actions based on the results of WAF application protection advanced engine scanning decisions. When a WAF application protection scan rule results in a block or monitor result, you can use http.request.detection.result.application_protection_set= to perform an action such as additional logging to manually identify the content of the request.</td>
</tr>
<tr>
<td>&quot;http.request.detection.result.validation=&quot;</td>
<td>Allows you to define policy actions based on the results of WAF validation decisions. When a WAF validation rule results in a block or monitor result, use this condition to perform an action such as additional logging to manually identify the content of the request.</td>
</tr>
<tr>
<td>&quot;http.request.log_details<a href="">header, body</a>&quot;</td>
<td>Outputs the contents of the header, body, or both from an HTTP request to access log fields x-bluecoat-request-details-header and x-bluecoat-request-details-body.</td>
</tr>
<tr>
<td>&quot;http.request.normalization.default()&quot;</td>
<td>Per transaction, normalize only the value of the specified attribute(s) using the specified normalization function(s).</td>
</tr>
</tbody>
</table>
• **rule** – Use the rule keyword to enable or disable certain blacklist or analytics_filter engine rules. The rule keyword is not applicable to other engines. Rules have the following format:

   AF-####-# or BL-####-#

   In the preceding example, AF refers to analytics_filter and BL refers to blacklist. The ####-## represents the rule and sub-pattern identifiers. When the blacklist and analytics_filter engines flag a detection within a request, the matching rules are included in the block or monitor details access log field(s).

• **version** – Specify the command injection engine version:

   • 2 - The legacy version used in versions prior to 6.6.5.1. This version targets chained command sequences, and requires command-separation characters to be present in the payload to be effective.

   • 3 - The current default version. The command injection engine detects a wider set of attacks, including non-chained command injection payloads. Symantec recommends that you use this version.

- **property** – A supported property.
- **block** - Denies the request and logs the action.
- **monitor** - Allows the request and logs the action.
- **ignore** - Allows the request and does not log the action.

### Supported Engines, Keywords, and Properties

Refer to the following supported keywords and properties.

Table A–2 Supported keywords and properties for `define application_protection_set`

<table>
<thead>
<tr>
<th>Engine</th>
<th>CPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blacklist</td>
<td>engine=blacklist</td>
</tr>
<tr>
<td></td>
<td>rule=(BL-2000-0, BL-2000-1)</td>
</tr>
<tr>
<td>Analytics Filter</td>
<td>engine=analytics_filter</td>
</tr>
<tr>
<td></td>
<td>rule=(AF-1000-0, AF-1000-1)</td>
</tr>
<tr>
<td>SQL injection</td>
<td>engine=injection.sql</td>
</tr>
<tr>
<td>Cross-site scripting</td>
<td>engine=xss</td>
</tr>
<tr>
<td>Code injection</td>
<td>engine=injection.code</td>
</tr>
<tr>
<td></td>
<td>language=(java, php, ssi, javascript)</td>
</tr>
<tr>
<td>HTML injection</td>
<td>engine=injection.html</td>
</tr>
<tr>
<td>Directory traversal</td>
<td>engine=reference.directory_traversal</td>
</tr>
</tbody>
</table>

**Note:** You do not have to use parentheses when specifying a single value.
Layer and Transaction Notes

- Applies to proxy transactions.
- Only alphanumeric, underscore, dash, and slash characters can be used with the define action name.

See Also

- "Support for HTTP Attributes" on page 49

Example

```c
define application_protection_set engines
  engine=blacklist
  engine=analytics_filter
  engine=reference.directory_traversal
  engine=xss
  engine=injection.command host=(windows, osx)
  engine=injection.html
  engine=injection.sql
  engine=injection.code language=java
end
```

**effective_date=**

Specifies the set of WAF rules selected by the enclosing define application_protection_set definition. Symantec delivers WAF rule updates for the Blacklist and Analytics Filter engines through the Web Application Protection (WAP) subscription. The effective_date condition allows WAF administrators to control rule selection and usage based on the date the rules were added.

For example, rules qualified in a pre-production environment can be set to block-mode, while new rules can be set to monitor-mode. This enables an organization to take advantage of new rules immediately without introducing new false-positives that block legitimate requests.
After the new rules are qualified, `effective_date` can be migrated forward in the production environment, thereby setting the new rules into block-mode.

