CORS Security

UP

CORS Security (Cross Origin Resource Scripting)

CORS Overview

Cross-origin resource sharing (CORS) is a mechanism that allows a web page, loaded from one domain, to make HTTP requests to access resources located in another domain. CORS is primarily targeted at web browser's Java Script (JS) and it XMLHttpRequests object. Java Script's security defaults prevent such "cross-domain" requests. CORS also affects other HTTP clients such as command line tools (curl) and language HTTP clients such as Apache's HTTPClient. The CORS standard defines a way in which the client can ask the server, or individual web application,whether it can access the cross-origin resource. The server's configuration determines whether access to the cross-domain resource will be granted or rejected. It is more powerful than only allowing same-origin requests, but it is more secure than simply allowing all such requests.

The W3C CORS standard works by adding new HTTP headers that allow servers to serve resources to permitted origin domains. Browser's JS engines support these headers and enforce the restrictions they establish. Additionally, for HTTP request methods that can cause side-effects on user data (in particular, for HTTP methods other than GET, or for POST usage with certain MIME types), the specification mandates that browsers “preflight” the request, soliciting supported methods from the server with an HTTP OPTIONS request header, and then, upon “approval” from the server, sending the actual request with the actual HTTP request method. Servers can also notify clients whether “credentials” (including Cookies and HTTP Authentication data) should be sent with requests. HTTP clients that are not running a JS engine may need to manually insert the CORS HTTP headers when the cross-domain resource is governed by a policy that requires all HTTP clients to send CORS headers.

CORS can be used as a modern alternative to the JSONP pattern. While JSONP supports only the GET request method, CORS also supports other types of HTTP requests. Using CORS enables a web programmer to use regular XMLHttpRequest, which supports better error handling than JSONP. On the other hand, JSONP works on legacy browsers which preclude CORS support. CORS is supported by most modern web browsers. Also, whilst JSONP can cause XSS issues where the external site is compromised, CORS allows websites to manually parse responses to ensure security.

A CORS enabled server or web application classifies all HTTP requests as:

1. A CORS request that contains the HTTP "Origin" header in any type of request
2. A CORS preflight request that contains the "Access-Control-Request-Method" header in an OPTIONS request
3. A "generic" request that does not contain any CORS HTTP headers

To initiate a cross-origin request, a browser sends the request with an Origin HTTP header. The value of this header is the site that served the page. For example, suppose a page on http://www.example-social-network.com attempts to access a user's data in online-personal-calendar.com. If the user's browser implements CORS, the following request header would be sent:

```
Origin: http://www.example-social-network.com
```

If online-personal-calendar.com allows the request, it sends an Access-Control-Allow-Origin header in its response. The value of the header indicates what origin sites are allowed. For example, a response to the previous request would contain the following:

```
Access-Control-Allow-Origin: http://www.example-social-network.com
```

If the server does not allow the cross-origin request, the browser will deliver an error to example-social-network.com page instead of the online-personal-calendar.com response.

To allow access to all pages, a server can send the following response header:

```
Access-Control-Allow-Origin: *
```

OpenEdge Support

OpenEdge uses a 3rd party Java open source package named CORS Filter. All of the primary CORS functionality resides in that product. OpenEdge has integrated it into their Java container web applications by implementing a Spring Security filter bean so that it can be configured from within the Spring Security configuration files, with all of the other web application security.

The default configuration for CORS is in the appSecurity-xxxxx.xml configuration files. Look for the name "OECORSFilter" to find where and how the CORS bean is included. The first thing that was added to the Spring Security configurations was the bean definition:

```
<b:bean id="OECORSFilter"
            class="com.progress.rest.security.OECORSFilter" >
            <!-- Examples:
            <b:property name="allowAll" value="false" />
            <b:property name="allowDomains"
value="http://studio.progress.com,http://mobile.progress.com" />
            <b:property name="allowSubdomains" value="false" />
            <b:property name="allowMethods" value="" />
            <b:property name="messageHeaders" value="" />
            <b:property name="responseHeaders" value="" />
            <b:property name="supportCredentials" value="true" />
            <b:property name="maxAge" value="-1" />
            -->
    </b:bean>
```

The bean is implemented in the Java class com.progress.rest.security.OECORSFilter.java. The bean has a number of properties that control how it operates:

Property	Datatype	Default	Range	Comments
allowAll	boolean	"true"	{"true"|"false"}	When true, accept client requests that do not include the CORS headers. When false all client requests must include CORS headers. *If CORS headers are present in the request*, **regardless of this property's setting,** *the request must pass the requirements imposed by the remaining properties.*
allowDomains	String	"*"	{"*"|"domain1[,domain2...]}	Which network domains are allowed. Must be in the format: {http|https}://host[:port]}. The value may be "*" for all domains, or a explicit list of comma separated domains. If the HTTP request is not from an allowed domain the request is denied.
allowSubdomains	boolean	"false"	{"true"|"false"}	Modify the domain checking to allow sub-domains.
allowMethods	String	"GET,PUT,POST,DELETE,OPTIONS"	Valid http method name	Allow a client to use these HTTP verbs. Use a comma separated list with no whitespace. If a HTTP verb is used that is not in the list, the request is denied.
messageHeaders	String	"(see (a) below)"	Any valid string value	A comma separated list of allowable HTTP request header names. If a header is not in the list, the request is denied.
responseHeaders	String	"(see (b) below)"	Any valid string value	A comma separated list of HTTP response headers the client application is allowed to access.
supportCredentials	boolean	"true"	{"true"|"false"}	Accept user credentials via COOKIES (such as the JSESSIONID cookie).
maxAge	integer	-1	{-1|+n}	Max time for a resource to be granted: -1 is infinity, do not use zero, positive numbers are in seconds

(a)
"Accept,Accept-Language,Content-Language,Content-Type,X-CLIENT-CONTEXT-ID,Origin,Access-Control-Request-Headers,Access-Control-Request-Method,Pragma,Cache-control"
(b) "Cache-Control,Content-Language,Content-Type,Expires,Last-Modified,X-CLIENT-CONTEXT-ID,Pragma"

The next part of the CORS configuration is to add the bean into the Spring Security security (filter) list. In the OE case, the CORS filter is inserted just before the built-in FILTER_SECURITY_INTERCEPTOR bean. This puts the CORS check after the user authentication filters and just before the filters that do the granting/revoking of user access to a resource. In this way the CORS filter may use the authenticated user's credentials for advanced filtering.

```
<custom-filter before="FILTER_SECURITY_INTERCEPTOR"
                       ref="OECORSFilter" />
```

Configurations

The configuration combinations can be extensive. A good policy is to keep it simple if possible.

*Leaving the default ("*allowAll*"="true") will allow "generic" requests to access resources without being required to send CORS headers

*To require all HTTP clients, not just the Java Script type, to use active CORS access control, turn the "*allowAll*" property to "false". The CORS filter will now begin checking all client's for the required CORS headers and control access to all resources. The default is "true", which allows any HTTP request, from any domain, as long as the request does not include any CORS headers.

All CORS requests (containing CORS headers) are always checked, but because the default domain list is "", all clients are allowed access to resources. To begin limiting client Domains to only certain ones, configure the "*allowDomains*" property with a comma separated list of Domain names. Always supply "http" or "https", the DNS domain name, and optionally a non-standard HTTP/HTTPS port. Do not append the '/' path separator, and only add a port # if not using the http defaults. Wildcard regular expressions are not supported in explicit client domain lists.

*The next level of control exists in which HTTP methods clients can use to access resources. The "*allowMethods*" property is a command delimited list of valid HTTP method names - all in upper case. It is a 'grant' list, so include the methods you want to allow. Do note that this list of method names is for all resources in the web application and will need to be coordinated with the resource authorization controls. The recommendation is to leave this the default value.

*The "*requestHeader*" and "*responseHeader*" properties are for advanced http client and server use, and should not be used unless all of the web application's clients are coded to use these headers. If you do specify this property, Copy the default list of headers from (a) or (b) respectively, and append the additional headers you want to allow.

*The "*maxAge*" property is a simple expiration for how long the client is granted resource access before they have to request access again. Using this property does require a client to intercept a denied response and actively request access again.

*The "*supportCredentials*" property controls whether the CORS filter allows the client to send user credentials in the form of a COOKIE. The default is "ture", which allows the client to use user login sessions via COOKIEs. If you want the client to not send COOKIE user credentials set the value to "false". You should only set this property to "false" when you want to supply totally stateless resources for anonymous users.

Debugging

Debugging CORS filtering is done by adjusting the debugging level in the application's log4j.properities file. Look for the debug trigger that includes '_com.progress.rest.security' and set the logging level to "DEBUG".