1 Background

One of the outputs of the SEE-GEO project was “façade” code to sit in front of an HTTP web service, intercept client requests, and check that the request URL contains a valid session identifier. Requests with a valid session id are forwarded to the web service behind the façade; all other requests are rejected, with the client asked to initiate authentication of the user to a federated SAML Service Provider (SP) component of the façade, at a separate URL containing a new session id. If authentication of the user succeeds there, the new session id becomes valid and can be used in subsequent client requests to the façade, which will be forwarded to the protected web service. A firewall ensures that only the façade can directly communicate with the underlying service.

The façade code produced by SEE-GEO was tailored to a specific web service, the Open Geospatial Consortium (OGC) Web Map Service (WMS). In addition, the SAML SP component was code developed within the project. The objectives of the WSTIERIA project included:

- support for arbitrary HTTP (non-SOAP) web services
- use of a standard SAML SP, specifically Shibboleth, in place of the SEE-GEO custom SP code.

From the description above it should be clear that the façade operates partly as an HTTP proxy. During January and February 2010, initially as part of familiarisation with the previous SEE-GEO work, it was decided to see whether an off-the-shelf web proxy could be made to do the session verification and proxying part of the job, with a Shibboleth SP handling the SAML parts. Initially the perl HTTP::Proxy package from the CPAN library was investigated. This appeared to have suitable functionality, except that it only operates as a forward proxy. That would have required reconfiguration of every client to know about the proxy, which would be unacceptable for our purpose. The next tool considered was the Apache web server (httpd). This has the required reverse-proxy functionality, where requests to a URL range managed by the web server are forwarded to some other URL range (which need not be on the same server). It turns out that it is possible to combine this with verification of session ids by means of the URL pattern matching and rewriting features of Apache (mod_rewrite). The rest of this note describes the method.

Although using Apache in this way was initially intended only for experimental purposes, it has a number of benefits that may make the technique suitable for providing federated access management to HTTP web services at a production level:

- Apache is already widely deployed at sites that run web services. It is part of most Linux distributions. Even if it is not already deployed, for example where Windows IIS is the primary web server, it is available as a well supported, well documented and simple to install download, sustained over the long term by an active and respected developer community. This will compare well for a cautious web service administrator against installing and relying on bespoke code produced by a short-term project.

- The proxy functionality within Apache is robust and unlikely to introduce operational problems. It is also likely to run faster than new code that has not been extensively tuned.

- Customisation for a particular web service can be done partly at the level of the Apache configuration language (httpd.conf), which for many administrators is already a familiar tool. The rest is achieved through short scripts, in any scripting language the administrator is familiar with (perl is used in the examples in this note).
Federated Access to an HTTP Web Service Using Apache

d) The façade approach means that federated access management can be added without touching the underlying web service in any way.

2 The Example Web Service

Since the previous SEE-GEO work had used OGC WMS services as examples, and these were still available, it was decided to continue using one of those example web services (a WMS providing access to EuroGlobalMap data). A WMS allows a client to obtain map images of a geographical area specified in the request. Like other OGC web services, a WMS uses only simple HTTP requests and responses. The request is expressed as CGI key/value pairs; the response is XML. Also as with other OGC services, a WMS must support a GetCapabilities request:

http://server:port/path?REQUEST=GetCapabilities&version=1.1.1&service=WMS

This returns an XML document describing the properties of the service. The particular server, port and path of the service for which federated access management is to be provided are not important and are left as placeholders in the text here.

3 Apache Configuration

Installation and basic configuration of Apache are as described at http://httpd.apache.org. Some modules must be added to the basic configuration by editing httpd.conf and un-commenting the following lines (which are not all located together):

LoadModule ext_filter_module modules/mod_ext_filter.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule rewrite_module modules/mod_rewrite.so

This enables adding the following URL-matching and rewriting rules to the configuration. We will use the URL path /wms for proxying to the protected web service, so the first order of business is to prevent clients from accessing that path directly:

RewriteRule ^/wms - [NC,F]

This rule will match /wms at the beginning (^) of a URL. The [F] flag causes any URL matching this rule to be rejected with a 403 (Forbidden) status code. The [NC] flag (“no case”) forces a case-insensitive pattern match for some additional protection. The ‘¬’ indicates that no URL substitution is to be performed by this rule, leaving the existing path untouched.

How should the client access the protected service? We will look for a session id if the URL from the client begins with /session/. The pattern-matching part of this rule should therefore be:

RewriteRule ^/session/(.*)

‘.’ matches any character and ‘*’ means “zero or more times”, so the whole of the rest of the URL path will be swallowed as the session id. Finer control could be obtained from a more specific pattern, but in the OGC example case the interesting parts of a GET request are in the query string (service=, REQUEST=, etc.) By default the query string is not matched against, so no harm is done.

