Managing CORS Preflight Scrutiny in Layer 7 Policy

Introduction

OWASP identifies a risk in that the CORS request preflight process is entirely managed on client side (by the browser) and that anything warranted by the web application during the request preflight process will be always followed. A user can thus create and send a final HTTP request (using tools like Curl, OWASP Zap Proxy, etc) without previously sending the first request for preflight and then bypass request preflight process in order to act on data in a unsafe way. (https://www.owasp.org/index.php/CORS_RequestPreflighScrutiny).

The CORS policy implementation in Layer 7 is designed to provide a configurable proxy for web services and applications to mitigate abuse of CORS Preflight processes by caching preflight requests and validating that subsequent complex requests correlate to an existing preflight request.

Note: This documentation applies to version 7.0 of the SecureSpan Gateway.

About CORS (Cross-Origin Resource Sharing)

User agents commonly apply same-origin security policy (SOSP) restrictions to network requests. These restrictions prevent a client-side Web application running in one origin from obtaining data retrieved from another origin, and also limit unsafe HTTP requests that can be automatically launched toward destinations that differ from the running application's origin.

The purpose of CORS is to provide a mechanism for bypassing SOSP restrictions by introducing several HTTP request and response headers and explicitly whitelisting aspects of the request such as origin URLs, allowed methods, allowed HTTP headers, etc. For complex requests the CORS specification defines a preflight request mechanism whereby a client can probe the application for allowed methods, etc.

Preflight vs Actual Requests

CORS requires clients to perform preflight requests to determine the allowed methods and headers before making actual requests to the resource. Preflight requests are made using the OPTIONS HTTP header with an access-control-request-method HTTP header set.

Preflight Caching Considerations

The internal cache mechanism of the Layer 7 Gateway does not provide for cluster aware caching out of the box. Thus if the actual request following a preflight request is routed to a different node in the cluster it will not be aware of the preflight context. This implementation provides a simple mechanism to update other nodes in a cluster for awareness of the preflight context by means of a simple PUT request to the other nodes if a complex request is encountered.

Installation

Installing the CORS policy requires the following steps, detailed below:

- Deploy the CORS Processing fragment
- Publish the CORS Caching services
- Configure cluster-wide properties
- Publish and configure service using the CORS policy fragment

Deploy the CORS Processor Fragment

The CORS Processor fragment does the bulk of the CORS processing. It returns true if the request message passes all of the CORS requirements, setting ${cors.preflight} in the process.

Steps

1. Create a new folder in the Services and Policies window for CORS.
2. Right mouse click the folder and select Create Policy.
3. Enter CORS Processor in the Name field then click OK (Figure 1). The view will change to the Policy Editing view.

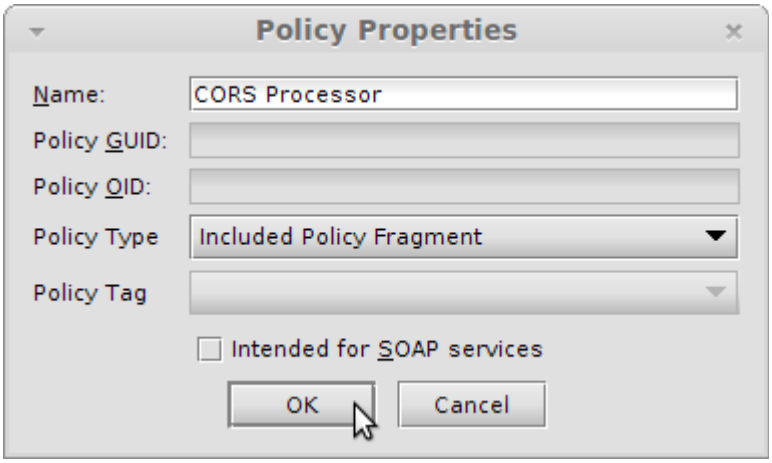

Figure 1: Creating the CORS Processor policy fragment

4. Click the Import Policy button then navigate to select the CORS Processor Policy v1.0.xml file included with this document.
5. Once the policy is loaded click the Save & Activate button (Figure 2).

