AAA Policy Extra Exercise - Summary

This simple example demonstrates how the AAA Policy object can be used to authenticate and authorize messages sent to a DataPower service.
In this case *Basic Auth* is the mechanism used to identify the sender of the message.
As you go through the exercise, you will see many security technologies that the DataPower supports.
For this simple example, authentication and authorization information is stored in an AAA file on the DataPower.
(This AAA file is at the bottom of this document.)

Build an AAAPolicy object

You are going to create an AAAPolicy object in the same domain as your *HelloWorld* MPGW.

AAA Policy: *HelloWorld-AAAPolicy*

- *Main* tab:
  - just enter 'Name' *HelloWorld-AAAPolicy*
  - add some relevant 'Comments'

- *Identity extraction* tab
  - select 'HTTP Authentication header'
    (notice all of the other options)

- *Authentication* tab
  - for 'Method' choose 'Use AAA information file'
    - AAA information file URL:
      - local:///
      - 'Upload' a file created from the text at the bottom of this document

- *Resource extraction* tab:
  - Resource information
    - select 'URL sent by client'

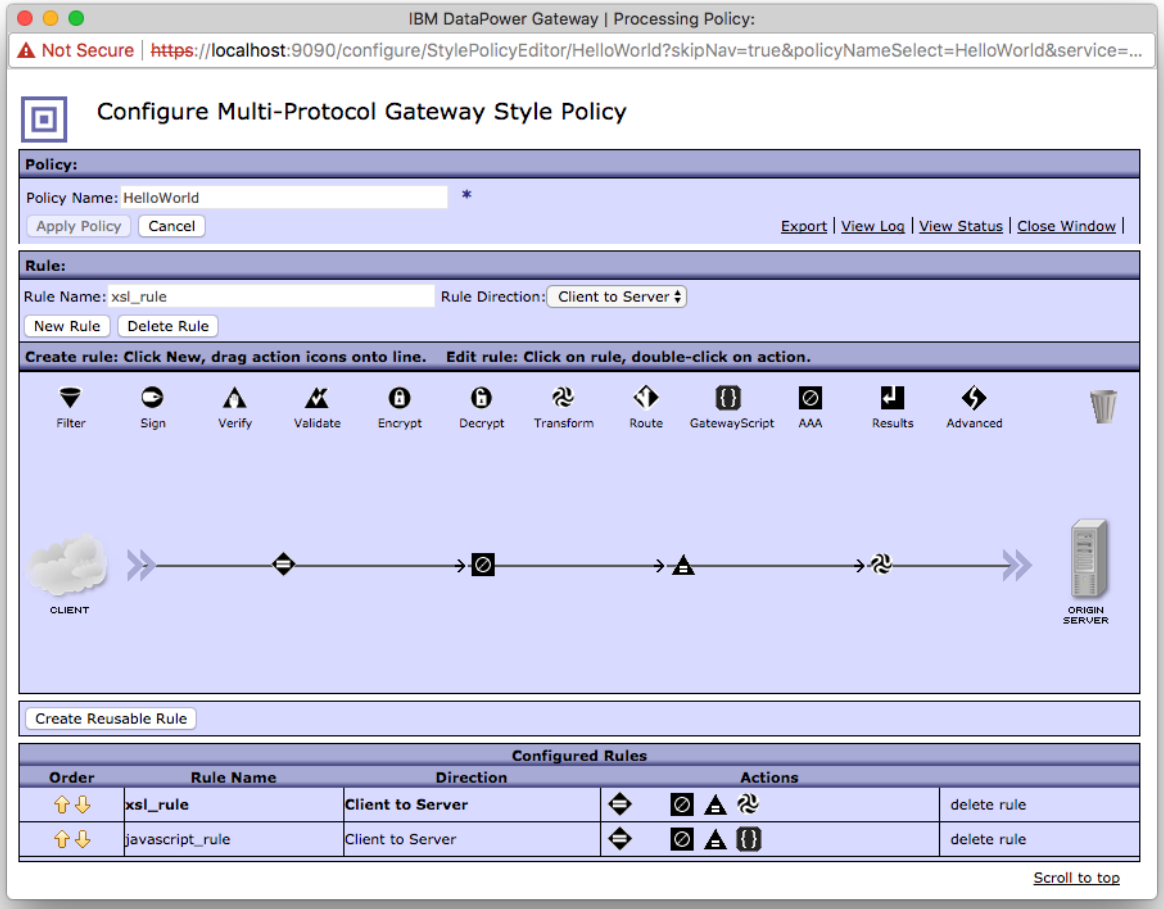
- *Authorization*
  - for 'Method' choose 'Use AAA information file'
    - AAA information file URL:
      - local:///
      - select the same file as already uploaded above

