Implementing OAuth on IBM WebSphere DataPower Appliances, Part 1: Introducing OAuth 2.0 support in DataPower

Paul Glezen (pglezen@us.ibm.com)  
Consulting IT Specialist  
IBM  

Shiu Fun Poon (shiufun@us.ibm.com)  
Security Architect  
IBM  

John Rasmussen (rasmussj@us.ibm.com)  
Senior Software Developer  
IBM  

May 2015  
(First published 01 August 2012)

This article is the first in a multi-part series of articles describing OAuth support on WebSphere® DataPower Appliances. Part 1 starts with an overview of OAuth and then describes DataPower support for OAuth roles. The rest of the article series will go into detail on several aspects of OAuth and DataPower usage and configuration.

View more content in this series

Introduction

Welcome to the IBM® WebSphere DataPower and OAuth article series. These articles provide an introduction to OAuth and the role that DataPower plays as a policy enforcement point (PEP) and authorization and token endpoint within OAuth architectures. If you are familiar with DataPower, you are aware of the purpose-built nature of these devices and the specialized hardware and firmware leveraged to provide highly optimized security, integration, and transformation features. We will discuss the features supporting OAuth in DataPower, including integration with IBM's Tivoli® Federated Identify Management V6.2.2 or later products.

DataPower firmware version 6.0 continues OAuth support. Some features include:

- Forms based login and added host name redirection to the IP address option
- Public client type for clients incapable of maintaining credential confidentiality
- SSL certificate credential for client authentication
• Refresh token generation with access token
• Revocation support including token cache and custom processes
• Scope support to limit resource access
• XSLT samples for scope customization
• Resource owner pre-authorization of authorization request
• Access token validation URL to validate DataPower-produced access tokens
• Implicit grant

Part 1 presents an introduction to OAuth and to the OAuth roles that DataPower supports. It establishes the foundation for the other parts in the series. As the roadmap in Figure 1 illustrates, Parts 1, 2, and 3 establish the fundamentals for configuring OAuth in DataPower. They should be understood before proceeding to the grant type scenarios. Parts 4, 5, and 6 demonstrate various OAuth grant types. You may read them in sequence or focus on the grant types that most interest you. Part 7 and onward describe advanced integration and customization scenarios.

**Figure 1. OAuth article series roadmap**

The rest of the articles are summarized below:

- **Part 2: Introducing the Web Token Service**: The Web Token Service (WTS) was introduced with OAuth support in mind. It is a new service proxy type, in addition to the common ones such as Multi-Protocol Gateway, Web Service Proxy, and Web Application Firewall oriented toward OAuth authorization server scenarios. This article describes how to create, configure, and debug the Web Token Service (WTS). It also explains typical use cases of the WTS and the advantages of using the WTS over other service types. Finally, it demonstrates manual configuration of a WTS.

- **Part 3: Form-based login policies and the role of AAA**: The heart of an OAuth configuration is AAA (authentication, authorization, and auditing). This article provides an overview of AAA policies and how they have been enhanced to provide OAuth support. The enhancements include OAuth token processing and form-based logins. These elements are then used to demonstrate the WTS creation wizard.

- **Part 4: Grant type scenario – Resource owner password credential**: This scenario-driven article addresses the resource owner password credential grant type (Section 4.3 of the OAuth 2.0 Specification).

- **Part 5: Grant type scenario – Client credentials**: This scenario-driven article addresses the client credential grant type (Section 4.4 of the OAuth 2.0 Specification).
Overview of OAuth

OAuth is an authorization framework that allows a resource owner to grant permission for access to their resources without the sharing of their credentials with a third party. The protocol defines a process that allows limited access to resources hosted by web-based services accessed over HTTP. This is not to be confused with web services and service-oriented architecture (SOA). While SOA architectures exist in a vast array of implementations, OAuth is more focused on the emerging Web 2.0 infrastructure and the popularity of APIs that exist to provide customizable access to an organization's applications. For example, eBay® provides an API to provide enhanced shopping experiences by integrating with third-party applications. Twitter® and Facebook® provide APIs that extend their applications by providing content sharing capabilities. Each of these integrations requires focused attention on all aspects of security and the need to consider all access to be untrusted until proven otherwise.