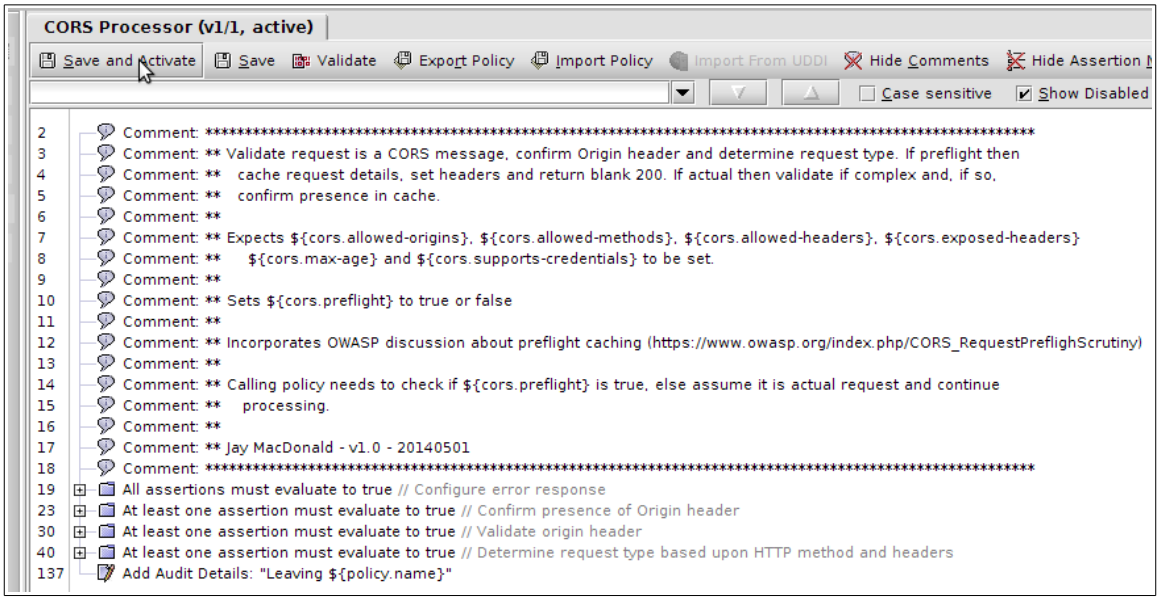

Figure 2: CORS Processor Policy

Publish the CORS Caching Service

The CORS Caching Service is needed for gateway nodes in a cluster that receive CORS preflight requests to notify other nodes in the cluster that the request has been preflighted.

Steps

1. Right mouse click on the CORS folder and select Publish Web API
2. In the Publish Web API Wizard dialogue set the Service Name to CORS Preflight Cache Service and the Gateway URL to /system/CORSCache (Figure 3) then click Finish.

