Introducing the CSRFPolicy in Alfresco Share

21 Replies

A few weeks back we added a CSRF filter to Share. It is now available on Alfresco Enterprise 4.1.4 and also to the Community on HEAD.

This blog post will be about how the new filter affects your current Alfresco Share installation or your custom Share extension module. In most cases the new filter will actually not affect your installation or custom code at all, however there might be a few edge cases where you need to change your code slightly. The post will also describe how you can configure the CSRF filter to work behind one or more proxies, how to run it with 3rd party plugins behaving badly, how it can be used to stop specific repository services from being accessible directly from the browser through Share's proxy and also how to turn off the filter.

To learn more about CSRF feel free to read OWASP's introduction article about CSRF: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

Note! This post misspells the word "referer" during the entire post to honour the spec that introduced the `Referer` header with the incorrect spelling :-)

How does the filter work?

The filter is implemented in a class named `org.alfresco.web.site.servlet.CSRFFilter` that reads a config section in `share-security-config.xml` named `CSRFPolicy` which will describe how and when the filter shall mitigate CSRF, summarized as:

- Each logged in user will receive a secret CSRF token.
- The token will be communicated to the browser using a cookie named `Alfresco-CSRF-Token`.
- When a logged in user performs a `POST`, `PUT` or `DELETE` http request against Alfresco Share the token MUST be passed in the request using one of the following methods:
    - As a custom http request header named `Alfresco-CSRF-Token`
    - As a url parameter named `Alfresco-CSRF-Token`.
      Note! Most often the header will be required, but in certain circumstances a header cannot be used and only then the token may be passed using a url parameter. The default config only

accepts the url parameter when the `Content-Type` header starts with `multipart/`.

- Every time the logged in user goes to a new Share page the token will be renewed.
- The filter will also check the `Referer` and `Original` http request headers matches the current domain (if present in the request).

To dig into the config, take a look at the latests revisionhttp://svn.alfresco.com/repos/alfresco-open-mirror/alfresco/HEAD/root/projects/slingshot/config/alfresco/share-security-config.xml. A detailed description of all available options can be found at the bottom of this post.

Do you need to tweak your code?

If your custom code only is reading data using GET requests you will not have to do anything.

If your custom code only is using the standard`Alfresco.util.Ajax`, `alfresco/core/CoreXhr` or `Alfresco.forms.Form` javascript classes when creating/updating/deleting data you will also not have to do anything. Everything will be handled for you by:

- `Alfresco.util.Ajax & alfresco/core/CoreXhr` – will automatically take the token from the cookie and add it as a request header for every request.
- `Alfresco.forms.Form` – will automatically take the token from the cookie and add it as a url parameter to when submitting an`multipart/form-data` request.
  (When submitting a form as JSON the `Alfresco.util.Ajax` will be used internally)

Note! In the unlikely event that you do need to change your code make sure to:

- *Always read the token from the cookie just before the request is sent.*
  Reason being that if you have multiple tabs opened in your browser one tab could, if accessing a new page, refresh the token in the session. If another tab then had saved its token in a variable it would suddenly be stale. But if all tabs always read the cookie just before submission they will always get the latest token even if the user changed pages in another tab. To read the token from the cookie simply use `Alfresco.util.CSRFPolicy.getToken()`.
- *Always check if the Alfresco.util.CSRFPolicy object exists before using it*
  That way your code will work in all versions of Alfresco Share.

Now let's take a look however at the scenarios when you might need to tweak your code or change the default configuration.

1. You are making an XMLHttpRequest with method POST, PUT or DELETE without using the Alfresco.util.Ajax or alfresco/core/CoreXhr classes

Perhaps you are using the native `XmlHttpRequest` object or a 3rd party library such as jQuery. If that is the case you will need to add code looking something like this to pass the token:

```
if (Alfresco.util.CSRFPolicy && Alfresco.util.CSRFPolicy.isFilterEnabled())
{
   xhrHeadersObject[Alfresco.util.CSRFPolicy.getHeader()] = Alfresco.util.CSRFPolicy.g
}
```

Or if your using YAHOO.util.DataSource to load data with POST requests your code shall look like this:

```
if (Alfresco.util.CSRFPolicy && Alfresco.util.CSRFPolicy.isFilterEnabled())
{
   yuiDataSource.connMgr.initHeader(Alfresco.util.CSRFPolicy.getHeader(), Alfresco.uti
}
```

