VMware NSX Advanced Load Balancer WAF Guide

VMware NSX Advanced Load Balancer 21.1.4

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Guide

The VMware NSX Advanced Load Balancer WAF guide helps you understand how to configure and deploy the Web Application Firewall (WAF) in NSX Advanced Load Balancer for security enforcement and intelligence.

You will also understand the following from the guide:

- How to use WAF protection and apply basic settings for WAF functionality for virtual services.
- Enable application learning for a virtual service.
- Understand and deploy WAF Signatures.
- Best practices that you can follow to implement WAF.
- Methods to leverage the log analytics of WAF events based on historical trend information.
- Real-time visibility into ongoing operations, application behavior analysis, and attack patterns.

Overview

1

This section explains about the NSX Advanced Load Balancer Web Application Firewall.

Web application firewalls (WAFs) are intended to protect HTTP and HTTPS applications from attacks and proactively prevent threats.

Traditional web application security solutions do not provide visibility and security insights that administrators can use to create an effective application security posture. Enterprises need real-time visibility into application traffic, user experience, security and threat landscape, and application performance to identify and protect against the most sophisticated attacks.

This chapter includes the following topics:

WAF Features

The NSX Advanced Load Balancer leverages software-defined architecture and its strategic location on the network to gain real-time application insights. The built-in WAF solution provides application security and networking teams with an elastic and analytics-driven solution, that scales and simplifies policy customization and administration through central management.

The NSX Advanced Load Balancer WAF plays an integral role in a defense-in-depth strategy that performs comprehensive threat analysis, mitigates risk, provides zero-day protection against unpublished exploits and optimizes application security.

WAF gives administrators an important point of security enforcement and intelligence. WAF protects web applications from common vulnerabilities identified by Open Web Application Security Project (OWASP), such as SQL Injection (SQLi) and Cross-site Scripting (XSS), while providing the ability to customize the rule set for each application.

WAF analyzes the security rules that match a particular transaction and provides that insight in real-time as applications and attack patterns are learned. This application intelligence, paired with intuitive one-click rule customization, allows WAF to sharply reduce false positives.

Core Security	Threat Detection	Application Protection
OWASP Top 10 attack protection, including HTTP validation, injection, data leakage protection, automated attack blocking, and application-specific security.	Accept-Listing rules that allow bypassing WAF for certain request properties. For example, to allow the DAST scanner IPs from WAF inspection, to exclude internal IP addresses from WAF inspection or to bypass WAF for all POST requests.	Learning mode for application behavior and structure helps profile applications, inform decisions and automatically create positive security rules.
Guided false positive mitigation with customizable paranoia levels that control the strictness of the policy based on the logs and analytics.	Positive security for allowed application behavior to block anomalies. Positive model engine is called before the signature engine, reducing false positives and the time required to reach a decision about the validity of the request.	Per-app deployment for precision protection of specific applications with different security policy levels while ensuring application performance.
Rate-limiting per app to limit L3/L4 and L7 traffic based on parameters such as Client IP, URL and Path.	Signatures protection against known threats through a denylist approach by analyzing every part of the incoming and outgoing requests against SQLi, XSS and other threats based on Core Rule Set (CRS).	On-demand autoscaling to elastically scale the number of WAF instances and application servers to handle unpredictable traffic without impacting performance.
Point-and-click policy with central control and ease of use by enabling users to create custom policies quickly and efficiently.		Application analytics for WAF events based on historical trend information and real-time visibility into ongoing operations, application behavior analysis, and attack patterns.
RBAC support to control write access to WAF profiles and policies and read access to applications, pools, and clouds.		

Feature List

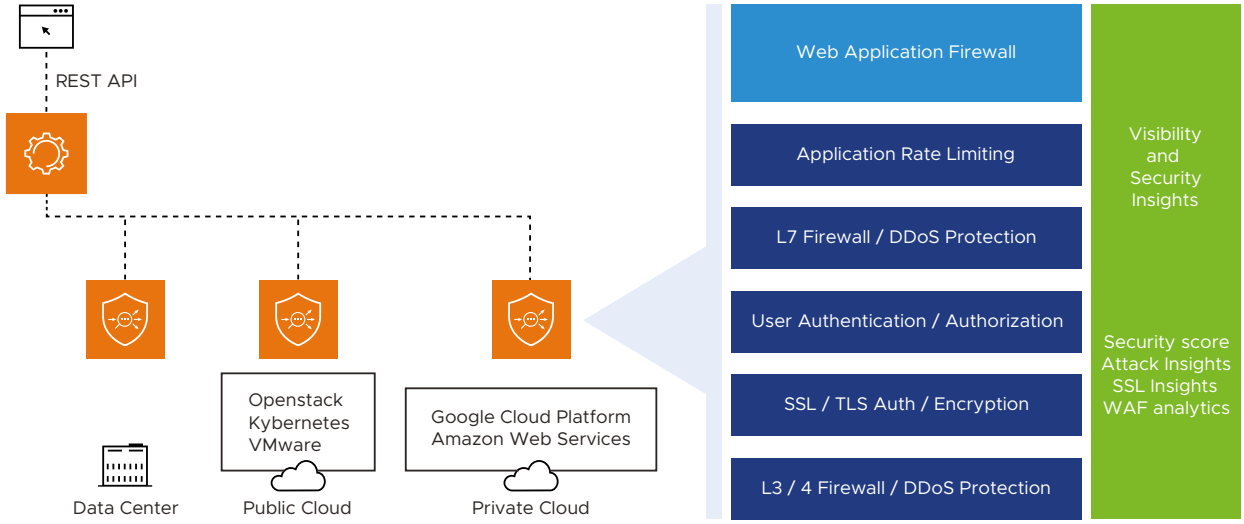
The following sections describe WAF features under three broad categories:

- Web Security and Application Attack Protection
- Administration and Configuration
- Logs and analytics

For a concise list of all WAF features, see What are the features provided as part of WAF ? in the FAQ section.

Web Security and Application Attack Protection

WAF provides a full application security stack to ensure web security and protection against application threats.



- OWASP Top 10 attack protection, including HTTP validation, injection, data leakage protection, automated attack blocking, and application-specific security.
- Positive security rules check the application traffic for allowed application behavior and block anomalous behavior. This engine is called before the signature engine, thereby reducing false positives and allowing for quicker decision to be taken about the validity of the request.
- Application behavior and structure is learned and can later be reused for other analytics and to automatically create positive security rules through Application Learning.
- The Allowlist defines rules for bypassing of WAF for some given request properties. For example, to allow the DAST scanner IPs from WAF inspection, to exclude internal IP addresses from WAF inspection or to bypass WAF for all POST request to /upload.php, and so on.
- IP geolocation
- HTTP RFC compliance
- File upload scanning
- DAST import
- Scripting for application logic flaws
- API protection for JSON XML.
- In addition to the protection of traditional HTTP Applications, WAF can also protect APIs or AJAX applications, by analyzing JSON or XML payloads.
- Support for multiple CRS versions
- Rate-limiting per application
- Brute force attack protection
- Basic DDOS protection
- HTTP Security policies

- L3-L7 security rules including ACLs

Administration and Configuration

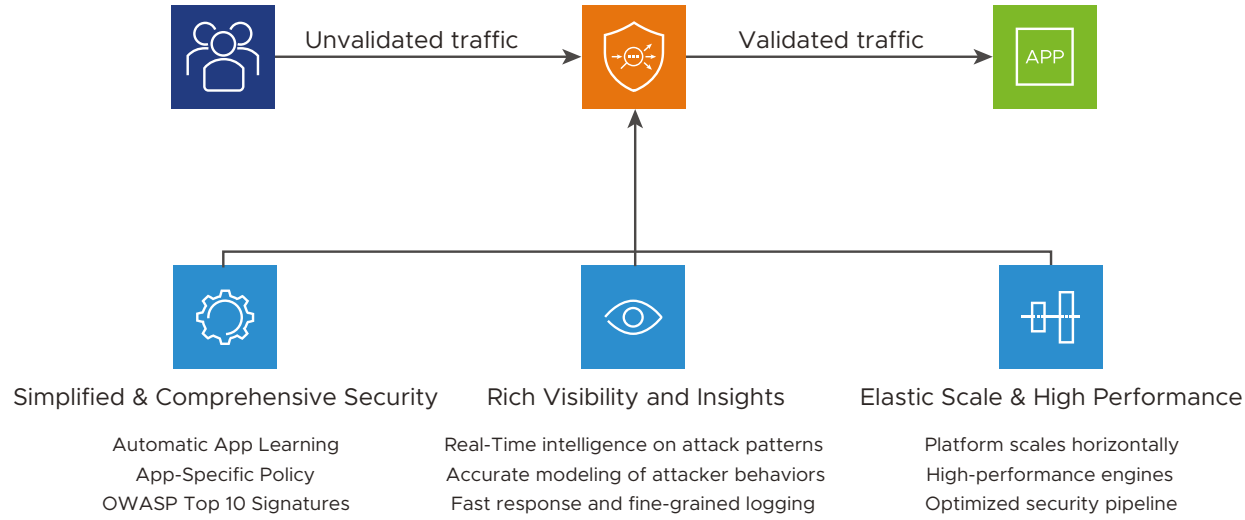
- WAF admin RBAC support provides granular Role-Based Access Control (RBAC). Users can have write access to WAF Profiles and Policies and read access to application virtual services, pools, clouds and so on.
- SSH and Web GUI access for administration.
- The NSX Advanced Load Balancer platform is 100% REST API based. So, deployment can be fully automated and all functionalities can be included into a CI-CD pipeline.
- Customizable error codes and error pages.
- Per-app deployment for precision protection of specific applications with different security policy levels while ensuring application performance.
- Easy to deploy rules.
- Easy to create custom rules which can be added for the application-specific use cases or any other custom requirement that might arise.
- Point-and-click simplicity for policies with central control.
- Elastic scale with highly performant, automatic scale-out architecture.
- Easy to deploy exclusions.
- Guided false positive mitigation with customizable paranoia levels that control the strictness of the policy based on the logs and analytics.
- Built-in event and alert mechanisms.
- SNMP support.
- NSX Advanced Load Balancer Cloud Services provide live security threat updates, such as IP reputation, signatures and more updates are sourced from industry-leading threat analysis companies and curated through the NSX Advanced Load Balancer Cloud Services.

Logs and analytics

- Application analytics for WAF events based on historical trend information and real-time visibility into ongoing operations, application behavior analysis, and attack patterns.
  - Granular security insights on traffic flows and rule matches to create precise and custom policies.
- Comprehensive log collection that includes pinpoint analysis of all security incidents that were blocked by WAF.

Architecture

The NSX Advanced Load Balancer WAF is built on the core design principles shown below to ensure that WAF is a simple yet comprehensive security solution.



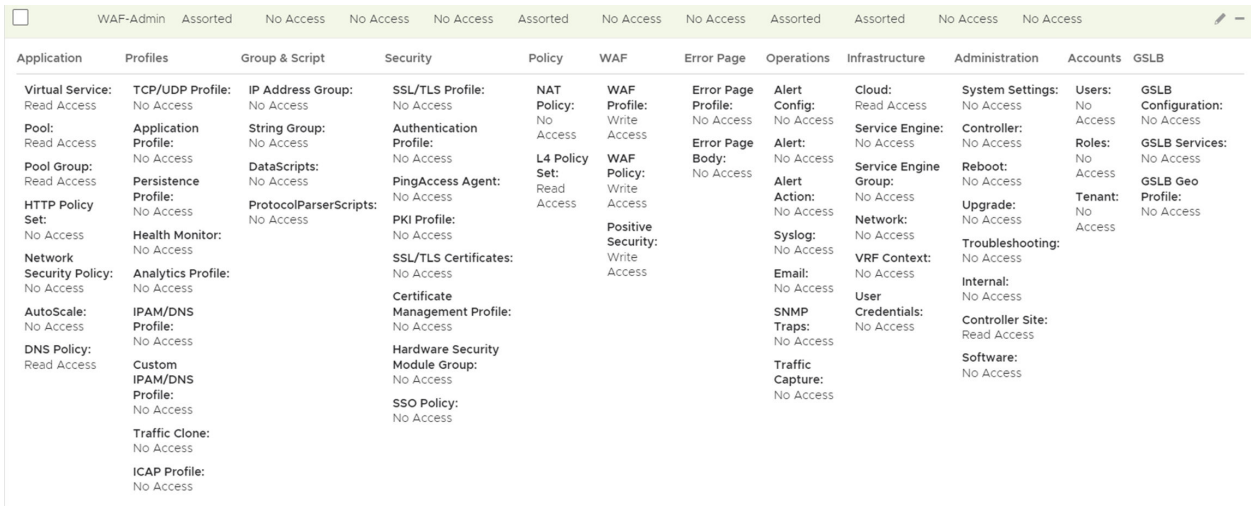
For information on WAF Process Flow, see What is WAF processing flow? in the FAQ section.

WAF Administrator Role

WAF administrator role assigns users specific access to several components in the NSX Advanced Load Balancer. This role differentiates access rights between the security team and other administrators. With this role, the team can independently check the security status and implement policy changes.

The WAF administrator role provides read access to essential components such as virtual service, pool, and pool groups. Components such as WAF Profile and WAF Policy are provided with write access.

To locate the WAF administrator role, navigate to **Administration > Accounts > Roles**. **WAF-admin** defines role access for all components as shown below:



Configuring WAF

2

This section discusses how to configure Web Application Firewall (WAF) on the NSX Advanced Load Balancer.

This chapter includes the following topics:

Enabling WAF on Virtual Server

Each virtual service can have one WAF Policy attached to it. This topic details the steps to configure WAF for a virtual service.

To add a WAF Policy to an existing virtual service:

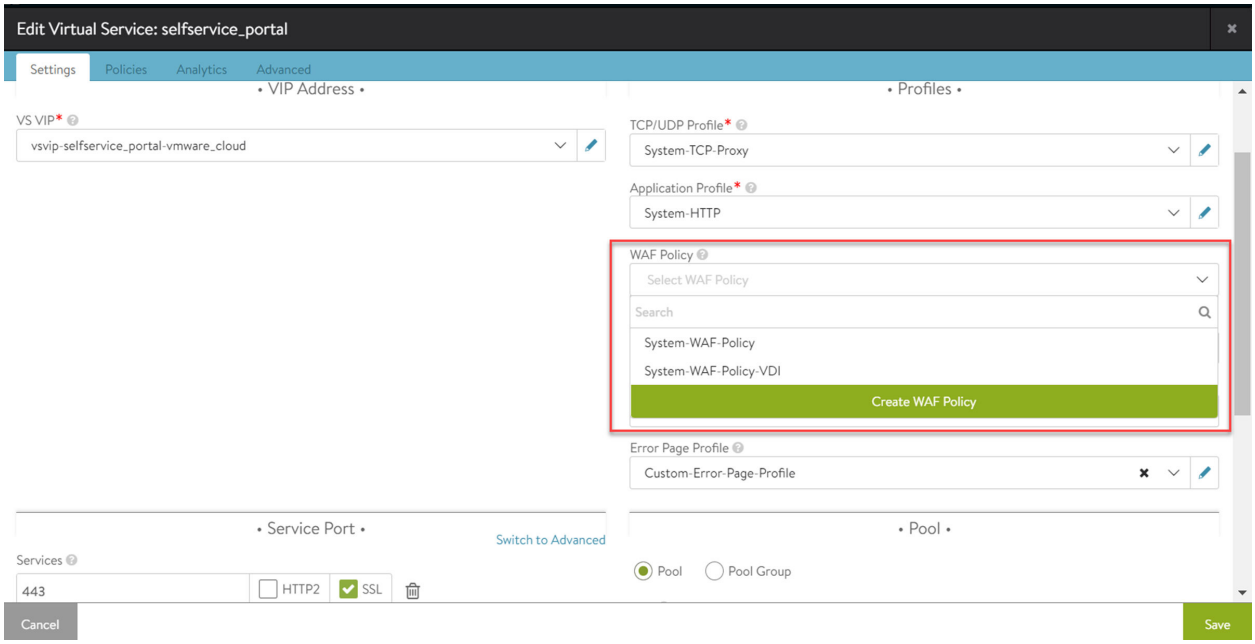
1 Navigate to **Applications > Virtual Services**.

2 Select the virtual service and click **edit** icon.

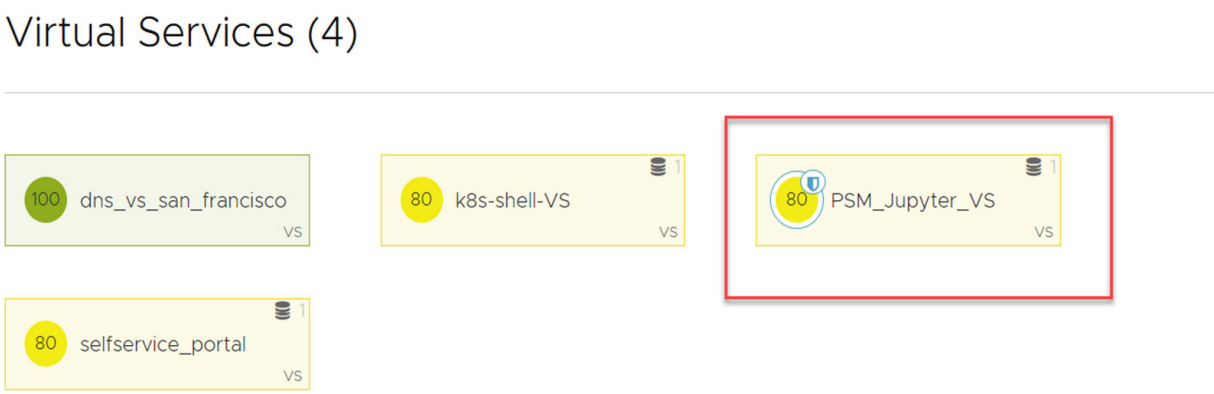
3 In the **Profiles** section, select an existing policy from the **WAF Policy** drop-down menu or create a new one.

4 **Save** the configuration.

Caution Attaching a WAF Policy to a virtual service immediately puts that policy into effect. If the policy is in Enforcement mode, it starts blocking requests. For new applications and untuned WAF policies, we recommend running the policy initially only in Detection mode.



Navigate to **Applications > Dashboard** to verify if WAF is enabled on the virtual service. If enabled, you will notice a halo and a shield on the attached virtual service object.



WAF Profile

A WAF Profile contains the settings for WAF functionality and is attached to a WAF Policy.

The WAF Profile is referenced by a WAF Policy and is defined for a specific set of applications that can be easily reused by multiple WAF Policies and virtual services.

The following are a few possible examples for WAF Profiles:

- An **Application Java Profile** that contains all the necessary elements for a specific type of Java application.

- An **Application PHP Profile** that contains all the necessary elements for a specific type of PHP application.
- An **API Profile** that contains configuration for API traffic.

Configuring WAF Profile

This section details the steps for configuring WAF Profile.

Note

- Navigate to **Templates > WAF > WAF Profile** to locate the default profile.
- `System-WAF-Profile` is the default profile that contains most commonly used web application settings served through a virtual service.
- For customizing a profile, it is highly recommended to create a new profile instead of editing the default profile `(System-WAF-Profile)`.

To create a new profile:

1 Navigate to **Templates > WAF > WAF Profile**.

2 Click **Create**.

3 Configure the new WAF Profile under the following tabs:

   a General

   b Default Actions

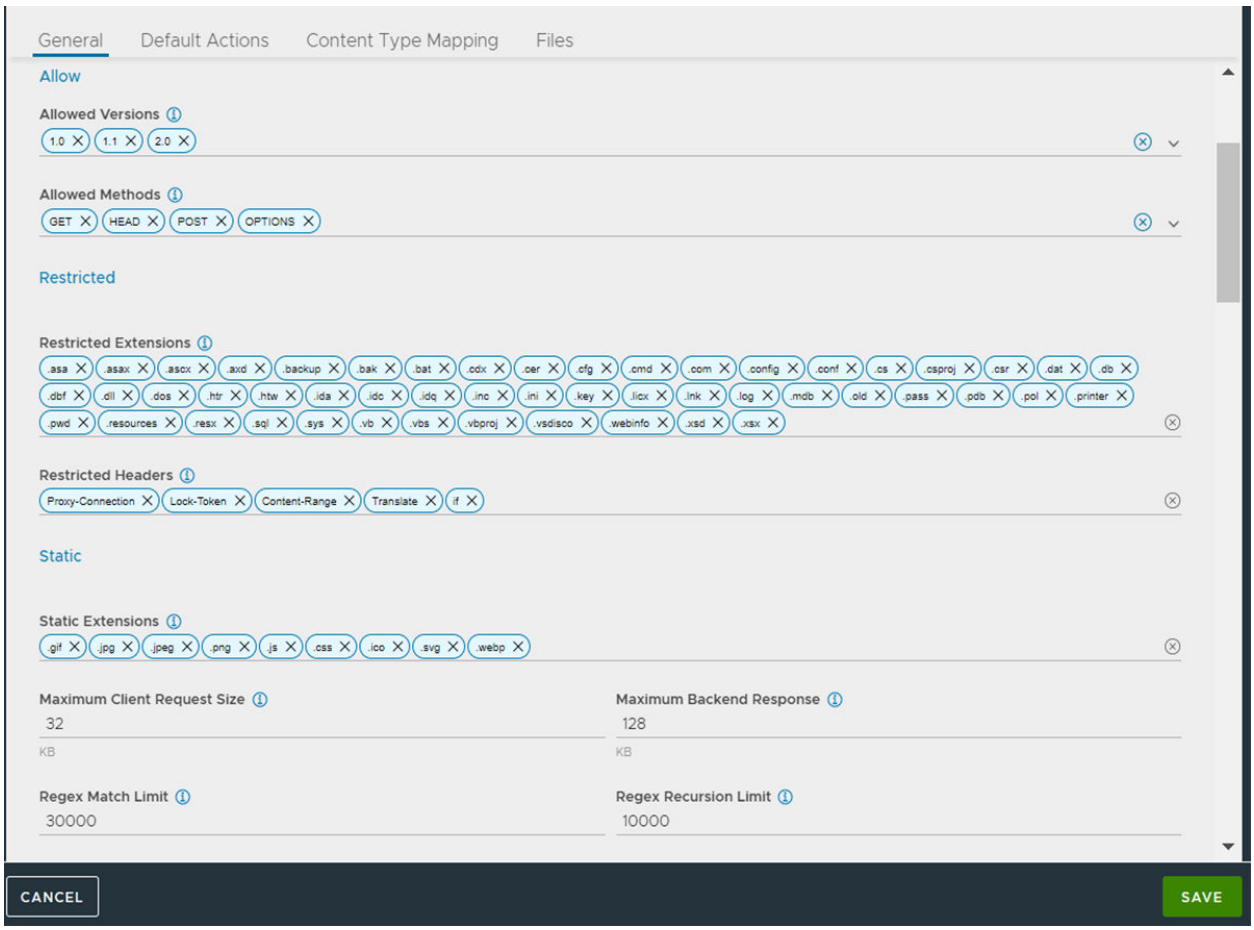
   c Content Type Mapping

   d Files

4 Click Save.

General

The `General` section in the `Create WAF Profile` screen is as shown below:



Provide the following details to configure the WAF Profile:

Field	Description	Additional Information
Name	Enter a relevant name for the profile.	
Allowed Versions	Enter the allowed HTTP versions for the profile.	**1.0**, **1.1**and **2.0** are the default entries.
Allowed Methods	Enter the allowed HTTP method for the profile. Different applications might need different methods.	Websites might use only the default HTTP methods i.e. **GET**, **HEAD**, **POST**, **OPTIONS**. APIs might use other HTTP methods such as PUT, DELETE, TRACE, CONNECT etc. You can also choose from the following additional options: ■ **PATCH** ■ **PROPFIND** ■ **PROPPATCH** ■ **MKCOL** ■ **COPY** ■ **MOVE** ■ **LOCK** ■ **UNLOCK**

Field	Description	Additional Information
Restricted Extensions	Enter extensions that must be restricted and blocked.	Generally, these are files that do not reside on a web server.
Restricted Headers	Enter headers that will not be allowed by WAF.	
Static Extensions	Enter the list of static file extensions that will bypass the WAF check.	A GET request without any parameter or dynamic part is classified as a static request. It does not contain any attack vector.

Other Settings

The following are some more settings that can be configured in the **General** Tab:

Field	Description	Additional Information
Maximum Client Request Size	This is the maximum size for the client request body scanned by WAF. WAF buffers up to *client_request_max_body_size* of the request body and inspects it. If the request body is larger than this, the remaining part of the request body is not inspected by WAF but is streamed directly to the backend server.	Example - 32
Maximum Backend Response	Enter the maximum response size in KB allowed by WAF.	Example - 128
Regex Match Limit	This is the limit for CPU utilization for each regular expression match when the processing rules.	Example - 30000
Regex Recursion Limit	Limit depth of recursion for each regular expression match when processing rules.	
Maximum Execution Time	This is the maximum time allowed for WAF processing for a single request.	Example - 50.
Argument Separator	Enter the separator for special applications that have different argument separators.	Example - &.
Cookie Format Version	Select the preferred cookie format version.	**Version 1 Cookies**have been deprecated. So, **Netscape Cookies** are recommended.
Ignore Request Body Parsing Errors due to partial Scanning	Select to ignore request body parsing errors due to partial scanning. When we have an incomplete request for inspection in WAF because of the **Maximum client request size** setting, the body parse very likely returns an error because the request body is incomplete. This error is ignored only if we present a partial request and *ignore_incomplete_request_body_error* is set. This is the default setting.	
Enable XXE Protection	Block or flag XML requests referring to External Entities.	Select or deselect the check box.

Default Actions

Each phase has a default action. The fields defined for the default action are **phase**, **action**, **status code**, **additional logging** and **WAF logs**.

The value of default action takes the following format:

<phase:n>, <action>, <status code>, <additional-log-level>, <WAF-log-level>

- <phase>: The following is the list of WAF phases and the allowed values for the phase:
  - **Request Header Phase** - **phase:1**.
  - **Request Body Phase** - **phase:2**.
  - **Response Header Phase** - **phase:3**.
  - **Response Body Phase** - **phase:4**.
- <action>: Two options for action are **pass** and **deny**.
- <status code>: If the request is denied by WAF, by default a 403 status code is sent to the client. The status code can also be customized (if required). For example, 403.
- <additional-log-level>: The additional logging level. For example, **log**.
- <WAF-log-level>: The WAF logging level. For example, **auditlog**.

The following is an example for default action for **Request Header Phase**:

phase:1,deny,status:403,log,auditlog.

Content Type Mapping

This tab is used to configure the accepted request content types for the profile.

WAF Profile contains a mapping from HTTP Content-Type value to the WAF request body processor. Request body processor is a WAF component responsible for parsing a particular format of a HTTP request, like application/x-www-form-urlencoded, JSON or XML. This section discusses more information on WAF request body parsers for various content types and how WAF uses them.

Though WAF ships with default settings for popular content types, on some occasions there can be a need to configure a specific content type used by a particular application. For example, when you configure application/xml Content-Type to use request body processor XML, it indicates that the XML request body processor will be used when WAF processes an HTTP request with application/xml Content-Type header. The XML request body processor parses the XML contents of the request and allows for use of XML variables in Signature Rules.

The following is a list of available request body parsers:

Request Body Parser	Description
URL Encoded	WAF request body parser for URL encoded data (application/x-www-form-urlencoded Content-Type).
Multipart	WAF request body parser for multipart form data (Content-Type multipart/form-data).
JSON	WAF request body parser for JSON data.
XML	WAF request body parser for XML data.
As String	WAF request body parser that does not attempt to interpret the body of incoming requests in any way and does not set ARGS WAF variables. REQUEST_BODY variable is set to the value of request body data. HTTP request is treated as a text and WAF attempts to scan for potential attack vectors in request data. However, it is advised to use more specific request body parser if possible.
Do Not Parse	WAF request body parser that does not attempt to parse incoming requests and does not set any WAF variables, effectively turning off WAF protection for a given content type. Use **ctl:forceRequestBodyVariable** in Custom Signature Rule to set *REQUEST_BODY* variable if it is needed.

When no request body processor is configured for a given Content-Type value, WAF assumes a String value.

Note When writing custom PRE or POST CRS (Core Rule Set) rules, you can use **ctl:requestBodyProcessor** action to set the request body processor when the conditions specified in the rule are met. Setting request body processor using **ctl:requestBodyProcessor** action takes precedence over **WAF Profile setting**.

Files

The static input data in a WAF Profile that is shared between virtual services is stored here. For example, the file name `sql-errors.data` has the default data set which contains strings for examining HTTP responses for data leakage protection.

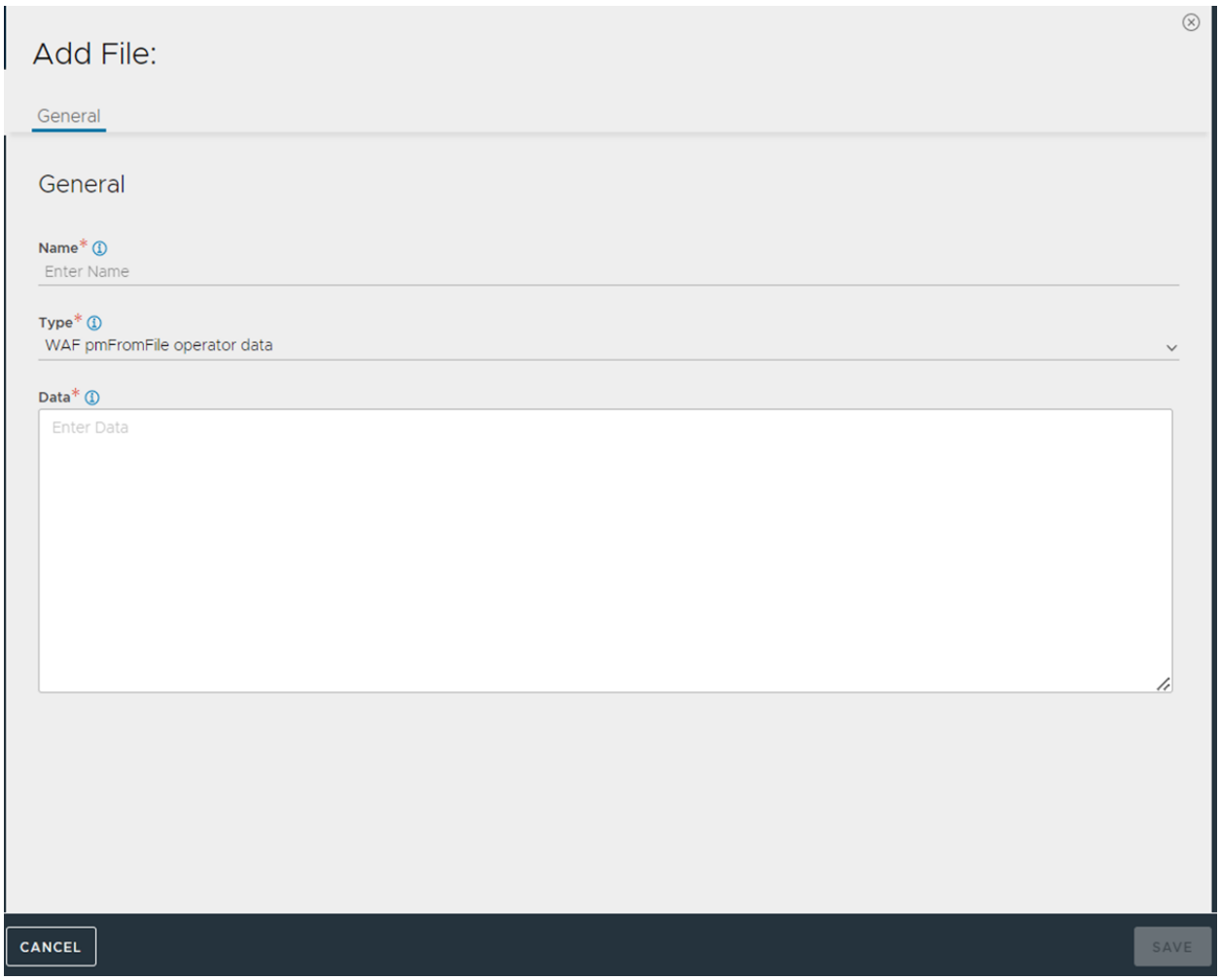
To create a new file:

1 Navigate to the **Files** tab.

2 Click **Add**.

3 Provide **Name**, **Type** and enter the relevant **Data**.

These files can be referred in the custom WAF Policy rules. For more information, see Custom Rules.



WAF Policy

This section discusses the WAF Policy on NSX Advanced Load Balancer.

WAF Policy is a specific set of protections for the application. This policy is enabled by associating it with a virtual service.

Configuring WAF Policy

This section discusses how to configure WAF Policy.

Note

- Navigate to **Templates > WAF > WAF Policy** to locate the default policy.
- `System-WAF-Policy` is the default policy in NSX Advanced Load Balancer, which is the recommended starting point for all new applications. For example, it contains the NSX Advanced Load Balancer OWASP CRS Signatures. For more information, see Updating CRS Rules.
- For customizing a policy, it is highly recommended to create a new policy instead of editing the default policy `(System-WAF-Policy)`.
- WAF policies that enable Application Learning cannot be shared between applications since they contain configuration tailored to that specific application.

The following are the steps to create a new policy:

1 Navigate to **Templates > WAF > WAF Policy**.

2 Click **Create**.

Note Create will clone the **System-WAF-Policy** and use it as the basis for the newly created WAF Policy.