We need somehow to look up the session id from the request in a list of valid session ids. The feature of mod_rewrite that will allow us to do this is called a RewriteMap. The text matched by a pattern in parentheses like (.*$) is always made available in a numbered working variable, in this case $1. The URL substitution part of the rule should look up the client’s session id ($1) in a RewriteMap called sessions, and if it is not found then substitute the URL /forbid, which will be rejected with a Forbidden status similarly to /wms above. The syntax for this is:

${sessions:$1}/forbid

If the session id is found in the map, the value found there is substituted (instead of /forbid). We will arrange that this value will be /wms, to initiate proxying as described below. (The previous rule to forbid direct use of...
/wms will not apply, because rules are applied strictly in order of appearance). Putting the pattern matching part together with the required lookup and substitution gives:

```
RewriteRule ^/session/(.*) ${sessions:$1}/forbid
RewriteRule ^/forbid [F]
```

The `sessions` map must also be defined, as follows. Various kinds of map are available. The simplest, a text file, is shown here (and should be adequate for small to moderate numbers of active sessions) but dbm-style databases and user-defined external programs are also supported:

```
RewriteMap sessions txt:/path/to/sessions.txt
```

The text file can be anywhere but must be readable and writable by Apache. Initially (manually) created empty, after some time it might contain:

```
12345 /wms
45678 /wms
90123 /wms
```

Here there are three valid session keys (12345, 45678 and 90123), all of which map to the URL prefix /wms, which will initiate proxying.

Another flag, [P], allows URLs matching a particular pattern to be proxied either to another path on the same server, or to a URL on a different server:

```
RewriteRule ^/wms(.*) http://server:port/path$1 [P]
```

Here, /wms at the start of a URL will initiate proxying, with the remainder of the URL ($1) being passed along to the destination server. In our case this will be the server hosting the web service to which access is being controlled (the service behind the façade). Note that this service need not be modified. It must simply be accessible from the IP address of the machine hosting the façade (i.e., doing the proxying) and, usually, inaccessible from elsewhere (by means of a firewall or its web server configuration).

## 4 Creating Authenticated Sessions

It has not yet been explained how session ids get into the sessions.txt file.

The principle is that an ordinary browser-based access is made to a new-session script hosted on a web site that requires federated authentication. This access is initiated either:

- by the user, before requests are made to the façade (this requires user education) or
- by a façade-aware client, in response to a “Forbidden” status code returned by the façade.

The URL of this script is completely unrelated to the façade URLs discussed in the previous section. If required, browser-based Identity Provider (IdP) discovery is performed by the SP hosting the new-session script in any of the usual ways (for example, via a federation Where Are You From service). The user enters credentials at the IdP and, if successfully authenticated, SAML assertions about the user are passed from the IdP to the SP. The new-session script decides whether the authenticated user is authorised to use the web service being protected by the façade. If so, it creates a new session id and adds it to the sessions.txt file in any of a number of possible ways:

- If the the new-session script SP is on the same web server as the façade, the script may modify the file directly.
- The same applies if the SP is on a different web server on the same system.
- If the new-session script SP is on a different system, it may still be able to update the sessions.txt file if a suitable mechanism has been established between the two systems beforehand (either a shared file system or access to an agent on the façade system controlled by IP-address restriction, ssh keys, SSL client certificates, or some other means).
While experimenting with the approach described here, we used a Shibboleth SP that was already set up on one system for other purposes to host the new-session script. When a user was authorised, this script invoked an agent on the system hosting the façade (the author’s desktop machine) by making an HTTP request to an IP-address protected script on the desktop system. We mention the possibilities here because some readers wishing to experiment may (for example) be in a position to add a new script easily to an existing Shibboleth SP but not to modify the configuration of its web server.

The final response from the new-session script to the user’s browser is an HTML page containing a URL that will be accepted by the façade (constructed by appending a randomly generated new session id to the /session URL path described in the previous section). The page should also contain instructions to the user to copy this URL and paste it into the client application. This requires no modification of client applications that may expect to have direct HTTP access to a web service. Alternatively, where a client application can be modified to handle “Forbidden” and initiate the browser flow, it could be further modified to obtain the required URL from a cookie planted by the new-session script. In either case, the client application can then proceed to access the façade URL as if it were directly accessing the underlying web service. Apache will proxy requests to the server running the actual web service and return the responses.

5 OGC Web Services

As mentioned previously, the example being used while experimenting with the approach described here was an OGC WMS. Mostly OGC web services require no special treatment. There is one exception however. An OGC web service must support the GetCapabilities request described previously. This returns an XML document describing the service, which includes the absolute URL of the service endpoint in multiple places, for example:

```
<GetMap>
  <Format>image/gif</Format>
  <Format>image/png</Format>
  <Format>image/jpg</Format>
  <Format>image/tif</Format>
  <Format>image/bmp</Format>
  <DCPType>
    <HTTP>
      <Get>
        <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
      </Get>
    </HTTP>
  </DCPType>
</GetMap>
```

This XML document is produced by the underlying OGC web service, not the façade. It will therefore contain the URL of the protected service endpoint, not of the façade. If the client uses these absolute URLs to access the service, it will attempt to bypass the façade and communicate with the underlying service directly. This attempt will be blocked, resulting in failure.