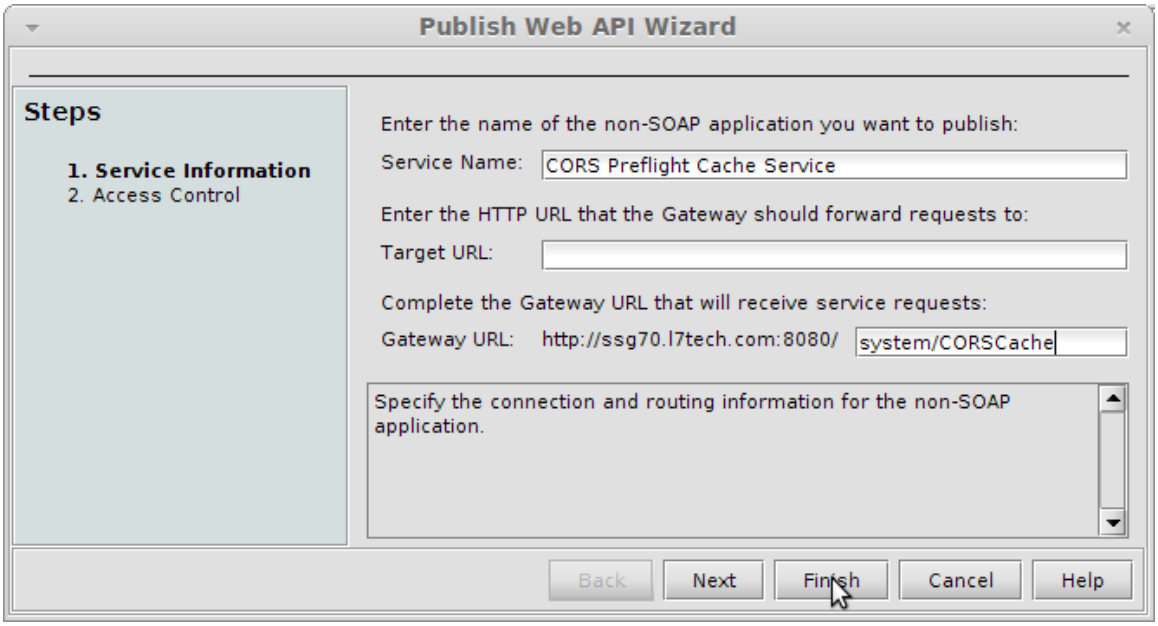

Figure 3: Publishing the CORS Preflight Cache Service

3. Click the Import Policy button then navigate to select the CORS Preflight Caching Service v1.0.xml file included with this document.
4. Once the policy is loaded click the Save & Activate button (Figure 4).

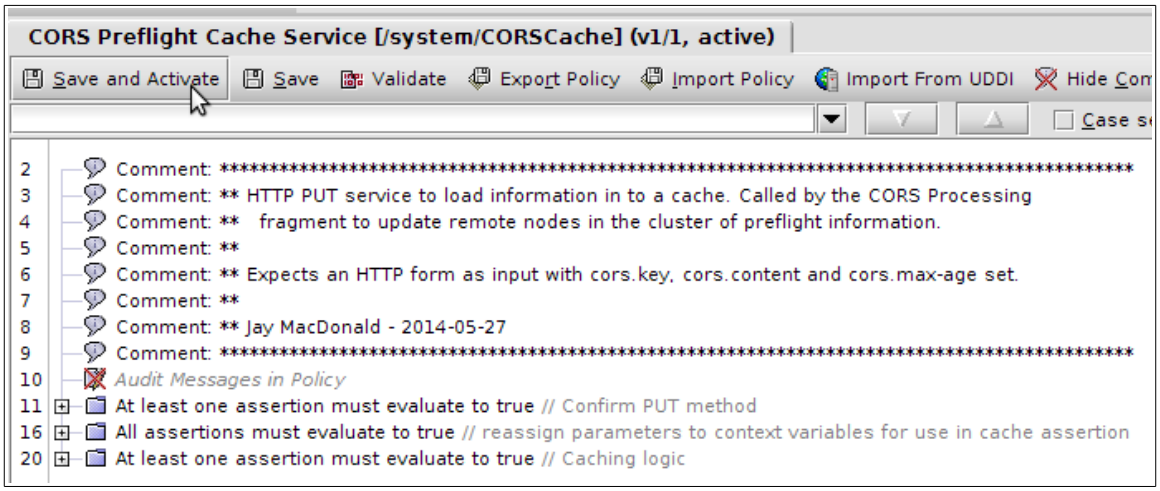

Figure 4: CORS Preflight Cache Service loaded

5. Right mouse click on the service in the Services and Policies window and select Service Properties.
6. Select the HTTP/FTP tab and disable all methods except PUT (Figure 5) then click the OK button.