2. You are making a form upload with enctype multipart/form-data without using Alfresco.forms.Form

When uploading a file by submitting a form with enctype `multipart/form-data` it is not possible to set a header on the request, the reason is not because of the enctype specifically but due to the fact that its not possible to set a header on any form submission in the browser. Therefor you need to pass the token as a url parameter instead. If you are using the`Alfresco.forms.Form` class this will be handled for you automatically but otherwise you have to add the token as a url parameter using code looking something

like this:

```
if (Alfresco.util.CSRFPolicy && Alfresco.util.CSRFPolicy.isFilterEnabled())
{
   url += "?" + Alfresco.util.CSRFPolicy.getParameter() + "=" + encodeURIComponent(Alf
}
```

3. You are using a flash movie inside Share to send http requests with method POST

If you are using a flash movie to upload files (it uses the`flash.net.FileReference` ActionScript class which will perform a`multipart/form-data` request) make sure to add the token as a url parameter in your Javascript before passing in the url to the flash movie. If your Flash movie is performing `application/json` or other text based `POST` requests (it uses the `flash.net.URLRequest` and/or `flash.net.navigateToURL` ActionScript classes and methods) then make sure to pass in the token and the name of the header so it can be set from the flash movie.

Note! Flash cannot use ActionScript to directly get hold of the token since it may not read the browser cookies or read http response headers. It is possible however to use the `flash.external.ExternalInterface` ActionScript class to call a custom javascript method you have included on the page.

4. You are writing a non-browser client, i.e. a mobile app

Such app should be targeted against the repo in which there is no CSRF filter, meaning you don’t have to do anything.

That’s it! If your don’t have code matching any of the described patterns you should not have to modify your code in any way. Please continue reading to be aware of other scenarios when you might be required to alter the default configuration.

Another system is sending POST requests to your Alfresco Share server

If there is a scenario in your system environment where servers from other domains actually shall be allowed to POST requests you will need to reconfigure the `CSRFPolicy` config in your `share-config-custom.xml` file to not check for a token or a `Referer` or `Origin` header. To do so simply:

1. Copy the entire `CSRFPolicy` config in `share-security-config.xml`
2. Paste it into your `share-config-custom.xml` file and make sure it is replacing the old config section:
   `<config evaluator="string-compare" condition="CSRFPolicy"replace="true">`
3. Place the following snippet as the first child to the `<filter>` element:

```
<rule>
   <request>
      <method>POST</method>
      <path>/page/trusted/call/1|/page/trusted/call/2</path>
   </request>
   <action name="assertReferer">
      <param name="always">false</param>
      <param name="referer">https://www.trustedserver.com/.*</param>
   </action>
   <action name="assertOrigin">
      <param name="always">false</param>
      <param name="origin">https://www.trustedserver.com</param>
   </action>
</rule>
```

The CSRF filter will compare the incoming request with all the rules' request elements to find one that match, when it does it will invoke all the defined actions for that rule and then let the normal Share processing begin. In this case it means that if the external page was `POST`:ing to`/page/trusted/call/1` or `/page/trusted/call/2` this (and no other) rule will be used and its actions will run. The first action will assert that full page url in the `Referer` header (if present)

equals `https://www.trustedserver.com/.*` and that the protocol and domain in the `Origin` header (if present) equals`https://www.trustedserver.com` .

I am running Alfresco Share behind one or more proxy server(s) and I get errors…

If you have placed Alfresco Share behind one or more proxy servers that rewrites all the urls before Alfresco Share is reached you might see errors in your log about the Referer or Origin headers not matching the current server. If that is the case see if its possible to also rewrite the Origin and Referer headers if they a) are present and b) match your proxy servers domain.
Note! When doing so make sure to not use a start-with-like-comparison without ending the proxy server's domain with a front slash ("/"), since`http://www.my-proxy-server.com.evil-server.se/csrf.html` obviously starts with `http://www.my-proxy-server.com` but not `http://www.my-proxy-server.com/`.