**Note:** This condition is only valid when a WAP subscription is present on the ProxySG appliance.

### Syntax

```
  effective_date=\textit{date\_range}
```

where:

- \textit{date\_range}—Specifies the date or a date range that determines rule set selection. If `effective_date` is not specified, all rules are selected.

### Layer and Transaction Notes

- Use in `<proxy>` layers.
- Applies to proxy transactions.

### See Also

- define application_protection_set

### Example

```
; Selects WAF rules added up to 2015-09-01.
define application_protection_set WAF_QualifiedRules
  engine=blacklist effective_date=..20150901
end
<proxy>
http.request.detection.WAF_QualifiedRules(block)

; Selects WAF rules added on 2015-09-02 or later.
define application_protection_set WAF_NewRules
  engine=blacklist effective_date=20150902..
end
<proxy>
http.request.detection.WAF_NewRules(monitor)

; Selects WAF rules added in 2015.
engine=blacklist effective_date=20150101..20151231
```

**http.csrf.authentication_link()**

This property is used to specify the authentication details to be included in a Cross-Site Request Forgery (CSRF) token and is used in conjunction with `http.csrf.token.insert` and `http.csrf.detection`. 
The information contained in a CSRF authentication token is encrypted and signed with a cryptographically-secure signature. An anti-CSRF token cannot be deciphered by an entity other than the appliance that generated it, or another appliance that has been configured with the same private key.

Anti-CSRF tokens are used to accurately identify subsequent POST requests as genuine, thereby preventing malicious actors from initiating unwanted actions on behalf of an authenticated user.

**Syntax**

```
http.csrf.authentication_link(userid)
http.csrf.authentication_link(client_ip)
http.csrf.authentication_link(userid,client_ip)
```

where:

- `userid`—The authenticated user’s username. This is the default.
- `client_ip`—The authenticated user’s IP address.
- `userid,client_ip`—The token will attempt to first use the `userid` as the identifier. If one is not present, the `client_ip` is used to identify the user in the authentication token.

**Layer and Transaction Notes**

- Use in `<proxy>` layers.
- When using this property with `client_ip`, make sure that the proxy generating the token can see the client’s actual IP address. Proxy deployments that use NAT, load-balancing, and proxy chain configurations can obscure a client’s original IP address.
- If client source IP addresses are obfuscated, but requests that reach the proxy include an `X-Forwarded-For` header from the child proxy in a proxy chain, you can use the CPL gesture `client.effective_address` to force the Proxy to use the appropriate client IP.

**See Also**

- `http.csrf.token.insert()`
- `http.csrf.detection()`

**Example**

Insert and validate a CSRF authentication token with a lifespan of 1200 seconds that attempts to use the `userid` to identify the user. If that is not available, the IP address is used. If a valid token is not found, block requests that fail CSRF validation:

```
<proxy>
  authenticate(WindowsIWARealm) authenticate.force(yes)
  authenticate.mode(auto)
<proxy>
http.csrf.token.insert(1200) \nhttp.csrf.authentication_link(userid,client_ip)
<proxy>
http.csrf.detection(block)
```
**http.csrf.detection()**

This property is used to validate a Cross-Site Request Forgery (CSRF) token in client HTTP POST requests, where http.csrf.token.insert is used. The specific type of validation information is determined in another policy gesture such as http.csrf.authentication_link.

When used in policy, only form-type HTTP POST requests (either URL-encoded or multipart-form) will be examined for CSRF tokens. CSRF tokens are defined with the http.csrf.token.insert property.

**Syntax**

```
http.csrf.detection(block|monitor|ignore)
```

where:

- **block**— The proxy will validate CSRF tokens wherever possible. If a valid token is not found, the request is denied and reported in the `x-bluecoat-waf-block-details` and `x-bluecoat-waf-attack-details` access log fields.

- **monitor**— The proxy will validate CSRF tokens wherever possible. If a token is found to be invalid, it will be reported in the `x-bluecoat-waf-monitor-details` and `x-bluecoat-waf-attack-family` access log fields.

- **ignore**— While CSRF tokens may or may not be present, no action will be taken. This has the same impact as not having this property in policy and is used for false positive mitigation.

**Layer and Transaction Notes**

- Use in `<proxy>` layers.