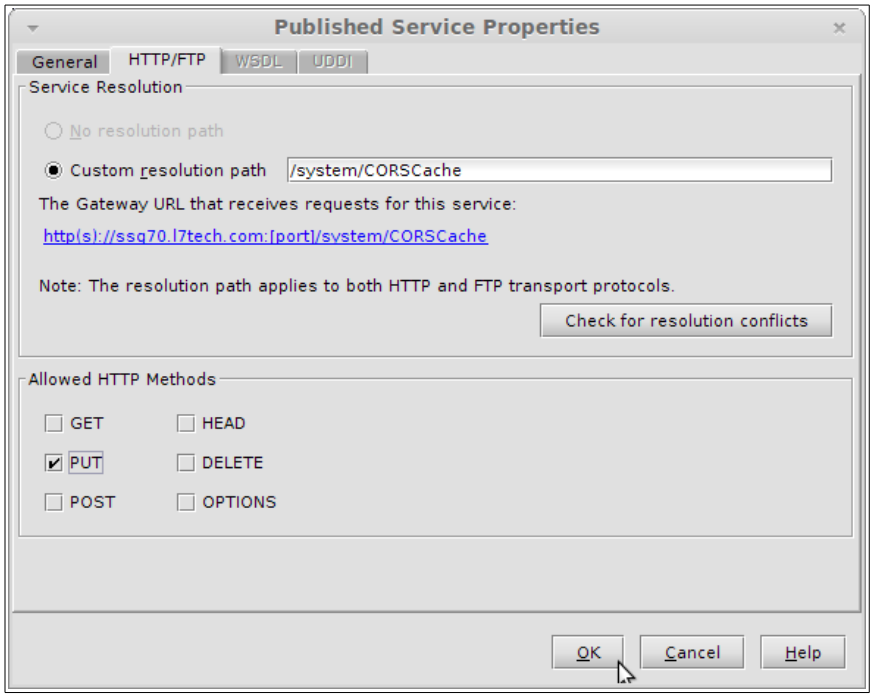

Figure 5: Setting the HTTP method

Configure Cluster-Wide Properties

Two cluster wide properties are used by the CORS Processing policy to manage maintaining state of preflight calls: cors.clusterNodes and cors.clusterNotify. Maintaining this state across the cluster does incur a slight increase in latency.

cors.clusterNotify

cors.clusterNotify is a boolean setting to indicate whether the policy should notify the rest of the cluster of a caching requirement. If the load balancer in front of the cluster is set to sticky then this should most likely be set to false, meaning we do not need cluster wide awareness of preflight calls.

cors.clusterNodes

cors.clusterNodes maintains a list of the nodes in the cluster as a comma delimited string of pairs mapping node name to IP address. The policy cycles through each entry to send the preflight state to be cached if cors.clusterNotify is true. If cors.clusterNotify is false this property must still be properly configured.

The values for the node names correspond to the node names set in the Gateway Status tab of the Dashboard (Figure 6). For example:

```
ssg70-a.l7tech.com:10.7.50.70,ssg70-b.l7tech.com:10.7.50.170,ssg70-c.l7tech.com:10.7.50.171
```

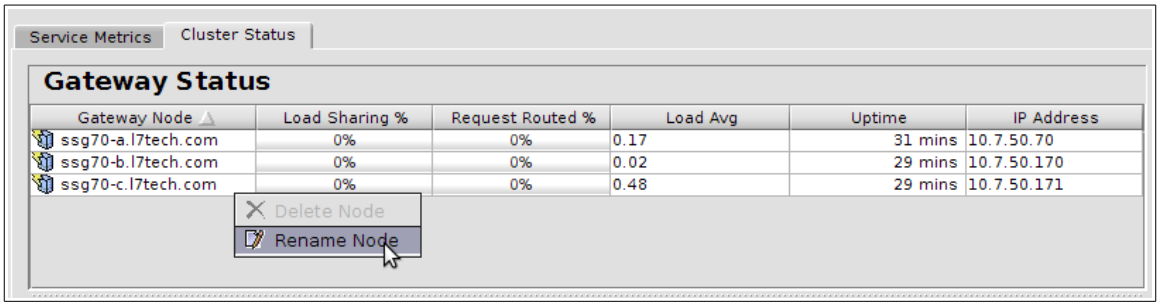

Figure 6: Determining the Gateway node name and IP address

Configure the Service

Enabling CORS in a service requires setting some context variables for configuring the CORS parameters and including some policy. Figure 7 illustrates a policy that leverages the CORS Processor fragment.

