Applying authentication to your web service

Lonneke Dikmans
Vennster
The Netherlands

Keywords:

Java, Web Service, REST, SOAP, Authentication. WS-Security, WebLogic Server, Glassfish

Introduction

Web Services can be secured in different ways. In this paper I look at authentication, applying different types of authentication to SOAP and REST and finally a case where the requirement was to apply different types of authentication to different types of clients.

Authentication

Before we continue, it is important to know what we mean by authentication.

Authentication is the process of verification that an individual or an entity is who it claims to be.

Individual refers to a person and entity refers to a system or an organization. Note that this has nothing to do with what you are allowed to do (authorization), just with verification of **identify**.

The following HTTP authentication mechanisms are available in web applications:

Type	**Explanation**
None	The resource (URL) is publicly available to everyone
Basic	Username and password. The password is sent as base64 encoding
Form	Username and password are collected using a form. Uses base64 encoding
Digest	Username and password, password is encrypted
Client-cert	Uses an X509 certificate (HTTP over SSL)
Mutual Authentication	Both client and server are authenticated using either a username/password or a certificate

Note that you can add SSL to Basic and Form authentication, making it more secure.

Authentication can be handled on three different locations in your infrastructure:

1) By the application server (configured in the application);
2) By the service bus (configured in the services that are exposed) if you use a service bus;
3) By the web tier (configured in the HTTP server in the DMZ).

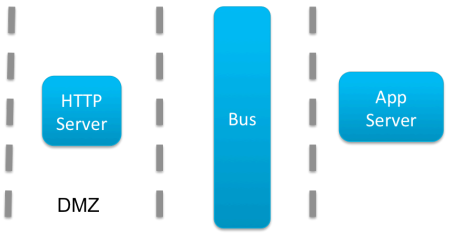

Illustration 1. Different locations where authentication of clients can be applied

Applying authentication to SOAP and REST services

You can apply authentication to SOAP web services both on the transport layer and the message layer. On the transport layer it offers the same types as 'a regular' web application. On the message layer you can apply WS-Security. SAML facilitate single sign on and identity federation between services.

REST services are very similar to regular 'web applications'. This means that all the HTTP authentication mechanisms apply. For single sign and identity federation you can use OpenID and OAuth.

Case: applying different types of authentication for different types of clients

Using certificates is common to authenticate systems. It is not very feasible to authenticate a person with a certificate. There are several (obvious) reasons for this:

1) It requires installation of the certificate. Users often don't have the technical skills to do that or don't have permission to do that on their system.
2) Persons change machines (from home to work, or from laptop to phone). It would require them to install certificates on all machines.

A government agency offered services to other agencies. The service had a number of operations that allowed other agencies to set a subscription on changes (update functionality) or to fetch data from a central registry. After a while it became clear that not all potential customers could use this web service channel. The agency decided to offer a web application for those clients that are not able to consume web services. This would only offer read functionality in a slightly different format (a new service operation).
The web services used mutual authentication based on certificates. For the client this would not work, for reasons stated above. This means that they needed different authentication mechanisms based on the type of client.

Option 1. Write different web applications
The quick and dirty solution would be to write two different web applications that use the same library or service in the back end. The disadvantage of this approach is that you deploy two different applications. If something changes in the library, you have to redeploy both web applications again.

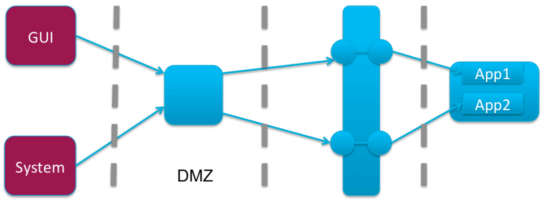

Illustration 2. Writing different applications for different clients

Option 2. Solve it at the application level: accept two types of authentication in the application