**See Also**

- http.csrf.authentication_link()
- http.csrf.token.insert()

**Example**

Using a CSRF authentication token with a lifespan of 1200 seconds that uses the userid to identify the client in their next HTTP POST request, block requests that fail CSRF validation:

```xml
<proxy>
  authenticate(WindowsIWArealm) authenticate.force(yes)
  authenticate.mode(auto)
<proxy>
<proxy>
  http.csrf.token.insert(1200) \
  http.csrf.authentication_link(userid)
<proxy>
  http.csrf.detection(block)
```
**http.csrf.token.insert()**

This property is used as a method to prevent Cross-Site Request Forgery (CSRF) attacks on WAF-protected web application servers. CSRF attacks are performed when a malicious site poses as a user who has recently authenticated on a web server, in order to act maliciously on that user’s behalf.

The anti-CSRF token inserted with this property includes data that the proxy uses to identify the user after they have authenticated against the web server. After the initial authentication exchange, a user’s subsequent HTTP POST request includes a token that identifies them. The proxy validates the token. If it is valid the user is permitted to access the web server. If the token is invalid or missing, the request may be blocked or logged, depending on the configuration of `http.csrf.detection()` in policy.

Previous releases of SGOS include CSRF protection in the form of referer header checking and CAPTCHAs. These methods are only suitable in limited cases.

**Syntax**

```plaintext
http.csrf.token.insert(yes|no|n)
```

where:

- **yes**—The proxy adds content to the response from the WAF-protected Web Application server, and an anti-CSRF token is inserted in subsequent client POST requests.
- **no**— The proxy will not add content to the response from the WAF-protected web application server and no anti-CSRF token will be inserted in subsequent requests.
- **n**— The time, in seconds, for which the token is valid. The time value takes the place of `(yes)` in policy. If no time value is specified, the anti-CSRF token is valid for 900 seconds.

**Layer and Transaction Notes**

- Use in `<proxy>` layers.

**See Also**

- `http.csrf.authentication_link()`
- `http.csrf.detection()`
- `http.csrf.token.name()`

**Example**

Using a CSRF token with a lifespan of 1200 seconds that uses the client’s authenticated userid to identify the client in their next HTTP POST request, block requests that fail CSRF validation:

```plaintext
<proxy>
  authenticate(WindowsIWArealm) authenticate.force(yes)
  authenticate.mode(auto)
<proxy>
  http.csrf.token.insert(1200)
  http.csrf.authentication_link(userid)
```
Appendix A: WAF Policy Reference

http.csrf.token.name()

(Introduced in version 6.7.4) Use this property in conjunction with other CSRF-related policy in WAF deployments. The anti-CSRF token that is inserted using http.csrf.token.insert() has a default name of CSRF-Token; thus, by specifying a custom token for legitimate requests, you can make it difficult for malicious users to determine the name of the token and identify the WAF solution in your deployment.

When a custom anti-CSRF token is set in CPL, you can view the source of a browser form to verify that it shows var antiCsrfTokenName = "<custom_token_name>".

A suspected CSRF attack is written to access logs when the expected token name—whether that is the default CSRF-Token or the custom name—is not present. Make sure that the access log format includes the x-bluecoat-waf-monitor-details and x-bluecoat-waf-attack-family fields.

Syntax

http.csrf.token.name(<string>)

where string is a custom token name.

Layer and Transaction Notes

- Use in <proxy> layers.
- Applies to HTTP POST transactions.

See Also

- http.csrf.authentication_link()
- http.csrf.detection()
- http.csrf.token.insert()

Example

; for the given form, insert anti-CSRF token valid for 10800 seconds ; validate CSRF tokens and log "token missing" if no tokens detected <proxy>
url.domain=bankform.com http.csrf.token.insert(10800) \ 
    http.csrf.detection(monitor)

; include the client IP from initial auth with the token details <proxy>
    http.csrf.authentication_link(client_ip)

; use the specified custom name for bankform.com <proxy>
    http.csrf.token.name(ccsrf-46536-bf-va78)
**http.request.body.data_type()**

This property allows an administrator to configure the ProxySG appliance to either evaluate and trust the content-type header, or treat the request as if a specific content-type was specified, regardless of the actual value in the request.