3 Configure the new WAF Policy under the following tabs:

a **Settings**

b **Learning**

c **Allowlist**

d **Positive Security**

e **Application Rules**

f **Signatures**

4 Click **Save** to create the WAF Policy.

Settings

Provide the following details to configure the WAF Policy:

Field	Description	Additional Information
Name	Enter a relevant name for the policy.	
WAF Profile	Choose the WAF Profile that must be attached to this policy. The profile contains common reusable settings that complement the WAF Policy.	For more information, see WAF Profile.

Field	Description	Additional Information
Policy Mode	Select one of the following modes: ■ Detection. ■ Enforcement. For more information, see Selecting a WAF Policy Mode.	It is recommended to use Detection Mode when onboarding a new Application. For more details, see WAF Mode. For more information on Mode delegation, see Mixed Mode and Enabling Mode Delegation.
Allow Mode Delegation	Enable this option to allow WAF rules to overwrite the Policy Mode selected, where specific action (Detection or Enforcement) can be defined for a single rule, irrespective of the action defined for the rule set.	**Allow Mode Delegation** check box is only enabled if the Policy Mode selected is **Detection**, since it is required for **Enforcement** mode.
Paranoia Level	Set the paranoia level for the WAF Policy. This is used to determine the rigidity of the policy and has a direct impact on potential false positive rate.	For more information, see What are the Paranoia Modes available in WAF? What are the considerations for choosing the mode?.
Geo DB	Geolocation Mapping Database used by the WAF Policy.	

Mode Delegation

With **Mode Delegation** option, the policies can be enabled to operate in the following two modes:

- **Detection**: In Detection mode, if a request matches a rule, the request is flagged with an application log message (marked FLAGGED) and allowed through.
- **Enforcement**: In Enforcement mode, if a request matches a rule, it is blocked by the Service Engine, and an application log message (marked REJECTED) is generated.

If Mode Delegation is enabled, individual WAF rules can overwrite the Policy Mode, resulting in different behavior from the rest of the rules. This is also called mixed mode and is another way of fine-tuning to avoid legitimate requests from being blocked due to Enforcement mode.

A few relevant use cases for enabling Mode Delegation are:

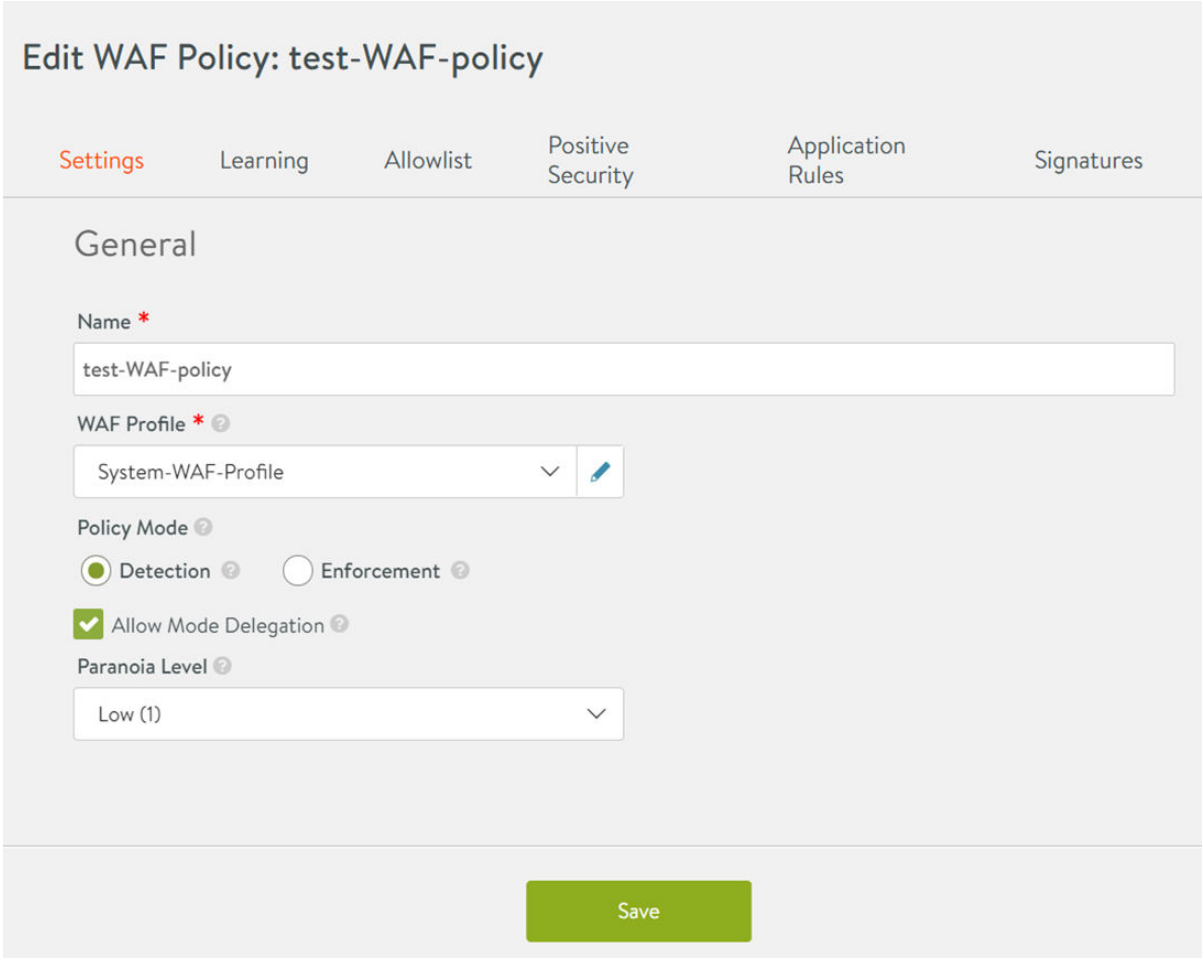
- Test new rules: You can configure manually written rules or new CRS rule updates with mixed mode enabled to avoid false positives. You will be able to introduce new rules to operate in Detection mode to ensure that legitimate requests are not rejected.
- Partial detection: You can configure a few rules in Enforcement mode, while still retaining the whole WAF Policy in Detection mode.

You can enable Mode Delegation through the following steps:

1 In the NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Policy**.

2 Click **Create** or edit an existing WAF Policy.

3 In the **Settings** tab under **Policy Mode**, select the check box for **Allow Mode Delegation** to enable mixed mode.



To enable Policy Mode for a certain rule:

- Navigate to the **Signatures** tab and select the **CRS Version**.
- Expand the **Group** that the **Rule** to be edited is part of.
- Click the **edit** icon for the **Rule** to be edited.
- Under **Rule Mode**, select the option **Use Policy Mode**.
- Click **Save**.

WAF Mode

The following section compares the two modes of WAF configuration policy.

Detectionand **Enforcement** are the two modes supported for a WAF Policy in NSX Advanced Load Balancer. Every policy runs in one of these modes to evaluate the requests and responses.

Name	Detection	Enforcement
Policy	Logs alerts during an attack, but no deny action is taken.	Rejects requests when a policy is matched and deny action is taken.
Operation	Evaluates the whole policy without stopping at the first rule hit.	Matches the first rule that rejects the request and implements the default action or returns a rule specific error code.
Log files	Contains the WAF log section where the policy violation was found and entries for every rule that is matched.	Contains WAF log section which has the first rule that rejected the request. **Note** This is to improve performance. If a request is already identified as an attack, further checks are not required.
Response Code	200 OK	Default is 403 Forbidden. This response code can be modified in the WAF Profile Default Actions section.

Selecting a WAF Policy Mode

You can select one of the following modes. The supported modes are:

- **Detection**: In Detection Mode, a rule gets processed, but will not perform a blocking action.
- **Enforcement**: In Enforcement mode, a rule gets processed and blocking actions are executed based on the defined default action. This Default Actions is configured in the WAF Profile. By default, the WAF rejects requests with attack vectors and the corresponding log entry is marked as **REJECTED**.

Note When a rule is configured using **Use Policy Mode**, the overall Policy Mode (either Enforcement or Detection) will be used.

Usage Recommendations

Follow the steps provided to enable a suitable mode for different usage scenarios.

For new applications:

1 Create a virtual service for the application.

2 Add WAF Policy in **Detection** mode.

3 Iterate through false positive mitigation.

4 Once no legitimate traffic is flagged by WAF, change to **Enforcement** mode.

For existing applications:

1 Add WAF Policy in **Detection** mode.

2 Iterate through false positive mitigation.

3 Once no legitimate traffic is flagged by WAF, change to **Enforcement** mode.

Note The time taken for evaluating **Detection** mode depends on several factors such as total number of requests seen, paranoia mode, and application coverage of request.

Application Learning for WAF

This section discusses Application Learning for WAF.

Application Learning enables the WAF feature on NSX Advanced Load Balancer in order to analyze a set of incoming traffic processed by the WAF Policy

When the Application Learning is enabled on a virtual service, the Service Engine collects data and sends it to the Controller for analysis. So, all learning takes place on the Controller. The traffic selection for Application Learning is based on the WAF Policy configured.

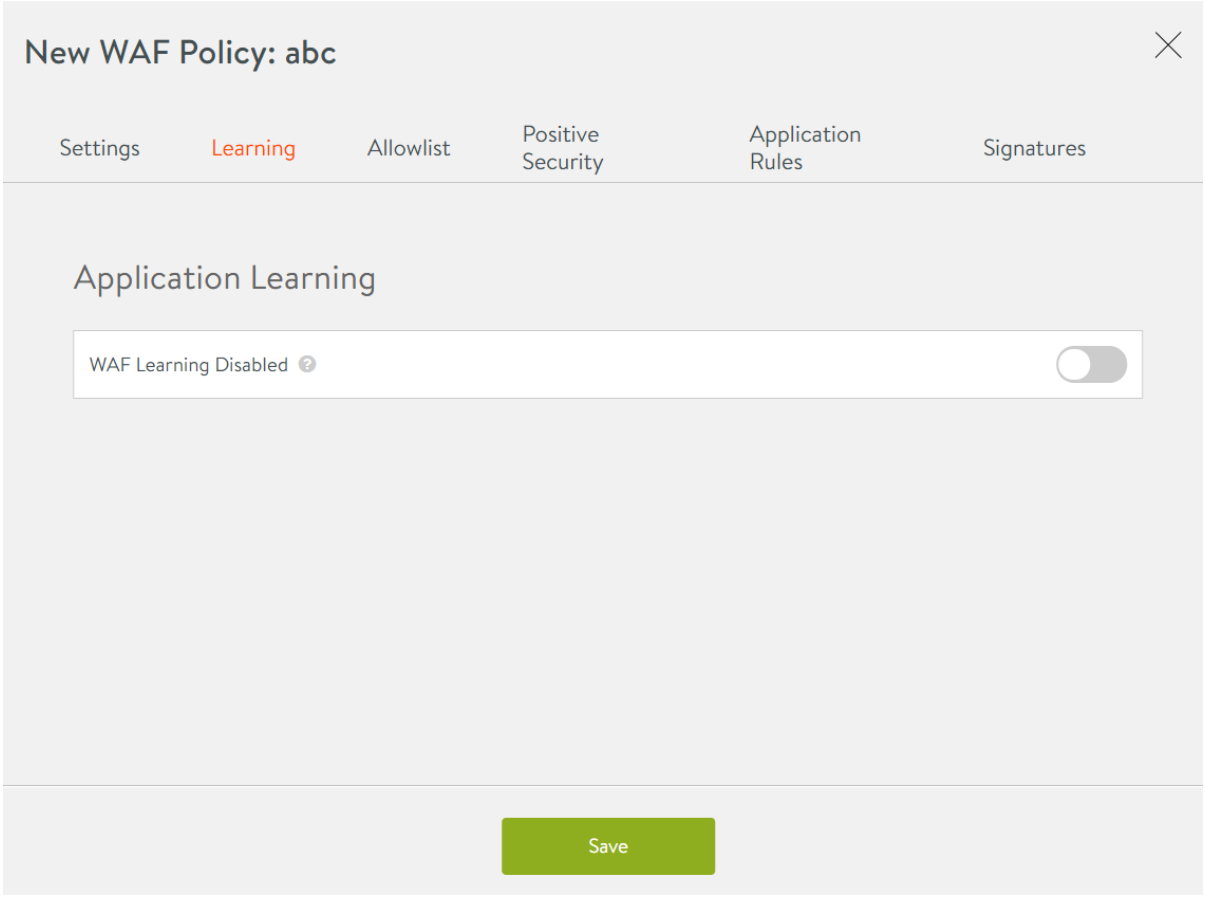
It parses all paths containing URI or BODY parameters of an HTTP request. This collection continues during a specified duration or time interval. Once the timer is hit, the Service Engine sends the data to the NSX Advanced Load Balancer Controller for analysis. These WAF configuration parameters are distributed across WAF Policies.

Learning

To enable the Learning option:

1 Navigate to **Templates > WAF > WAF Policy**.

2 Select the policy for which **Application Learning**must be **enabled**. The following screenshot exhibits the option to enable Application Learning :



3 Enable Application Learning for the selected WAF Policy by moving the slider. Once the option is enabled, the additional configuration options will be available to edit as follows:

Note

- Enabling Application Learning for WAF does not automatically create a Learning Group. The warning message, `A Learning Group is required for WAF Learning` is displayed, along with a link to **Create Learning Group**, to prompt the user to create a Learning Group.
- If **Per URI Learning** is **ENABLED**, the learning algorithm programs the URI and param combinations when they reach the confidence score. If **DISABLED**, the learning algorithm programs the params independently from the URI. This can be useful when URIs are generated for each session.

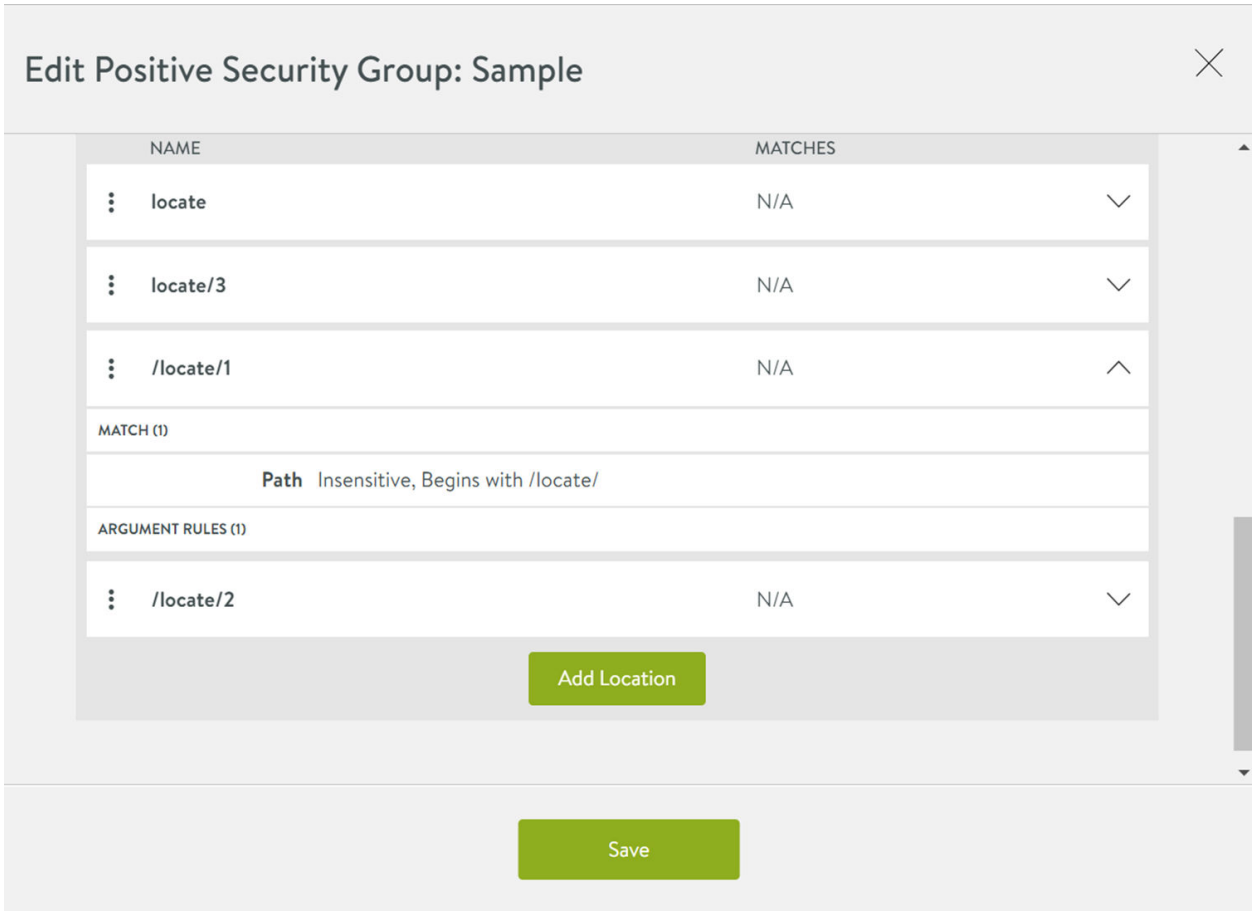
Field	Description	Additional information
Sampling	Percent of the requests subjected to Application learning.	Range (1 to 100%).
Enable Auto Rule Updates	Enable Application Learning based rule updates on the WAF Profile. Rules will be programmed in dedicated WAF learning group.	Select or deselect the check box.

Field	Description	Additional information
Auto Promote Rules w/ Confidence	Minimum confidence label required for auto rule updates.	Low Probable High Very High (99.99 -100%)
Learning Interval	Frequency with which SE publishes Application learning data to controller.	Range (1 to 60 min). Example- 30 min
Max Parameters	Maximum number of params to learn for an application.	Range (10 to 1000). Example- 100
Min Hits to Learn	Minimum number of occurrences required for a param to qualify for learning.	Range (10 to 1000). Example- 100
Per URI Learning	Learn the params per URI path.	Select or deselect the checkbox.
Max URI	Maximum number of URI paths to learn for an application. This value can be set higher for more complex applications.	Range (10 -10000).

Consolidation of Learning Data

The Learning Data that is generated will be consolidated into prefixes combining similar URLs. This is done automatically to reduce the overall number of locations within Positive Security Model. It does not change the behavior, but just the manageability of the number of rules.

The following example shows the details of a location when the prefix is considered for the similar URIs instead of the full URIs. Similar URIs are consolidated into one location using a prefix (for example, /locate/) and accordingly the path is matched as **Begins with** (instead of equals).



Adaptive Application Learning

This section explains Adaptive Application Learning.

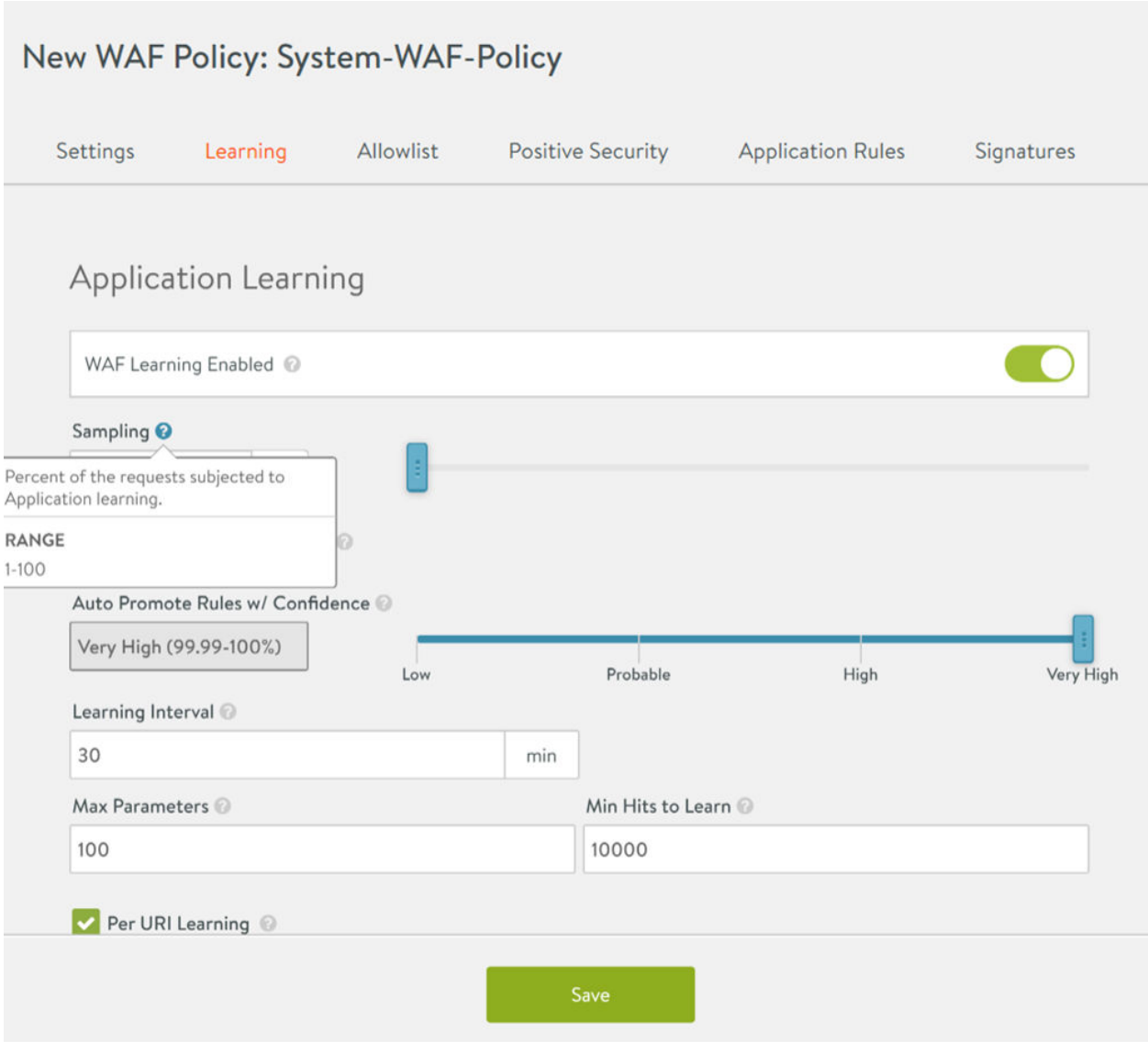
The NSX Advanced Load Balancer supports Adaptive Application Learning. In Adaptive Application Learning, the Controller takes control of adjusting WAF Learning parameters to facilitate effective Application Learning, while the SE only enforces it.

In Adaptive Application Learning, manual modification of sampling percentage and learning interval is not required.

To improve user experience and optimize resource usage while maximizing the application learning progress, a feedback system is created for tuning these parameters, once learning is enabled on an application.

For effective Application Learning, the followings parameters are continuously adjusted by the Controller:

- Sampling Percentage
- Learning Interval



The following sections provide more information on how to manually use these parameters.

Sampling Percentage

For a WAF Policy, sampling is the process of assigning a percentage of the incoming requests to participate in the Application Learning process. This topic details the steps to enable sampling and modify sampling percentage through the application UI.

The sampling rate controls the frequency of the Service Engine collecting data while analyzing happens in the Controller.

If the value of sampling frequency is set to 50%, the Service Engine will only collect 50% of the incoming requests or every alternate request.

It is recommended to use the sampling percentage of 100% in the initial phase of Application Learning. This helps in faster data aggregation and more efficient application learning.

Using a lower sampling rate conserves SE resources when no new data is available for learning. When the learning is in progress, the quantum of URI information that is sent to the NSX Advanced Load Balancer Controllertends to peak or fall.

The NSX Advanced Load Balancer Controller sends the adjusted sampling percent (reduced) to the SEs. After the sampling, the SEs have to inspect or evaluate only a small percentage of the incoming traffic. The maximum sampling percent for the application learning is set to 100%, the minimum percentage can be set as 1%.

By default, the value of the `enable_adaptive_config` parameter available under the analytics profile, is set to `true`.

To disable adaptive learning, set the value of `enable_adaptive_config` to `false`.

In adaptive application Learning, when a new type of traffic is received by SEs, the NSX Advanced Load Balancer Controller changes the sampling rate for the learning.

The option to manually set the Sampling Percentage is available under the **Learning** tab of WAF Policy.

Note It is recommended to keep the default setting for sampling rate.

Learning Interval

This section provides information on Learning Interval in Application Learning.

Learning interval is the time period or duration, after which the Service Engine sends data related to Application Learning to the NSX Advanced Load Balancer Controller. By default, this duration is set to 30 minutes. This means that the Service Engines sends data to NSX Advanced Load Balancer Controller every half an hour for further processing. Based on the learning activities or the amount of Application Learning data, the value of this parameter can be increased or decreased.

Note It is recommended to keep the default 30 minutes for production systems. For staging or test environments it can be set lower for quicker visibility of learned data.

Application Rules

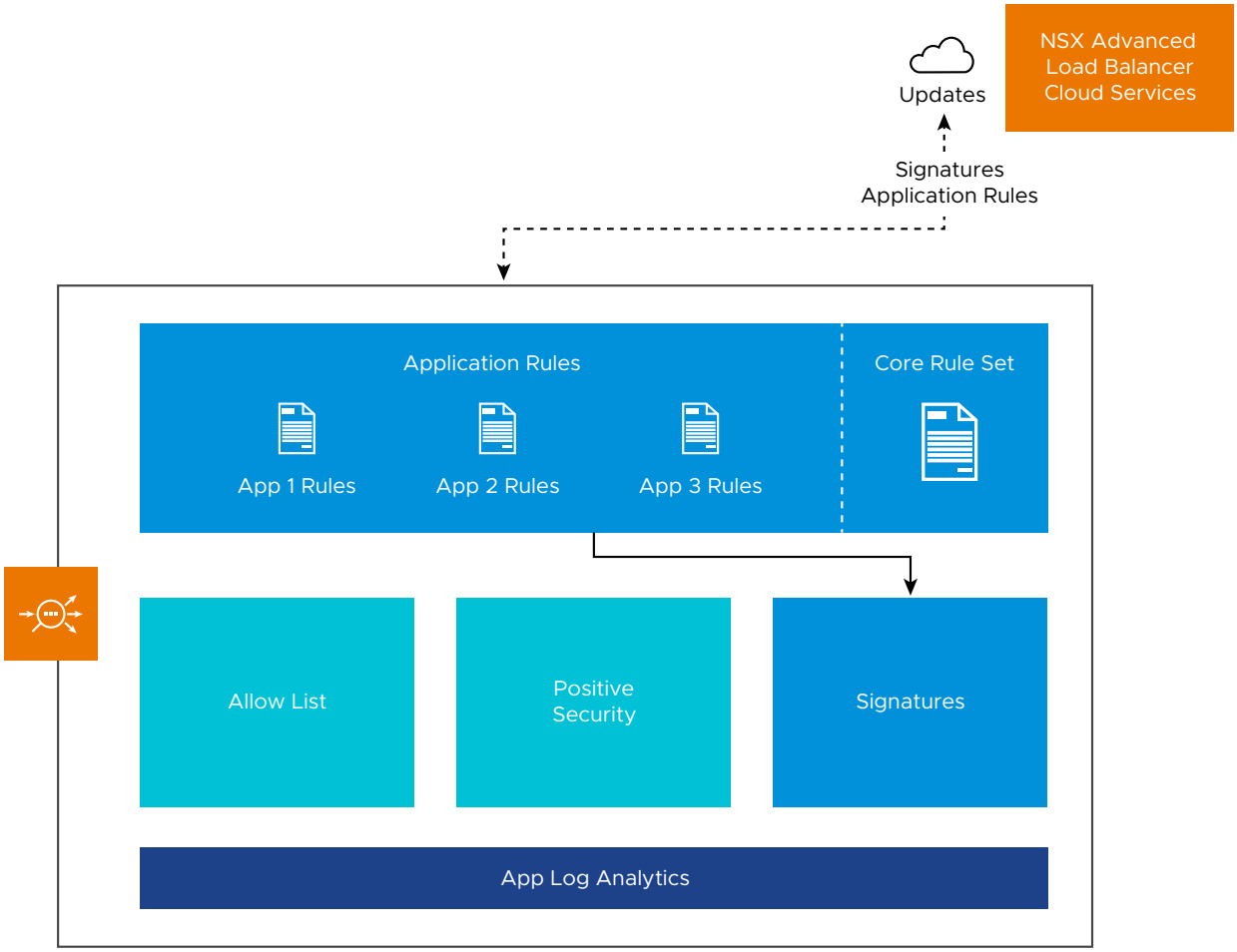
This topic explains the application-specific rules for WAF. To secure applications with known vulnerabilities, WAF supports the use of Application Rules.

Note This feature works only if you have the NSX Advanced Load Balancer Enterprise License for Cloud Services.

These Application Rules are specifically designed to block attacks on known application vulnerabilities (many of them with Common Vulnerabilities and Exposures (CVEs)). They are automatically updated using the Application Rules of the NSX Advanced Load Balancer Cloud Services. When the admin enables this feature, more than 5000 applications can be selected, thereby activating specific rules for that application in the WAF Policy.

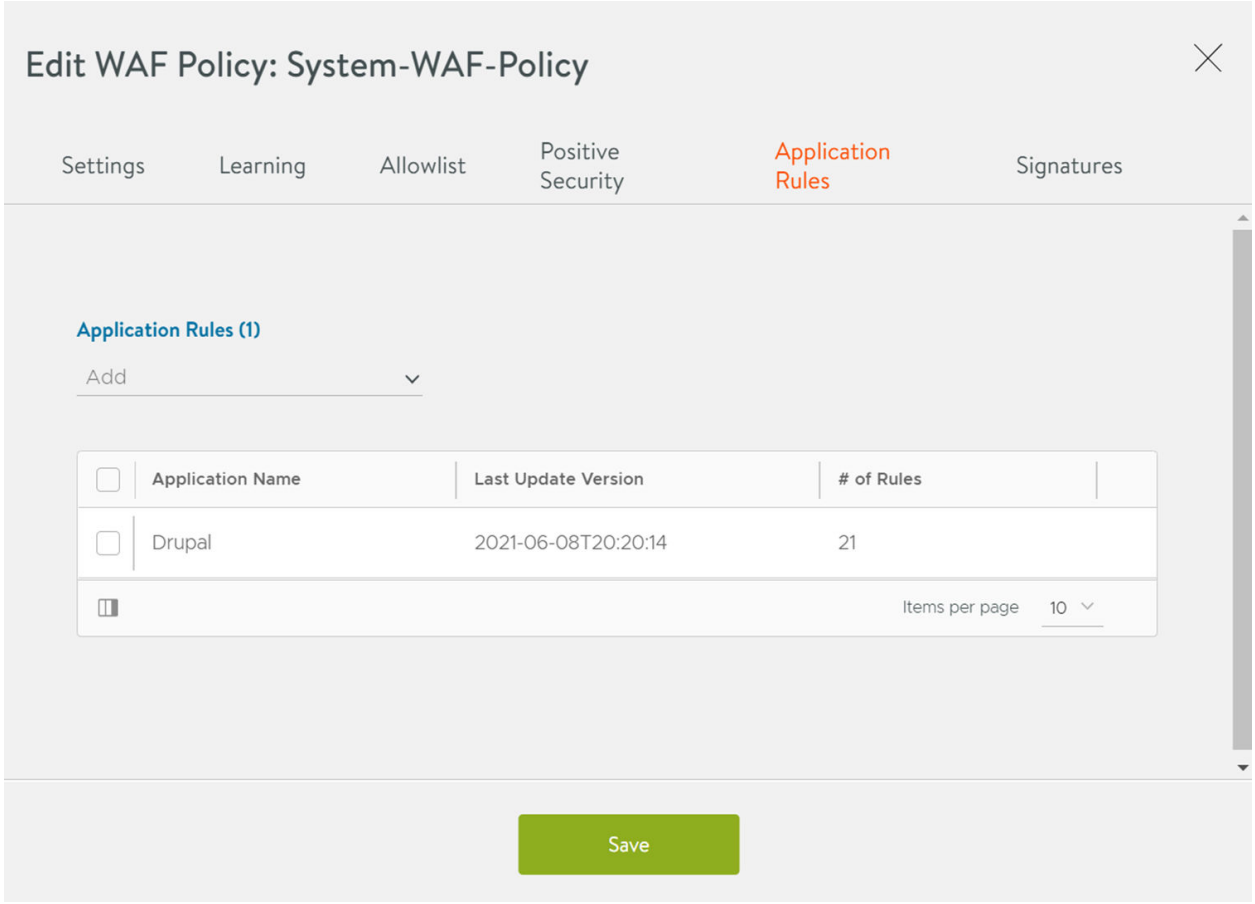
For more information on Application Rules for Cloud Services, see Application Rules Service.

For more information on WAF Policy, see WAF Policy



Configuring Application Rules

To configure Application Rules, navigate to **Templates > WAF > WAF Policy** and click the **Edit** icon for the desired policy. Navigate to the **Application Rules** tab. You can choose the Application Rules by clicking the **Application Rules** drop-down menu. When an application is selected, it is added to the list of currently configured applications and will showcase the application name, the last update time stamp, and the number of rules for this application.



In this example, Drupal has been chosen as the Application to protect. The policy now contains 21 rules that are active and protect against specific exploits known for Drupal.

To remove an application from the list, click **delete** icon for that item.