In WebLogic you can put a comma-separated list of possible authentication mechanisms. It will try the first and then use the other ones as fallbacks. However, this will be true for all requests, independent of the resource that is being secured. This does not fit the requirement that is should be different depending on the operation (URL) that is called. The listing of how this would look in your web.xml is shown below:

```
<login-config>
   <auth-method>CLIENT-CERT, BASIC</auth-method>
   <realm-name>myrealm</realm-name>
</login-config>
```

Another option is to create an EJB rather than a web application, and secure each endpoint differently:

```
<webservice-endpoint>
   <port-component-name>VIFA_GUI_PORT</port-component-name>
   <login-config>
      <auth-method>BASIC</auth-method>
      <realm>myRealm</realm>
   </login-config>
   <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</webservice-endpoint>
```

This would have worked, but the current solution was a web application, not an EJB.

Apart from defining authentication on transport level, the authentication could also be defined on the message level. You can apply a predefined policy from OWSM for example: `wss11_saml_or_username_token_with_message_protection_service_policy.` This works the same as the multiple log-in configurations, the policy checks whatever token the client uses. Alternatively you could create a custom policy.

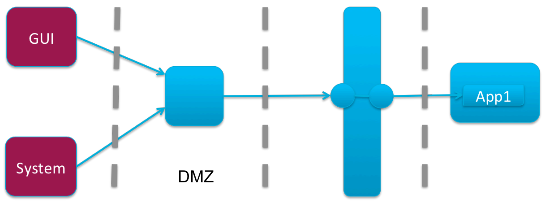

Illustration 3. Using different authentication types in one application

The disadvantage of this approach is that it would change security from transport level to message level security, effectively breaking the existing code.

Option 3. Use the service bus

The service could offer two proxy services, both pointing to the same business service. One proxy would expose the operations for our GUI, using Digest authentication or mutual authentication based on username/password. The other proxy would expose the operations for the systems (setting subscriptions etc.), applying CLIENT-Certification.

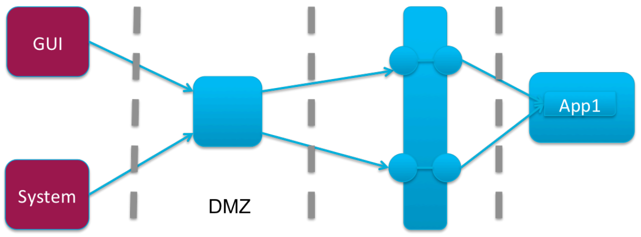

Illustration 4. Using the service bus

The same is true for Mule, this can be achieved by configuring a *Security Endpoint Filter* on an HTTP endpoint.

Option 4. Use the HTTP server
In the current solution authentication was handled by the Web-Tier. SSL was terminated before it reached the service bus. For this reason the agency decided to solve the security in that layer for the GUI as well. The URL for the GUI was different than the system parameters and a different type of authentication was configured for that resource.

Conclusion
For services you can enforce authentication on two levels: On the transport level or on the message level. If you use transport layer security, there are three locations where you can enforce this: at the web tier, the service bus or the application server.

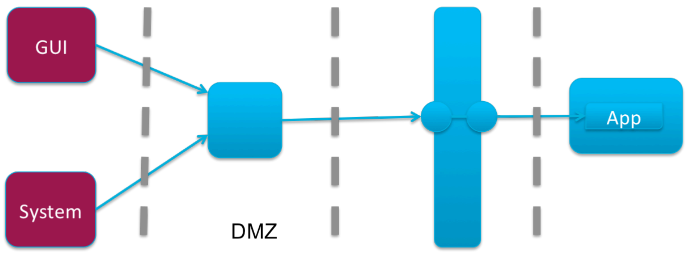

Illustration 5. The final solution, using the HTTP Server

In this case SSL was terminated before it reached the enterprise service bus. The application and service bus were secured based on IP-address. The application server can only be reached from the server that runs the service bus. The server that was running the service bus can only be reached from the web server in the DMZ.

Contact address:

Lonneke Dikmans
Vennster
Postbus 31457
6503 CL, Nijmegen, the Netherlands

Phone:	+31(0)6-15083349
Email	lonneke.dikmans@vennster.nl
Blog:	http://blog.vennster.nl