In established implementations, protected resources are accessed by providing credentials that can be authenticated and used to authorize access to the resource. If a resource owner wishes to grant access to a third-party, credentials must be provided, authenticated, and access authorized by the authorization service that protects the resources.

These established architectures expose several issues, including the inherit weakness of password authentication. Passwords are vulnerable; a “man in the middle,” or perhaps a “woman looking over your shoulder,” could obtain your credentials and gain access to your resources. And most likely, a resource owner has to share his password with the entity that needs access. If a password is changed, the entity no longer has access to the resource. A more fundamental limitation is that authentication and authorization are all under the control of the resource owner, thereby limiting the sharing of authentication information. IT organizations have played an integral part in these security decisions limiting flexibility. Who you are is one component of access control that can be externalized from the authorization process by providing a federation of identity management across organizational boundaries. OAuth eliminates the need for sharing the user's credential with a third party entity.

An OAuth resource owner grants a third party client (or application) access to his or her resources by first authenticating with the authorization service. An OAuth authorization service issues an access token to the third party client (or application). In a typical OAuth use case (see Figure 2),
Sally (resource owner) may provide a printing service (third-party, the client) access to certain pictures (resource) by authenticating to an authentication service and providing limited access (access token) through the authorization process.

Figure 2. OAuth print service access token flow

The current version of OAuth, 2.0, is published in draft form by the Internet Engineering Task Force (IETF). You may wish to review the current [OAuth IETF documents](https://tools.ietf.org/html/draft-ietf-oauth-v2-0) for additional information. Contrary to the reputation IT specifications have for difficulty, this specification is actually quite readable. Familiarity with the specification is encouraged since an OAuth solution usually involves participation among different groups that may not frequently interact. Establishing a consistent set of terms and clarifying the required scenario is important.

OAuth terms

This section describes the roles involved in an OAuth scenario. We have mentioned them in our introduction, but let's review them once more:

- **Resource owner**: An entity (possibly a person or end user) that is capable of granting access to a protected resource. Often this entity is a person interacting via a web browser. For those new to OAuth, it is common to confuse this with the **client** role (see below).

- **Resource server**: A process (typically an HTTP server) that is capable of accepting and responding to resource requests and resolving access tokens.

- **Authorization server**: A process (typically a Secure Token Service or STS) that is capable of authenticating a resource owner and issuing access tokens. The authorization service performs multiple functions and, therefore, exposes multiple service endpoints. There are authorization endpoints and token endpoints.

- **Client**: An entity making a request for a resource upon approval of the resource owner. Since the client is making the request on behalf of the resource owner, it's easy to confuse the two. The resource owner is a "client" of the OAuth client. Moreover, the resource owner is the role that typically uses a web browser. But in OAuth terminology, the **client** is the client of the resource server.
It is important to distinguish between the "OAuth client" and the "OAuth resource owner".

**Protocol flow**

Now let's review the traditional OAuth authorization code grant type protocol flow shown in Figure 3.

**Figure 3. Traditional OAuth authorization code grant type protocol flow**

1. A resource owner (usually a person with a browser) issues a request to the OAuth client that requires access to one of the resource owner's resources on the resource server.
2. The client routes the resource owner (via an HTTP redirect) to the authorization server. This redirect contains the client ID and the scope of the requested grant.
3. The authorization server authenticates the resource owner. This process involves no interaction with the client. The client never sees the resource owner credentials.
4. Upon successful authentication of the resource owner and permission from the resource owner to allow the client access to the requested scope, the authorization server returns an authorization code to the client. The return path is an HTTP redirect through the resource owner's browser.
5. When the client is ready to access a resource, it requests an access token directly from the authorization server (no redirects through the resource owner). The request contains the authorization code that is only valid for the client. It also contains the client secret that identifies the client. Upon validation by the authorization server, it returns an access token that will be accepted by the resource server for a limited time.
6. The client accesses the resource server with the access token. Assuming the access token is still valid and the requested resource is within the scope granted earlier, the resource is served by the resource server to the client.