Updating Application Rules

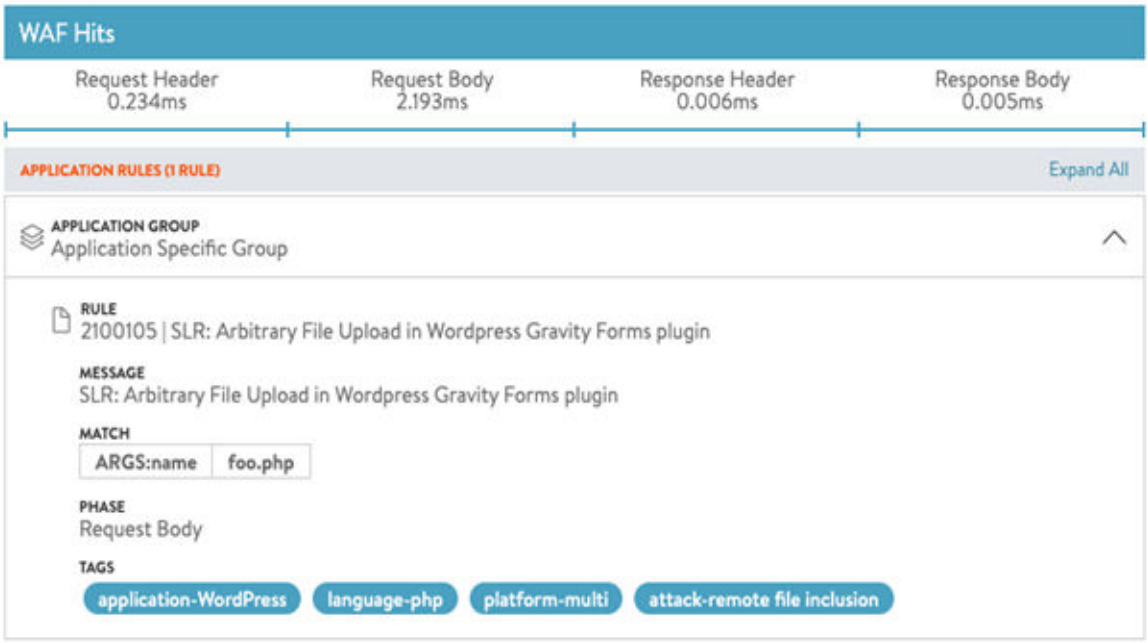
You need to register to the Application Rules of the NSX Advanced Load Balancer Controller using Cloud Services, for the Application Rules feature to work. For more information on configuring through Cloud Services, see Application Rules Service. After this initial configuration, Application Rules are updated automatically on a daily basis. Updates are applied to any WAF Policy with a configured application.

Note If a rule has been disabled, this change is retained when an update is applied.

App Log Analytics

When an Application Rule is hit, the App Log Analytics captures the event.

It contains information about the type of attack that it has detected. It is seamlessly integrated with other parts of the WAF functionality.

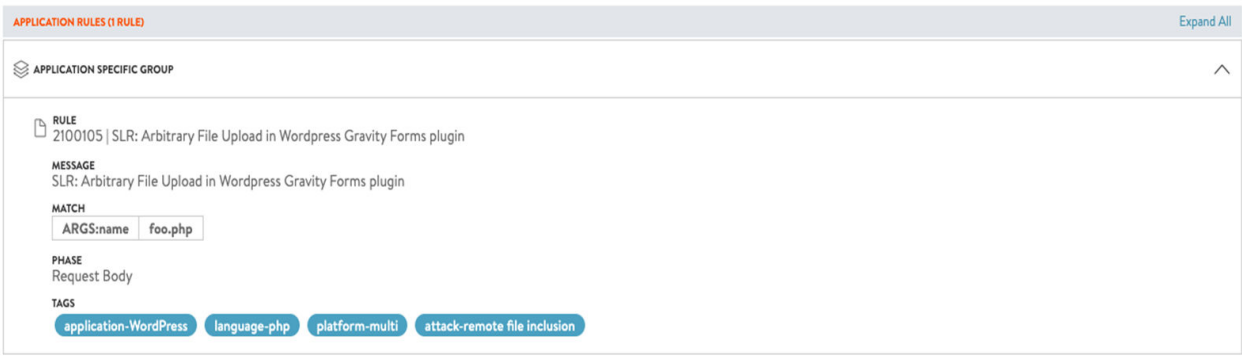


Tuning of Application Rules

The nature of the specific protection provided by Application Rules makes it easy to deploy. There are no false positives because the Signatures target only specific attack vectors.

Note In contrast to rules, the rule text is not shown.

If there is a need to tune a rule, it can be disabled using CLI.



Steps to Configure through CLI

You can disable rule using the rule ID 2100105 as shown in the following example.

```
ssh to Controller
shell (login)
configure wafpolicy System-WAF-Policy
application_signatures
where
CTRL/F or CMND/F for rule ID: 2100105 -> note index
rules index 14160
```

```
no enable
save
save
save
```

The rule is disabled.

```
| rules[500]     |                                                                  |
|   index        | 14160                                                            |
|   name         | SLR: Arbitrary File Upload in Wordpress Gravity Forms plugin     |
|   rule_id      | 2100105                                                          |
|   enable       | False                                                            |
|   rule         | <sensitive>                                                      |
|   tags[1]      | application-WordPress                                            |
|   tags[2]      | language-php                                                     |
|   tags[3]      | platform-multi                                                   |
|   tags[4]      | attack-remote file inclusion                                     |
|   is_sensitive | True                                                |
```

Troubleshooting

- Application Rules template is empty.
  - Check if the Controller is registered with NSX Advanced Load Balancer Cloud Services.
  - Check if **Enable auto-sync Application Rules DB updates** opt in is set.
  - Check config audit log for `ALBSERVICES` entries.
- Application Rules block a Legitimate Request.
  - Check steps on disabling an Application Rule in Steps to Configure through CLI
- Application Rules are not getting updated.
  - Check *ALBSERVICES* config audit log for details.
  - Update interval is daily. Update check can be done more frequently. You can check again the next day.

Note Individual Application rules are only updated once an Application has a new CVE. For more information on CVE, see What is a CVE?

For more details on Cloud Services connection, see Getting Started.

Note Exceptions are not available for Application Rules. The main reason is that these rules are specific and normally only match a specific parameter on a specific URL. Excluding this parameter from the rule is the same as disabling the rule and therefore entirely removing the protection.

WAF Allowlist

The Allowlist functionality allows the definition of match conditions for requests to perform associated actions. This section discusses examples and use cases for configuring Allowlist Rules in NSX Advanced Load Balancer.

Use cases

- Allow access from the internal network.
- A security scanner that scans the application, directly bypassing WAF protection.
- Do not check special parts of the URL space, for example "`/upload/*`".
- Run parts of the application in **Detection** mode.
- When the request comes from a specific IP range.
- When the request matches the URL pattern specified using the HTTP Method match type.

Configuring Allowlist Rules

This section discusses how to configure Allowlist Rules.

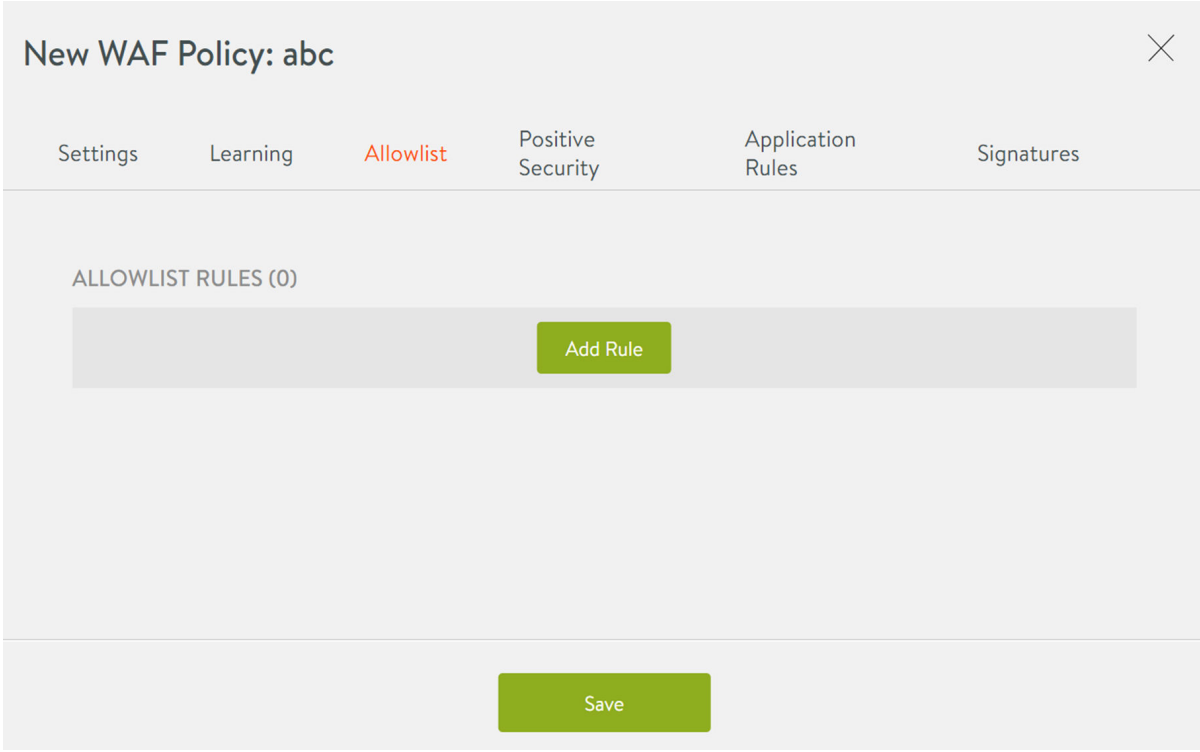
To define Allowlist Rules do the following:

1 From the NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Policy**.

2 Click **Create** or edit an existing **WAF Policy**.

3 Enter the required details under the **Settings**tab.

4 Click **Allowlist** tab.



5 Click **Add Rule** button.

6 In the **New Allowlist Rule** screen, enter the details as shown below:

Table 2-1. General

Field	Description
Rule Enabled	By default, the **Allowlist** rule is enabled. Click the toggle button to disable it.
Name	Enter a relevant name for the rule.
Description	Enter a description to define the rule.
Sampling	Percentage of sampled traffic ranging from 0 to 100.

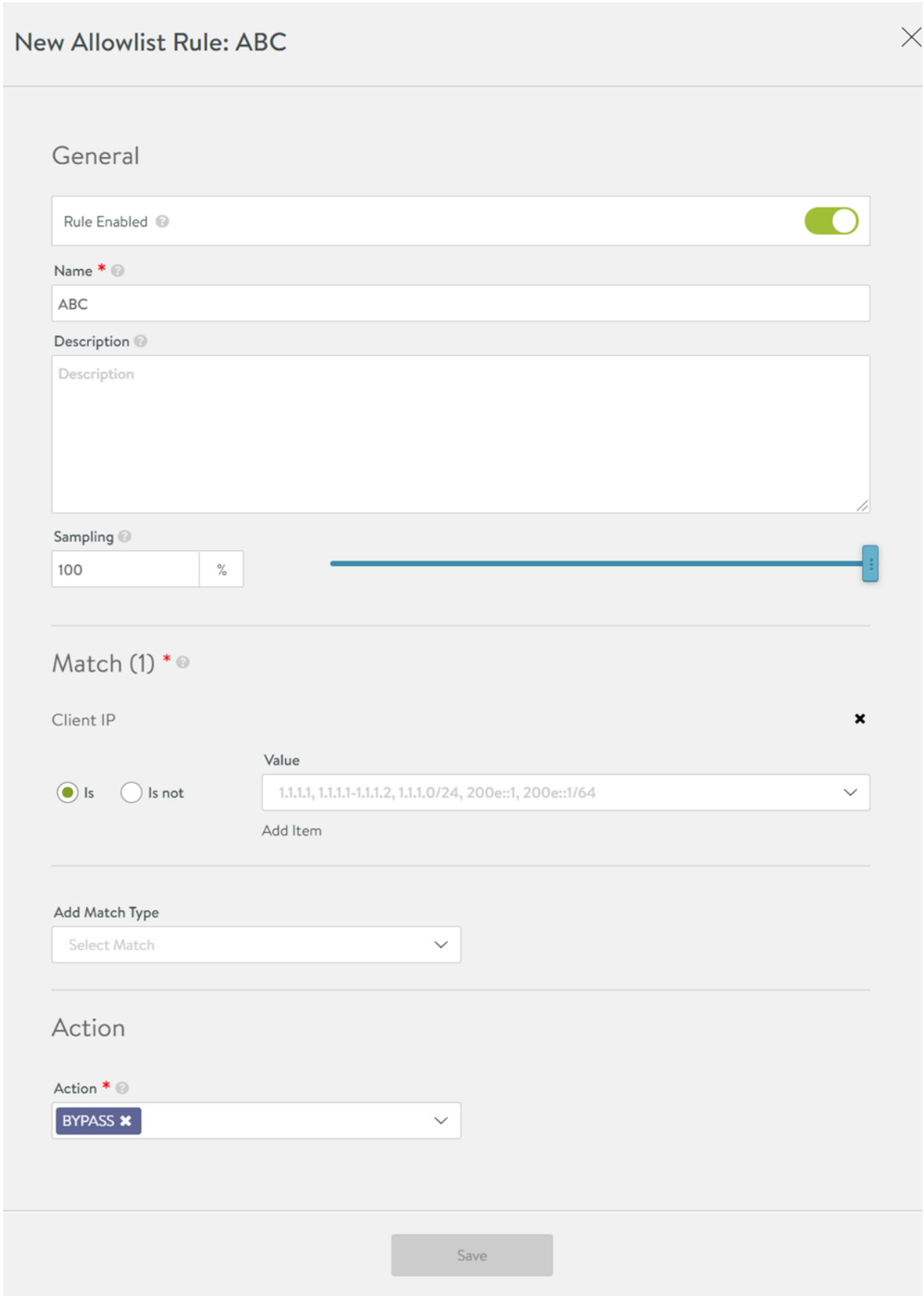
Table 2-2. Match

Field	Description
Add Match Type	Select a **Match Type** from the following options: ■ Client IP ■ HTTP Method ■ Path ■ Host Header

Table 2-3. Action

Field	Description
Action	From the following options, select the action to be performed when the request matches the criteria specified: ■ **BYPASS**: When Bypass is selected, WAF does not execute any further rules and the request is allowed. ■ **CONTINUE**: Selecting Continue stops the Allowlist execution and directs WAF to continue its activity. ■ **DETECTION MODE**: When enabled, the WAF Engine will be set to **DETECTION MODE** for that request.

The **New Allowlist Rule** screen is as shown below:


New Allowlist Rule: ABC
General
Rule Enabled
Name *
ABC
Description
Description
Sampling
100
%
Match (1) *
Client IP
Value
Is
Is not
1.1.1.1, 1.1.1.1-1.1.1.2, 1.1.1.0/24, 200e::1, 200e::1/64
Add Item
Add Match Type
Select Match
Action
Action *
BYPASS
Save

6. Click **Save**.

Match Type

Match type can be used to select a trusted list of client IPs or client IP groups. This section explains configuration of Match Type.

Client IP

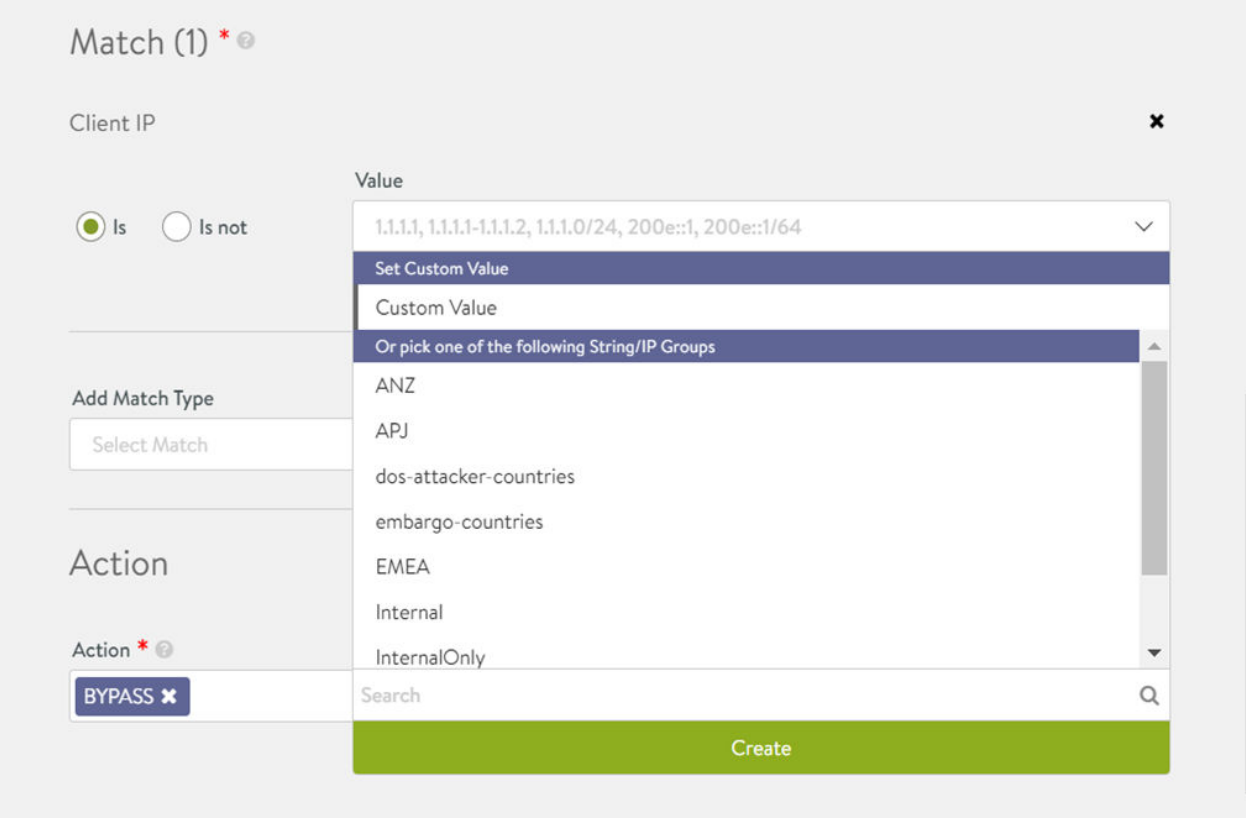
To configure a match rule for the client IPs:

1 Select the match type as **Client IP** under **Add Match Type**.

2 Select **Is** or **Is Not** to provide permissions accordingly.

3 Click the drop-down menu under **Value**.

4 Select either **Custom Value** and enter the IP Addresses manually or select **Internal**.



Note This client IP match type supports IP Groups. For more information, see IP Group.

HTTP Method

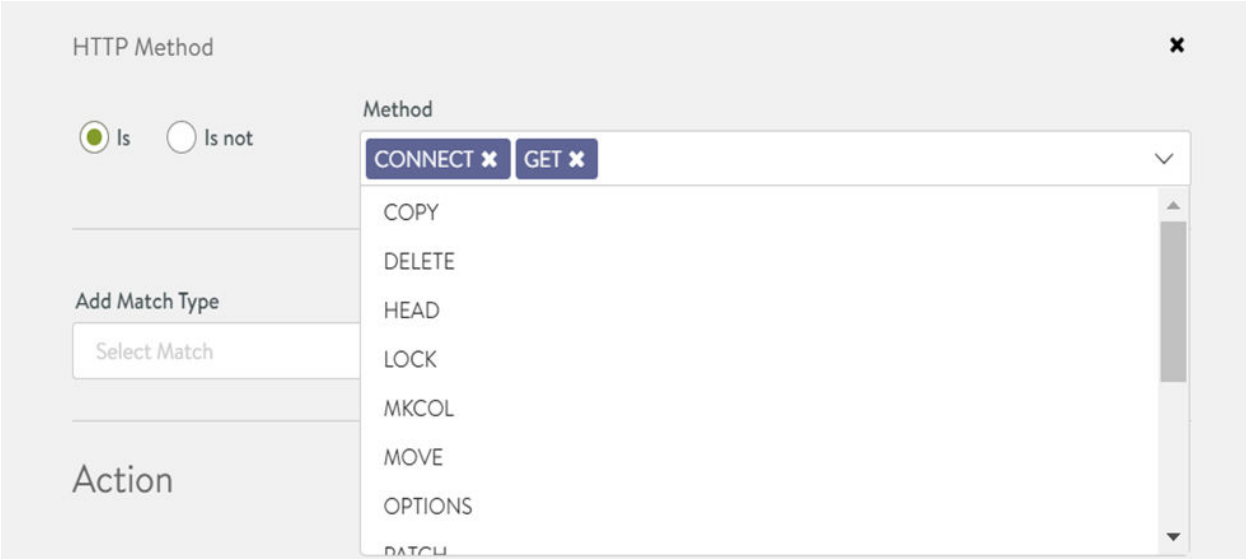
Use this to select only specific types of HTTP requests using the HTTP request methods like GET, CONNECT, DELETE, and more.

To define WAF Allowlist rules based on HTTP Method:

1 Select the match type as **HTTP Method** under **Add Match Type**.

2 Select **Is** or **Is Not** to match or negate the selection under **Method**.

3 Select one or more methods under **Method** field as shown below:



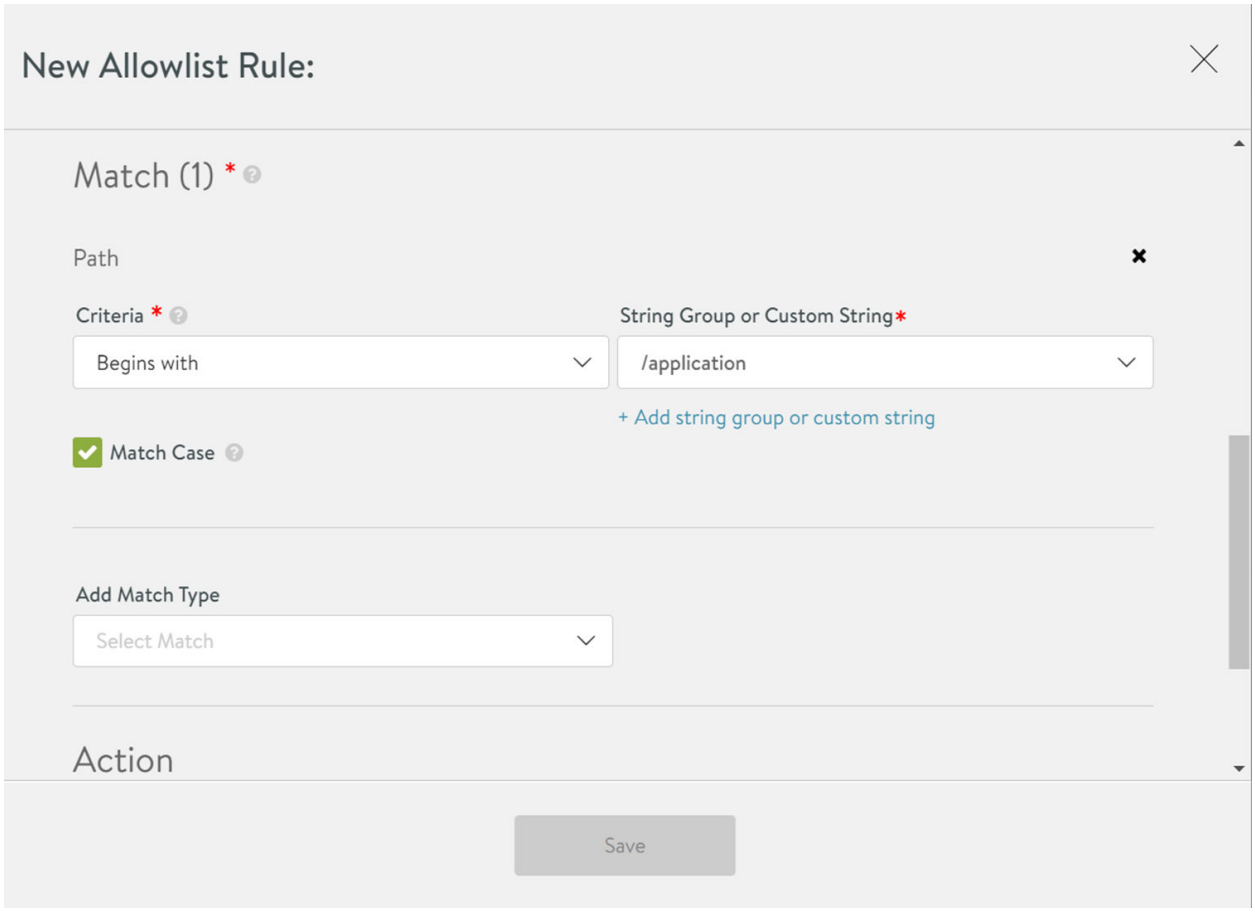
Path

To configure WAF Allowlist for specific URLs:

1 Select the match type as **Path** under **Add Match Type**.

2 Select the **Criteria** and **String Group or Custom String** which needs to be matched in the URL. For example, select **Begins with** and enter **/application** in the **String Group or Custom String** field, to allow all URLs with this prefix.

3 Select **Match Case** to enable case sensitivity.



Host Header

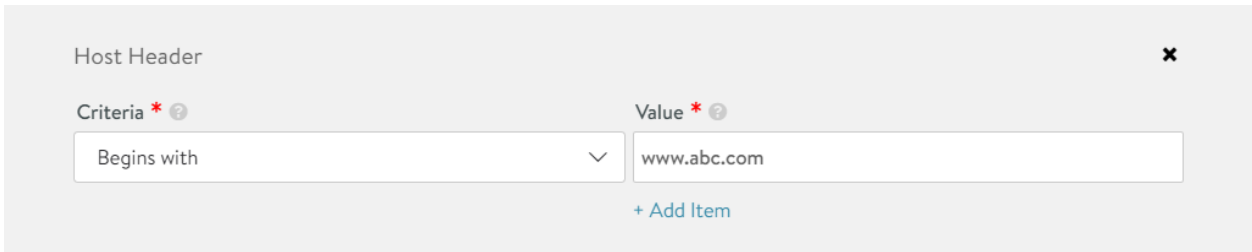
Use this method to apply rules to only requests that match the specified host header criterion.

To configure WAF Allowlist for specific Host Headers:

1 Select the Match Type as **Host Header** under **Add Match Type**.

2 Select the **Criteria** which need to be matched in the URL.

3 Enter the **Value**.



Use IP Group in a WAF Allowlist

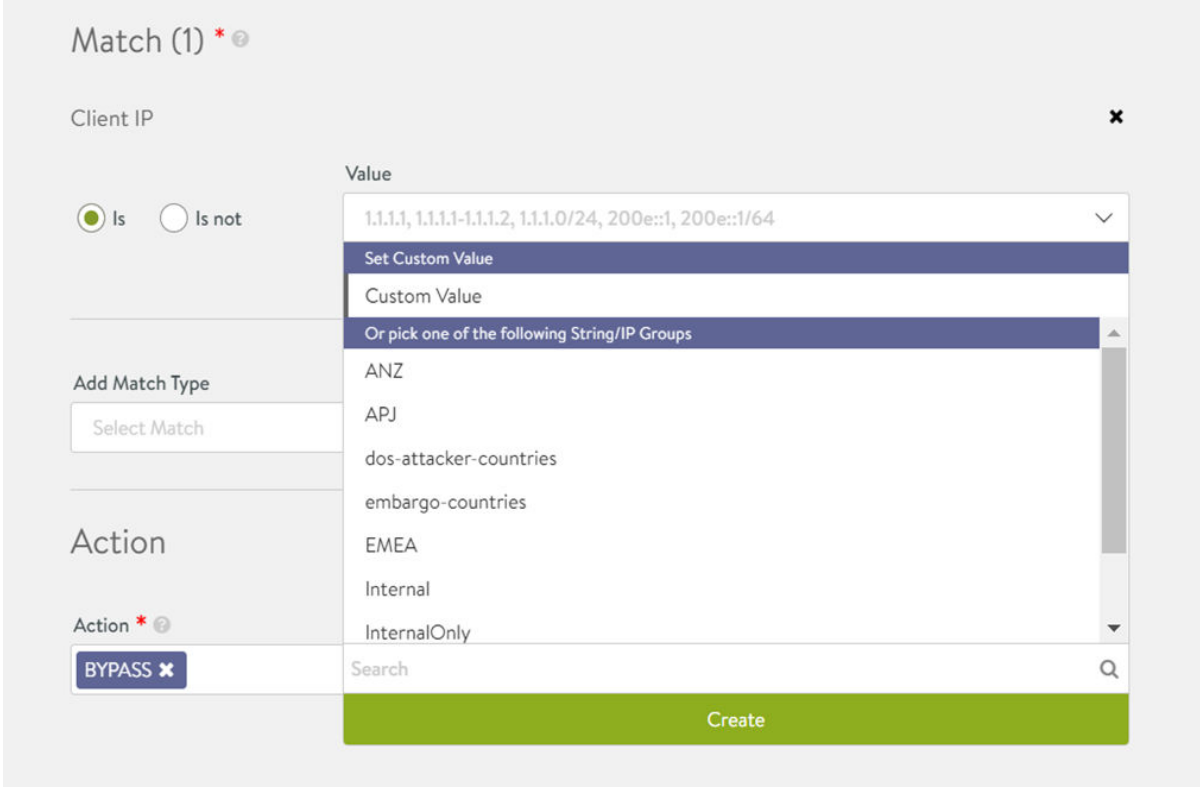
This section explains how to use a previously configured IP group in a WAF Allowlist for making all requests from IPs in the IP group bypass WAF checks.

Prerequisites

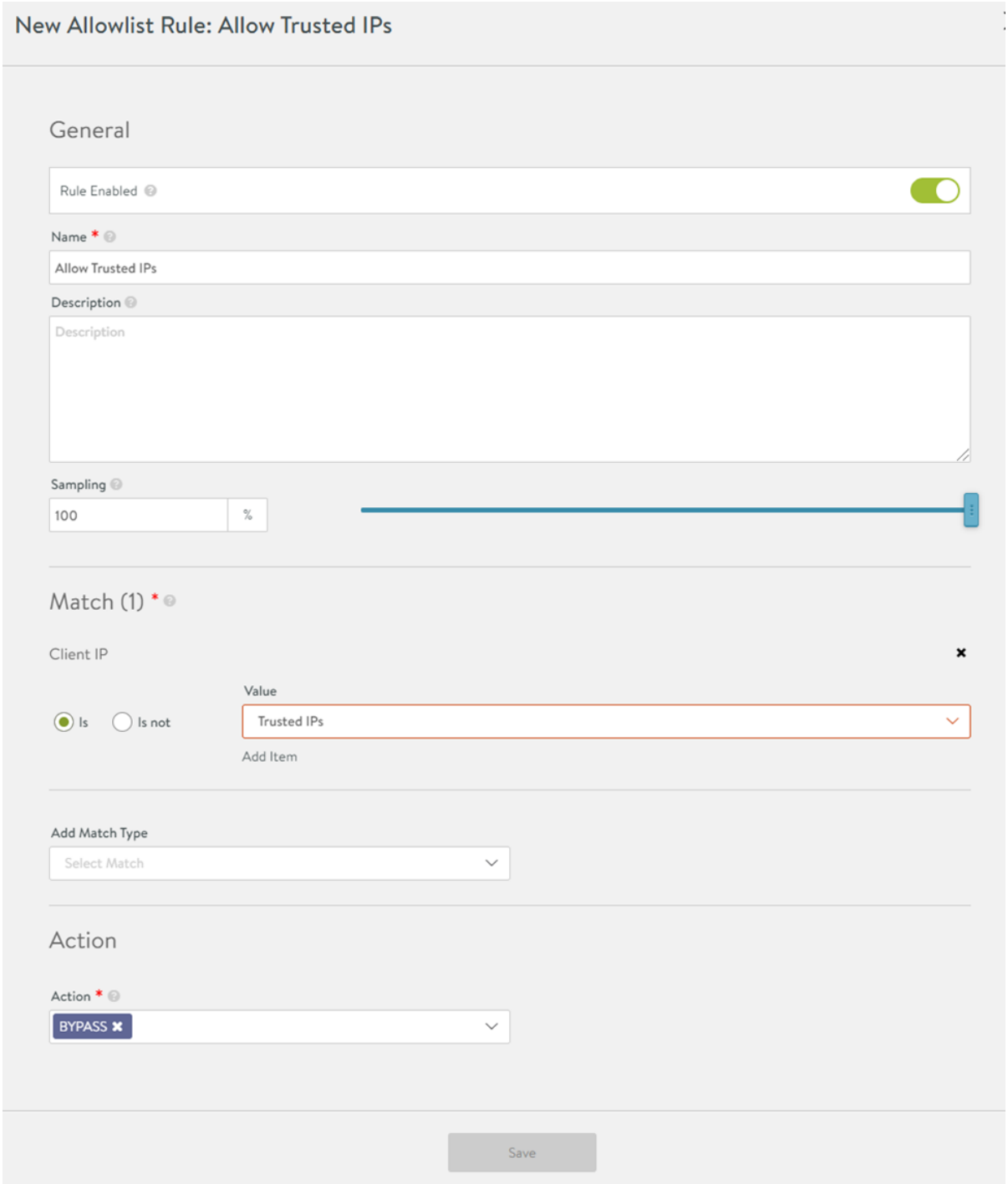
An IP Group with the list of IP addresses that need to be bypassed by WAF must be created. For more information on creating an IP Group, see IP Group.