If that is not possible you will have reconfigure the filter to accept requests from specific domains by simply:

1. Copy the entire `CSRFPolicy` config in `share-security-config.xml`
2. Paste it into your `share-config-custom.xml` file and make sure it is replacing the old config section: `<config evaluator="string-compare" condition="CSRFPolicy" replace="true">`
3. Modify every referer action to accept additional urls by changing:

```
<action name="assertReferer">
   <param name="always">false</param>
</action>
```

…to…

```
<action name="assertReferer">
   <param name="always">false</param>
   <param name="referer">https://www.proxyserver1.com/.*|https://www.proxyserver2.com/.*</pa
```

```
</action>
```

4. Modify every origin action to accept additional urls by changing:

```
<action name="assertOrigin">
   <param name="always">false</param>
</action>
```

…to…

```
<action name="assertOrigin">
   <param name="always">false</param>
   <param name="origin">https://www.proxyserver1.com|https://www.proxyserver2.com</param>
</action>
```

The `Referer` will contain the entire url from which the request was submitted but the `Origin` will only include the protocol and domain (hence the `.*` wildcard at the end of the `referer` parameter).

In case you wondered, the current domain will continue to be accepted, meaning that you can login and use Share using its own “internal” domain.

I just want to disable the filter!

There is no real reason why you should need to turn off the filter if it is configured correctly and you are running a standard Share installation. However if you have installed a 3rd party plugin that is not using the Alfresco provided classes for sending `XMLHttpRequest`s or submitting forms you should

contact the plugin developer and ask him to read this blog post so he/she can update the plugin. You will then have to make a decision to either uninstall your plugin OR lower the security level in the filter and not check for tokens anymore (at least until a new version of the plugin has been released). To stop checking for tokens, but continuing to check the Origin and Referer headers when available for logged in users, just add the following code snippet in your `share-config-custom.xml` file:

```
<config evaluator="string-compare" condition="CSRFPolicy" replace="true">
   <filter>
      <rule>
         <request>
            <method>POST|PUT|DELETE</method>
            <session>
               <attribute name="_alf_USER_ID">.*</attribute>
            </session>
         </request>
         <action name="assertReferer">
            <param name="always">false</param>
         </action>
         <action name="assertOrigin">
            <param name="always">false</param>
         </action>
      </rule>
   </filter>
</config>
```

If you have custom code that fails for the reason mentioned in the previous section make sure to fix them instead of disabling the filter. It shouldn't take long.

If you still, for what ever reason, want to disable the filter just add the following code snippet in your `share-config-custom.xml` file:

```
<config evaluator="string-compare" condition="CSRFPolicy" replace="true">
```

```
    <filter/>
</config>
```

I have a repository webscript or service that I don't want to be accessible through Share's proxy…

This is a bonus feature of having a configurable CSRF filter, that it can be used to completely block certain services in the repository. Perhaps you have an API that only shall be accessible from a) other clients than Share OR b) from server side Java or Javascript code running on the Share server (rather than in the browser as a Java Applet or client side Javascript).

If that is the case you can add the urls to those services to the CSRF filter and make sure it throws an error when they are accessed. To do this simply:

1. Copy the entire `CSRFPolicy` config in `share-security-config.xml`
2. Paste it into your `share-config-custom.xml` file and make sure it is replacing the old config section: `<config evaluator="string-compare" condition="CSRFPolicy" replace="true">`
3. Add the following code snippet as the first child to the <filter> element:

```
<rule>
   <request>
      <path>/proxy/alfresco/acme/special/services/.*</path>
   </request>
   <action name="throwError">
      <param name="message">It is not allowed to access this url from your browse
   </action>
</rule>
```

A detailed description of the CSRFPolicy configuration

The next code snippet is will give you a detailed description of all available options in the `CSRFPolicy` configuration. It is probably only worth reading in case you're really interested or have run into trouble.