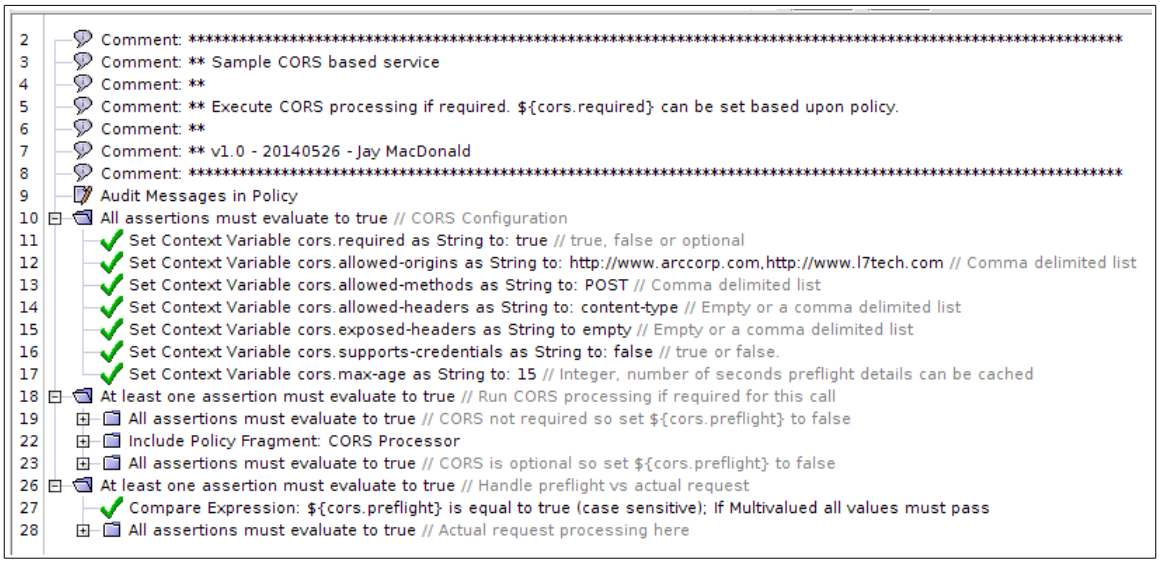

Figure 7: Service policy using the CORS Processor fragment

Prior to calling the CORS Processing fragment the following variables must be set:

cors.required

String value control CORS processing. Must be one of true, false or optional. This can be set as required by policy, for instance to determine if CORS is required from specific network ranges, etc. The optional setting allows for legacy systems that may not have CORS implemented on the client side. If optional is to be set cluster wide create a cluster-wide property cors,required and reference it as ${gateway.cors.required} when setting this in policy.

cors.allowed-origins

Comma delimited list of origins that are allowed in the Origin header of the request. Must be case sensitive match.

cors.allowed-methods

Comma delimited list of methods that are allowed for the resource

cors.allowed-headers

Comma delimited list of HTTP headers that the resource will accept outside of simple request headers

cors.exposed-headers

Comma delimited list of headers that may be exposed by the resource

cors.supports-credentials

Flag to indicate if the resource supports credentials

cors.max-age

Maximum time for a preflight response to be cached. Clients can cache preflight responses so subsequent requests are not require to make preflight calls. Note: This is also the amount of time a Gateway can cache the preflight details for request scrutiny.

Version History

Date/Ver	Edited By	Comments
20140602	Jay MacDonald	Initial Version for SecureSpan v7.0
20140603	Jay MacDonald	Added content around cors.clusterNodes configuration