Such location-dependency in an application protocol may well be seen as a drawback, potentially causing problems also for other web intermediaries, but it is a fact. The façade for an OGC web service must therefore be not just a proxy but also a rewriting proxy, changing these endpoint URLs on the fly to point to the façade rather than the underlying service (similarly to the operation of products like EZproxy).

It turns out that Apache is flexible enough to handle this too. Proxyed responses can be run through a text filter before being passed back to the client, like this:

```
<Proxy http://server:port>
  SetOutputFilter munge
</Proxy>
```

The server and port here are those of the service being protected (no path is needed). The filter (named munge here) must be defined beforehand.
ExtFilterDefine munge cmd="/path/to/perl.exe /path/to/urlRewrite.pl"

This will invoke an external command (cmd), in this case to run a perl interpreter on a script called urlRewrite.pl. A simple (string rather than XML based) version of such a script is shown below:

```perl
while (<>) {
    s|http://server:port/path|http://façade/session/$ENV{SESSION}|g;
    print;
}
```

This looks for the server and port of the underlying service in each line of standard input and substitutes those of the façade, appending the required session id from the environment variable SESSION. This variable needs to be set at the time the session id is looked up in the session map. Previously we had:

```
RewriteRule ^/session/(.*) $\{sessions:$1\}/forbid
```

We can set the SESSION environmental variable here to the required value (held in $1) using the `[E]` flag:

```
RewriteRule ^/session/(.*) $\{sessions:$1\}/forbid` [E=SESSION:$1]
```

A useful addition would be only to go to the trouble of running the filter for GetCapabilities requests, rather than on every request. The filter definition can be modified so that it will only be run when another environmental variable is set:

```
ExtFilterDefine munge cmd="/path/to/perl.exe /path/to/urlRewrite.pl"  
EnableEnv=DO_FILTER
```

The DO_FILTER environmental should be set only for requests with REQUEST=GetCapabilities in the query string. As mentioned previously, a normal RewriteRule will not match the query string part of a URL. To do that, a RewriteCond rule is required:

```
RewriteCond %{QUERY_STRING} REQUEST=GetCapabilities
```

A RewriteCond applies only to the immediately following RewriteRule, in this case:

```
RewriteRule ^/wms - [E=DO_FILTER:TRUE]
```

The combination of this RewriteCond and RewriteRule will match URLs starting with /wms where the query string contains a GetCapabilities request. If there is a match, the request URL is left unmodified (`-`) but the DO_FILTER environmental variable is set to TRUE, enabling the rewriting filter.

### 6 Testing the Method

To check that the method worked as expected in practice, we used the following components:

- an existing IP-address protected OGC WMS serving EuroGlobalMap data, completely unmodified
- an open-source WMS client (the Quantum GIS desktop application), also unmodified
- an existing Shibboleth SP setup
- a new Apache installation (on a Windows desktop) configured as described previously

The screen shots below show the output from the new-session script on the Shibboleth SP and then from Quantum GIS when opening the new-session URL displayed by that script and displaying some of its map layers. It can be seen that the WMS client in the unmodified Quantum GIS happily works with the façade in place of the underlying EuroGlobalMap WMS and that, due to the Shibboleth SP, only authenticated users from UK federation IdPs can obtain a valid session URL, and then only if they are authorised by the new-session script. (For testing, the script simply required that the user should have a scoped affiliation value of member@edina.ac.uk).
Successfully Authenticated

The temporary access link is:
http://twmpa.ucc.ed.ac.uk/session/6594803

Create a new WMS connection

Connection details

Name: ProtEuroGlobalMap
URL: http://twmpa.ucc.ed.ac.uk/session/6594803
7 Collected Configuration

The Apache configuration described previously is shown collected below, in the required order, and with one extra statement (RewriteEngine on) that is needed to enable mod_rewrite processing:

```
RewriteEngine on
RewriteMap sessions txt:/path/to/sessions.txt
RewriteRule ^/wms - [NC,F]
RewriteRule ^/session/(.*) ${sessions:$1|/forbid} [E=SESSION:$1]
RewriteRule ^/forbid - [F]
RewriteCond %{QUERY_STRING} REQUEST=GetCapabilities
RewriteRule ^/wms - [E=DO_FILTER:TRUE]
RewriteRule ^/wms(.*) http://server:port/path$1 [P]

ExtFilterDefine munge cmd="/path/to/perl.exe /path/to/urlRewrite.pl" \  
  EnableEnv=DO_FILTER

<Proxy http://server:port>
  SetOutputFilter munge
</Proxy>
```