Security setup for RESTful web services

Syed Ahmed

This article helps you understand commonly used options for securing web services. It also provides instructions on how to perform security configuration settings for a RESTful web service and how to set up authentication against IBM® Bluepages enterprise directory and authorization using IBM Bluegroups.

Introduction

This article begins with a brief definition of web services and an overview of security aspects such as authentication and authorization. It then provides a detailed step-by-step process of how to apply authentication and authorization for a RESTful web service.

Although this article talks specifically about security configuration for RESTful web services, a similar configuration can be implemented for SOAP-based web services as well. You will see in the instructions that the HTTP methods implemented as part of the RESTful web service resources need to be configured in the web resource collection.

Overview of a web service

A web service provides a method of communication between two application systems hosted on the network. They can be hosted over the internet or within two different systems in the enterprise network. The application that publishes the service is called the "Service Provider", and the one that consumes the service is called the "Service Consumer".

Types of web services

There are two commonly used types of web services:

- **JAX-RPC based web services**, in which a Service Provider publishes the service definition using a Web Services Definition Language (WSDL) and the Service Consumer sends a serialized XML message wrapped in a SOAP envelop across the wire.
• **JAX-RS based web services**, in which a Service Provider publishes the Resource name that can be used to consume the service, and the Service Consumer uses stateless operations from the HTTP protocol and sends requests and receives response messages. Because of the stateless nature of the operations, web services are called Representational State Transfer (REST) services. The message payload can either be in XML or JSON format.

**Overview of security**

The main objective of software security is to apply a set of measures to prevent vulnerabilities that can evade privacy of controlled data or misuse of resources. From this objective, we derive two main aspects of security: authentication and authorization.

RESTful web services differentiate themselves from SOAP-based web services mainly in the simplicity of their design and implementation. However, they can become vulnerable when they are unsecured, especially when serving controlled data. This is quite true of any service, not just RESTful web services. In this article, we will look into some of the options to secure them and look closely into one of these options in terms of how to apply it.

**Authentication**

Authentication validates if you are the right person who can login to the software system. For example, if a user, such as "ahmed", is the authenticated user to login to the Linux server "pumbaa".

The following are some options that are commonly used for setting up authentication:

• **Basic authentication**: Basic authentication is one of the easiest ways to implement security as it does not require overhead of additional APIs, besides the APIs used in the implementation framework itself. As the name implies, it is basic in nature where you send user name and password in Base64 encoding with no advanced options. It should always be used with SSL encryption to prevent credentials being decoded.

• **OAuth 1.0a**: OAuth1 is a signature based protocol. It is a widely-used, well-tested, and very secure protocol. The protocol uses a cryptographic signature, such as HMAC-SHA1, which is a value that combines the token secret, nonce, and other request based information. The biggest advantage of OAuth 1 is that it does not require passing the token secret across the wire directly. Because of this, it completely eliminates the possibility of tapping the password from over the wire. This can be safely used without SSL. However, this higher level of security offered in this protocol comes with an overhead of complex signature generating process.

• **OAuth 2**: This protocol completely eliminates signatures; hence, it is much simpler. It requires Transport Layer Security, which handles encryption. OAuth2 is less widely used and less secured as compared to OAuth 1.0a. Therefore, it can be used for less sensitive data.

• **Custom security protocol**: This approach should be avoided not only because no one, besides you, will use your custom protocol. It puts the responsibility of maintenance on the creator. For very secure data, you do not want to leave open holes unless you know what you are doing.
Because of advantages and disadvantages mentioned above, it is safer and simpler to choose Basic Auth with SSL for most REST services. However, high overhead based OAuth 1.0a protocol can be used for extremely sensitive data in less common situations.

**Authorization**

Authorization validates if you are the right person to have access to the resources. For example, whether or not a user, such as "ahmed", is authorized to perform a read or write operation on a file such as `/home/admin/resources` on the Linux server "pumbaa".

Authorization can be set up on the resource managed by an operating system using operating system groups, or on a resource in a web application deployed on a J2EE server (for example, WebSphere® Application Server) using an LDAP user group (for example, a group in an IBM Bluegroups application).

**Development and runtime environment versions**

The screenshots and instructions in this article use the following software:

- **Operating system**: Windows® 7 Professional
- **Eclipse Integrated Development environment**: Rational® Software Architect version 8.0.4.2
- **Application Server runtime environment**: WebSphere Application Server version 7.0.0.29
- **Feature Packs**: Web 2.0 Mobile feature pack is applied on the IDE as well as the server.