Cheers and thanks for reading this far!

```
<config evaluator="string-compare" condition="CSRFPolicy">

  <!--
    (Mandatory) Only 1 client element is allowed.
    Describes what names are used to communicate the token back and forth
    between the server and the client.
  -->
  <client>

    <!--
      (Mandatory) A client element must have exactly 1 cookie element.
      Name of the cookie that will hold the token, used by the client side to
      grab the value.
    -->
    <cookie>

    <!--
      (Mandatory) A client element must have exactly 1 header element.
      Name of the custom Http header to place the token in when sending a request
    -->
    <header/>

    <!--
      (Mandatory) A client element must have exactly 1 parameter element.
      Name of the parameter to place the token in when sending a request
    -->
    <parameter/>
```

```
</client>

<!--
  (Mandatory) Only 1 filter element is allowed.
  The filter will look for 1 rule with a matching request and execute its
  actions (if any). An empty filter element means the CSRF filter is disabled,
  in other words will allow all requests to pass.
-->
<filter>

  <!--
    (Optional) Zero or more rule elements are allowed.
    A rule contains a description of a request and a set of actions to execute.
  -->
  <rule>

    <!-- (Mandatory) A rule element must have exactly 1 request element -->
    <request>

      <!--
        (Optional) A request element may have exactly 1 method element.
        Holds a regular expression that will be matched against the request's
        method.
      -->
      <method/>

      <!--
        (Optional) A request element may have exactly 1 path element.
        Holds a regular expression that will be matched against the request's
        "share path", i.e. /page/start-workflow or /proxy/alfresco/api/people
      -->
      <path/>

      <!--
        (Optional) A request element may have any number of header elements.
        Holds a regular expression that will be matched by the header specified
        by the name attribute.
      -->
```

```
    <header name=""/>

    <!-- (Optional) A request element may have exactly 1 session element -->
    <session>
      <!--
        (Optional) A session may have multipe attribute elements.
        Holds a regular expression that will be matched by the session
        attribute specified by the name attribute. A closed attribute element
        indicates that the session attribute does not exist.
        I.e. <attribute name="Alfresco-CSRFToken"/> means that the token has
        not yet been created.
      -->
      <attribute name=""/>
    </session>

  </request>

  <!--
    (Optional) A rule element may have multiple action elements.
    Below is a list of all available actions:
  -->

  <!-- Generate the token -->
  <action name="generateToken">
    <!--
      (Mandatory) An "generateToken" action may have exactly 1 "session"
      param.
      Holds the name of the session attribute in which to place the token,
      shall match the client element's session element above.
    -->
    <param name="session"/>

    <!--
      (Mandatory) A "generateToken" action may have exactly 1 "cookie" param.
      Holds the name of the cookie in which to place the token, shall match
      the client element's cookie element above.
    -->
    <param name="cookie"/>
```

```
</action>

<!-- Clear the token value -->
<action name="clearToken">
  <!--
    (Mandatory) A "clearToken" action may have exactly 1 "session" param.
    Holds the name of the session attribute which value shall be cleared,
    shall match the client element's session element above.
  -->
  <param name="session"/>

  <!--
    (Mandatory) A "clearToken" action may have exactly 1 "session" param.
    Holds the name of the cookie which value shall be cleared, shall match
    the client element's cookie element above.
  -->
  <param name="cookie"/>
</action>

<!--
  Assert the request's Referer header matches the current domain.
  If not an error will be thrown.
-->
<action name="assertReferer">
  <!--
    (Mandatory) An "assertReferer" action may have exactly 1 "always" param.
    Decides when to compare the incoming requests Referer header to the
    current domain, if set to:
    - true: Always compare, even when no Referer header was provided in the
            request.
    - false: Only compare if a Referer header was provided in the request
  -->
  <param name="always"/>

  <!--
    (Optional) An "assertReferer" action may have 0 or 1 "referer" param.
    Holds a regular expression that will be matched against the incoming
    Referer header if the incoming Referer header does not match Share's domain.
```

```
    -->
    <param name="referer"/>
  </action>

  <!--
    Assert the requets's Origin header matches the current domain.
    If not an error will be thrown.
  -->
  <action name="assertOrigin">
    <!--
      (Mandatory) An "assertOrigin" action may have exactly 1 "always" param.
      Decides when to compare the incoming requests Origin header to the
      current domain, if set to:
      - true: Always compare, even when no Origin header was provided in the
              request.
      - false: only compare if a Origin header was provided in the request
    -->
    <param name="always"/>

    <!--
      (Optional) An "assertOrigin" action may have 0 or 1 "origin" param.
      Holds a regular expression that will be matched against the incoming
      Origin header if the incoming Origin header does not match Share's
      domain.
    -->
    <param name="origin"/>
  </action>

  <!-- Will throw an error -->
  <action name="throwError">

    <!--
      (Optional) A "throwError" action may have exactly 1 "message" param.
      Holds the error message that will be used when throwing the error.
    -->
    <param name="message"/>
  </action>
</rule>
```

```
    </filter>

</config>
```