Apply and *Save Configuration*

Adding AAA Control to a MPGW: *HelloWorld*

MPGW: *HelloWorld*

Add to each of the two rules in the MPGW Policy, an AAA action based on the *AAAPolicy* object created above, placing these just after the *Match* action:



Testing

Using a terminal window, you are going to test using the curl command. The **-u** option inserts *Basic Auth* information into the HTTP message.

`curl http://<dp_public_ip>:<mpgw_helloworld_port>/xsl -uanderson:passwArd`
- this one should succeed
- open up the multi-step probe to check how this is processed
- open up the system log after setting the debug-level to debug and rerun

`curl http://<dp_public_ip>:<mpgw_helloworld_port>/xsl -uanderson:badpass`
- this one should fail
- open up the multi-step probe to check how this is halted, looking for the error codes in the service variables after the AAA action
- open up the system log and rerun

`curl http://<dp_public_ip>:<mpgw_helloworld_port>/xsl -uingram:passwIrd`
- this one should fail

`curl http://<dp_public_ip>:<mpgw_helloworld_port>/javascript -uingram:passwIrd`
- this one should succeed

`curl http://<dp_public_ip>:<mpgw_helloworld_port>/xsl -uedwards:passwErd`
- this one should succeed

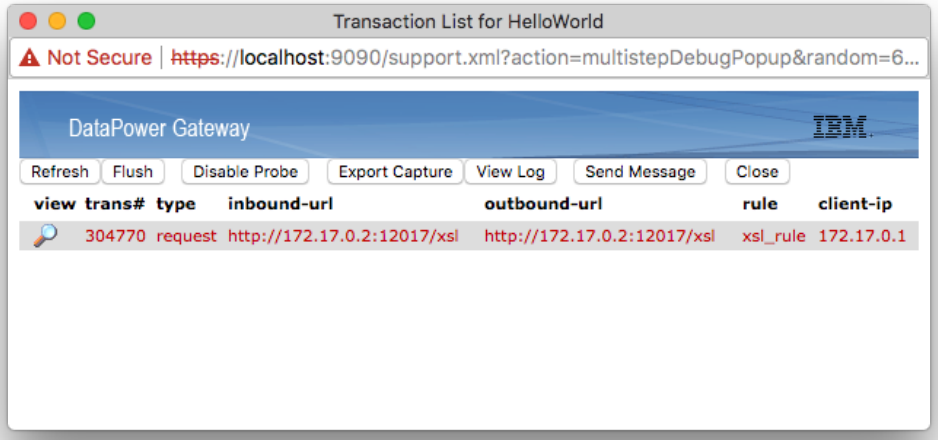
`curl http://<dp_public_ip>:<mpgw_helloworld_port>/javascript -uedwards:passwErd`
- this one should succeed

Sample multi-step probe service variables:

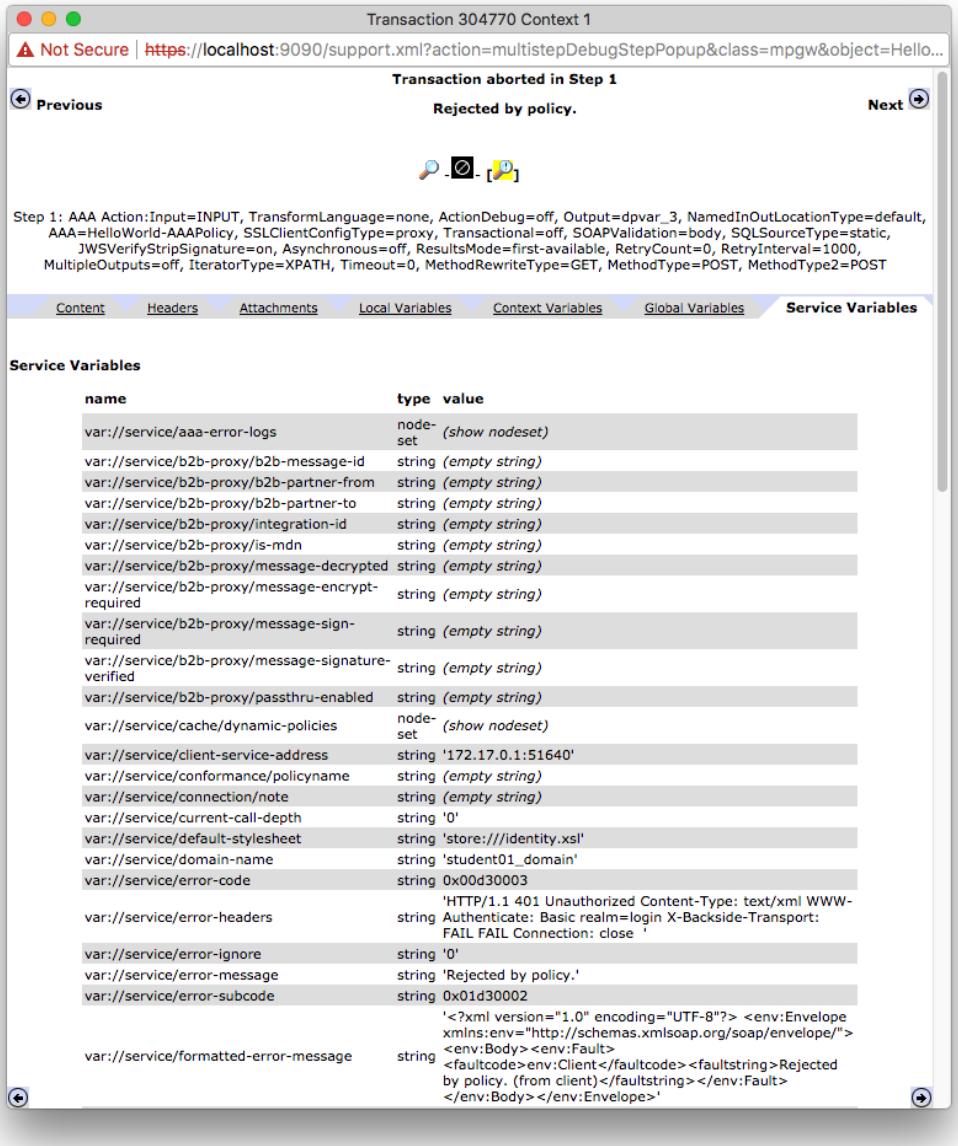
```
var://service/error-code                  0x00d30003
var://service/error-headers               'HTTP/1.1 401 Unauthorized
                                                 Content-Type: text/xml
                                                 WWW-Authenticate: Basic realm=login
                                                 X-Backside-Transport: FAIL FAIL
                                                 Connection: close'
var://service/error-ignore                '0'
var://service/error-message               'Rejected by policy.'
var://service/error-subcode               0x01d30002
var://service/formatted-error-message     string '<env:Envelope ... Rejected by policy. ... </env:Envelope>'
```

Sample multi-step probe windows

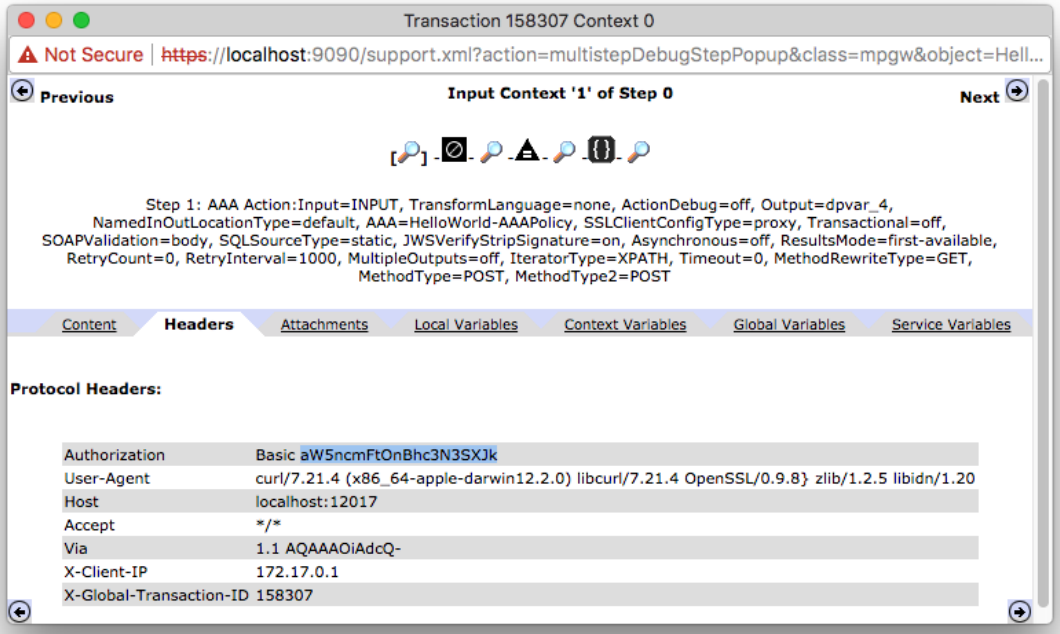
The following message rejected:



Service variables after the rejection:



Incoming message includes Basic Auth information in HTTP Header:



The base64 string **`aW5ncmFtOnBhc3N3SXJk`** decoded is **`ingram:passwIrd`**.

Add an Error Rule

Add an ***error*** rule to the MPGW that uses the `error-handling-log.xsl` file from here.
This will send entries to the logging system.

Add a further Transform action (XSL) that outputs (returns to the client) some **relevant** information back to the client.
This should be of the form of some HTML that includes output from some of the service variables and some relevant feedback indicating authentication failure.

Test that this is working as you would expect:

- ensure the log target is working and that output is going to the log file
- ensure that an appropriate error response is being received by the client

AAA Info File

```
<AAAInfo xmlns="http://www.datapower.com/AAAInfo">
  <FormatVersion>1</FormatVersion>
  <Filename>local:///AAA-HelloWorld.xml</Filename>
  <Summary>A simple example of an AAA file.
```

```
          This approach most likely to be used for development.</Summary>

  <!-- Authentication -->
  <Authenticate>
    <Username>anderson</Username>
    <Password>passwArd</Password>
    <OutputCredential>x-group</OutputCredential>
  </Authenticate>

  <Authenticate>
    <Username>ingram</Username>
    <Password>passwIrd</Password>
    <OutputCredential>j-group</OutputCredential>
  </Authenticate>

  <Authenticate>
    <Username>edwards</Username>
    <Password>passwErd</Password>
    <OutputCredential>a-group</OutputCredential>
  </Authenticate>

  <!-- Authorization -->
  <!-- Members of this group can call the XSL rule -->
  <Authorize>
    <InputCredential>x-group</InputCredential>
    <InputResource>.*xsl</InputResource>
    <Access>allow</Access>
  </Authorize>

  <!-- Members of this group can call the javascript rule -->
  <Authorize>
    <InputCredential>j-group</InputCredential>
    <InputResource>.*javascript</InputResource>
    <Access>allow</Access>
  </Authorize>

  <!-- Members of this group can call all rules -->
  <Authorize>
    <InputCredential>a-group</InputCredential>
    <InputResource>.*</InputResource>
    <Access>allow</Access>
  </Authorize>

</AAAInfo>
```

Review

Review your implementation of the AAA (basic-auth) for the HelloWorld MPGW:
- did you use the same AAA Policy object on both request rules?
- is it good idea to re-use this same object on both rules?

Further Enhancements

A - Multiple Error Rules

Change the error rule you created earlier so that the match is based on the error code that is generated when there is an AAA failure.
Test it to ensure that it still works

Add an extra error rule that matches on all error codes, placing it below the existing one.
This should use the same logging XSL above.
Also add to this a further Transform action that outputs (returns back to the client) some selected service variables and an indication that there was some unknown error.
This can be tested by using in the request a URI that is not in the matches of any of the request rules.

B - Software Modularization - Using Multiple Services

(Save a CHECKPOINT before starting this!)
Anticipating that the security aspects might eventually become more extensive
(AAA/basic auth, SQL injection, ...) build another MPGW, named *Security*, that provides a front-end service that feeds into *HelloWorld* MPGW and move your existing AAA Policy object into a single request rule that matches all URIs (*) in that new service.
Then the existing HelloWorld MPGW just deals with the "business" logic, and the new MPGW takes care of, in a centralised manner, all of the security checking.

Hints:

- the new MPGW front handler takes over the ip/host-alias, and port that the *HelloWorld* MPGW service originally used
- the backend of the new service would go to 127.0.0.1 (perhaps used "localhost")
- the front end of the *HelloWorld* would also use 127.0.0.1 but the same original port