**Syntax**

```plaintext
http.request.body.data_type(auto|json|multipartform|none|webform|xml)
```

where:

- **auto** (default)—Evaluate content-type, and resolve to one of the following:
  - multipartform
  - xml
  - url encoded
  - json
  - other (none)
- **json**—Act as if content-type: json is specified
- **multipartform**—Act as if multipart-form content-type is specified
- **none**—Act as if no content-type is present
- **webform**—Act as if application/x-www-form-url encoded is specified
- **xml**—Act as if body content-type is XML

**Layer and Transaction Notes**

- Use in `<proxy>` layer.
- Applies to all HTTP transactions.

**See Also**

- Refer to the *Content Policy Language Reference* for related gestures, such as `define xml_schema-type_schema`

**Example**

```plaintext
; Act as if body content-type is XML
<proxy>
  http.request.body.data_type(xml)
```

**http.request.body.inspection_size()**

Specify the maximum number of bytes of an HTTP request body that Web Application Firewall (WAF) content nature detection engines or policy can scan per transaction. The maximum is 65536 bytes. If the request body is chunk-encoded or compressed and the request data is a type for which WAF engines are enabled, the ProxySG appliance unchunks and decompresses the data.
Appendix A: WAF Policy Reference

For example, if the request body is chunked and includes SQL statements, and the SQL injection engine is enabled, the appliance unchunks the data. The `http.request.body.inspection_size()` property considers the size of the unchunked data.

While WAF engines scan the HTTP request body, contact to the origin content server (OCS) does not occur until the engines scan the specified amount of request body bytes or reach the end of the HTTP request body. Because the OCS connection could be delayed, this could lead to increased latency of the transaction.

If the `http.request.data.N=` condition is already specified in policy, the WAF engines scan up to the number of bytes specified in the condition and you do not have to use the `http.request.body.inspection_size()` property. If policy includes both `http.request.data.N=` and `http.request.body.inspection_size()`, WAF engines use the greatest value specified for scanning.

The `http.request.body.inspection_size()` property can be combined with the `http.request.detection.other.threshold_exceeded()` property to control WAF behavior when a large request is processed.

**Syntax**

```
http.request.body.inspection_size(N)
```

where \( N \) specifies the maximum number of bytes in the HTTP request body to scan. The maximum is 65536 bytes.

**Layer and Transaction Notes**

- Use in `<cache>` and `<proxy>` layers.
- Applies to all HTTP requests that have a request body.

**See Also**

- `http.request.data.N=`
- `http.request.detection.other.threshold_exceeded()`

**Example**

WAF engines will scan up to 50000 bytes of an HTTP request body and block any requests larger than this value.

```
<proxy><cache>
http.request.body.inspection_size(50000) \nhttp.request.detection.other.threshold_exceeded(block)
```

**http.request.detection.bypass_cache_hit()**

To improve performance, you can bypass Web Application Firewall (WAF) scanning for requests whose response is served from the object cache on the appliance.

If policy includes this property, the `x-bluecoat-waf-scan-info` field in the `bcreporterwarp_v1` access log format indicates if WAF processing is intentionally skipped due to cache hit optimization being bypassed:

- If WAF engines scan a transaction, the field reports `WAF_SCANNED`. 
If WAF evaluation does not occur due to the presence of the `http.request.detection.bypass_cache_hit(yes)` property or the absence of WAF policy, the field reports `WAF_SCAN_BYPASSED`.

If no WAF policy is present, the field reports `WAF_DISABLED`.

The WAF is designed to protect backend web application servers in a reverse proxy deployment. If the object cache can serve the response for a given request, the request is not required to make the round trip between the appliance and origin content server (OCS).

In reverse proxy deployments with a high cache-hit ratio, enabling this property can have a significant positive impact on performance.

The `http.request.detection.bypass_cache_hit()` property allows you to optimize for either performance or security; thus, it is important to understand the trade-offs and implications of using this feature. A misconfigured web application server, which allows caching via the `cache-control` response header on dynamic responses, causes the appliance to store responses in the object cache. If a protected web application returns cacheable dynamic responses while this property is enabled, the appliance could serve malicious responses from cache. To address this potential issue, do one of the following:

- Correct the cache directives that the web application server served.
- Use the `rewrite()` property in a `define action` block to rewrite the `cache-control` header, as follows:
  
  ```
  rewrite(response.header.cache-control, "regex_pattern", "string")
  ```
- Use the `bypass_cache(no)` property to bypass caching for specific requests or `cache(no)` to avoid storing specific responses in the cache.
- Clear the object cache with the `#clear-cache object-cache` CLI command. For maximum security, run this command before using `http.request.detection.bypass_cache_hit()` in an existing deployment, as well as whenever WAF policy or content from the Application Protection subscription is updated.