Procedure

1 Select the previously created IP group as the value for the **Match** option while creating a new Allowlist Rule.



2 Select the desired action and save the WAF Allowlist, as shown below.



3 The above screen shows an Allowlist Rule using IP group. **Action** is set as **BYPASS** for any client IP address which is part of the IP address group created in the previous step.

Sampling Traffic to WAF Allowlist

Sampling is used to enhance the Allowlist feature for the WAF traffic by exposing only a particular percentage of traffic for WAF. This section discusses how to set the sampling of traffic in WAF Allowlist rules.

Sampling is beneficial if we want to send a subset of all traffic through WAF. The *sampling_rate*slider is used to allot a range for each Allowlist rule. The value for *sampling_rate* can range from 0 to 100%.

If the request is in the sampling range, the configured action is applied to the request. For example, if sampling is set to 50%, every alternate request will trigger the action.

Positive Security and Learning

This section discusses Positive Security and Learning feature for WAF.

Positive Security rules define allowed application behavior. These rules can be created by the Learning Engine, scanner import or manually. A Positive Security rule will match when the request (or parts of the request) matches the behavior defined in the rules. This is in contrast to Signatures, which detect attack patterns and will match when an attack pattern is found.

Both Positive Security and Signatures allow similar concepts for rules.

- Enable / Disable.
- Mode (Detection / Enforcement) by rule.
- Paranoia levels of rules.

Reasons for Using the Positive Security Model

- Since Positive Security defines application behavior, it can reduce the attack surface by only allowing known good traffic.
- Positive Security can result in better performance. Instead of checking a value against a long list of known attack vectors using Signatures, the validation is against a single regular expression.

Configuring Positive Security Groups

This section discusses how to configure Positive Security Groups.

Create a Positive Security Group

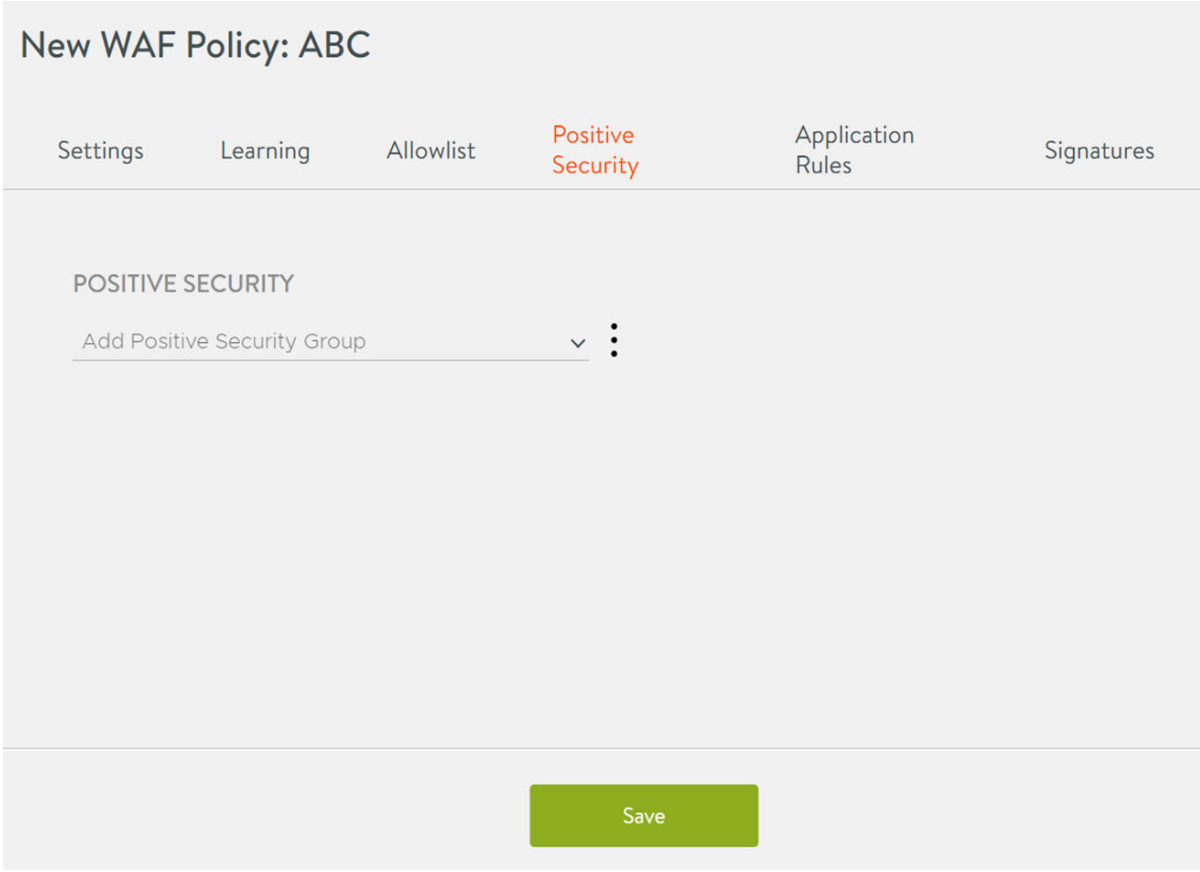
This topic details the steps to create a Positive Security Group.

Procedure

1. From the NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Policy**.
2. Click **Create** or **Edit** an existing **WAF Policy**.

3 Enter the required details under the **Settings** tab.

4 Click the **Positive Security** tab.



5 Create a new Positive Security Group by clicking on the more icon (three dots) and clicking **Create**.

6 In the **New Positive Security Group** screen, enter the details as shown below:

Field	Description	Additional Information
Group Enabled	Toggle this slider to enable or disable this WAF rule group.	
Name	Enter a relevant name for the policy.	
Description	Enter a description to identify the group.	
Learning Group	Select this check box to enable learning for the group.	This field cannot be changed after the group is created.
Hit Action	Select **Allow parameter** or **No operation** from the drop-down menu.	If a rule in this group matches the `match_value` pattern, this action will be executed.

Field	Description	Additional Information
Miss Action	Select either **Block** or **No operation**from the drop-down menu.	If a rule in this group does not match the `match_value` pattern, this action will be executed.
Location	Click **Add Location** to add a location and configure Match Type and Argument Rules for this Positive Security Group. For more information on adding a location, see Creating a Location.	Positive Security rules are created in locations. Locations are derived from URLs.

7 Click **Save**.

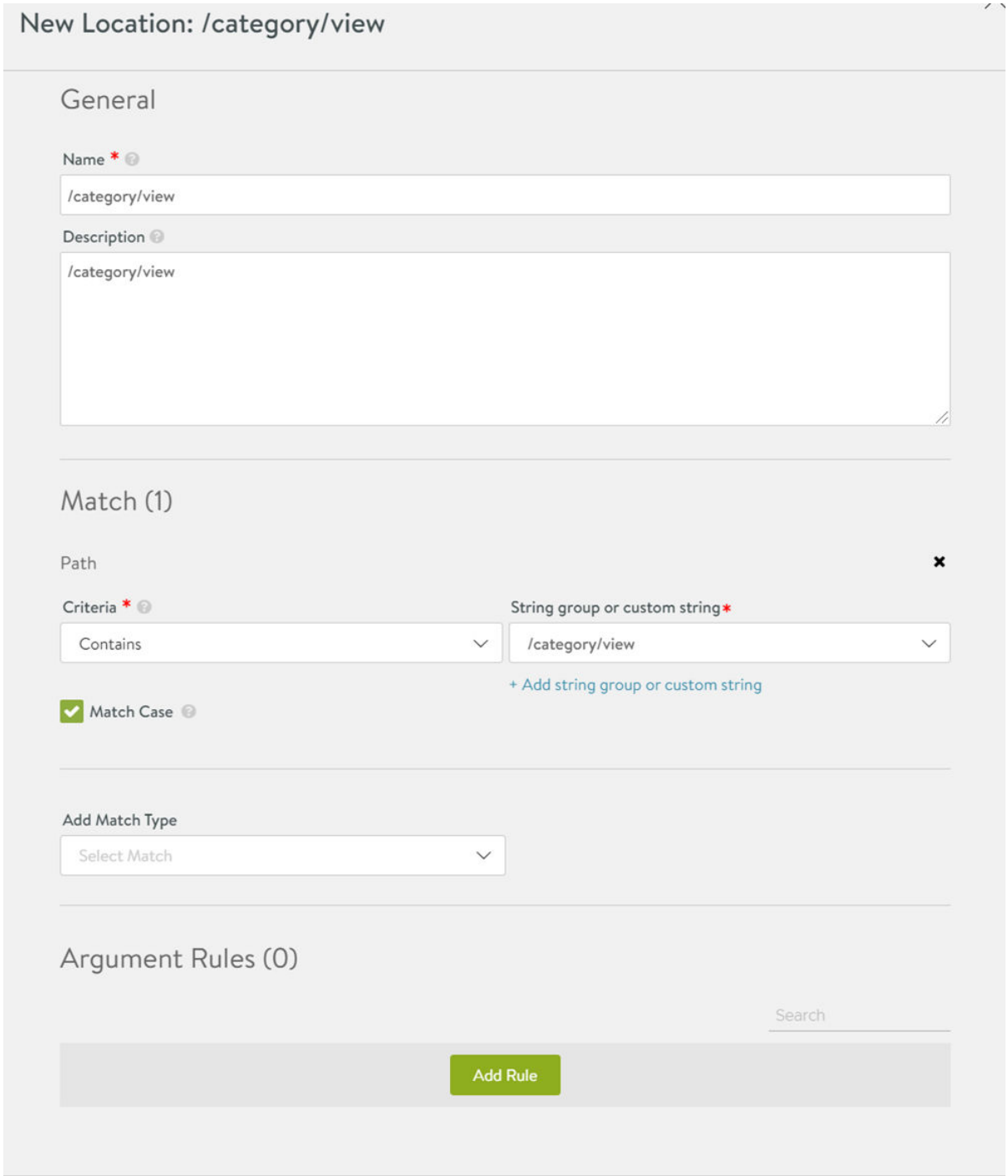
Creating a Location

This section details the steps to create a Location for a Positive Security Group.

Enter details in the **New Location** screen as shown below:

Procedure

1 Enter a unique **Name** to identify the location.



2 Enter the **Description**.

3 Select a **Match Type**, for example, **Path**.

4 In the **Criteria** field, select the criterion to use for matching the HTTP request in the URI.

5 Enter a string value in **Custom String**.

6 Select **Match Case** to enable case sensitivity.

7 To add another match type, select one from the **Add Match Type** drop-down menu.

8 Click **Add Rule** to create a new rule.

9 Click **Save**.

Creating an Argument Rule

This section discusses creation of an Argument Rule.

In the **New Argument Rule** screen, do the following:

Procedure

1 Click the **Rule Enabled** toggle button to enable or disable the rule. The Rule is enabled by default.

2 Enter a **Rule ID** that is unique for this group.

3 Enter the rule **Name**.

4 Enter a **Description** for the rule.

5 Select one of the following:

- **Use Policy Mode**.
- **Detection**.
- **Enforcement**.

  For more information on choosing a mode, see Selecting a WAF Policy Mode

6 Select a Paranoia mode. For more information, see What are the Paranoia Modes available in WAF? What are the considerations for choosing the mode? in FAQ section. This defines the Paranoia mode (set by the overall policy) in which this rule gets executed.

7 Define the **Match Elements** as shown below:

a Enter the **Value Max Length** to define the maximum length of the match value.

b Select a **Value** to identify the expression which describes the expected value. This value can be a **Pattern** or **String Group/Key**. To know more about String Groups, see String Groups Support.

c Enable **Arguments Case Sensitive**, if required. This ensures that the match value has the same case as specified in the match value pattern.

8 Click **Add Match Element** and define the match elements as shown below:

   a In the field **Name**, select the variable collection. This is a dictionary of all parsed parts of the incoming request. If the match must happen on a POST argument, choose **ARGS**. The drop-down menu gives all available options.

   b Select the criteria to match from the **Criteria** drop-down menu. Criteria is the method for locating Match Element. Equals indicates that the provided **Sub Element** must be equal to the corresponding request parameter.

   You can choose other methods. For example, the regular expression match interprets the **Sub Element** as a regular expression.

   c Enter a **Sub Element**. This is the name of the element you are matching on. If the match should be on a parameter `foo`, enter `foo` in the **Sub Element** field.

   d Select the **Excluded** check box, if you need to exclude the element mentioned under **Name** and **Sub Element**. This negates the match.

   e Select **Case Sensitive** check box for a case-sensitive match.

9 Click **Save**.

String Groups Support

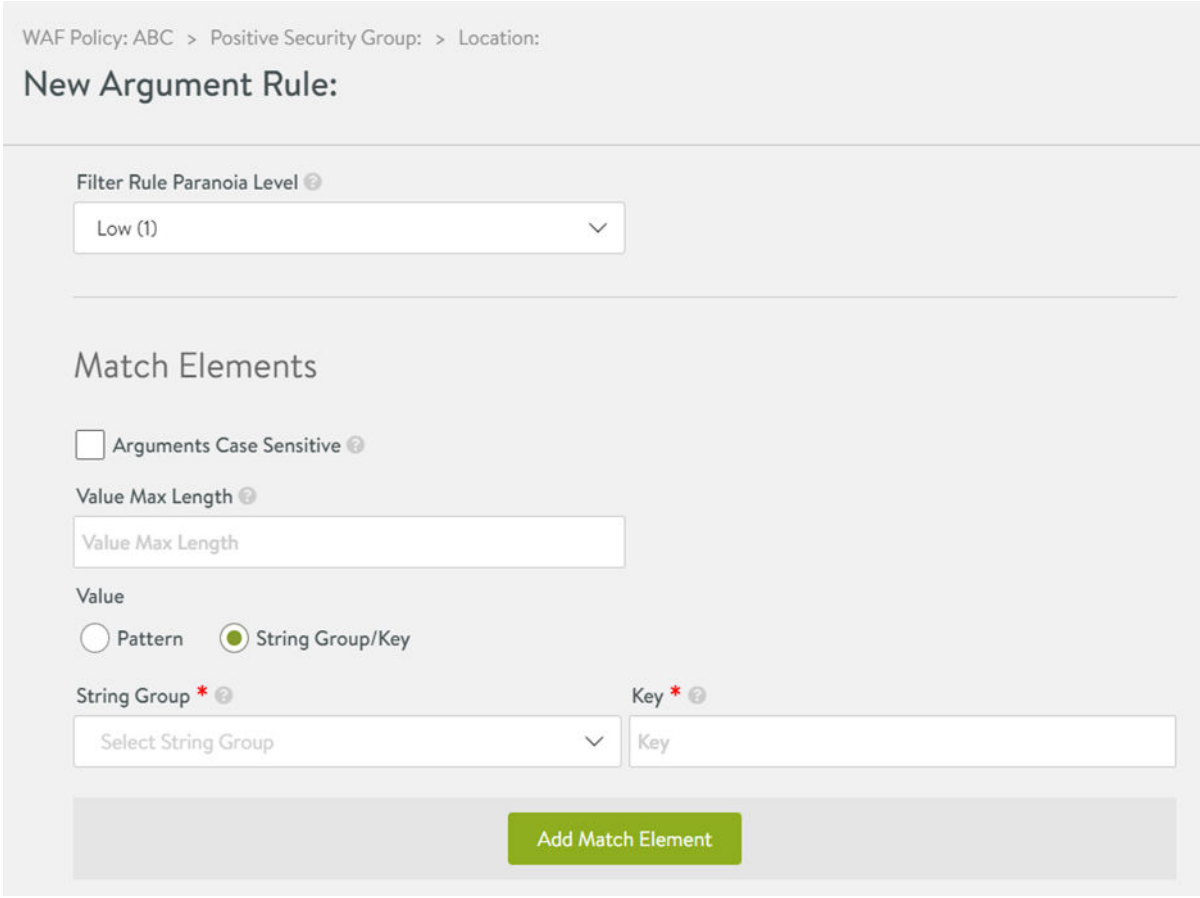
The NSX Advanced Load Balancer supports String Groups in addition to the match value pattern. The following topic explains configuration of new String Groups for Argument Rules.

The String Group consists of the following parts:

- String Group – UUID of the String Group containing key used in the match element.
- Key – PCRE-supported regular expression.

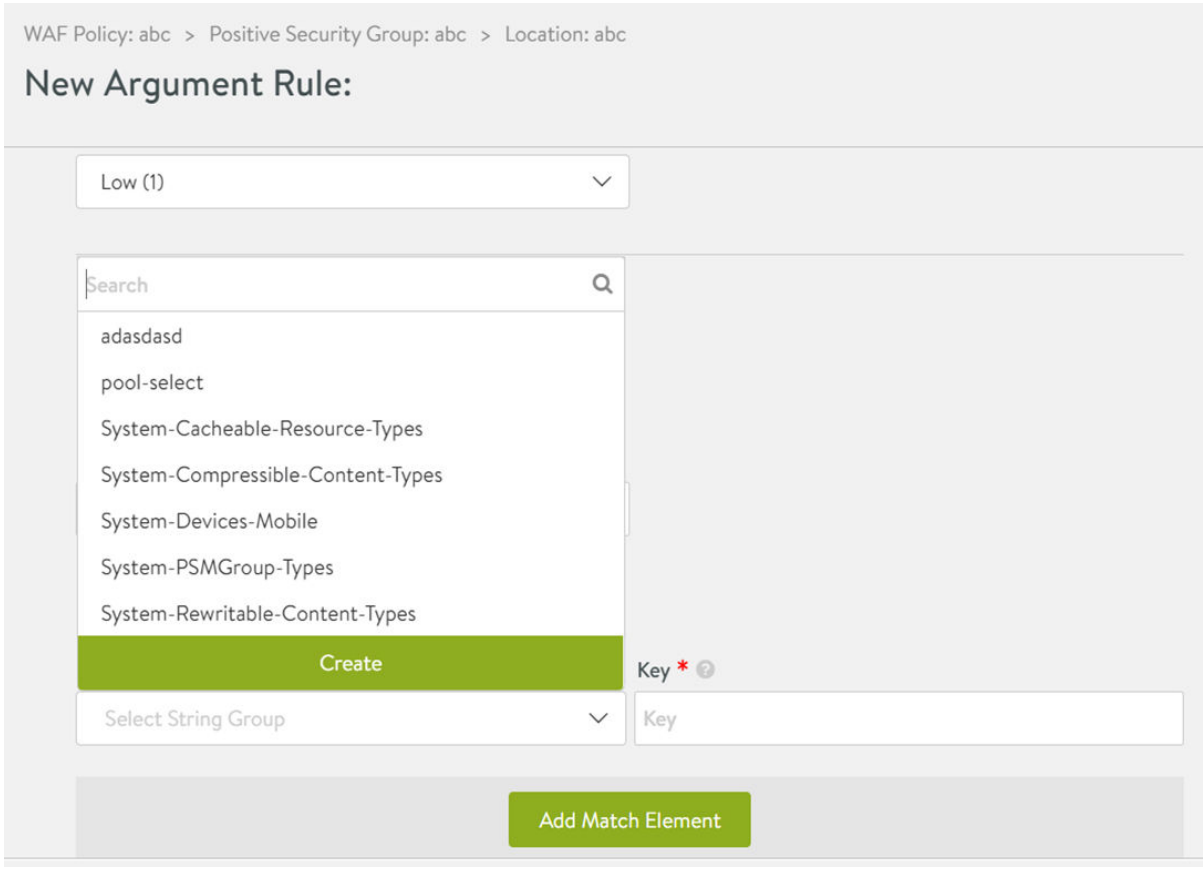
1 Navigate to **Templates > WAF > WAF Policy > Positive Security tab**.

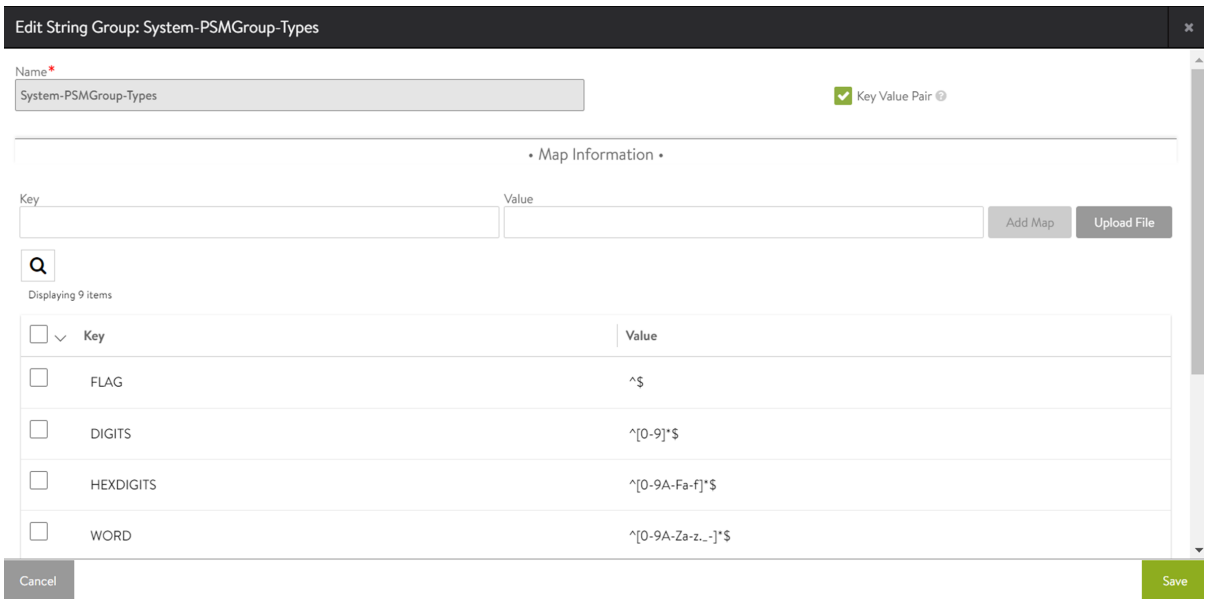
2 The option to use String Groups available under **Match Elements** while creating a **New Argument Rule** is shown below:



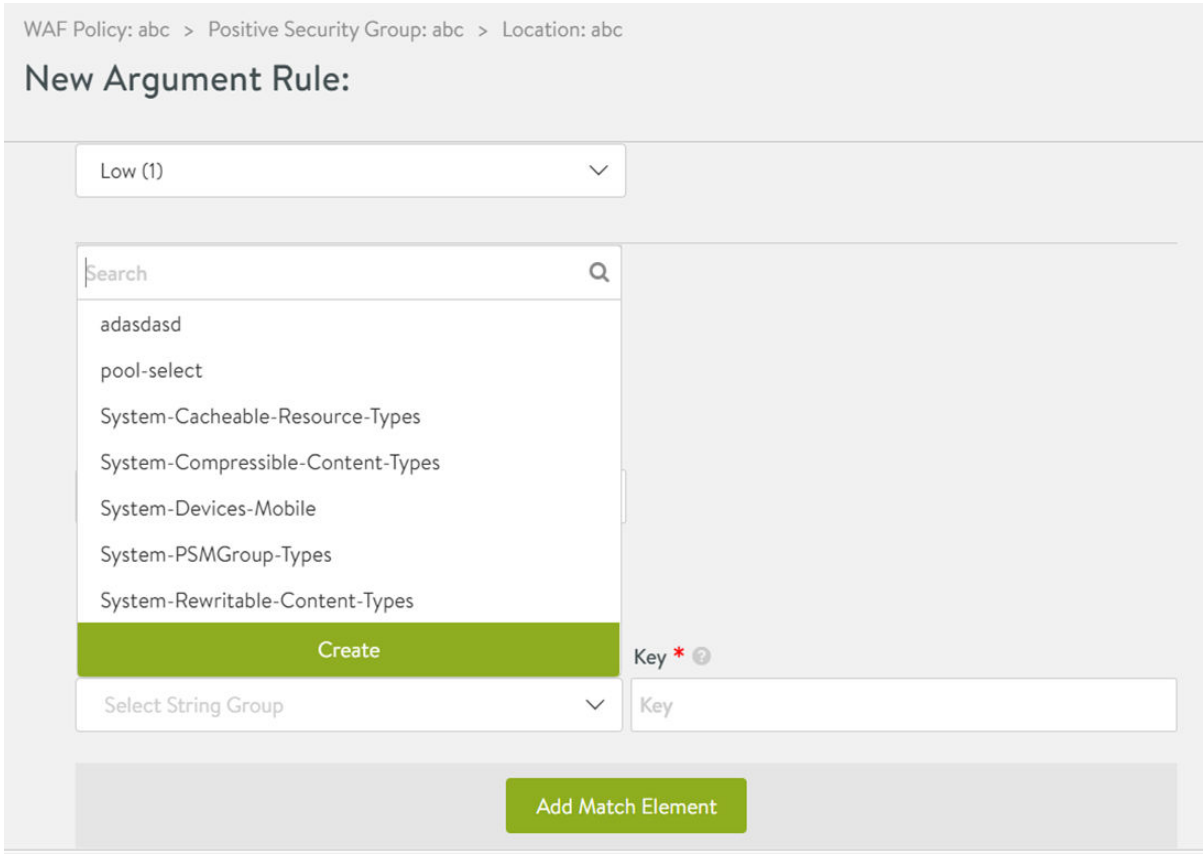
3 For the String Group, select the default **System-PSMGroup-Types** from the drop-down menu or create a new String Group.

4 For the default **System-PSMGroup-Types**, select one of the **KEY NAMES** as shown below:

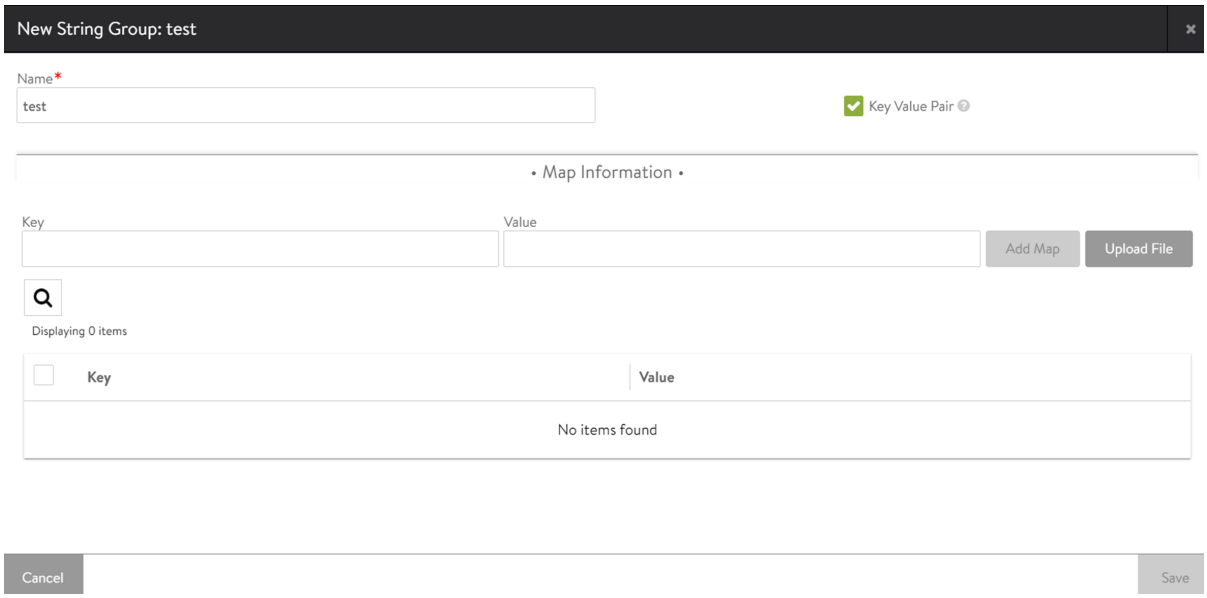




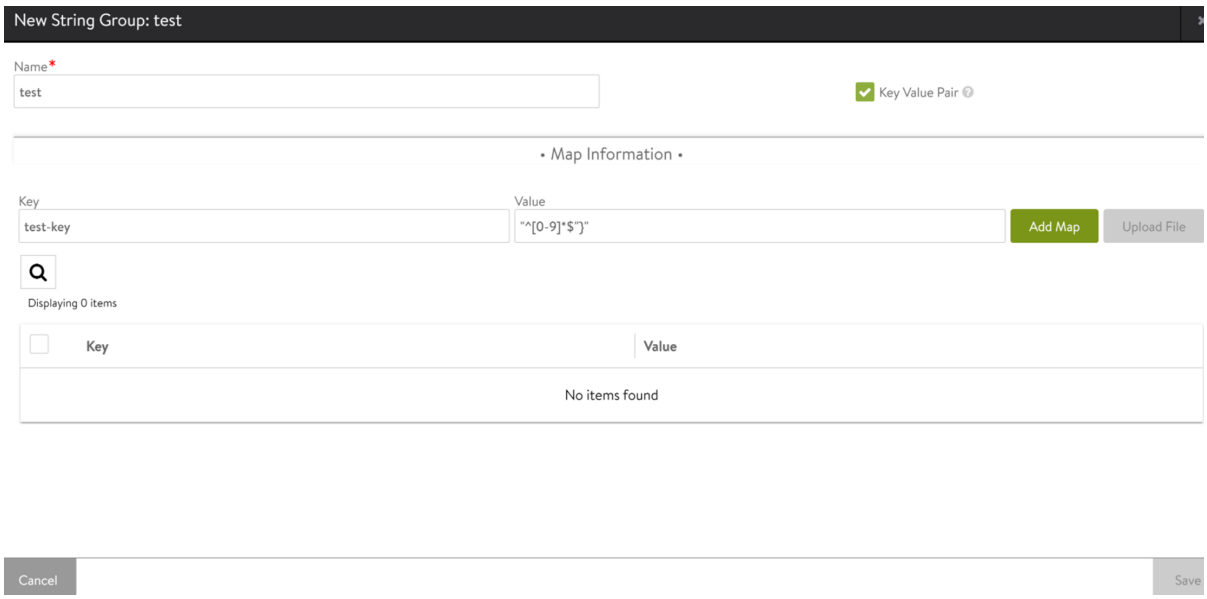
5 To create a new String Group, select **create** from the drop-down menu as shown below:



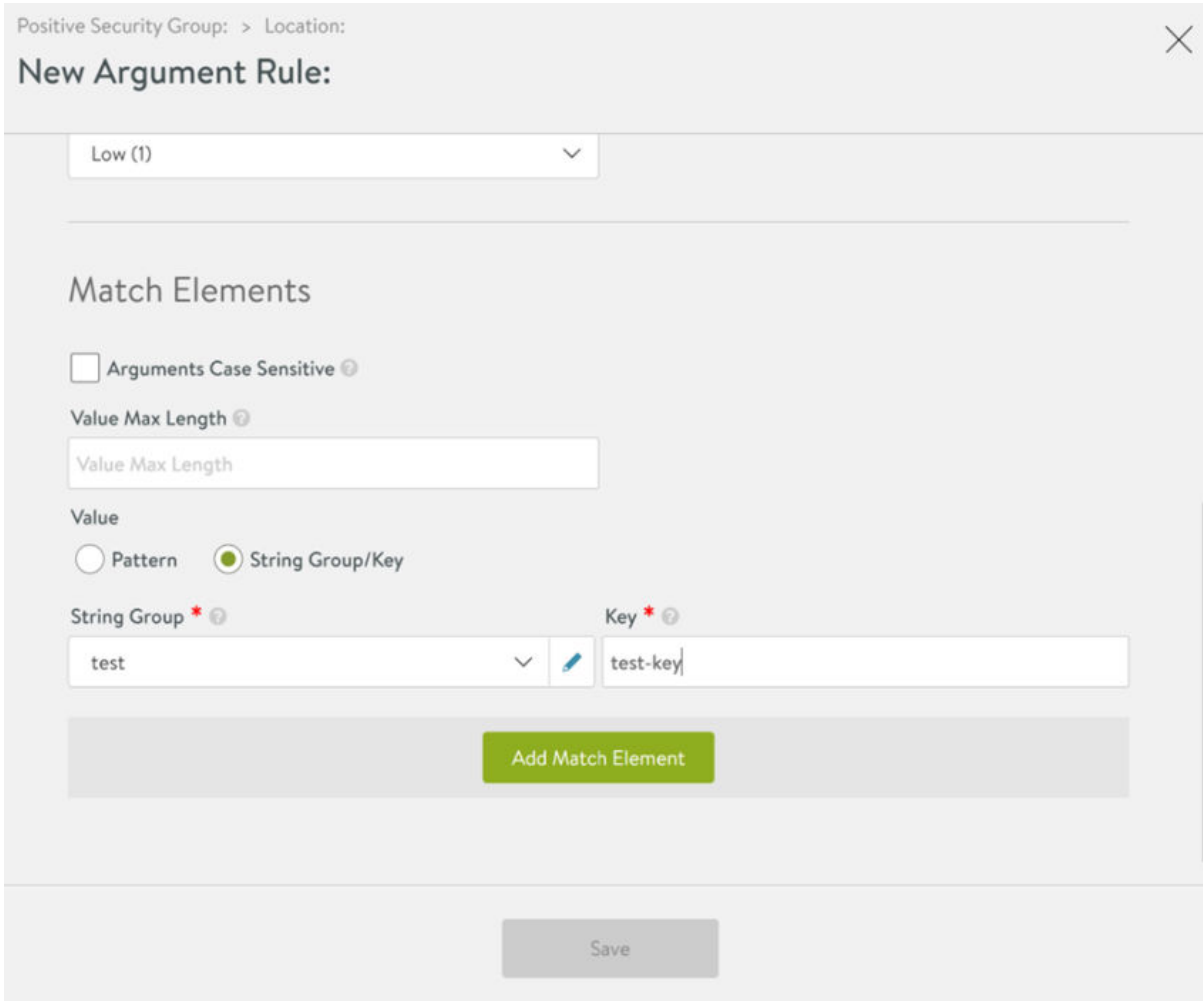
6 Provide the **Name**, select the check box for **Key Value Pair**.