There are a few things to keep in mind about this scenario.

- All six steps could proceed in straight succession. However, there are also cases where the first four occur (outlined in the peach-colored box) as one exchange. Then Steps 5 and 6 (outlined in green) occur later or as needed.
• The exchange in the peach-colored box does not involve the client's secret.
• The exchange in the green-colored box does not involve the resource owner's credentials.
• The client identity and secret are pre-arranged between the client and authorization server.

**Client types**

We've discussed the resource owner's "granting" of access rights to its protected resources. The type of grant can vary, depending on the level of trust between the client and the resource owner, the client's relationship to the protected resource, and also by the type of implementation. OAuth has defined two general client types, depending on their ability to maintain the confidentiality of their credentials:

- **Confidential**: For clients who are capable of maintaining confidentiality of their credentials.
- **Public**: For those clients who are not capable of maintaining confidentiality of their credentials.

Each client must register with the authorization server prior to engaging in an OAuth protocol exchange. Successful registration results in the establishment of a "client identifier" and other metadata on the authentication server used in subsequent interactions.

**Grant types**

Section 4 of the OAuth 2.0 Specification defines four grant types. They are listed below next to their specification section number:

- 4.1 - Authorization Code
- 4.2 - Implicit
- 4.3 - Resource Owner Password Credentials
- 4.4 - Client Credentials

The authorization code grant type was described in detail above.

Implicit grants are available as a simplified method of access token generation for browser-based clients. Rather than generating the intermediate grant, the access token is issued directly to the client after authenticating the resource owner. While this improves efficiency, it is a less secure method than intermediate authorization code grants.

The resource owner password credential grant may be used by a trusted client directly to obtain an access token. There must be a high level of trust between the client and the resource owner.

The client credential grant type is used when the client is the owner of the resource. In this case, the client receives an access token upon providing its client credentials only. There is no separate resource owner authentication.

**Access tokens**

An access token provides access rights to a resource as well as constraints, such as scope and duration. This token is typically an opaque string, which may take the form of a pointer to retrieve
the actual access token or provide self-contained information. Access tokens may take different forms when used to access protected resources.

- A **bearer token** is opaque data. This requires secured transmission protocols, such as HTTPS, to protect the token.
- A **MAC token** uses a supplied MAC key to sign sections of the request for integrity verification.
- **Refresh tokens** may be issued by the authentication server at access token generation. They may be used by the client to obtain additional access tokens when existing access tokens have expired, or in order to obtain a more limited set of permissions than originally defined.

The internal format of the access token is **not** addressed by the OAuth 2.0 specification. The resource server enforcement point must understand the tokens generated by the authorization server.

**Two-legged vs. three-legged flows**

The scenarios we have describe so far have dealt with the typical OAuth architecture where a resource owner provides a client access to their resources. This access is authorized by the authorization service and provided by the resource server. In this "three-legged flow", the resource owner does not provide their credentials to the client and, therefore, the client requires an authorization code grant as was demonstrated in Figure 2.

However, the specification also provides for a scenario where the client may be the resource owner. Or, the client can be a trusted entity that has been provided the resource owner’s credentials. In these **two-legged** use cases, the roles of the client and resource owner collapse. It requires a resource owner password credential or a client credential grant type.

**DataPower's support for OAuth**

This section introduces capabilities provided initially by the 5.0 version and further enhanced by the 6.0 version of the DataPower firmware. Detailed discussions and specific uses cases will be demonstrated later in this article series.

DataPower can act as an OAuth authorization service responding to authorization endpoint and token endpoint requests. DataPower can act as an Enforcement Point (EP) for a resource server receiving OAuth 2.0 requests. DataPower can integrate with IBM's Tivoli Federated Identity Manager V6.2.2 or greater in this EP role. Interactions with Federated Identity Manager use the WS-Trust protocol for token verification. DataPower supports bearer tokens with confidentiality and integrity support provided by the underlying secured transport (HTTPS).