**Note:** To determine when the Application Protection subscription was last updated (or, if Notify Only is enabled, a newer database is available for download), issue the `#(config application-protection)view` command and check the Last successful download details.

Refer to the *Command Line Interface Reference* for details on the `#clear-cache object-cache` and `#(config application-protection)view` commands.

**Syntax**

`http.request.detection.bypass_cache_hit(yes|no)`

where:

- `yes` means WAF engines do not scan requests that result in a cache hit.
- `no` means that WAF engines scan all requests; this is the default behavior.
**Layer and Transaction Notes**

- Use in `<proxy>` layers.
- Applies to all proxy transactions.

**See Also**

- `#clear-cache object-cache` CLI command
- `#(config application-protection)view` CLI command

**Example**

```
; Avoid WAF scanning for all requests to www.bluecoat.com
; That result in an object cache hit
<proxy>
    url.domain="www.bluecoat.com" /
        http.request.detection.bypass_cache_hit(yes)
```

### http.request.detection.exception()

Specify a built-in or user-defined exception message to return to the user when a WAF engine or property blocks a request. If this property is not specified, the default message for a blocked request is `invalid_request`.

To set a WAF exception, use `define application_protection_set` with the corresponding `http.request.detection` block action in a rule.

**Note:** The `exception()` property does not override this property for WAF block exceptions and this property does not override the `exception()` property for standard exceptions.

**Syntax**

```
http.request.detection.exception(exception_id, details, format_string)
```

where:

- `exception_id` is one of the following:
  - A built-in exception such as `silent_denied`
  - A user-defined exception in the form `user_defined.exception_id`

- `details` is a text string, enclosed within quotation marks, for the exception message. The message is substituted for `$(exception.details)` when the exception occurs.

- `format_string` is text defined with `define string` and substituted for `$(exception.format)`. The named string overrides the format field of the exception and can include substitutions.

**Layer and Transaction Notes**

- Use in `<proxy>` layer.
- Applies to all client-initiated HTTP transactions.
See Also

define application_protection_set

Example

; block requests for the XSS engine
; return a gateway error when XSS engine is triggered
; and return a policy_denied error for other responses
; when HTTP status code is 200

define application_protection_set EngConf
gen=xxs
end

define string ErrorMessage
; define an HTML page for the error message
><html>
><head>
><title>Gateway Error</title>
><meta http-equiv=refresh content="10;$(url)">
></head>
><body>
>Cannot complete your request.
></body>
></html>
end

<proxy>
http.request.detection.EngConf(block)
</proxy>

<proxy>
http.request.detection.exception(gateway_error, "Gateway Error", ErrorMessage)
</proxy>

<proxy>
http.response.code=200 exception(policy_denied)

http.request.detection.other()

Enables and defines settings for request validation in HTTP requests. The set of validation occurs after the URI path and all names and values are normalized in the query string, cookie, and body in JSON, URL-encoded and Multipart-Form-encoded formats.

Syntax

http.request.detection.other.[attribute]{block|monitor|ignore}

where

attribute is one of the following:
• null_bytes - Detects content that contains null bytes.
• invalid_form_data - Detects invalid Multipart-Form evasion techniques.
• invalid_json - Detects invalid JSON data in the request body.
• parameter_pollution - Detects multiple instances of parameters with the same name.
• parameter_pollution_separator("char") - When multiple instances of parameters with the same name exist, the values are concatenated and separated by the specified character. For example, when you specify a comma (",") as the separator, field=1&field=2 are concatenated to field=1,2 before WAF engines analyze the values.

Note:  To preserve WAF engine effectiveness in detecting threats:
- Use this attribute only in environments where the backend system supports parameter concatenation.
- Specify a useful separator character; typically, backend systems (such as ASP) support the comma as a string separator.

• multiple_encoding - Detects request data encoded more times than what is specified in "http.request.normalization.default()".
• invalid_encoding - Detects invalid UTF-8 encoding in full and half Unicode. Applies only to normalizations specified in the http.request.normalization.default() property.
• multiple_header - Detects headers declared more than once, with the same name, in a request. Does not consider any legitimate, known headers that are allowed to appear multiple times.
• threshold_exceeded - Triggers when the http.request.body.inspection_size() limit is reached.