**Assumptions, prerequisites, and expectations**

- It is assumed that you are familiar with basic understanding of RESTful web services and intermediate level experience with development environment such as Rational Application Developer/Rational Software Architect for creating and developing an Enterprise Application Archive (EAR) application and have an EAR file available to deploy and test.
- In the following instructions and setup, the name of the EAR file referenced is "IWMAsAServiceEAR" and the web module name is "IWMAsAServiceAppWeb". The role that is configured is named as "IWMaaSRole" and the bluegroup name used is "IWM Services users". The authentication is configured using "bluepages.ibm.com" as the name of the Enterprise Directory Server.
- Basic authentication and authorization are used in the security setup instructions in this article.
- Screen captures are taken using the IDE Rational Software Architect and admin console of the unit test environment. Therefore, the screen captures and instructions may not perfectly match with the standalone server configuration pages or pages on the clustered servers' admin console. Refer to the [WebSphere Application Server Information Center](https://www.ibm.com/support/knowledgecenter/en/SSAW57_7.0.0/com.ibm.websphere.wlp.de.doc/ae/ae-conceptible戀.jpg) and consult with your WebSphere Application Server admin for any help needed for such environments.

**Security setup instructions in the application**

1. Open `web.xml` of the Web Project and select the application name in the design editor and click the **Add** button as shown in Figure 1.
2. Enter security in the search type-ahead and select **Security Role** and click **OK** as shown in Figure 2.

**Figure 2. Select security role**

3. Enter the role name as **IWMaaSRole** as shown in Figure 3 and save.

**Figure 3. Enter role name**

4. Open **web.xml** of the web project (if not open) and select the application name in the Design editor and click the **Add** button as shown in Figure 4.
5. Enter `security` in the search type-ahead and select **Security Constraint** and click **OK** as shown in Figure 5.

**Figure 5. Select the Security Constraint**

6. Enter the display name as `RESTSecurity` and click **Add** and type `IWMaaSRole` as shown in Figure 6.

**Figure 6. Enter Constraint name**

7. Select **Web Resource Collection** and enter the Web Resource Name as `iwmaaS` and add `GET`, `PUT`, `POST`, and all other HTTP methods needed as shown in Figure 7 (method names are in uppercase).
8. Select the security role (**IWMaaSRole**) and click the **Add** button. Highlight **Special Subject** and click **OK** as shown in Figure 8.

**Figure 8. Add Special Subject to the Select Security Role**

9. From the Details section on the right-hand side, choose **All Authenticated In Trusted Realms** from the drop down as shown in Figure 9.

**Figure 9. Choose "All Authenticated In Trusted Realms" as the type for the special subject**
Security setup instructions on the server

1. Double-click the server name under the **Servers** view and set the values (if you do not already have security enabled on your local test environment) as shown in Figure 10.

   **Figure 10. Enable the security on the server**

2. Enable global and application security as shown in Figure 11 (Java 2 security is not enabled below, but it can be enabled if needed – leave it unchecked for now). After selecting **Federated Repository**, click the **Set as current** button.
3. Click the **Configure** button and enter the primary administrative user name (for example, admin) as shown in Figure 12.

**Figure 12. Enter primary administrative user name**
4. Click the **Manage repositories** link to create a repository resource entry as shown in Figure 13.
   
   **Figure 13. Open the "Manage repositories" to create a repository resource**

5. Click the **Add** button as shown in Figure 14.
   
   **Figure 14. Add repository**

6. Fill in the repository identifier, primary host name, and so on in the screen that shows up and click the **OK** button (at the bottom of the screen), as shown in Figure 15.
   
   **Figure 15. Configure repository**

7. Click the Repository ID link for the repository created above as shown in Figure 16.
Figure 16. Open the Repository definition for additional configuration

8. At the bottom of the screen, click the **LDAP entity types** link under additional properties as shown in Figure 17.

   Figure 17. Open the "LDAP entity types" property

9. For each of the entity types, click the links and configure the object classes as shown in Figure 18. Each time, click the **Apply** and **Ok** button and save to the master configuration when it prompts you.

   Figure 18. Configure object classes for the LDAP entity types

10. Using the **Group attribute definition** link, navigate to the path shown in Figure 19 and specify object classes for member and uniqueMember.
11. Using the left navigation (Applications > Application Types > WebSphere enterprise application), go to the list of applications and click the IWMAsAServiceEAR application as shown in Figure 20.

**Figure 20. Open the EAR Application**

12. Click the Security role to user/group mapping link as shown in Figure 21.
13. Configure the mapping of special subjects and groups (if not already updated by application install) as shown in Figure 22. You need to select the check box first and click the drop down Map Special Subjects, select All authenticated in application’s realm, and then once again select the check box and click the Map Groups button.

This concludes the step-by-step instructions on how to configure application security for RESTful web services.

**Conclusion**

In this article, you learned about various aspects of security essentials and the importance of securing a RESTful web service. You also learned how to configure and setup security for a
RESTful web service using an Enterprise Directory, such as Bluepages and Bluegroups for authentication and authorization.
Related topics

- Design and implement RESTful web services with Rational Software Architect
- Constructing REST services with WebSphere Application Server, Part 1: A simple RESTful implementation