7 Provide the name of the new key, enter a PCRE supported expression under the **Value** field, and click **Add Map** option as shown below:



8 Provide the name of the key created in the previous step as shown below.



Note The maximum number of String Groups that the NSX Advanced Load Balancer can support is 100. A String Group supports a maximum of 1000 key values.

Configuring CRS Rules for WAF Signatures

Core Rule Set (CRS) is a set of protection rules that the WAF Policy uses. This section discusses how to configure and update CRS rules.

WAF Signatures is a security service delivered through Cloud Services. The WAF Signature service is based on Opt In basis, that is disabled by default.

For more information on configuring WAF Signatures in the NSX Advanced Load Balancer through auto and manual modes, see Web Application Firewall (WAF) Signatures Service in the Cloud Services Guide.

Updating CRS Rules

If the CRS version is updated, all new CRS rules will be in **Detection** mode. With this, you can update the CRS ruleset without any risk in production. However, these new rules must be moved into **Enforcement** mode (or **Use Policy Mode**) manually.

Note

- This feature works only if you have the NSX Advanced Load Balancer Enterprise License for Cloud Services.
- For Controllers setup in the ENTERPRISE_WITH_CLOUD_SERVICES tier, the WAF Signatures Service automatically pushes new rules to Controllers where this service is enabled.
- For Controllers without this setup, you can opt-in to receive notifications with download links when new rules become available. Once downloaded, the rules can be manually uploaded to the NSX Advanced Load Balancer Controller.

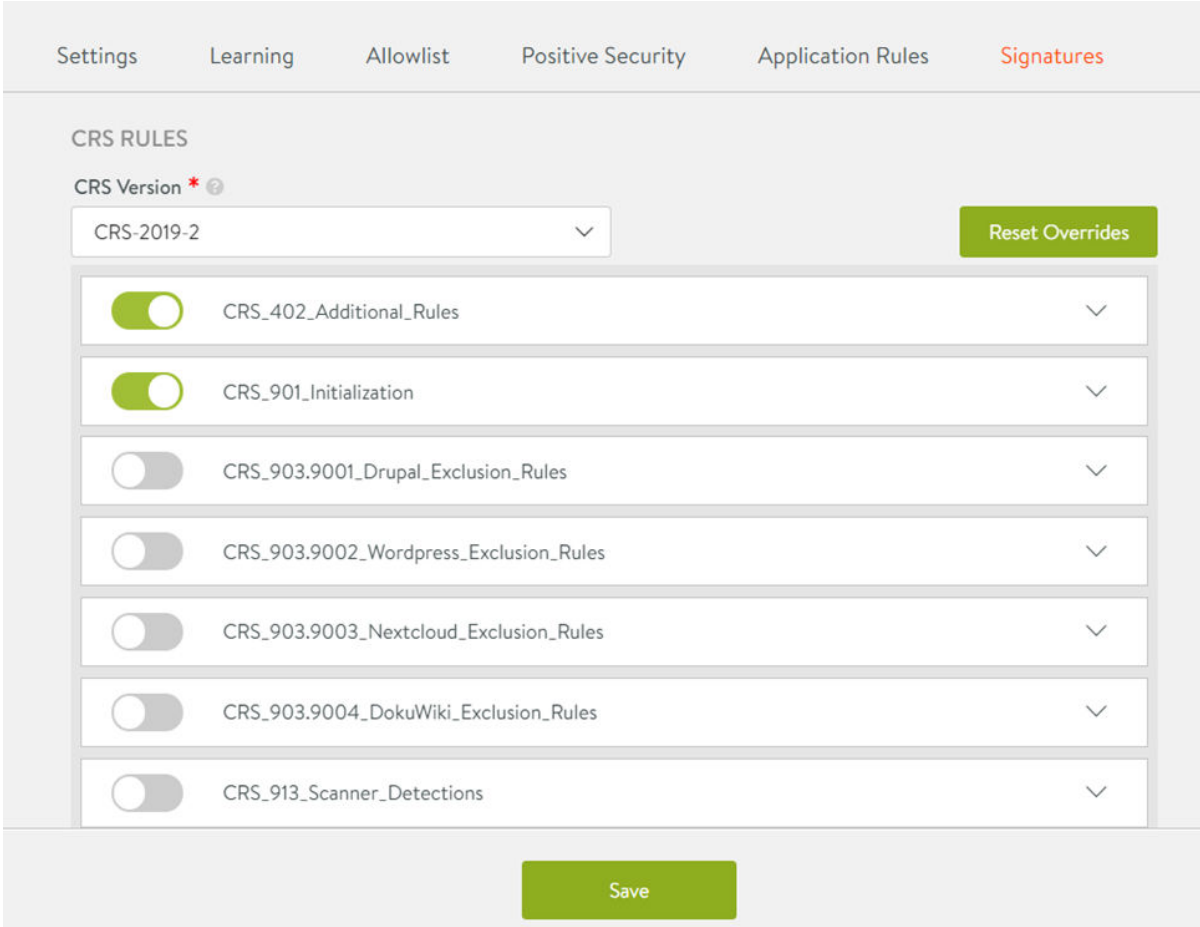
All updated rules will continue to remain in the same mode and the existing Exceptions will be applied to the rules.

To update CRS Rules do the following:

1 Navigate to **Templates > WAF > WAF Policy**.

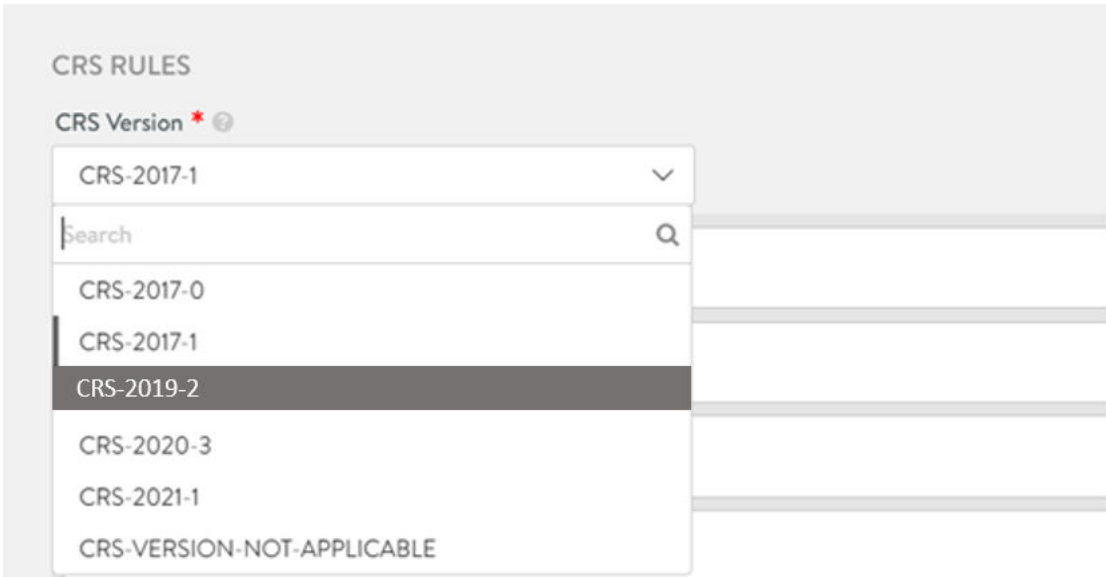
2 Click the **Edit** icon for the WAF Policy.

3 Under the **Signatures** tab, scroll down to the **CRS Rules** section.

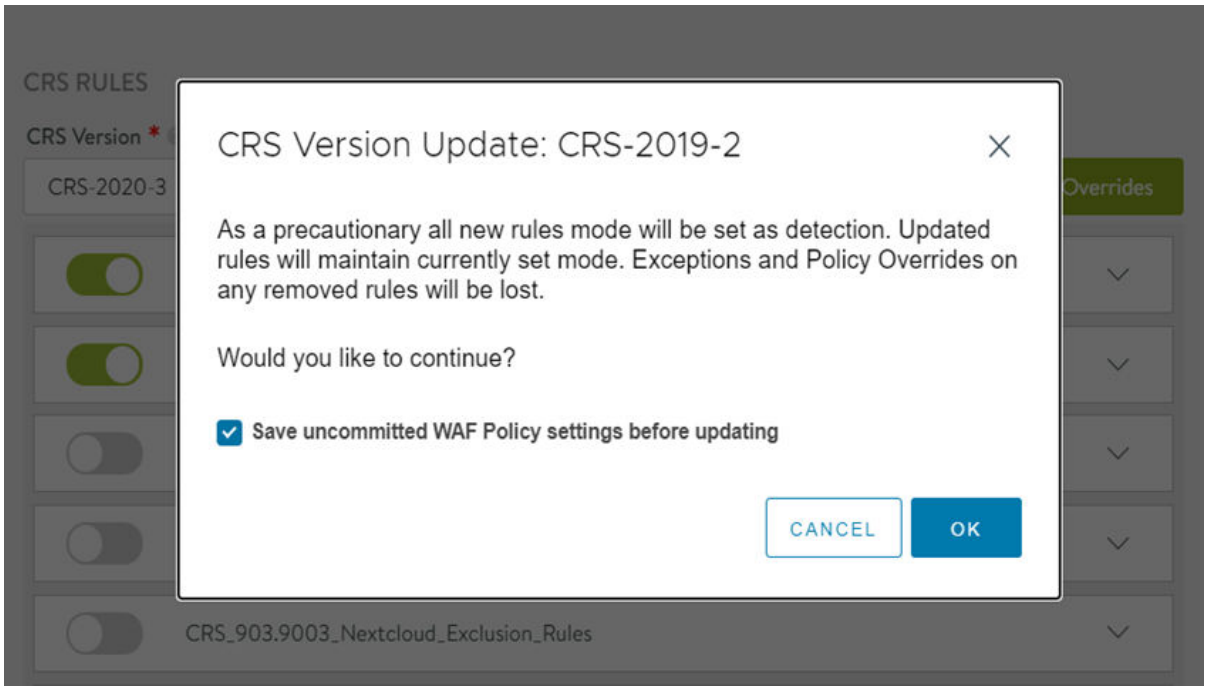


Note The **Reset Overrides** button resets all rule mode changes back to inherited mode. It is very useful after a CRS update, when new rules have been tested and are now ready to be part of the WAF policy mode.

4 Select the required **CRS Version** from the drop-down menu.



5 The change log is displayed as shown below. Click **OK** to confirm and update the CRS version.



The final step in WAF processing is a Signature check. CRS can be configured under the **Signatures** tab. You can configure to execute custom rules before CRS (Pre-CRS rules) or after CRS (Post-CRS rules).

Note When using features like Anomaly Detection, the CRS Group *CRS_901_Initialization* must be enabled, without which required anomaly thresholds are not configured to the defaults. It is generally recommended to keep this group enabled.

Exceptions

Exceptions are a common way of tuning a WAF Policy to work with an application. The following section lists common use cases for creating Exceptions. It also explains how to configure Exceptions, the recommended workflow, and the Match Elements and XML Exceptions that can be configured as part of it.

These are created when the regular traffic of the application matches specific WAF rules. The following are a few other reasons for creating Exceptions:

- For false positive mitigation.
- For applications transmitting data that might appear like an attack. For instance, transferring HTML content in query parameters.
- For applications with special requirements that are not allowed in the WAF Policy. For instance, accessing an application using its IP address.

- You can use NSX Advanced Load Balancer Recommendation system to create Exceptions or add them manually. For more information, see Reviewing Recommendations to Remediate False Positives.

To define an Exception manually.

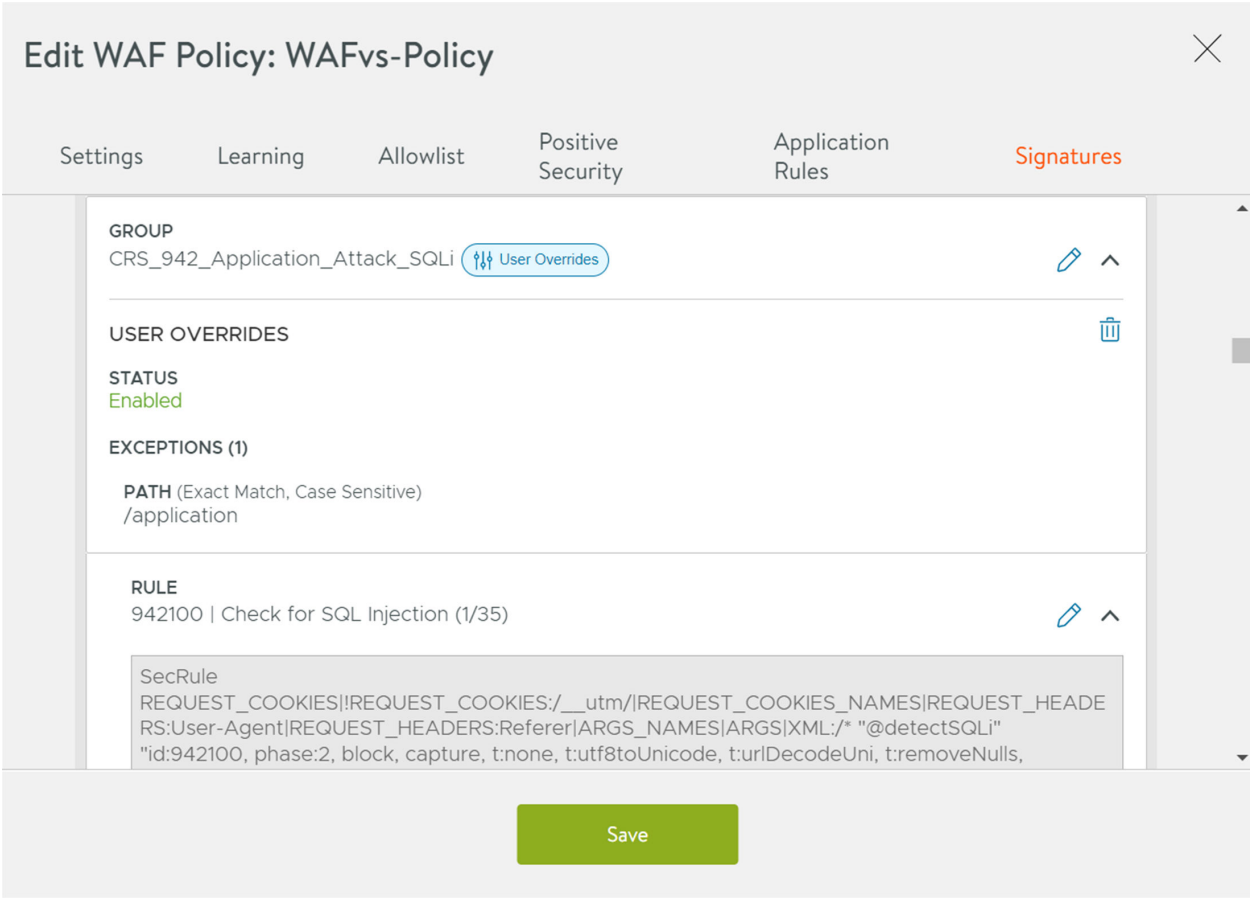
- Click **+Add Exception** to manually configure Exceptions.
- Configure Exceptions for IP address or subnet, path, or any match element. For example,

```
Subnet- 10.0.0.0/8, Path- /application , Match Element - REQUEST_BODY
```

- Configure the following options for Path and Match Element, as required:
  - **Case Sensitive** - The case of the characters have to match.
  - **Regex Match** - The pattern of the string of characters have to match.

Note Exceptions can be created on a CRS group or rule level.

The rule configured with Exception **PATH** - `/application` is as shown below:



Supported Match Elements

Exceptions can be created for the following match elements:

ARGS, ARGS_GET, ARGS_POST, ARGS_NAMES, FILESQUERY_STRING, REQUEST_BASENAME, REQUEST_BODY, REQUEST_URI, REQUEST_URI_RAW, REQUEST_COOKIES, REQUEST_HEADERS, RESPONSE_HEADERS, XML.

For example, creating an Exception for *ARGS:password* at a WAF rule level implies that the rule will not examine the *password* HTTP parameter (sent in URL or request body - as JSON, XML or HTML form). The rule will continue checking other parts of a HTTP request that are not specified as Exceptions.

XML Exceptions

WAF rules use an XML variable (with an XPath expression) to specify XML request body fragments that must be examined.

Example:

```
SecRule XML:/* text_to_match "id:1,phase:2,log,deny,status:403".
```

If a text representation of an XML request matches text_to_match, the WAF blocks the request. XML Exceptions are also specified using the XPath expressions, for example, `'XML:/*'`.

WAF rules use `XML:/*` expression to enable processing of the whole XML request body. Using XPath expressions in WAF Exceptions is a powerful way to configure WAF to skip checks for the whole XML document or to selectively exclude only parts of the XML document from WAF inspection.

In a common scenario, when a CRS rule generates a false positive WAF hit, it must be sufficient to follow the steps described in the Recommended Assisted Workflow section to disable XML processing by a given rule. It is also possible for a user to manually configure an exception and specify the XPath expression to exclude parts of the document from WAF rules processing.

Example: XML request body:

```
<example>
<username>joe</username>
<password>!@#$%</password>
</example>
```

The WAF rule generates a hit:

match element: XML:/*

When following the Recommended Assisted Workflow option, the match element XML:/* is used to disable processing of the whole XML document from the rule 12345. You can also create a rule-level exception and specify the match element as `XML:/example/password` to exclude only an element responsible for a WAF hit. As a result, WAF will continue examining of the remaining parts of the XML document.

Pre-CRS Rules

This section explains how to configure pre-CRS rules.

The custom rules that are applied before the supplied OWASP CRS are called Pre-CRS rules. For more information, see Custom Rules Examples.

To define Pre-CRS rules do the following:

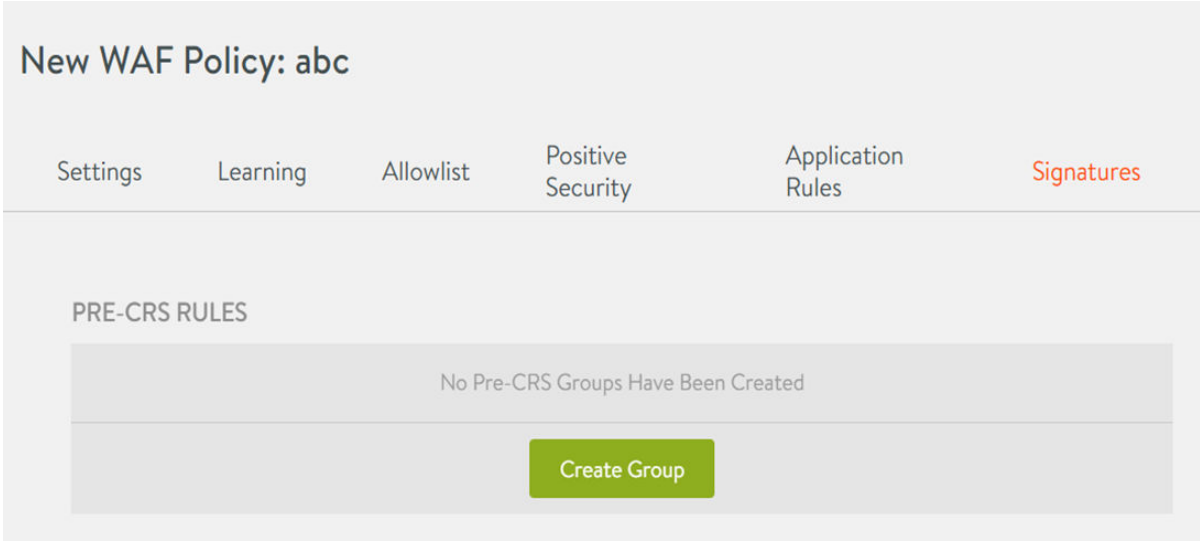
1 From the NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Policy**.

2 Click **Create** Or Edit an existing **WAF Policy**.

3 Enter the required details under the Settings.

4 Click the **Signatures** tab.

5 Under **Pre-CRS rules**, click **Create Group**.



6 Enter the **Group Name**. Every rule is configured within a group.

7 Click the **Create Rule** button. Rules are enabled by default. To disable a Rule, deselect the **Enable Rule** check box.

8 Enter a **Name** for the rule.

9 Select one of the following options:

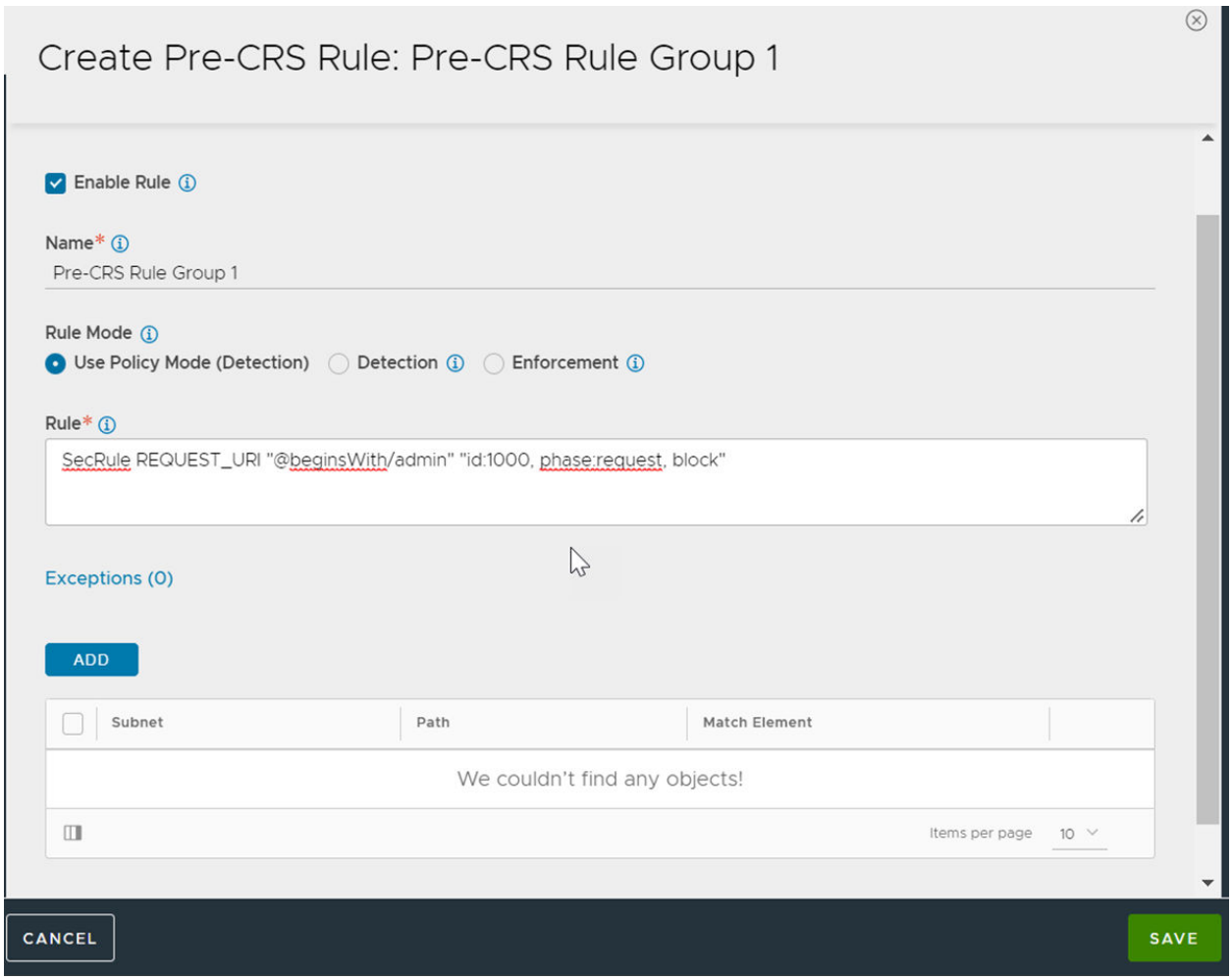
   a Use Policy Mode

   b Detection.

   c Enforcement.

   For more information on selecting modes, see Selecting a WAF Policy Mode

10 Enter the **Rule** in the text box. The Rule is specified in Modsec language.



Post-CRS Rules

This section lists the steps to configure custom rules to be applied after the supplied OWASP CRS is configured.

To configure Post-CRS rules:

1 Under the **Signatures** tab, scroll down to the **Post-CRS Rules** section.

2 Create **Groups** and **Rules** as discussed in the Pre-CRS Rules section.

3 Click **Save**.

Mixed Mode and Enabling Mode Delegation

WAF Policy can be configured to operate in either Detection or Enforcement mode.

With the Mode Delegation option in NSX Advanced Load Balancer, the policies can be enabled to operate in Detection and Enforcement modes. For more information on enabling mode delegation, see Mode Delegation

WAF Rate Limiting

Though rate limiting is primarily done outside WAF, owing to specific customer requirements, it is also included as part of WAF. The following topic explains Rate Limiting with respect to WAF.

Use Case for WAF Rate Limiting

The main use case addressed as part of Rate Limiting in WAF is to limit the calls to the `/login` action to 10 requests per minute per IP.

To address this need, the *@ratelimit* operator is implemented as an extension within the rule language.

The operator *@ratelimit* accepts 3 arguments (macros not supported).

- requests - (number)
- time unit - (number) + (time unit (s, m, h))
- burst size - (optional number)

The rate limiter is uniquely identifiable by virtual service and rule-id. So every rule gets its own rate limiter. The rate limiter key is the *match_element_name* and the *match_element_value*.

The current use case is to limit the number of POST requests to /login to 10 requests per minute per IP. An example rule can be as follows.

```
SecRule REQUEST_METHOD "^POST$" "id:42,phase:1,t:none,block,chain"
    SecRule REQUEST_URI "@contains /login" "t:none,chain"
       SecRule REMOTE_ADDR "@rateLimit 10 1m"
```

The following is the explanation of the rule.

- If the request is a POST request
  - AND the URI contains `/login`
    - For every IP (REMOTE_ADDR), limit the number of request to 10 per 1m (1 minute).
    - If this limit is exceeded, `block` the request.

Best Practices for Working with WAF

3

This section discusses the best practices for working with WAF.

This chapter includes the following topics:

Creating Exceptions

This section explains creation of Exceptions.

Exceptions in WAF

This section discusses Exceptions in WAF.

Exceptions allow for tuning a WAF Policy to work with an application. They are generated when the regular traffic of an application and the configured WAF rules match.

The following are a few reasons for creating Exceptions:

1. Applications do not conform with the **System-WAF-Policy**.
2. The application transmits data that resembles an attack to the WAF. For example, transferring HTML content in query parameters.
3. The application has special requirements that are not allowed in the WAF Policy. For example, accessing the application on their direct IP address.

Recommended Assisted Workflow

The steps in mitigating a false positive are given below:

1 Identify a potential false positive.

> **Note** False positives can occur in large numbers for different client IP addresses.

2 Eliminate the false positive by adding an Exception to the rule.

3 Exceptions can be created either at a group or rule level. Exceptions are activated immediately after they are created.

Create Exceptions

This section discusses Creating Exceptions for WAF policies.

To create Exceptions:

Procedure

1 From the NSX Advanced Load Balancer UI, navigate to **Applications > Virtual Services**.

2 Click the **Virtual Service** mapped to the WAF Policy and navigate to **Logs**.

3 Filter the WAF log analytics. You can analyse the WAF logs based on parameters like the client IP, URI, the type of request, etc.

4 **WAF Hits** displays all the rules that were matched.

5 Click **+Add Group Exceptions** or **+Add Rule Exception** to create an Exception for a false positive remediation.

6 **Save** the Exception.

Alternatively, Exceptions can be manually defined for a group or a rule within the WAF Policy. This can be done at the Pre-CRS, CRS, or Post-CRS levels.

In the following example, HTML is added through the parameters.

Request	Match Element	False Positive Reason
POST /foo/bar_form.php HTTP/1.1 Host: boofar.com name1=value1&name2=value2&img= <img+src='/images/foo.png'>	ARGS:img	XSS rules match "<img...

Workflow for Mitigation

In order to mitigate for false positives, you need to exclude the parameters from being checked for XSS at the Rule Group level. The following is a typical mitigation workflow that an administrator user follows to handle false positives using Exceptions.

Admin is aware that many requests are denied.

Procedure

1 Scanning the App log analytics shows requests from many IPs that got blocked because of the offending `ARGS:img`.

2 Clicking the offending parameter opens **Analytics** and shows that this `ARGS:img` had n number of denied requests the previous day.

3 Admin identifies this as a standard functionality within the application (might ask Dev team).

4 Admin clicks **Add Exception**.

5 Admin chooses one or more of the suggestions: parameter, IP and/or parameter.

6 New Exception (`NONE, "foo/bar_form.php", ARGS:img`) is put in place for the CRS XSS rule group.

Results

False positives are handled using Exceptions.

WAF Exceptions with Regex Matching for Arguments

This section discusses configuring WAF exceptions with regex matching for arguments. NSX Advanced Load Balancer supports Regex for URL matching.

NSX Advanced Load Balancer WAF uses PCRE (Perl Compatible Regular Expressions) as regex. NSX Advanced Load Balancer supports configuring regular expressions for arguments. The **match_element** field under WAF Policy can be configured to use a regular expression instead of just a keyword.

Configuring Regex Matching for Arguments through CLI

1 Login to the Controller shell and enter the command to edit the required WAF policy.

2 Search for `match_element_criteria` by typing slash ('/'), followed by the keyword `match_element_criteria`.

3 Configure the desired regex in the `match_element` field as shown below. Under `match_element`, set the `match_case` field to `SENSITIVE` and the `match_op` field to `REGEX_MATCH`.

```
exclude_list:
       - match_element: ARGS:regex
         match_element_criteria:
           match_case: SENSITIVE
           match_op: REGEX_MATCH
         uri_match_criteria:
           match_case: SENSITIVE
           match_op: REGEX_MATCH
         uri_path: ^/test.php
```

4 Hit `Esc` and enter `:wq`.

5 Type `save` to save the configuration.

```
save
```

Example: Configuration Example

The argument name can have several fixed and dynamic parts. Consider an example of an URL as follows:

`https://appname.com/typo/test_doc.php?data[news][1234][body]=Some_long_text_expected`

Here, `data[news][" and "][body]` are the fixed parts and the number `[1234]` is a dynamic value that varies with each request. An example attack on this application will be as follows:

`https://appname.com/typo/test_doc.php?data[news][1234][body]=%3Cscript%3Ealert(1)%3C/script%3E`

The regex required for creating an exception for this example would be:

```
URL Regex:              ^/typo/test_doc.php
Match element Regex:    ARGS:.data\[news\]\[.*\]\[body\]
```

Use `ARGS:.data` instead of `ARGS:data` to make this a valid regular expression.

The WAF Policy configuration would be as follows:

```
exclude_list:
 - match_element: ARGS:data\[news\]\[.*\]\[body\]
   match_element_criteria:
     match_case: SENSITIVE
     match_op: REGEX_MATCH
   uri_match_criteria:
     match_case: SENSITIVE
     match_op: REGEX_MATCH
   uri_path: ^/typo/test_doc.php
```

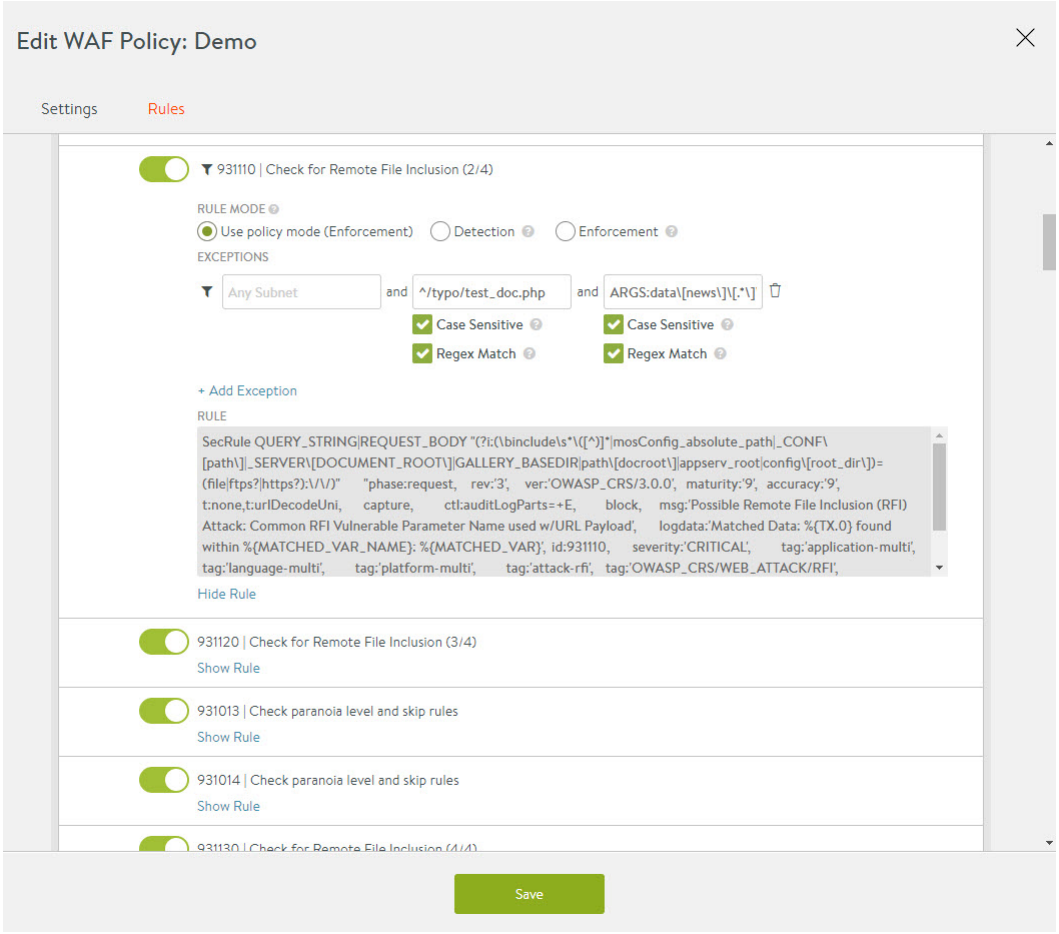
Configuring Regex Matching for Arguments through UI

1 On the NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Policy**. Click on the policy to be edited or create a new policy as required.