- block - Denies the request and logs the action.
- monitor - Allows the request and logs the action.
- ignore - Allows the request and does not log the action.

Layer and Transaction Notes
- Use in <proxy> layer.
- Applies to all HTTP transactions.

See Also
- http.request.normalization.default().
- http.request.body.inspection_size()

Example
For an example, see "Add WAF Policy" on page 18.
**http.request.detection.result.application_protection_set=**

Allows you to define policy actions based on the results of WAF application protection advanced engine scanning decisions. When a WAF application protection scan rule results in a block or monitor result, you can use `http.request.detection.result.application_protection_set=` to perform an action such as additional logging to manually identify the content of the request.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

**Syntax**

```
http.request.detection.result.application_protection_set=[block|monitor]
```

**Layer and Transaction Notes**

- Use in `<proxy>` layers.
- Applies to all transactions that have already been processed by a WAF application protection scan rule.

**See Also**

- `define application_protection_set`
- `http.request.detect.result.validation=`
- `http.request.log_details[header, body] (yes|no)`
- `http.request.body.inspection_size()`

**Example**

When a WAF application protection set action results in a block or monitor result, log the full header and body:

```verbatim
define application_protection_set SecurityEngines
  engine=Injection.sql
  engine=xss
end

<proxy>
  http.request.detection.SecurityEngines (monitor)
</proxy>
```

**http.request.detection.result.validation=**

Allows you to define policy actions based on the results of WAF validation decisions. When a WAF validation rule results in a block or monitor result, you can use `http.request.detection.result.validation=` to perform an action such as additional logging to manually identify the content of the request.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

**Syntax**

```
http.request.detection.result.validation=[block|monitor]
```
Layer and Transaction Notes

- Use in `<proxy>` layers.
- Applies to all transactions that have already been processed by a WAF validation rule:
  - `http.request.detection.other.null_byte(monitor|block)`
  - `http.request.detection.other.invalid_encoding(monitor|block)`
  - `http.request.detection.other.invalid_form_data(monitor|block)`
  - `http.request.detection.other.invalid_json(monitor|block)`
  - `http.request.detection.other.multiple_encoding(monitor|block)`
  - `http.request.detection.other.multiple_header(monitor|block)`
  - `http.request.detection.other.threshold_exceeded(monitor|block)`

See Also

- `http.request.detect.result.application_protection_set=`
- `http.request.log_details[header, body](yes|no)`
- `http.request.body.inspection_size()`

Example

When a WAF validation results in a block or monitor result, log the full header and body:

```xml
<proxy>
  http.request.normalization.default(auto)
<proxy>
  http.request.detection.other.invalid_form_data(monitor)
<proxy>
  http.request.detect.result.validation=(monitor||block) \ 
    http.request.log_details[header, body] (yes)
```

`http.request.log_details[header, body]()`

Outputs the contents of the header, body, or both from an HTTP request to access log fields `x-bluecoat-request-details-header` and `x-bluecoat-request-details-body`.

In a WAF deployment, administrators can use this information to validate detections, in an effort to rule out false-positives.

In a standard forward proxy deployment, administrators can use this information to log the data users are transmitting to the Internet.

Syntax

`http.request.log_details[header, body](yes|no)`

where:

- header—Outputs the HTTP headers from user requests to `x-bluecoat-request-details-header`.
- body—Outputs the HTTP body from user requests to `x-bluecoat-request-details-body`.

By default, only the first 8 kB of body contents are logged. You can increase this with either of the following policy gestures:
- http.request.data=
- http.request.body.inspection_size()

- yes—Enables the access log fields, x-bluecoat-request-details-header and x-bluecoat-request-details-body in the bcreporterwarp_v1 access log format, and outputs the contents of request body, header, or both.

- no—The default state for this property. When set to no, the bcreporterwarp_v1 access log format does not include the additional access log fields.

Layer and Transaction Notes
- Use in <proxy> layer.

See Also
- http.request.body.inspection_size()
- http.request.data=
- http.request.detection.result.application_protection_set=
- http.request.detection.result.validation=

Example
Always log header contents, and conditionally log the full body when a WAF detection occurs:
<proxy>
  http.request.normalization.default(auto)
<proxy>
  http.request.detection.other.null_byte(monitor)
<proxy>
  http.request.log_details[header](yes)
<proxy>
  http.request.detection.result.validation=(monitor||block) \  
  http.request.log_details[body](yes)
  http.request.detection.result.application_protection_set=(monitor||block) \  
  http.request.log_details[body](yes)

http.request.normalization.default()
Per transaction, normalize only the value of specified attribute(s) using the specified normalization function(s).