The following new DataPower objects and configurations are provided to support OAuth interactions:

- **The OAuth Client Profile** is a new configuration object that holds the metadata defined during the client registration process, such as client-id, redirection-URL, scope, and lifetime. Authorization forms, a shared-secret, supported grant types, and roles that the client will use
in subsequent protocol exchanges are defined here as well. The OAuth Client Group is used to create collections of OAuth clients with the same authorization policies.

- The AAA Policy has been enhanced to provide OAuth specific capabilities including:
  - Extract Identity OAuth Method used to obtain an OAuth Client ID and shared secret.
  - Extract Resource Processing Metadata used to obtain the requested OAuth scope.
- The Web Token Service (WTS) is a new service configuration object that can implement an OAuth token endpoint and as an authorization endpoint. A wizard is provided for the creation of WTS policies based on a configured AAA policy.

Figure 4 shows the earlier OAuth protocol exchange of Figure 2, which is updated to use DataPower as both an authorization service and as the enforcement point for the resource server:

- When acting as the authorization service authorization endpoint, DataPower is used to obtain authorization from resource owner and provides authorization for the client.
- As the token endpoint, DataPower provides access tokens.
- When acting as the enforcement point to protect the resource service, DataPower validates the access tokens used to access protected resources.

**Figure 4. DataPower as an authorization service and enforcement point for a resource server**

Figure 5 shows the integration of Tivoli Federated Identity Manager for authorization services. This integration leverages Tivoli’s enterprise level federation capabilities.
Conclusion

This article introduced fundamental concepts of OAuth and OAuth support provided by DataPower. The DataPower appliance products comprise a time-tested, highly secured, and optimized security, integration, and transformation platform. The features described allow for the implementation and integration of OAuth protocol exchanges.

For additional information, see the rest of the article series as follows:

- Part 2: Introducing the Web Token Service
- Part 3: Form-based login policies and the role of AAA
- Part 4: Grant type scenario for resource owner password credentials
- Part 5: Grant type scenario for Client credentials
- Part 6: Grant type scenario for authorization codes
- Part 7: Integration – IBM Federated Identity Manager
- Part 8: Grant type scenario for implicit grants
- Part 9: Customization scenario for scope and identity support
- Part 10: Customization scenario for additional extension points

Acknowledgments

Many individuals contributed to the implementation of OAuth in DataPower. The authors of this article series wish to thank members of the IBM Software Services for WebSphere organization for their contributions, including: Martin Lansche, Ken Hygh, Cindy Claudant, Nicholas A. Glowacki, Tamika Moody, and Timothy Baker.
Resources

- IBM WebSphere DataPower SOA Appliance Handbook
- IETF Web Authorization Protocol document (OAuth)
- WebSphere DataPower Knowledge Center
- WebSphere DataPower library
- developerWorks WebSphere zone
- WebSphere DataPower discussion forum
About the authors

Paul Glezen

Paul Glezen is an IT Specialist with IBM Software Services for WebSphere (ISSW). He started application server consulting in 1998 with the IBM CORBA-based C++/Java product known as Component Broker. His current product focuses are DataPower and WebSphere Application Server development and security configurations.

Shiu Fun Poon

Shiu Fun Poon is the Security Architect in the WebSphere DataPower Appliance development team. She has worked in many different areas and features of DataPower appliances. She currently leads DataPower's support for OAuth 2.0 functionality. Her past DataPower projects included WS-Security Policy, various WS-Security features, appliance RAS features, 7198/7199 appliance release, and third party integration with the appliance.

John Rasmussen

John Rasmussen is a Senior Software Engineer with IBM’s Software Group. He has worked with DataPower Corporation and IBM since 2002 as a product development engineer and services specialist, assisting many clients in the implementation of DataPower appliances. John has experience in software development and security, including work with McCormack & Dodge and Fidelity Investments, and as an independent developer of application software and security systems.

Trademarks (www.ibm.com/developerworks/ibm/trademarks/)