2 Under the **Rules** tab, navigate to the relevant rule under the rule sets. Click the drop-down menu for a rule to expand the configuration options. Click **+ Add Exception** to configure the exception.

3 Under the **EXCEPTIONS** field enter the regular expression and select the check box for **Regex Match**.

4 Save the configuration.



Importing the CRS files and CRS Update from NSX Advanced Load BalancerPortal

This topic discusses the steps to download the CRS JSON file from the customer portal and upload it to NSX Advanced Load Balancer.

Though multiple CRS versions are supported by the NSX Advanced Load Balancer, only curated CRS files that can be downloaded from the customer portal are supported.

Download the CRS File

When CRS updates become available, an event is generated on the NSX Advanced Load Balancer Controller with a signed download link.

For steps to enable CRS update notifications and download the updated CRS files, see Service Details in the Cloud Services Guide.

Upload the CRS File

For steps to upload the CRS file in the Controller, see Service Details in the Cloud Services Guide.

Custom Rules

This topic lists possible use cases for configuring custom rules.

WAF supports custom rules that can be added for any application-specific use case or for other custom requirements. Custom security rules are based on the ModSecurity language. For more information about the ModSecurity language, see OWASP ModSecurity Core Rule Set.

Custom rules can be configured and executed Pre-CRS and Post-CRS. For more information, see Pre-CRS Rules and Post-CRS Rules.

Custom Rules Examples

This section provides a list of examples for Custom Rules.

Bypassing WAF

You can bypass certain requests from going through WAF.

The following are a few ways to bypass WAF:

Through Content Length

WAF can be bypassed if the content length is greater than the defined value.

Custom Rule Syntax:

```
SecRule 'Variable:"value"' "phase:1,id:4000100,nolog,pass,ctl:ruleEngine=off"
```

Example:

In this example, if the value of the content-length header is greater than **1048576**, the request skips WAF.

```
SecRule REQUEST_HEADERS:Content-Length "@gt 1048576"
"phase:1,id:4000100,nolog,pass,ctl:ruleEngine=off"
```

Through Chunked Transfer Encoding

WAF can be bypassed based on the transfer encoding type.

Custom Rule Syntax:

```
SecRule 'Variable "@match criteria"'""
```

Partial buffering for chunked-encoded payload is supported. The remaining payload is streamed while maintaining the original chunk boundaries sent from the client.

Example:

In this example, if the form of encoding used to transfer is chunked, the request skips WAF.

```
SecRule REQUEST_HEADERS:Transfer-Encoding "@contains chunked"
"phase:1,id:4000101,nolog,pass,ctl:ruleEngine=off"
```

Based on Specific Patterns of the Requested Path

WAF can be bypassed according to certain patterns of the requested path.

Custom Rule Syntax:

SecRule 'Variable:"value"""id:4000102,phase:1,t:none,pass,ctl:ruleEngine=off"

Example:

In this example, any request that begins with the string “/IDMProv/login.do” will bypass WAF.

```
SecRule REQUEST_URI “@beginsWith /IDMProv/login.do”
“id:4000102,phase:1,t:none,pass,ctl:ruleEngine=off”
```

Allowlist Requests

The following section explains how to add requests that match certain conditions to the Allowlist.

Custom Rule Syntax:

```
SecRule 'Variable "@match criteria"'
"id:4000104,phase:1,t:none,pass,ctl:ruleEngine=off,chain" SecRule REMOTE_ADDR "@ipMatch
10.0.0.0/8" "t:none"
```

Example:

In this example, all requests from `10.0.0.0/8` to all URLs starting with `/admin` are added to Allowlist. Since there are two conditions to be fulfilled, a chain rule is used.

```
SecRule REQUEST_URI "@beginsWith /admin"
"id:4000104,phase:1,t:none,pass,ctl:ruleEngine=off,chain" SecRule REMOTE_ADDR "@ipMatch
10.0.0.0/8" "t:none"
```

Enabling Customizable XSS Keywords

WAF protects against XSS attacks. This section details steps for enabling customizable XSS keywords to perform a case-insensitive match of the XSS keywords and block them.

Custom Rule Syntax:

```
SecRule 'variable "@pmfromfile xss-keywords.data"' "msg:'Node-Validator Blacklist Keywords',
id:4099802, severity:'CRITICAL', phase:request,
t:none,t:utf8toUnicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,t:lowercase,t:
removeNulls, rev:'2', ver:'OWASP_CRS/3.0.0', maturity:'1', accuracy:'8', block,
ctl:auditLogParts=+E, capture, tag:'application-multi', tag:'language-multi', tag:'platform-
multi', tag:'attack-xss', tag:'OWASP_CRS/WEB_ATTACK/XSS', tag:'WASCTC/WASC-8', tag:'WASCTC/
WASC-22', tag:'OWASP_TOP_10/A3', tag:'OWASP_AppSensor/IE1', tag:'CAPEC-242', logdata:'Matched
Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}', setvar:'tx.msg=%{rule.msg}',
setvar:tx.xss_score=+%{tx.critical_anomaly_score}, setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score}, setvar:tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/XSS-%
{matched_var_name}=%{tx.0}"
```

Example

```
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/*
"@pmfromfile xss-keywords.data" "msg:'Node-Validator Blacklist Keywords', id:4099802,
severity:'CRITICAL', phase:request,
t:none,t:utf8toUnicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,t:lowercase,t:
removeNulls, rev:'2', ver:'OWASP_CRS/3.0.0', maturity:'1', accuracy:'8', block,
ctl:auditLogParts=+E, capture, tag:'application-multi', tag:'language-multi', tag:'platform-
multi', tag:'attack-xss', tag:'OWASP_CRS/WEB_ATTACK/XSS', tag:'WASCTC/WASC-8', tag:'WASCTC/
WASC-22', tag:'OWASP_TOP_10/A3', tag:'OWASP_AppSensor/IE1', tag:'CAPEC-242', logdata:'Matched
Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}', setvar:'tx.msg=%{rule.msg}',
setvar:tx.xss_score=+%{tx.critical_anomaly_score}, setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score}, setvar:tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/XSS-%
{matched_var_name}=%{tx.0}"

Create the data file xss-
keywords.datadocument.cookiedocument.write.parentnode.innerhtmlwindow.location-moz-binding<!
[cdata[
```

In this example,

`Keyword = document.cookie`

Remove `document.cookie` from the `xss-keywords` data file and send the curl request shown below:

```
<%code>curl -v -b cookies -X GET 'http://172.20.0.49/vulnerabilities/
xss_r/?name=%3Cscript%3Edocument.location%3D%27http%3A%2F%2F172.20.0.49%2Flogin.php%3F+
%27%2520%2Bdocument.cookie%3C%2Fscript%3E#'</code>
```

Note Alternatively, to retain the `document.cookie`, remove the Exceptions or enable the rules above and empty (not delete) the `xss-keywords.data`.

Enabling Special Mode for Specific Applications

Some rules can create false positives for certain known applications. The following section explains how to allow the application to coexist with the CRS.

Custom Rule Syntax:

```
SecRule 'variable"@unconditionalMatch"'
"id:4099803,phase:1,pass,setvar:'TX:crs_exclusions_=1'"
```

Example:

In this example, Wordpress is added to the CRS Exception list.

```
SecRule REMOTE_ADDR "@unconditionalMatch"
"id:4099803,phase:1,pass,setvar:'TX:crs_exclusions_wordpress=1'"
```

Note In addition to this, enable the **CRS_903_Application_Specific_Exclusions group** in the UI.

Allow Other HTTP Methods in WAF

This section explains how to overwrite the list of HTTP methods allowed in a WAF Profile and allow more methods.

Custom Rule Syntax:

```
SecRule 'variable "@unconditionalMatch"'
"id:4099804,phase:1,pass,setvar:'tx.allowed_methods=GET HEAD POST PUT OPTIONS DELETE PATCH'"
```

Example:

In this example, the allowed HTTP methods are `GET`, `HEAD`, `POST`, `PUT`, `OPTIONS`, `DELETE`, and `PATCH` . `@unconditionalMatch` forces the rule to always return `true`.

```
SecRule REMOTE_ADDRESS "@unconditionalMatch"
"id:4099804,phase:1,pass,setvar:'tx.allowed_methods=GET HEAD POST PUT OPTIONS DELETE PATCH'"
```

Note You can overwrite the list of methods in the Pre-CRS rules of WAF Policy, if needed.

More Examples for Custom Rules

This section provides more examples for Custom Rules.

- To exclude all host header entries not in the list.

```
SecRule REQUEST_HEADERS:Host "!@pm ct-vs1.local ct-vs2.local" "msg:'Found bad
hostname in request', severity:'CRITICAL', id:4913102, rev:'2', phase:request,
block, t:none, t:lowercase, ver:'OWASP_CRS/3.0.0', maturity:'9', accuracy:'9',
capture, logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %
{MATCHED_VAR}', tag:'application-multi', tag:'language-multi', tag:'platform-multi',
tag:'attack-reputation-scanner', tag:'OWASP_CRS/AUTOMATION/SECURITY_SCANNER', tag:'WASCTC/
WASC-21', tag:'OWASP_TOP_10/A7', tag:'PCI/6.5.10', setvar:'tx.msg=%{rule.msg}',
setvar:tx.anomaly_score=+%{tx.critical_anomaly_score}, setvar:tx.%{rule.id}-OWASP_CRS/
AUTOMATION/SECURITY_SCANNER-%{matched_var_name}=%{matched_var}, setvar:ip.reput_block_flag=1,
```

```
expirevar:ip.reput_block_flag=%{tx.reput_block_duration}, setvar:'ip.reput_block_reason=%
{rule.msg}'"
```

- To bypass WAF engine for a specific IP address or subnet.

```
SecRule REMOTE_ADDR "@ipMatch 10.0.0.0/8" "id:10000,phase:1,nolog,pass,ctl:ruleEngine=Off"
```

- To check the length of an input parameter.

```
SecRule ARGS:foo "@ge 24" "id:10001,t:length,phase:2,block,log,auditlog,msg:'Size of foo
parameter too big'"
```

- To check for java runtime and getruntime for specific CVE.

```
SecRule ARGS "@rx java\.lang\.runtime|getruntime" "id:4050100, phase:request, t:none,
t:lowercase, block, msg:'Java Injection found', tag:'application-multi', tag:'language-java',
tag:'framework-spring',tag:'CVE-2018-1273', severity:'CRITICAL'"
```

- To bypass a special parameter for a specific rule.

```
SecRule REQUEST_URI "@contains /vulnerabilities/fi/"
id:4000088,phase:1,t:none,nolog,pass,ctl:ruleRemoveTargetById=930120;ARGS:page
```

- To configure Positive Rule in ModSec.

```
SecRule ARGS:id "!@rx ^[0-9]+$" id:12345,phase:2,t:none,block,log,auditlog,msg:'id is not a
number'
```

- To XXE through Custom Rule.

```
SecRule REQBODY_PROCESSOR "@streq xml"
id:4099801,phase:2,t:none,t:trim,t:lowercase,block,chain SecRule REQUEST_BODY "@rx <!ENTITY\s+
[^>\s]*\s+SYSTEM"
```

- To use `detectSQL` Operator on last path element.

```
SecRule REQUEST_FILENAME “@rx ^/(?:[^/]/)(.*)$” \ “id:4099819,\ phase:2,\ block,\
capture,\ t:none,t:utf8toUnicode,t:urlDecodeUni,t:removeNulls,\ msg:’SQL Injection Attack
Detected via libinjection’,\ logdata:’Matched Data: %{TX.0} found within %
{MATCHED_VAR_NAME}: %{MATCHED_VAR}’,\ tag:’application-multi’,\ tag:’language-multi’,\
tag:’platform-multi’,\ tag:’attack-sqli’,\ tag:’OWASP_CRS/WEB_ATTACK/SQL_INJECTION’,\
tag:’WASCTC/WASC-19’,\ tag:’OWASP_TOP_10/A1’,\ tag:’OWASP_AppSensor/CIE1’,\ tag:’PCI/6.5.2’,\
tag:’paranoia-level/3’,\ ver:’OWASP_CRS/3.1.0’,\ severity:’CRITICAL’,\ chain” SecRule
TX:1 “@detectSQLi” \ “setvar:’tx.anomaly_score_pl3=+%{tx.critical_anomaly_score}’,\
setvar:’tx.sql_injection_score=+%{tx.critical_anomaly_score}’,\ setvar:’tx.msg=%{rule.msg}’,\
setvar:’tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/SQL_INJECTION-%{MATCHED_VAR_NAME}=%{MATCHED_VAR}’”
```

- To detect HTTP DeSync attack.

```
SecRule &REQUEST_HEADERS:Content-Length "@gt 0" "id:4099820,phase:1,t:none,block,msg:'HTTP
Desync attack detected',chain" SecRule REQUEST_HEADERS:Transfer-Encoding "@contains chunked"
"t:none,t:lowercase"
```

Upload Handling in WAF

While handling large requests, such as a file upload, certain requests can be blocked with a `413 Request Entity Too Large` message.

NSX Advanced Load Balancer can trigger an alert or block uploads for requests such as the following:

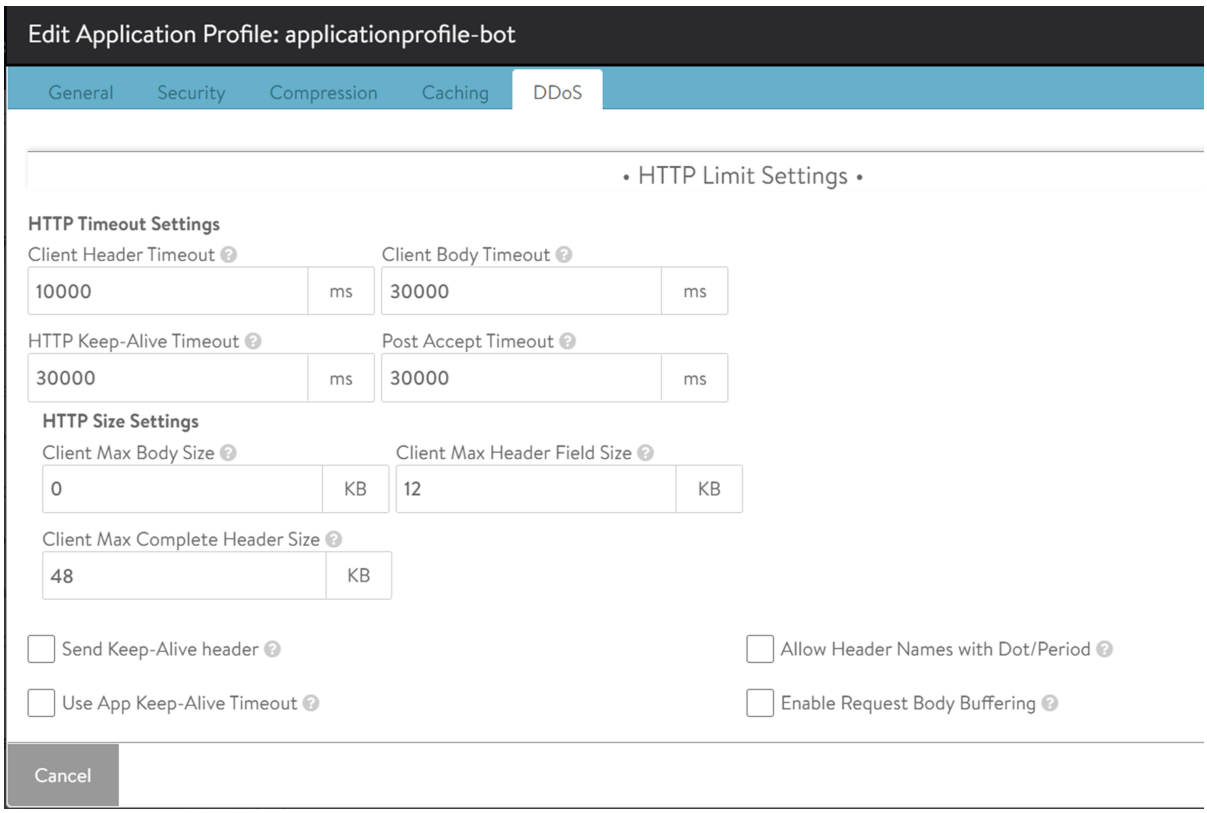
- File size exceeding the limit set under the **Client Max Body Size** field in the virtual service HTTP profile.
- Random match of `System-Default-Policy` rules for binary upload data, which will result in WAF blocking or flagging the request based on the mode.
- Exceeding the regex match limit with rules running too long on the input, which will result in WAF terminating the execution.

Large requests (such as a regular file upload) are handled differently, as these requests will be processed partially, as explained below.

Parameters Handling Large Uploads

The following two configuration parameters are considered for large file uploads:

- **Client Max Body Size**: In NSX Advanced Load Balancer UI, navigate to **Templates > Profiles > Application** to configure the HTTP policy. Under the **DDoS** tab, the **Client Max Body Size** field defines the maximum body size of a client request. This value limits the size of a client POST as a part of a single HTTP request. In the case of a WAF bypass rule, this value is overridden and not considered. Otherwise, if the value configured is lesser than the **Client Max Complete Header Size** configured under **WAF Profile**, the buffering will fail with a `413 error message` in the proxy before reaching the WAF. To mitigate this, update the value configured under **Client Max Header Field Size** to be higher than the value configured under **Client Max Complete Header Size**.



Note If the **Client Max Body Size** is set to a default value of zero, which refers to no-limit, then this value will always be greater than the **Client Max Complete Header Size**.

- **Client Max Complete Header Size**: In NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Profile** to configure a **WAF Profile**. Under the **Settings > Other Settings** section, the **Client Max Complete Header Size** field defines the maximum allowed size for the client request body to be scanned by WAF. If the client request size is larger than the value configured in this field, then the partial body is scanned through WAF as allowed per the defined size.

  If WAF rejects the request due to a `System-Default-Policy` rule match, the rest of the request body is discarded. If WAF allows the request, the rest of the body is streamed in the backend.

  Ensure that the check box for **Ignore request body parsing errors due to partial scanning** is selected to avoid any errors triggered by partial scanning.

Note

- This is supported for both HTTP 1.0 and HTTP 2.0.
- This is supported only for `content-length` request and not for chunked encoded POST.
- If WAF is activated in detection mode and a chunked encoded POST is received, then that POST will be rejected if the size of the POST is greater than the **Client Max Complete Header Size** defined under WAF profile.

Example: Examples

The following are the examples for specific uploads and the corresponding WAF log entries:

Example 1:

- Request is denied with a `413 message` as the client request size exceeds the maximum value configured in HTTP profile.
- As WAF did not inspect the request, WAF status is PASSED.

This error can be mitigated by increasing the value of **Client Max Body Size**.

Example 2:

- Size limit has been increased and so no limit was hit.
- Request is denied with a `403 message`.
- Coincidentally, as parts of the PDF matched the WAF CRS rules, the request is rejected and the status is REJECTED.

Bypassing WAF

You can bypass certain large requests or particular upload requests from going through WAF. Few file extensions are bypassed from the WAF check, as they are static content. For more information on configuring static extensions, see Configuring WAF Profile.

You can also bypass uploads completely using the **Modsec bypass rules**, as explained in the section below.

Modsec Bypass Rules

The following are a few examples of modsec bypass rules. It is recommended to configure these using URLs. The ID should either be within the local range of 0 to 99.999 or a private reserved range, as explained in the ModSecurity Handbook. The numbers are chosen here to illustrate the example. Ensure unique rule IDs in your deployment.

Single URL:

```
SecRule REQUEST_URI "@rx /app/upload/"
id:90001,phase:1,t:none,nolog,pass,ctl:ruleEngine=off
```

Multiple URLs:

```
SecRule REQUEST_URI "@rx /app/upload/|/app/upload_two/|/app/upload_three/"
id:90002,phase:1,t:none,nolog,pass,ctl:ruleEngine=off
```

This rule can be altered using other OPERATORS such as `@contains`, `@startwith`.

By Content-Length:

```
SecRule REQUEST_HEADERS:Content-Length “@gt 1048576”
phase:1,id:90003,nolog,pass,ctl:ruleEngine=off
```

Note It is recommended to configure rules using the URL, instead of Content-Length.

Effect on Malicious Uploads

These are cases where an attacker might want to smuggle a malicious upload onto a server. Such an upload might contain malware, ransomware, viruses, and other file-based exploits (pdf reader exploits), among many others. It is recommended to use a virus malware scanning tool on the upload directory of the application to detect the attacks and mitigate them.

Effect of Upload Bypass on Application Security

If large binary data bypass is configured with the right scope of uploading requests or URLs, then WAF will not be able to inspect the data and the impact will be minimal.

You can bypass certain large requests or particular upload requests from going through WAF. Few file extensions are bypassed from the WAF check, as they are static content.

Note Large uploads will be cached during traffic processing. Even when a part of the request is not needed and bypassed, it will be cached until other parts of that request have been inspected. Therefore, excluding only the upload parameter will not help achieve the best result.

Vulnerability Scanner (DAST)

A Dynamic Application Security Testing (DAST) scanner is a tool to identify potential security issues in applications.

The NSX Advanced Load Balancer SDK provides a script called `avi-iwaf-vpatch.py` that imports DAST scanner results. The imported results are used to construct WAF Policy that protects from the security threats found by the scanner. The technique is also called virtual patching.

The NSX Advanced Load Balancer SDK supports the following DAST scanners:

- OWASP ZAP Attack Proxy
- Qualys Web App Scanning

The supported format for DAST scanners is an XML file containing scanner results report.

Integrating DAST with WAF Workflow

This section discusses the steps to integrate DAST.

The script is delivered as part of NSX Advanced Load Balancer SDK, which is available on Controller in the DAST directory. The following are the steps to integrate DAST with WAF.

1 Run a scan against a web application not protected by WAF.

2 If you find any issues, the `avi-iwaf-vpatch.py` uses the output of the scan to generate WAF Policy rules. Enable WAF.

3 Scan again. The subsequent scans will not report issues for problems handled by WAF Policy.

The `avi-iwaf-vpatch.py` generates NSX Advanced Load Balancer WAF Policy Positive Security rules. It creates a WAF Policy Positive Security group containing all the rules covering DAST scan issues. The `avi-iwaf-vpatch.py` automatically creates Positive Security locations for each vulnerable URL reported by the scanner, and Positive Security rules for each supported issue.

Note The `avi-iwaf-vpatch.py` does not generate rules to protect from all the potential issues found. The script will generate rules related to parameter security, for example, URL parameters, HTML form fields and XML or JSON attributes.

Implementing DAST on WAF

The tool for importing DAST results into the NSX Advanced Load Balancer is written in the Python language. It can be run with the following command line example.

You can use the following format for python:

`avi-iwaf-vpatch.py PARAMETERS FILENAME.`

The `PARAMETERS` are as follows:

- -c — hostname or IP address of the NSX Advanced Load Balancer Controller to connect to.
- -u — username to log in to Controller.
- -p — password.
- -t — tenant.
- -g — (optional) WAF Policy PSM group name.
- -v — verbose output.
- -f — force apply changes.

`FILENAME` is a `DAST` scan output in `XML` format.

When you run the script without `-f` option, the system only prints what it would do. Only after `-force` is set, the system attempts to connect to the NSX Advanced Load Balancer Controller and write WAF Policy. If a group name is not specified using `-g`, the system creates a group named *zap* or *qualysweb*, depending on the scanner being used. The Scanner type is auto-detected based on the XML file format.

For example, `python ./avi-iwaf-vpatch.py -c 127.0.0.1 -g zap_group ./zap_results.xml --verbose`.

Limitations of DAST Scanner Integration

This section explains the limitations of the import script and the manual changes that can be applied.

DAST scanners can report multiple issues that are not handled by the `avi-iwaf-vpatch.py` script. Though many of them might be beyond the scope of WAF, some can be mitigated by appropriate settings in NSX Advanced Load Balancer. Following are some examples:

1 Issues related to Clickjacking can be mitigated by adding a `X-Frame-Options` HTTP header.

2 In the NSX Advanced Load Balancer admin UI, navigate to **Applications > Virtual Services** and edit a virtual service.

3 Click **Policies** tab and navigate to **HTTP Response** tab.

4 Click the + icon to add a new header.

5 Some issues related to cookies can be as follows:

   a A cookie has been set without the *HttpOnly* flag.

   b Cookie Does Not Contain the *secure* Attribute.

   These can be set by selecting appropriate options under **Application Profile > Security**.

WAF in Anomaly Score Mode

This section explains how Anomaly Scoring mode works.

By default, WAF Policy can be configured to operate in either Detection mode or Enforcement mode. The policy flags or rejects a request based on the match of a rule. Alternatively, the Anomaly Scoring mode can be used. For more information, see WAF Mode and WAF Policy

Note When using features like Anomaly Detection, the CRS Group *CRS_901_Initialization* must be enabled, without which required anomaly thresholds are not configured to the defaults. It is recommended to keep this group enabled.

Anomaly Scoring

The rules that match, add up to a request-based threshold. If the threshold is reached, the request is blocked. This topic explains the configuration and modification of multiple threshold limits.

Within the Service Engine, by default, multiple thresholds are set and can be changed.

```
Default thresholds

setvar:tx.sql_injection_score_threshold=15,
setvar:tx.xss_score_threshold=15,
setvar:tx.rfi_score_threshold=5,
setvar:tx.lfi_score_threshold=5,
setvar:tx.rce_score_threshold=5,
setvar:tx.command_injection_score_threshold=5,
setvar:tx.php_injection_score_threshold=5,
setvar:tx.http_violation_score_threshold=5,
setvar:tx.trojan_score_threshold=5,
setvar:tx.session_fixation_score_threshold=5,
```

```
setvar:tx.inbound_anomaly_score_threshold=5,
setvar:tx.outbound_anomaly_score_threshold=4
```

The most frequently used threshold is `inbound_anomaly_score_threshold`, which is used to deny in the default CRS rule 949110 - inbound anomaly score.

```
setvar:tx.critical_anomaly_score=5,
setvar:tx.error_anomaly_score=4,
setvar:tx.warning_anomaly_score=3,
setvar:tx.notice_anomaly_score=2"
```

When the WAF Policy is executed, rules that match will add to the specific thresholds.

For example: 931120. Check for RFI (3/4).

`setvar:tx.anomaly_score=+%{tx.critical_anomaly_score}`

`setvar:tx.rfi_score=+%{tx.critical_anomaly_score}`

When all rules are evaluated, the rule 949110 - inbound anomaly score will check the threshold `tx.anomaly_score and trigger` and deny if it was reached.

```
SecRule TX:ANOMALY_SCORE "@ge %{tx.inbound_anomaly_score_threshold}" "msg:'Inbound Anomaly
Score Exceeded (Total Score: %{TX.ANOMALY_SCORE})',     severity:CRITICAL,     phase:request,
id:949110,     t:none,     deny,    log, tag:'application-multi', tag:'language-
multi',     tag:'platform-multi', tag:'attack-generic', setvar:tx.inbound_tx_msg=%{tx.msg},
setvar:tx.inbound_anomaly_score=%{tx.anomaly_score}"
```

A log file entry is created.

To view the log:

1 Navigate to **Applications > Virtual Services**.

2 Click the virtual service mapped to the **WAF Policy**.

3 Click **Logs**.

4 Expand a Log entry and view the Signature CRS Rules.

Setting Up WAF in Anomaly Scoring mode

This section discusses setting Up WAF in Anomaly Scoring mode.

Changing the Default Behavior of WAF Profile

Note A rule can have different disruptive actions. Most of the rules use `block` as the disruptive action. `Block`triggers the rule engine to execute the default action mentioned in the WAF Profile attached to the policy. This default action contains the **deny** action that then triggers the flag (**Detection**) or reject (**Enforcement**) of the request.

For example: Default Action `phase:1,pass,status:403,log,auditlog`.

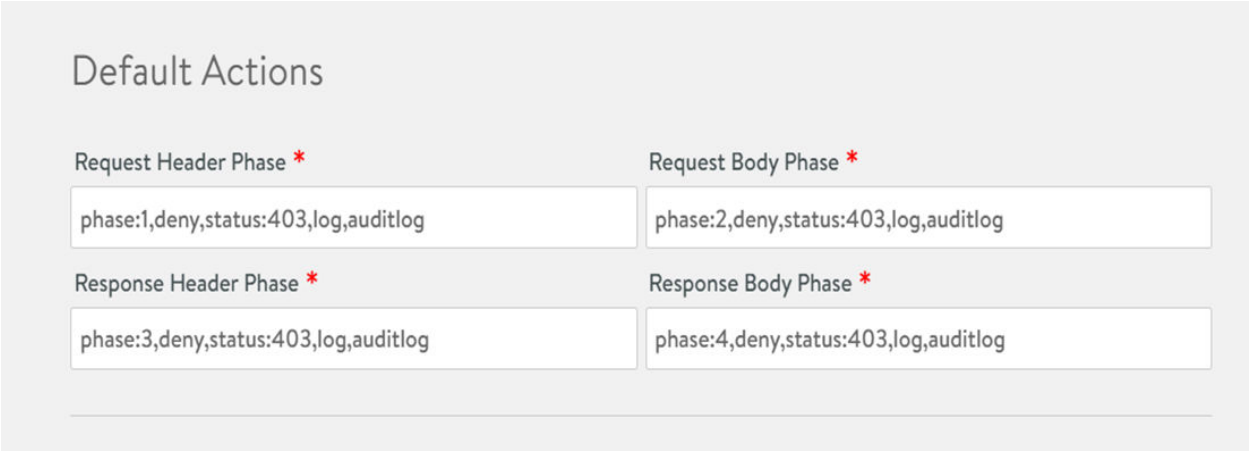
The new default action needs to be **pass**. It needs to be changed for all phases of the WAF handling.

To modify the default action:

1 From the NSX Advanced Load Balancer UI, navigate to **Templates > WAF > WAF Profile**.

2 Click the **Edit** icon against the required policy.

3 Modify the **Default Actions** in the **Edit WAF Profile** screen as shown below:



Changing individual thresholds and blocking of different threshold variables (by group, for example)

Threshold or score variable can be changed using Pre-CRS custom rule.

```
SecRule REMOTE_ADDR "@unconditionalMatch"
"id:4099803,phase:1,pass,setvar:tx.rfi_score_threshold=2"
```

For blocking by using different thresholds, a custom Post-CRS rule is required.

Note It is recommended to disable the CRS rule 949110, which denies the request on the overall score.

This following rule is for blocking the rule.

```
SecRule TX:RFI_SCORE "@ge %{tx.rfi_score_threshold}""msg:'Inbound
RFI-Anomaly Score Exceeded (Total Score: %{tx.rfi_score})',
severity:CRITICAL,phase:request,id:1949110,t:none,
deny,log,tag:'application-multi',tag:'language-multi',
tag:'platform-multi',tag:'attack-generic',
setvar:tx.inbound_tx_msg=%{tx.msg}"
```

This rule blocks only on the `tx.rfi_score_threshold` and the accumulated `tx.rfi_score` variable.

Similar rules can be created for all other groups of attacks.

Note Within ModSecurity language, the variable (for example, `TX:RFI_SCORE`) must be written with a ":" (colon).