Syntax
http.request.normalization.default("function: {attribute,..}")

where:
- auto - Provides the recommended normalization settings. See "Example" on page 47 for more information.
- function - Accepts one or more of the following:
  - urlDecode - Decode URL-encoded strings.
  - htmlEntityDecode - Decode characters encoded as HTML entities.
Appendix A: WAF Policy Reference

- **urlDecodeUni** - Same as `urlDecode`, with support for Microsoft-specific %u encoding.
- **cmdLine** - Delete or replace characters used as Windows and Linux command line escape characters.
- **cssDecode** - Decode characters encoded using CSS 2.0 escape characters.
- **jsDecode** - Decode JavaScript escape sequences.
- **compressWhiteSpace** - Convert whitespace characters (0x20, \f, \t, \n, \r, \v, 0xa0) to spaces (ASCII 0x20) and compress multiple consecutive space characters into one.
- **lowercase** - Convert all characters to lowercase. You cannot use this with the .case_sensitive modifier.
- **normalizePath** - Remove multiple slashes, directory self-references, and directory back-references (except when at the beginning of the input).
- **normalizePathWin** - Same as `normalizePath`, but first convert backslash characters to forward slashes.
- **removeWhitespace** - Delete all whitespace characters.
- **removeNulls** - Remove all null bytes.
- **replaceComments** - Replace each occurrence of a C-style comment /* ... */ with a single space (multiple consecutive occurrences of zre not compressed).
- **utf8toUnicode** - Convert all UTF-8 characters sequences to Unicode. This normalization function is not applied on POST body data if the content-type is unknown or not specified.
- **base64Decode** - Decode partial and complete Base64-encoded strings.
- **trimDecode** - Delete whitespace prefixes and suffixes.

**attribute** - One or more of the values/names described in "Support for HTTP Attributes" on page 49.

Layer and Transaction Notes

- Use in `<proxy>` layer.

See Also

- "Support for HTTP Attributes" on page 49

Example

Symantec recommends that you specify this property as follows:

```xml
<proxy>
  http.request.normalization.default(auto)
</proxy>
```

The `auto` option expands to the following normalization setting:

```xml
http.request.normalization.default("urlDecode:(path),urlDecode:jsDecode:htmlEntityDecode:trimDecode:(header_name,header,cookie_name,cookie),urlDecode:urlDecode:jsDecode:htmlEntityDecode:utf8toUnicode:trimDecode:(arg_name,arg)")
```
Appendix B: Support for HTTP Attributes

The `http.request.normalization.default()` property and define `application.protection_set` block support the following attributes.

Table B–1  Supported HTTP Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>All argument names found in the URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.</td>
</tr>
<tr>
<td>value</td>
<td>All named and unnamed argument values found in URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.</td>
</tr>
<tr>
<td>query_arg_name</td>
<td>All argument names found in the URL query string.</td>
</tr>
<tr>
<td>query_arg</td>
<td>All named and unnamed argument values found in the URL query string.</td>
</tr>
<tr>
<td>arg_name</td>
<td>All argument names found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).</td>
</tr>
<tr>
<td>arg</td>
<td>All named and unnamed argument values found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).</td>
</tr>
<tr>
<td>cookie_name</td>
<td>All argument names found in all Cookie and Cookie2 headers.</td>
</tr>
<tr>
<td>cookie</td>
<td>All named and unnamed argument values found in all Cookie and Cookie2 headers.</td>
</tr>
<tr>
<td>post_arg_name</td>
<td>All argument names found in the post body (URL-encoded and Multipart-form encoded formats).</td>
</tr>
<tr>
<td>post_arg</td>
<td>All named and unnamed argument values found in the post body (URL-encoded and Multipart-form encoded formats).</td>
</tr>
<tr>
<td>header_name</td>
<td>All header names.</td>
</tr>
<tr>
<td>header</td>
<td>All header values.</td>
</tr>
<tr>
<td>path</td>
<td>Path of the URL. This attribute does not have name=value format.</td>
</tr>
</tbody>
</table>

In addition, the following existing conditions also support the attributes:

- `http.request[attribute_name].modifier=`