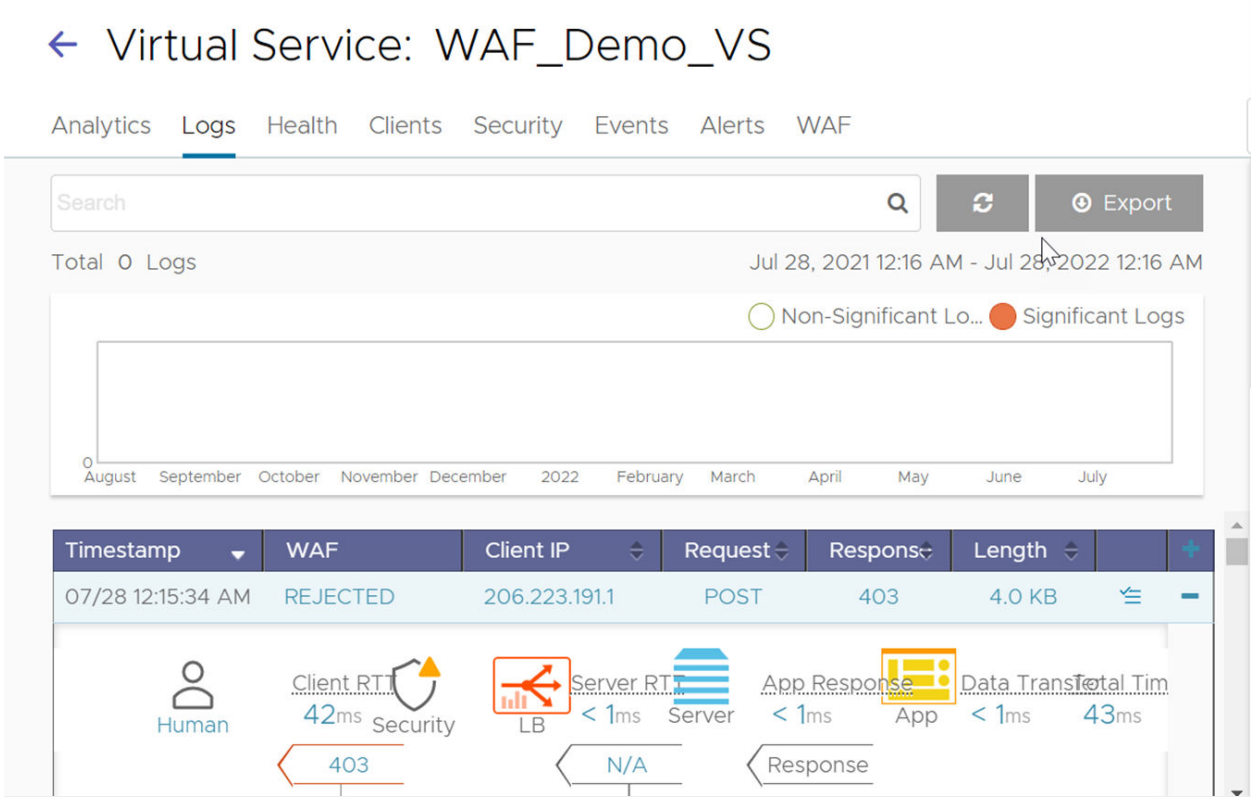
In the **Actions** list and in the **Operator**, it is written using a . `(dot)`. (For example, `tx.rfi_score`). If this is not done correctly, the rule will not match as intended.

Reviewing Recommendations to Remediate False Positives

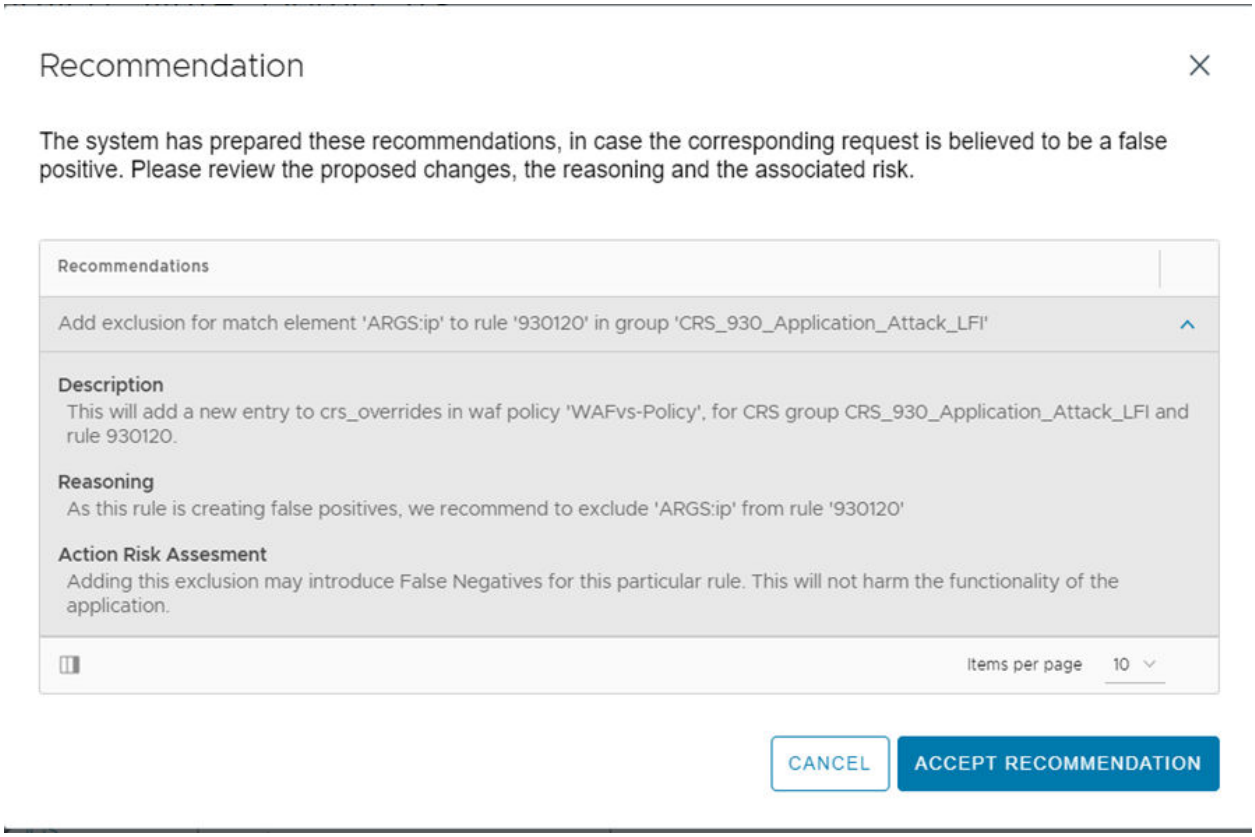
The system-generated Recommendations of NSX Advanced Load Balancer help you ascertain and remediate false positives.

To view the Recommendations:

- Navigate to **Applications > Virtual Services** and click a virtual service associated with WAF.
- Navigate to the **Logs** tab. Click the **Recommendation** icon corresponding to a `FLAGGED` or `REJECTED` log entry.



The **Recommendation** popup window appears with one or more Recommendations to remediate false positives.



- Review the proposed Recommendations. Expand the Recommendation to get more details. These details include a reasoning and a risk assessment for this change.
- After reviewing the Recommendation, if you decide that the log entry represents a false positive, click **ACCEPT RECOMMENDATION** to apply the change to the system configuration.

Note

- The system can recommend multiple changes at the same time. In this case, clicking **ACCEPT RECOMMENDATION** will apply all of the Recommendations to the active configuration.
- When system Recommendations are already applied to the configuration, the status message "All Recommendations are already applied" is displayed in the **Recommendation** page.
- When the system is not able to generate a Recommendation, you can still use the existing Exceptions system.

Recommended Assisted Workflow

The following are the recommended workflow steps to configure Exceptions:

Note This workflow is the older way of using Recommendations. This is replaced by the Recommendation system discussed earlier in this topic.

1 Using WAF Analytics and finding possible false positives.
   a False positives can occur in large numbers and for different client IP addresses.
   b To understand the context for false positives, consult the application owner if required.
2 In the log, choose the WAF hit entry that you want to add the Exception for, and click **+ Add Exception**.
   a The modal dialog generates a set of suggested values.
   b These values are pre-computed from the log entry and related findings. In many cases it is advised to review and if required, change (broaden) the scope of the values.
3 **Save** the Exception to apply it to the policy.

IP Reputation

4

This section discusses the types of IP reputation supported by the NSX Advanced Load Balancer.

Note This feature works only if you have the NSX Advanced Load Balancer Enterprise License for Cloud Services.

IP reputation service is a tool to identify or categorize IP addresses based on their threats. Support for IP reputation is supported through NSX Advanced Load Balancer Cloud Services.

Webroot is a service provider that provides a real-time database for various security threats. The NSX Advanced Load Balancer uses the IP reputation service of Webroot for receiving a database containing bad IP addresses. Bad IP addresses areaddresses that can pose security threats to network services and applications. The availability of the IP reputation database helps to apply various network and security policies to block communication from these IP addresses. It uses the database that contains the list of IP addresses and the categories of security threats associated with them.

IP Reputation Types

The following are the supported IP reputation types:

IP Reputation Type	Description	Values
Spam Source	IP address known to be a spam source. Spam sources include tunneling spam messages through a proxy, anomalous SMTP, or forum spam activities.	0
Windows exploit	IP address offering or distributing malware, shell code, rootkits, worms or viruses.	1
Web attacks	IP address known to be source of web attacks, including cross-site scripting, iFrame injection, SQL injection, cross domain injection, or domain password brute force attack.	2

IP Reputation Type	Description	Values
Botnet	IP address known to be a bot command and control channel, or infected machine controlled by a bot master.	3
Scanner	IP address known to be a scanner, such as probes, host scan, domain scan and password brute force attack.	4
DoS	DoS or DDoS attack, anomalous sync flood or anomalous traffic Detection.	5
Reputation	IP address known to be infected with malware or identified to contact malware distribution points.	6
Phishing	IP address hosting phishing sites or other kinds of fraud activities such as Ad click fraud or gaming fraud.	7
Proxy	IP address providing proxy services.	8
Cloud	IP address originating from a cloud.	9
Mobile threats	IP addresses of malicious and unwanted mobile applications.	10
Tor proxy	IP addresses acting as exit nodes for the Tor network.	11
All threats	Used if you want to protect against anything suspicious.	32

Use Case

- The IP reputation service provides insight into the possible security threats to networks and applications.
- It enhances the layer of protection and increases the performance of web applications as malicious IP addresses are blocked at Layer 4 (IP reputation in Network Security Policy) or Layer 7 (IP reputation in HTTP Policy).
- For example, you can block bad IP addresses or run any other action available in Network Security Policy or HTTP policies.
- It is helpful in differentiating legitimate traffic from malicious traffic.

Prerequisites

Cloud Services on NSX Advanced Load Balancer is a mandatory feature requirement for IP reputation service and must be enabled and registered with the NSX Advanced Load Balancer Controller.

This chapter includes the following topics:

IP Reputation Service

This section discusses the steps to enable the IP reputation service on an NSX Advanced Load Balancer Controller.

IP reputation service is part of the NSX Advanced Load Balancer Cloud Services and is disabled by default on the Controller.

For more information on enabling an IP Reputation Service, see IP Reputation Service.

Configuring Network Security Policy

This section explains the steps to configure Network Security Policy to deny connection attempts from bad IP addresses. This enables the NSX Advanced Load Balancer to drop TCP SYN packets originating from bad IP addresses, causing connection attempts to time out. This section outlines the steps to configure network security policy using the NSX Advanced Load Balancer UI and CLI. It also discusses IP Reputation check through DataScripts.

Using NSX Advanced Load Balancer UI

- Navigate to **Applications > Virtual Services** and click **Edit**.
- Click **Policies** in the **Edit Virtual Service** popup window.
- Select **System-IP reputation-Webroot-DB** from the **IP Reputation DB** drop-down menu under **Network Security**.
- Click **+** icon to create a rule or choose the existing one.
- Select **IP Reputation** from the **Add New Match** drop-down menu to create the matching rule to filter the request.
- Choose an option from the **IP Reputation Type** drop-down menu.
- Set the **Action** to **Allow**, **Deny** or **Rate Limit** against the request for which the IP reputation will be detected.
- Click **Save Rule**.

Using NSX Advanced Load Balancer CLI

A virtual service on NSX Advanced Load Balancer is configured with the Network Security Policy (Layer 4) to enable the IP reputation service to block or take the desired action against the malicious IP addresses. The virtual service can be configured to reject connections from the listed bad IP addresses.

The following describes the packet flow for the Network Security Policy:

1 All the configured Network Security Policies are evaluated when a client connects to the virtual service.

2 The corresponding action for the policy is executed when the following conditions are met:

   a When there is a match for the Network Security Policy (the IP address of the client is present in the IP reputation database).

   b The configured match target matches the IP reputation type listed against the client IP address.

```
[admin:controller]: > configure albservicesconfig
Updating an existing object. Currently, the object is:
+-----------------------------------------------+-------------------------------------+
| Field                                         | Value                               |
+-----------------------------------------------+-------------------------------------+
| uuid                                          | default                             |
| portal_url                                    | https://portal.avinetworks.com      |
| polling_interval                              | 10                                  |
| asset_contact                                 |                                     |
|   name                                        | John Doe                            |
|   email                                       | xxxxxxxxx                           |
|   phone                                       | (xxxxxx)                            |
| feature_opt_in_status                         |                                     |
|   enable_auto_download_waf_signatures         | False                               |
|   enable_waf_signatures_notifications         | True                                |
|   enable_auto_case_creation_on_system_failure | False                               |
|   enable_auto_case_creation_on_se_failure     | False                               |
|   enable_ip_reputation                        | False                               |
| proactive_support_defaults                    |                                     |
|   attach_tech_support                         | True                                |
|   case_severity                               | Severity 5                          |
|   attach_core_dump                            | False                               |
| use_split_proxy                               | False                               |
| ip_reputation_config                          |                                     |
|   ip_reputation_sync_interval                 | 60 min                              |
|   ip_reputation_file_object_expiry_duration   | 3 days                              |
+-----------------------------------------------+-------------------------------------+
```

The default sync interval is 60 minutes and can be changed to any value between 2 and 60 minutes. Use the `ip_reputation_sync_interval` option to change the sync interval.

```
[admin:controller]: albservicesconfig> ip_reputation_config [admin:controller]:
albservicesconfig:ip_reputation_config> ip_reputation_sync_interval 10
```

Note If the IP reputation service is enabled on an NSX Advanced Load Balancer Controller, the Webroot's IP reputation database is available by default.

IP Reputation in HTTP Policies

You can use IP reputation database in **HTTP Security** policy and **HTTP Request** policy. This can be configured in the same way as Configuring Network Security Policy.

IP Reputation in DataScript

You can use the DB for IP reputation check in L7 DataScript using a Lua function.

```
is_good, reputation_type = avi.utils.get_ip_reputation(ip_addr)
```

- The first return value for `is_good` is `true` or `false`, which indicates if the given IP is of good reputation.
- The second return value is a bitmap of IP reputation type and is valid only if `is_good` is false. For instance, value 1 indicates Spam Source (bit 0 set), and value 17 indicates Spam Source and Scanner (bits 0 and 4 sets).

For more information, see the IP reputation table in IP Reputation Types.

The function uses the IP reputation database configured for a `VSDataScriptSet`.

Note You can use API or CLI to configure `VSDataScriptSet` to use `IPReputationDB`.

The Lua function accepts both IPv4 and IPv6 addresses. However, it will always return `true` for IPv6 addresses because the IP reputation database currently contains only IPv4 address information.

The format of the `ip_addr` parameters is expected to be as returned by `avi.vs.client_ip()`, which is a presentation format, for example, 1.2.3.4.

```
message VSDataScriptSet {
  ...
  optional string ip_reputation_db_uuid = 58 [
    (refers_to) = "IPReputationDB",
    ...
  ]
  ...
}
```

Configuration Workflow

When a script uses the `avi.utils.get_ip_reputation(ip_addr)` function, `IPReputationDB`must be configured at the `VSDataScriptSet` level.

Troubleshooting IP Reputation

This section discusses the steps to check IP reputation and its related events.

Checking IP Reputation Database

Login to the NSX Advanced Load Balancer CLI and use the `show ipreputationdb <pdb_name> entries filter ip_addr <ip_addr>` command to check if a given IP address is categorized as a bad IP address in the reputation database.

```
[admin:controller]: >show ipreputationdb System-IPReputation-Webroot-DB entries filter
ip_addr 1.2.3.4
```

You can check the reputation of a given IP using the following command:

```
[admin:controller]: >show ipreputationdb System-IPReputation-Webroot-DB data filter ip_addr
1.2.3.4
```

You can also use the first command to check the IP reputation. The difference is that it queries all the Service Engines for the reputation status of the IP address, which might be helpful while debugging, but expensive when having a large Service Engine cluster.

The second (newer) command checks IP reputation status directly on the Controller.

The database can be checked using the following API endpoint:

```
/api/ipreputationdb/ipreputationdb-UUID/data?ip_addr=1.2.3.4
```

where the `ipreputationdb-UUID` can be obtained using:

```
/api/ipreputationdb/?name=System-IPReputation-Webroot-DB
```

Enable Logging of IP Reputation Events

NSX Advanced Load Balancer collects various alerts and events related to IP reputation service. You can enable logging for the specific virtual service to capture the blocked requests. To enable logging, use the following steps:

1 Select **Network Security** under **Policies**, and click **Edit** against the created Rule.

2 Enter the **IP Reputation Type** and action required.

3 Click **Save Rule**.

The log event shows the following information:

- Source IP Address.

- Destination IP Address.
- Matched rule.

Navigate to **Operations > Events** and filter the events using the required keywords (for example, albservice) to view the alert event for IP reputation database synchronization failure.

For more information on filtering with keywords, see Table 1: Events Search.

Analytics and Insights

5

This topic details the analytics and metrics that is collected for WAF.

This chapter includes the following topics:

WAF Log Analytics

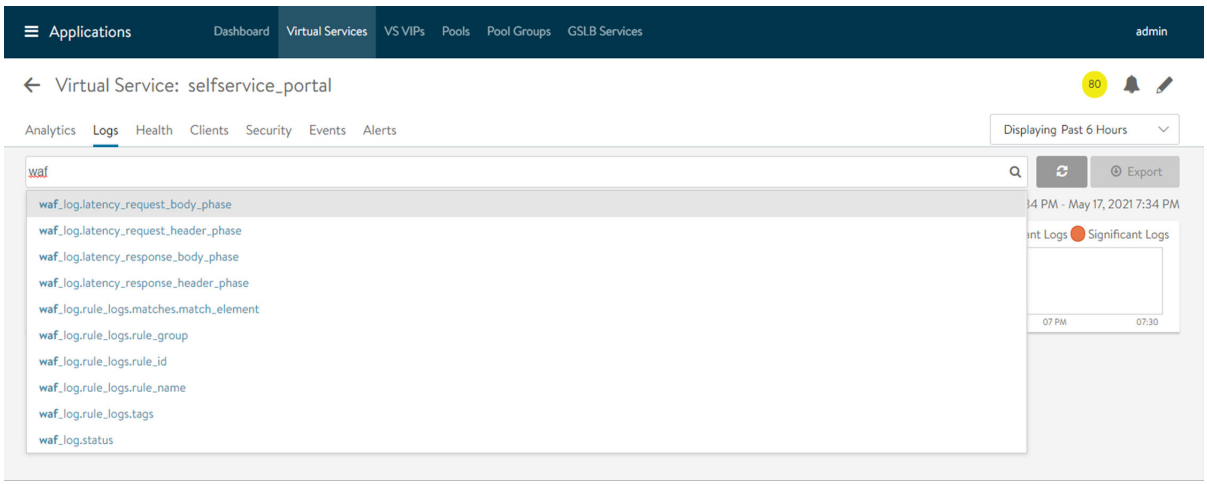
This section explains the WAF Logs Analytics feature available for virtual service in the NSX Advanced Load Balancer.

When a WAF Policy is attached to a virtual service, specific WAF logs are generated. To view the log files:

1 Navigate to **Applications > Virtual Services**.

2 Click the **Virtual Service** mapped to the WAF Policy and navigate to **Logs**.

3 The logs can be filtered to view specific WAF entries. Type WAF in the search bar to populate the available options.



Note The same filters can be used for WAF Analytics.

Analyzing WAF Logs

The following section explores further into WAF Logs Analytics.

Log fields

The following are the fields in a WAF log entry:

- `Timestamp`: Time of capturing the log.
- `WAF`: Result of WAF evaluation. For more information, see the WAF Status section.
- `Client IP`: IP address of the client.
- `URI`: URL of the evaluated traffic.
- `Request`: Request type.
- `Response`: Response code.
- `Length`: Size of the response body.
- `Duration/Timeline`: Duration of the traffic.

WAF Status

This column in the WAF Logs Analytics entry refers to the result of WAF evaluation. The following are the possible outcomes:

- `REJECTED`: Policy is in Enforcement mode and the request was rejected.
- `FLAGGED`: Policy is in Detection mode and the request was logged, but not rejected.
- `PASSED`: Request passed the WAF Policy without any match.
- `-`: The request was not evaluated by WAF.
- `BYPASSED`: When the request matches with the Allowlist and is bypassed.

Log Recommendations are used to help remediate false positives, if any. These Recommendations correspond to each `REJECTED` or `FLAGGED` log entry.

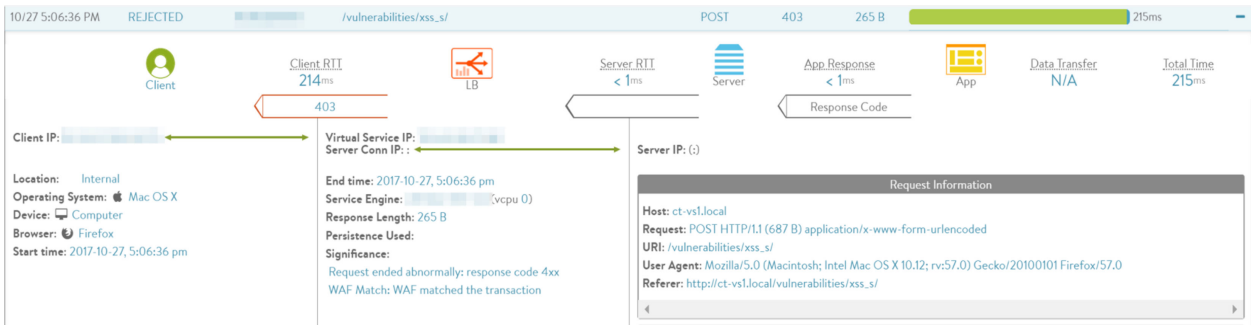
Note The system generates Recommendations that suggest what you can do to mitigate a false positive. However, it is entirely within your discretion to decide if a log entry represents a false positive.

Detailed log information

Clicking on the **+** sign at the end of each log entry expands the panel to provide more details.

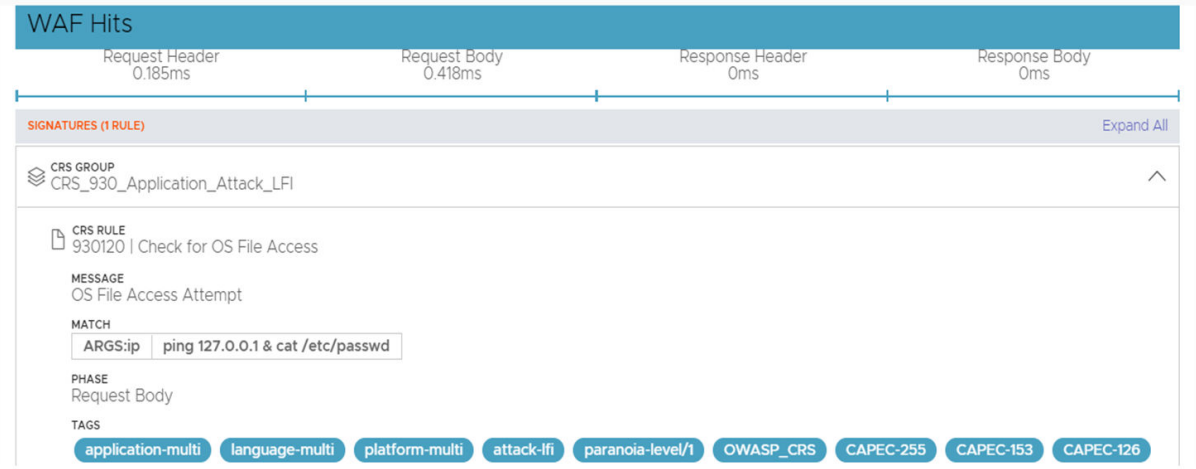
- **Significance**: Indicates WAF Policy match.

Note This is the first indicator of a matched WAF Policy and does not indicate if the request was rejected or not.



- **WAF response time**: Displays the execution time for all four WAF evaluation phases.

- **WAF Hits**: Displays the rules that were matched. All rules that were matched will have an entry consisting of the following fields:
  - **Group name**
  - **Rule name**
  - **Rule ID**
  - **Rule message**
  - **Part of the request or response that was matched, along with the offending string**
  - **Match phase**
  - **Tags** assigned to the rule



- **Add Exceptions**: Under the **WAF Hits** section, click **+ Add Exceptions**, to create an Exception for a false positive remediation.
- Exceptions can be created either at a group or rule level. The created Exceptions are activated immediately.

WAF Analytics

The section explores the WAF Analytics features available for a virtual service under its logs.

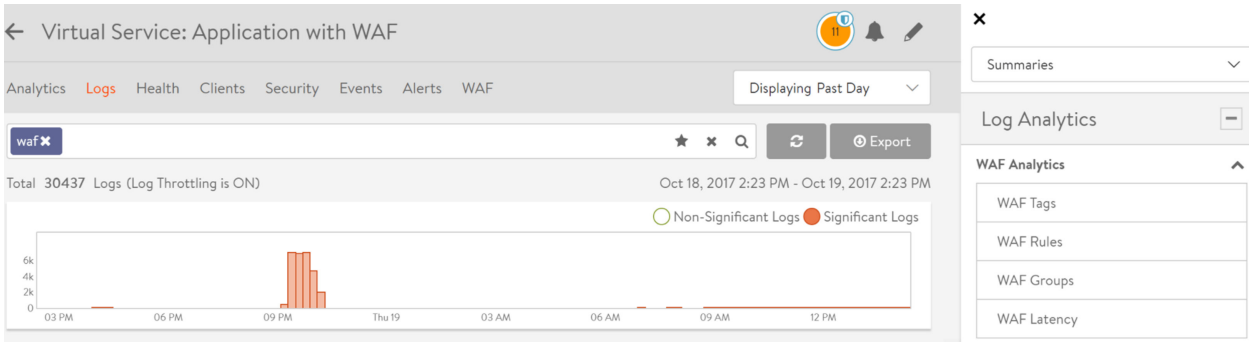
1 Navigate to **Applications > Virtual Services**.

2 Click on the virtual service mapped to the WAF Policy.

3 Navigate to **Logs** and click right side panel to access **Log Analytics**.

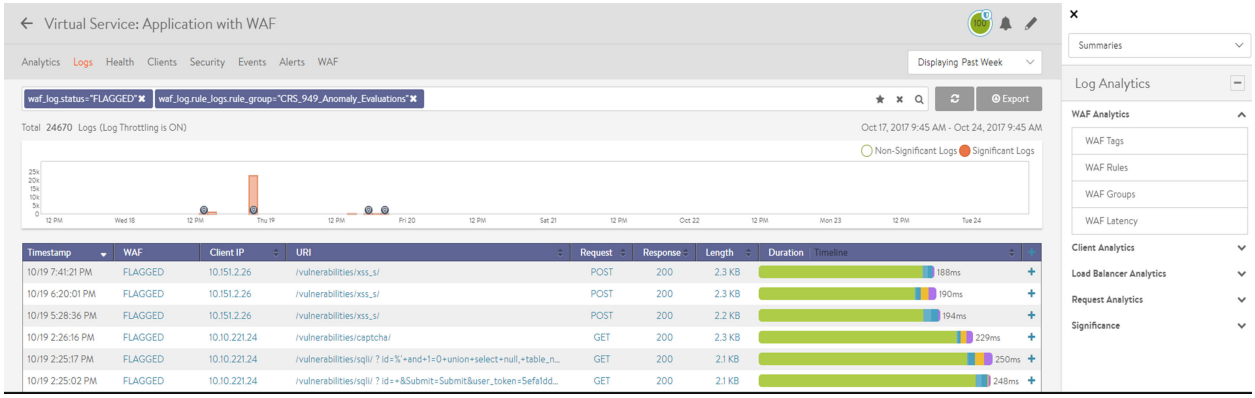
The **Log Analytics** tab provides an option for WAF analytics under the following sections:

- **WAF Tags**.
- **WAF Rules**.
- **WAF Groups**.
- **WAF Latency**.



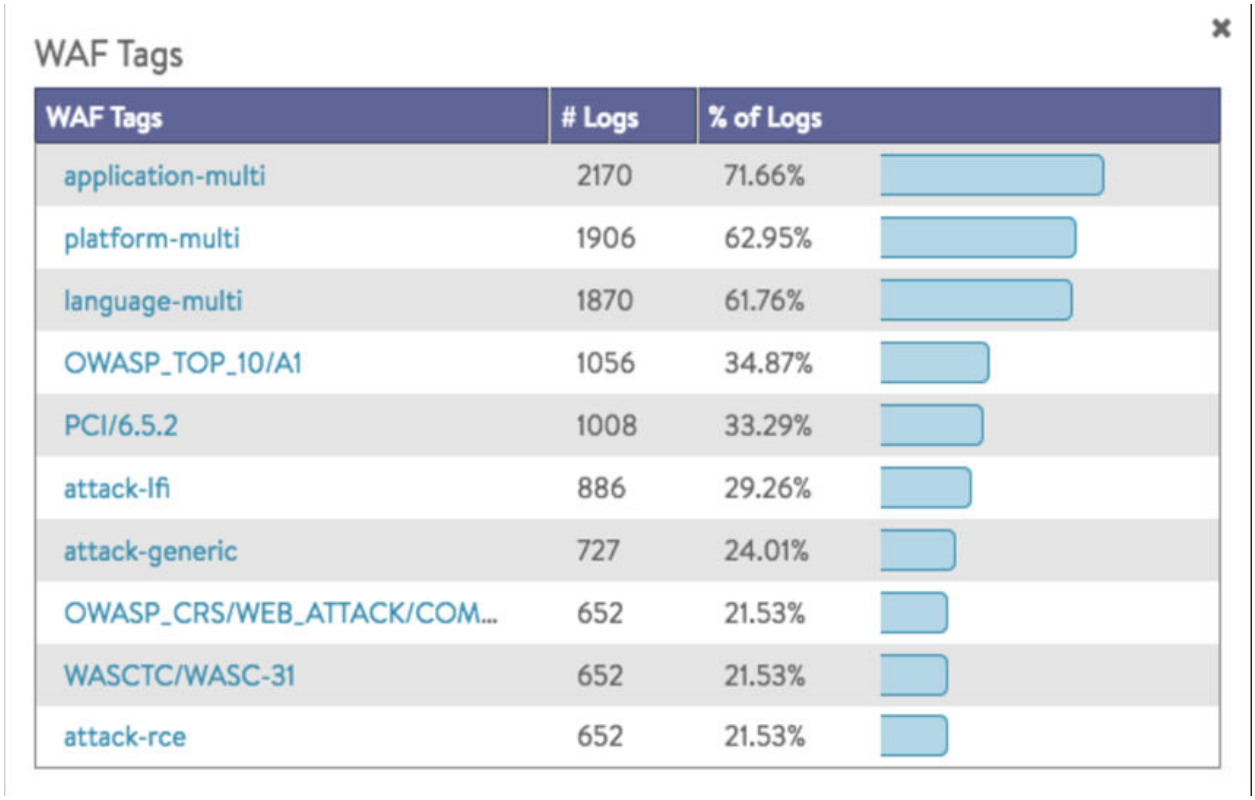
Each section provides an insight into the currently filtered traffic. Analytics can be generated based on the time frame chosen, such as **Displaying Past Week**, **Displaying Past 6 Hours**, and so on. The new WAF log analytics items can now be used in conjunction with the already existent analytics.

The following screenshot shows a sample of logs displayed on choosing **FLAGGED** WAF status filter along with **CRS_949_Anomaly_Evaluations** rule group under **WAF Groups** in the **Analytics** tab.



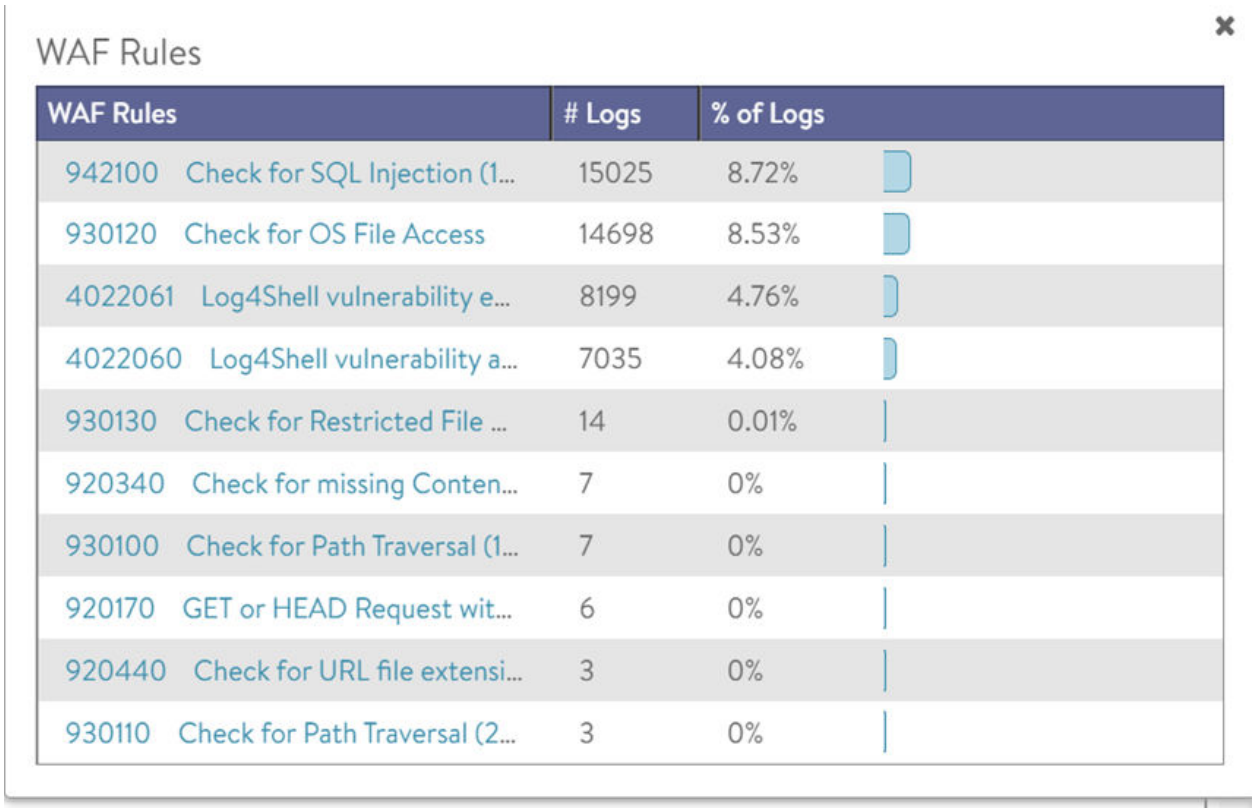
WAF Tags

The following screen capture gives you an overview of the tags that were hit during the selected time frame.



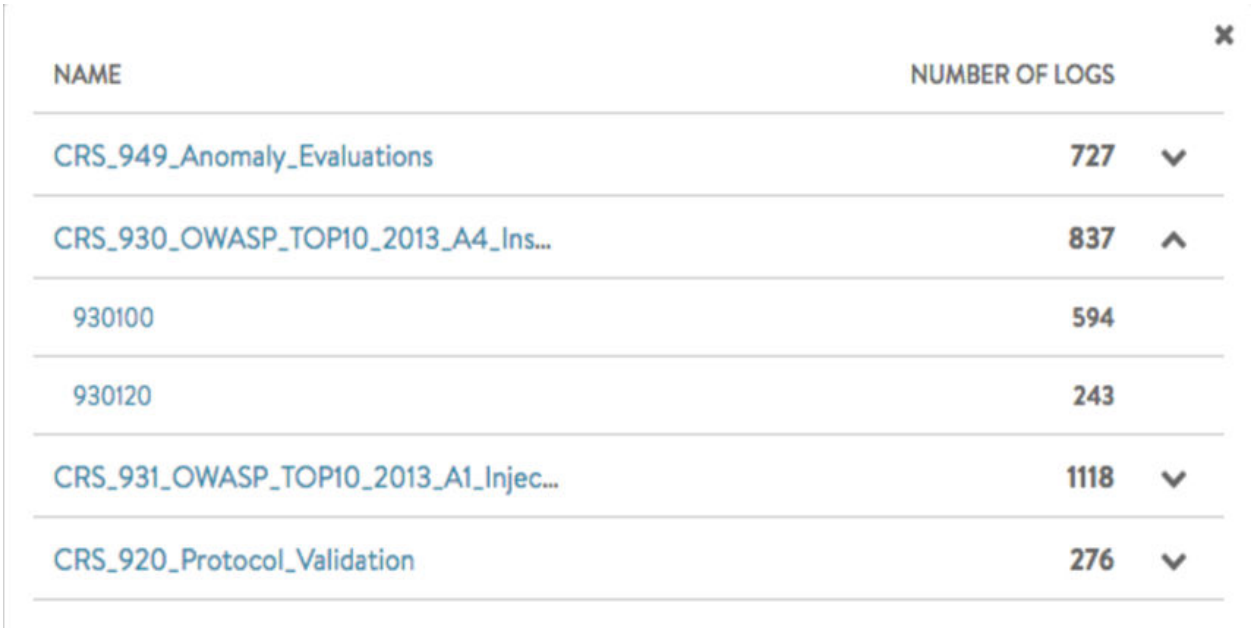
WAF Rules

The following screen capture shows an overview of the rules that were hit during the selected time frame.



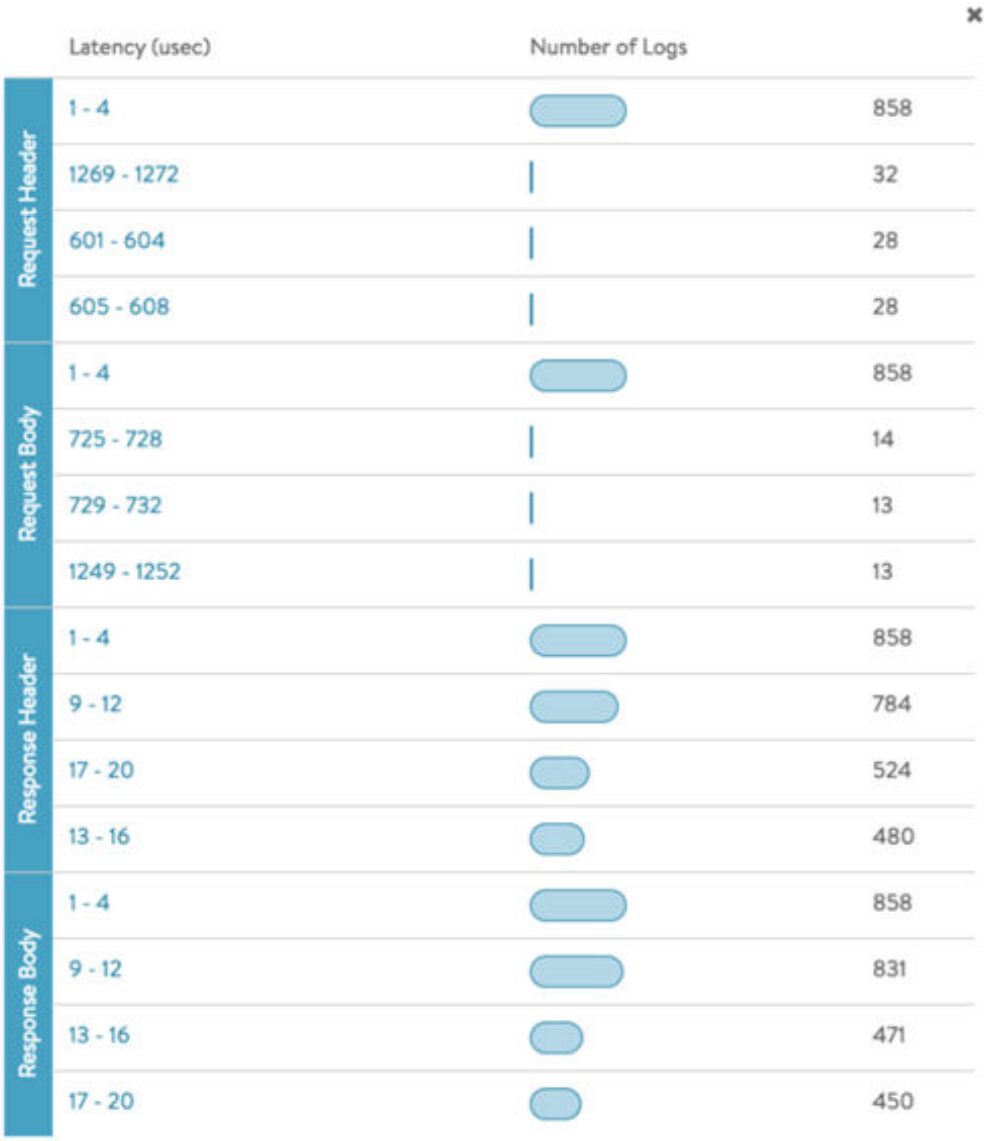
WAF Groups

Check the image below for an overview of the WAF groups that were hit during the selected time frame. Groups can be expanded to show the distribution by rule.



WAF Latency

The following screen captures the summary of WAF latency in microseconds for the log entries in a given time frame.



WAF Metrics

This topic provides an overview of WAF Metrics and steps to view WAF Metrics from the application UI.

To view WAF related metrics do the following:

1 Navigate to **Applications > Virtual Services**.

2 Click the **Virtual Service** mapped to the WAF Policy and navigate to **WAF**.

The chart in this tab displays WAF rule hits against the chosen time frame to help analyze denied requests and their corresponding trigger.

The following fields show specific hit counts for each listed element:

- **Group**

- **Rule**
- **Tag**
- **Client IP**
- **Path**
- **Match Element**

All elements in each field are displayed with the corresponding hit count. On discovering a false positive, any rule or group can be disabled using the toggle button.

You can click on any element in each field to create a specific filter. The field **Popular Combinations** displays the known combinations and their hit counts related to the chosen filter. The filter can be reset by clicking **Reset filters**.

Preview Exceptions

This section discusses previewing Exceptions from the WAF Analytics UI.

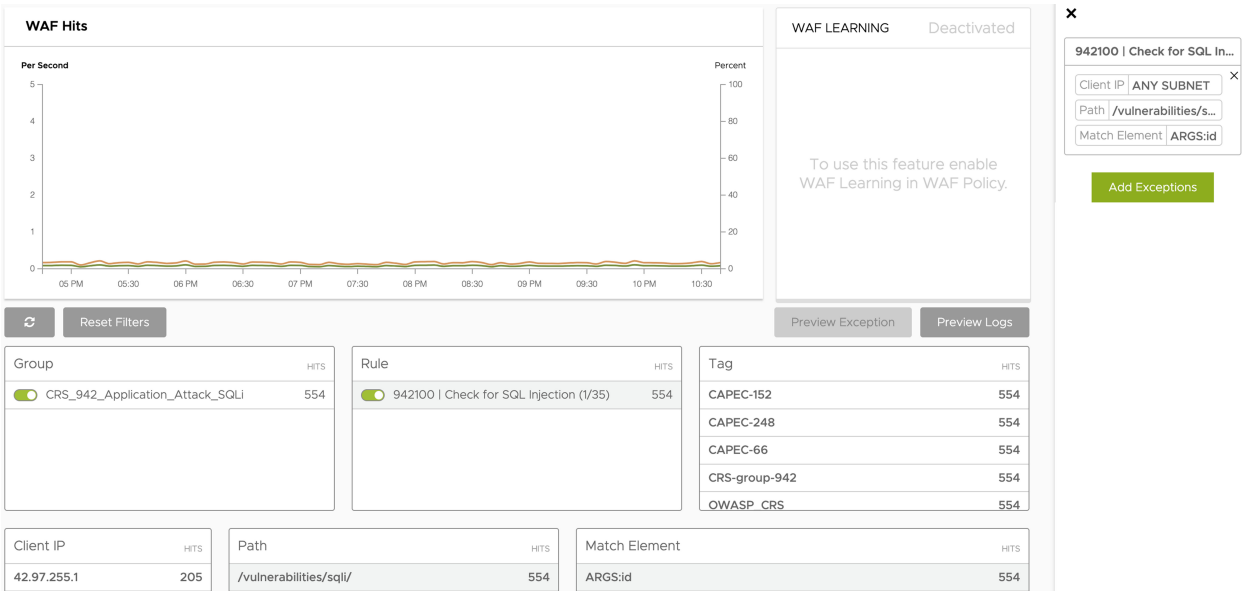
On choosing a specific filter under **Client IP**, **Path**, and **Match Element**, you can add an Exception for the selected combination.

1 Click **Preview Exception** to view the Exception on the right-side pane.

2 To add this Exception, click the **Add Exceptions** icon. The policy is updated immediately.

Note For previewing and creating Exceptions, ensure that the required rule is selected as a part of the filter.

For example, clicking `ARGS:ip` under **Match Element** provides a Preview Exception option as shown below:



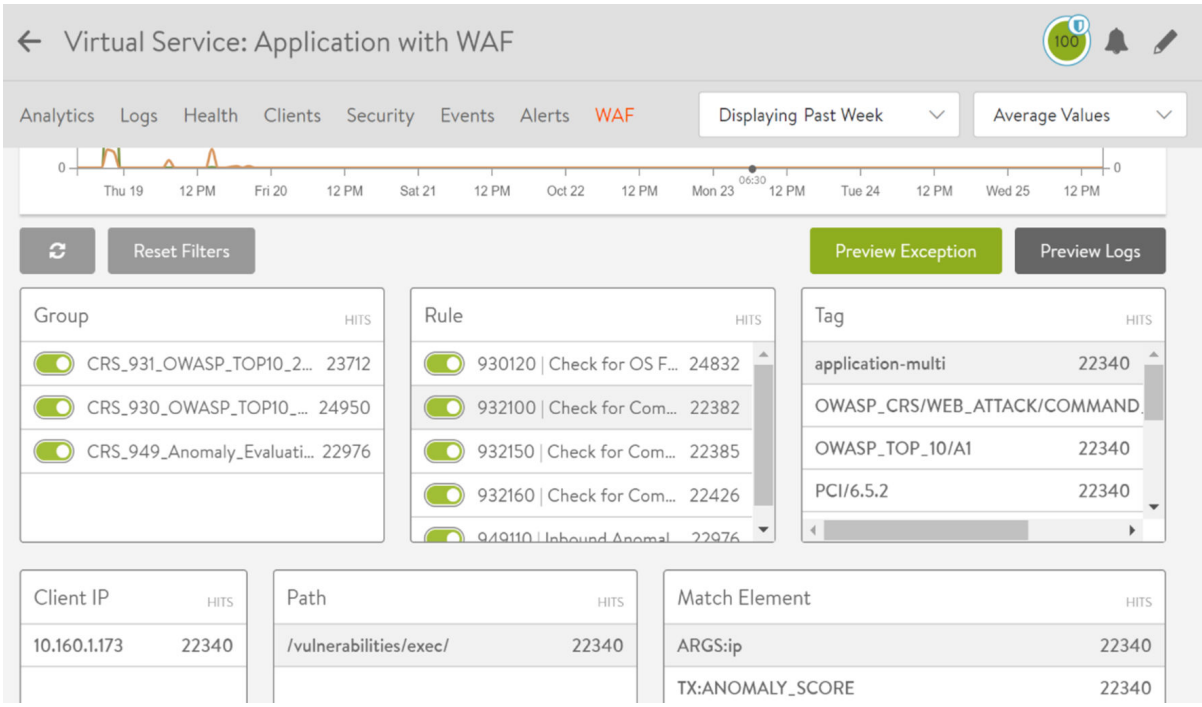
You can choose multiple field elements to create a more specific Exception entry.

Preview Logs

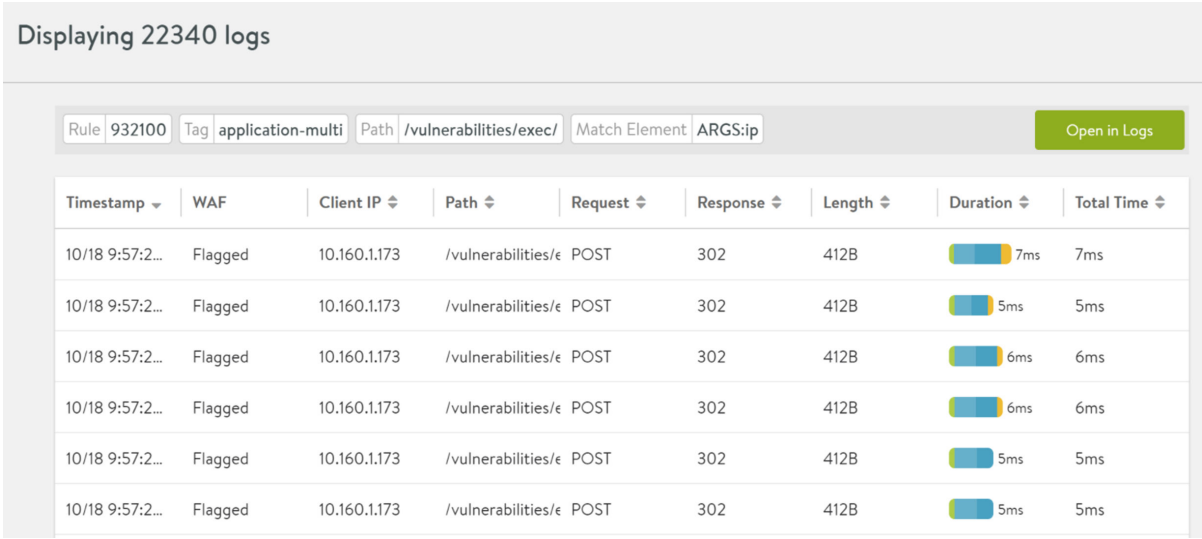
This section explains how to view Preview logs and events for debugging WAF Signatures.

1 You can filter preview logs for a particular combination by clicking the **Preview Logs** button.

2 In the following example, the grayed out elements in the screenshot represent the filter elements that are selected.



3 Click the **Preview Logs** button to view a log table as shown below.



For information on viewing events for debugging WAF Signature, see Viewing Events for Debugging WAF Signature.

Note There can be few instances where WAF hits (**Flagged**) are not displayed in the **WAF** tab of the virtual service but when filtered for WAF, they are displayed under the **Logs** tab of the same virtual service. WAF hits not getting displayed in the WAF tab is an expected behavior, as WAF analytics dashboard uses log APIs with filter where all logs marked as internal are not displayed (`waf_log.rule_logs.matches.is_internal != True`). It is possible that all of the WAF hits on VS Logs do not match this filter, and so no data is shown on the WAF dashboard.

Metrics for WAF Sizing

A set of L7 metrics has been introduced to facilitate WAF sizing. Based on the data provided by these metrics, you can plan your WAF resource allocation.

Note WAF must be enabled on the NSX Advanced Load Balancer for analyzing WAF Bypass metric. The other metrics are primarily HTTP based and also used for WAF sizing.

WAF Bypass

A class of HTTP requests that are considered safe need not be inspected by WAF and can be directly forwarded to the application. For example, `HTTP GET` with no parameters to fetch a `.jpg` image.

The **Static Extensions** field in the WAF profile configuration includes all requests that are termed safe. By default, .gif, .jpg, .jpeg, .png, .pdf, .js, .css, .ico, .svg and .webp are the included extensions.

Parameterless `HTTP GET` for such files are considered safe and will bypass WAF. These HTTP requests are important from a WAF sizing perspective. If web browsers interact with web applications that mostly issue requests bypassing WAF, WAF will not require many resources. Alternatively, if the bypass ratio is low, WAF inspects most of the HTTP requests, resulting in increased resource consumption.

The following Layer 7 metrics provide data on the impact of WAF bypass.

- *l7_client.sum_waf_disabled* – Total number of requests bypassing WAF in a given metrics interval.
- *l7_client.avg_waf_disabled* – Average number of transactions per second bypassing WAF.
- *l7_client.pct_waf_disabled* – Transactions bypassing WAF as the percentage of total requests received.

HTTP headers count

WAF inspects HTTP headers sent from the browser to the web application. With more number of headers to process, the WAF might need more resources. The following Layer 7 metrics provide data on the number of HTTP headers processed.

- *l7_client.sum_http_headers_count* – Total number of HTTP headers across all requests in a given metrics interval.
- *l7_client.avg_http_headers_count* – Average number of HTTP headers per request.

HTTP headers size

The size of the HTTP headers processed by WAF has a direct impact on the resource utilization. The following Layer 7 metrics provide data on the size of HTTP header processed:

- *l7_client.sum_http_headers_bytes* – Total size of HTTP request headers in a given metrics interval.
- *l7_client.avg_http_headers_bytes* – Average size of HTTP headers per request.

HTTP request method ratio

The number of HTTP POST or HTTP GET received by WAF is indicative of the web application behavior. The following Layer 7 metrics are percentage values that indicate the GET and POST requests received:

- *l7_client.pct_get_reqs* - Number of HTTP GET requests as a percentage of total requests received.
- *l7_client.pct_post_reqs* – Number of HTTP POST requests as a percentage of total requests received.

A value of 60 for *l7_client.pct_get_reqs* and 39 for *l7_client.pct_post_reqs* indicate that 60% of requests received were `GET` and 39% were `POST` . The remaining 1% is considered implicit.

HTTP POST size

The size of HTTP POST requests has a direct impact on WAF resource utilization. Higher the size, higher is the resource utilization. The following Layer 7 metrics provide data on the POST request size.

- *l7_client.sum_post_bytes* – Total size of HTTP POST requests.
- *l7_client.avg_post_bytes* – Average size of a HTTP POST request.

HTTP parameters

For higher number of HTTP parameters, more resources will be required by WAF. With the addition of each parameter, WAF consumes significantly more resources and its performance slows down. The following Layer 7 metrics provide data on the HTTP parameters count.

- *l7_client.sum_http_params_count* – Total number of HTTP request parameters.

- *l7_client.avg_http_params_count* – Average number of HTTP request parameters per request.
- *l7_client.sum_reqs_with_params* – Total number of HTTP requests containing at least one parameter.
- *l7_client.avg_params_per_req* – Average number of HTTP request parameters per request, taking into account only requests with parameters.

l7_client.sum_http_params_count and *l7_client.avg_http_params_count* consider all requests, including the ones with no parameters to calculate the value. However, *l7_client.sum_reqs_with_params* and *l7_client.avg_params_per_req* consider only requests that contain at least one or more parameters.

HTTP URI length

WAF resource utilization increases with the increase in the HTTP URI length. The following Layer 7 metrics provide data on the HTTP URI length.

- *l7_client.sum_uri_length* – Total length of HTTP request URIs.
- *l7_client.avg_uri_length* – Average length of HTTP URI per request.

NSX Advanced Load Balancer WAF Core Rule Set

6

NSX Advanced Load Balancer CRS is the default Signature-based protection for NSX Advanced Load Balancer WAF. This topic elaborates on CRS versions and feature inclusions.

Released versions are based on the OWASP ModSecurity CRS with heavy modifications to fit the NSX Advanced Load Balancer configuration model. This modified CRS is solely used by NSX Advanced Load Balancer WAF as the changes done to include benefits to rule performance, accuracy, and manageability, are specifically for the NSX Advanced Load Balancer.

For more information, see OWASP ModSecurity Core Rule Set (CRS).

You can create Custom Rules in NSX Advanced Load Balancer and apply the rules to a WAF Policy.

The version history of CRS updates in NSX Advanced Load Balancer are as presented in the table.

Name	Upstream version	Comments
CRS-2021-4	3.3.2	Improve rules to protect against Log4J vulnerability.
CRS-2021-3	3.3.2	Add rules to protect against Log4J vulnerability.
CRS-2021-2	3.3.2	OWASP CRS from version 3.3 to 3.3.2.
CRS-2021-1	3.3.2	OWASP CRS from version 3.2 to 3.3.
CRS-2020-3	3.2	Bugs fixes and performance improvements.
CRS-2020-2	3.2	Bugs fixes and performance improvements.
CRS-2020-1	3.2	Bugs fixed.
CRS-2019-3	3.2	OWASP CRS updated from version 3.1.1 to 3.2.
CRS-2019-2	3.1.1	A new rule group `CRS_402_Additional_Rules`(set of rules provided by NSX Advanced Load Balancer) is supported.

Name	Upstream version	Comments
CRS-VERSION-NOT-APPLICABLE	None	Enabled WAF Policy will not contain CRS rules.
CRS-2019-1	3.1.0	Rule optimizations and reorganization.
CRS-2017-1	3.0.2	Initial release version of 17.2.
CRS-2017-0	3.0 (beta)	Pre-release version.

NSX Advanced Load Balancer CRS Release Notes

CRS-2021-4

The following changes have been made between the release CRS-2021-3 and CRS-2021-4:

- Improve detection of `CVE-2021-44228` and `CVE-2021-45046` (Log4Shell).
- Reduce potential false positives in the Log4Shell detection rules.

CRS-2021-3

In CRS-2021-3, two rules have been added in group `CRS_402_Additional_Rules` to protect against `CVE-2021-44228`.

CRS-2021-2

The following changes have been made between the release CRS-2021-2 and `CRS-2021-1`:

- Based on OWASP CRS 3.3.2.
- Removed 3 rules in the *CRS_903.9001_Drupal_Exclusion_Rules* group.
- Fixed the names for some rules, for example, *rule 950130*.
- Removed redundant rules, *901120* and *901160*.
- Added NSX Advanced Load Balancer rules to detect Cross-Site Scripting and SQL Injection in the PATH name.
- Added NSX Advanced Load Balancer rule to detect unencoded # in URL.
- Every rule now has a tag which marks it as group membership, for example, CRS-group-980. This enables the user to exclude whole groups dynamically using ModSecurity control actions (For example, by using `ctl:ruleRemoveTargetByTag` or `ctl:ruleRemoveByTag`).
- Every rule with a block or deny action is now is guaranteed to have a paranoia-level tags.

- Improved the error message of rule *4022030* by including the reason for the parsing error in the log message.
- Fixed a false positive for the rule *931130*.

CRS-2021-1

The following changes have been made between the release CRS-2021-1 and CRS-2020-3:

- Based on OWASP CRS version 3.3.
- New Tags based on CAPEC (Common Attack Pattern Enumeration and Classification) give the user more information about the nature of an attack. The CAPEC IDs can be looked upon at https://capec.mitre.org/ to give more information about the impact of an attack detected by WAF.
- Added Exceptions for phpBB from upcoming OWASP CRS Version 3.4:
  - Incorporate certain fixes which will be added in the upcoming release.
  - Rule 920420 will not accept partial content types anymore.
  - Rule 920350 handles IPv6 addresses correctly.
  - Reduces false positives for rules 920470, 941120, 942230 and 942190.

CRS-2020-3

The following changes have been made between the release CRS-2020-2 and CRS-2020-3:

- Rule 920450 is now working as expected.
- The regex for rules 920470 and 920480 is updated to avoid false positives.

CRS-2020-2

The following changes have been made between the release CRS-2020-1 and CRS-2020-2:

- Rule 920180 no longer creates false positives for HTTP/2 requests. This bug has been fixed and performance improvement.
- Performance improvements for rules 941120, 942210, and, 942260.

CRS-2020-1

The following changes have been made between the release CRS-2019-3 and CRS-2020-1:

- Older systems could not update to CRS-2019-3. This bug has been fixed.
- Disable rule 920300 per default (this rule checked for Accept-Encoding header and was only generating log entries but never rejected a request).

CRS-2019-3

The following changes have been made between the release of CRS-2019-2 and CRS-2019-3:

- Introduce rules for special attack types. New groups have been included to:
  - Reduce false positives for xenForo.
  - Protect against NodeJS attacks.
- Moved two rules which handled input parsing failure into the CRS_402_Additional_Rules group.

CRS-2019-2

The following changes have been made between the release of CRS-2019-1 and CRS-2019-2:

- A new rule group with rules provided by NSX Advanced Load BalancerCRS_402_Additional_Rules has been created:

  This group contains two new rules to detect attacks on the HTTP protocol level, like the HTTP desync attack.

  Note The NSX Advanced Load Balancer is not vulnerable to this attack. However, these two rules will provide more visibility.

- The OWASP CRS is updated from version 3.1 to version 3.1.1 as follows:
  - Some rules are updated to avoid false positives.
  - Some rules are updated to make the pattern more efficient (avoid ReDOS attacks).
  - Fixed some false negatives in rules 920240 and 920400.

CRS-2019-1

The following changes have been made between the initial release of CRS-2017-1 and CRS-2019-1:

- Updated the OWASP CRS from version 3.0 to 3.1.
- Added groups which include Exceptions for special applications.
- Recreated the group structure from OWASP CRS (created more groups).
- Disabled rule 920350 (Detect if Host Header is an IP address) in the default installation.

FAQs

7

This section discusses some commonly asked questions regarding WAF.

What are the traditional WAF challenges addressed by the NSX Advanced Load Balancer?

The following are the solutions that WAF provides to secure customer applications.

- Security: Combine different verification methods to provide a comprehensive security layer (Signatures, Positive Rules, Client Reputation, Machine Learning, Outlier analysis and others).
- Automation: WAF solution that can be driven by any of the current automation frameworks (Ansible, Terraform etc) and can be integrated into a SDLC (Secure Development Life Cycle).
- Observability: WAF solution that provides deep insights into the traffic, application behavior and clients.
- Ease of use & Simplicity: WAF solution gathers data, learns from the data and auto-tunes the policy or helps the admin to adjust the policy quickly.
- Scalability: WAF solution that caters to small and large applications in a similar manner.
- Performance: WAF solution that uses the resources to the best effort and provides measurements to validate it.

What are the features provided as part of WAF ?

The WAF features are as follows:

- OWASP Top 10 Protection.
- Input Validation – XSS, SQLi etc.
- Positive security Model through Application Learning.
- Scripting for application logic flaws - Using Data Scripts.
- API protection for JSON, XML.
- Simplified Policy Definition.
- Real-time Insights.

- Elasticity and Automation.

Other security features provided by the NSX Advanced Load Balancer are:

- Application Rate Limiting.
- DDoS Protection.
- L3/L4 ACLs.
- L7 Rules/Policies etc.

For comprehensive information on all WAF features, see WAF Features.

Does the NSX Advanced Load Balancer provide WAF as a service?

As of today, the WAF (or LB) is not offered as a cloud service and is deployed to the customer environment. However, WAF-as-a-service offering is planned in the future. The WAF (or LB) is similar to a physical WAF in terms of on-prem deployment and has better operational, scale, performance, and visibility. The NSX Advanced Load Balancer also offers Controller-as-a-Service that includes WAF as part of LB offering.

What is the sizing recommendation for WAF?

Recommendations for running WAF and other Application Security features on the NSX Advanced Load Balancer platform are 2vCPUs and at least 4GB of RAM. If more performance is required for the protection of the virtual services, additional Service Cores might be required. For detailed analysis of and applications requirement, please reach out to your VMware specialist, who will happily assist.

Do we need separate license for WAF?

Since WAF solution is part of a large LB/ADC offering, a separate license for WAF is not required. However, make sure the Service Engine sizing is adjusted based on the WAF requirements.

What is Positive Security Model?

Positive Security Model, also called Application specific policy, describes the application behavior and provides an input validation by setting an accepted range (and length) of characters. If the input validation specification is not as expected, it reports a policy violation.

Example: `product_id=[0-9]{0..63}`.

For more information on Positive Security, see Positive Security and Learning

What do Signatures do?

Signatures perform input and output validation by analyzing all incoming and outgoing traffic. Signatures contain detection for OWASP Top 10 attack vector among many other uses.

What is an Allowlist?

See WAF Allowlist

What is Application Learning?

In a system, Application Learning is the method of collecting statistical information of an application's normal usage to generate a Positive Security Model. For more information, see Application Learning for WAF

What is WAF processing flow?

The WAF processing flow is as follows:

1. For an incoming request, the Allowlist policies are checked. If there is any matching condition, the request is added to the Allowlist i.e., WAF processing is turned off for that request.
2. If none of the conditions match, the Positive Security engine checks if the request is in line with the learned data.
3. If a request is marked illegitimate by the Positive Security engine, it is flagged or blocked immediately.
4. If Positive Security marks the request as legitimate, the request is sent to the Signature engine which checks it against the Signatures, in order to identify any attacks.
5. If WAF finds an attack vector, it blocks the request.
6. If WAF Policy is in Enforcement mode, it blocks the request.
7. If WAF Policy is in Detection mode, though it flags the request, it does not block them.

For a detailed view of WAF Architecture, see Architecture.

How does the system behave when creating WAF policy using CLI and API?

While creating the WAF Policy using the CLI and the API, the following two fields are deprecated:

- `crs_groups`.
- `application_signatures.rules`.

The above mentioned groups and rules are now taken directly from the referenced `wafcrs` and the respective `wafapplicationsignatureprovider` object.

The following new fields are available instead of the deprecated fields.

- `crs_overrides.`
- `application_signatures.rule_overrides.`

These fields are used to perform configuration changes, like setting the *mode* attribute or adding the `exclude_list` settings for a rule or group.

See the snippet below to check the overrides.

```
{
    "name": "Example Policy 1",
    "waf_mode": "WAF_MODE_ENFORCEMENT",
    "waf_profile_ref": "/api/wafprofile?name=System-WAF-Profile",
    "waf_crs_ref": "/api/wafcrs?name=CRS-2020-3",
    "crs_overrides": [
        {
            "name": "CRS_903.9002_Wordpress_Exclusion_Rules",
            "enable": true
        },
        {
            "name": "CRS_920_Protocol_Validation",
            "rule_overrides": [
                {
                    "rule_id": "920310",
                    "enable": false
                },
                {
                    "rule_id": "920311",
                    "enable": false
                }
            ]
        },
        {
            "name": "CRS_930_Application_Attack_LFI",
            "rule_overrides": [
                {
                    "rule_id": "930120",
                    "exclude_list": [
                        {
                            "match_element": "ARGS:path",
                            "match_element_criteria": {
                                "match_case": "INSENSITIVE"
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```

What is a false positive?

A false positive is a legitimate request that is flagged as an attack.

What are the Paranoia Modes available in WAF? What are the considerations for choosing the mode?

The available paranoia modes are:

- 1- Low (Default and recommended mode).
- 2- Medium.
- 3- High.
- 4- Extreme.

The following are the two aspects that must be considered while setting the paranoia mode.

- Risk level of an application.
- Resources available for policy tuning.

For more information on paranoia mode, see OWASP CRS Paranoia Mode.

What is a false negative?

An attack that is not detected is called a false negative.

What is Exception for false positive mitigation?

An Exception adds a matching condition of `<IP, URL, parameter>` in front of a Signature Rule or a Rule Group. For more information on Exceptions, see Exceptions

What is a CVE?

CVE or Common Vulnerabilities and Exposures is a list of publicly disclosed computer security flaws. When a CVE is mentioned, it refers to a security flaw that has been assigned a CVE ID number.