A Performance Optimization Model towards OAuth 2.0 Adoption in the Enterprise

M. Noureddine[1], R. Bashroush[2]

[1] Microsoft Corporation, Seattle, USA
Moustafa.Noureddine@Microsoft.com
[2] University of East London, London, UK
rabih@uel.ac.uk

Abstract. As Cloud software (Software-as-a-Service) become more and more ubiquitous, the scale and performance expectations become an important factor impacting architectural decisions for security protocol adoption. WS-Trust[6] and WS-Federation[7] are enterprise scale protocols but lacked wide adoption due to complexity. OAuth 1.0 emerged as an industry standard for unifying identity management for major SaaS players. However, OAuth 1.0 soon was proven to fail performance criteria for enterprise adoption. With the introduction of OAuth 2.0 some of the performance concerns were addressed. This paper proposes an optimization to OAuth 2.0 for enterprise adoption. This optimization is achieved by introducing manageability steps to pre-establish trust amongst the client and the protected resource server. In this model, the client needs to set up trust with the protected resource server as well as with the authorization server. These clients are called highly trusted clients. We believe such optimization makes it feasible to adopt OAuth in the enterprise where scale and performance are critical factors.

Keywords: OAuth; Access Delegation; Authorization Servers.

1 Introduction

OAuth is a claim-based security protocol that enables users to grant third-party access to their protected resources without sharing their passwords. OAuth 1.0 [1] was published in December 2007 and quickly become the industry standard for web-based access delegation. However, OAuth 1.0 faced lots of challenges to make it into the enterprise domain mainly due to the lack of performance optimization capabilities currently on offer by the protocol. Microsoft, Google, and other large organizations [3] proposed OAuth WRAP (Web Resource Authorization Profiles) to solve the performance challenges and facilitate adoption by the enterprise. One of the main optimizations is the introduction of an independent Authorization Server. OAuth adopted the WRAP recommendation into OAuth 2.0. In this work, we introduce an additional optimization where the Authorization Server is configured with explicit authorization table so that access grants are rejected at the Authorization Server

before getting to the protected resource server. This reduces the amount of processing some popular protected resources servers would have to do and alleviates the risk of potential threats such as Denial-of-Service (DoS) attacks and Distributed DoS (DDoS).

In the next section, we discuss the drivers behind the introduction of OAuth2.0 and present its architecture. In section 3, we argue the modifications suggested to OAuth2.0 in order to facilitate Enterprise adoption of the protocol. Section 4 then provides preliminary results from experiments currently being conducted using the modified version of OAuth2.0. Finally, section 5 rounds off the paper by providing a preview of planned future work.

2 Introduction to OAuth 2.0

Although OAuth 2.0 is a new protocol, it still retains the overall architecture and approach established by the previous versions. As large providers started using OAuth 1.0, the community realized that the protocol does not scale well. It required: state management across different steps; temporary credentials management; and provided no isolation of the Authorization server from the protected resource itself. In addition, OAuth 1.0 required that the protected resources' endpoints have access to the client credentials in order to validate the request. This broke the typical architecture of most large providers in which a centralized authorization server is used for issuing credentials, and a separate server is used for API calls. OAuth 1.0 required the use of both sets of credentials: the client credentials and the token credentials, which made the separation very hard [2].

As the deployment of Cloud hosted enterprise software evolves (such as Exchange Online and SharePoint Online), there is a growing trend for a variety of applications to access resources through an API over HTTP or other protocols. Often these resources require authorization for access to such Protected Resources. The systems that are trusted to make authorization decisions may be independent from the Protected Resources for scalability and security reasons. The OAuth Web Resource Authorization Profiles (OAuth WRAP) enable a Protected Resource to delegate the authorization to access a Protected Resource to one or more trusted authorities. Clients that wish to access a Protected Resource first obtain authorization from a trusted authority (Authorization Server). Different credentials and profiles can be used to obtain this authorization, but once authorized the Client is provided with an Access Token, and possibly a Refresh Token to obtain new Access Tokens. The Authorization Server typically includes authorization information in the Access Token and digitally signs the Access Token. The Protected Resource can verify that an Access Token received from a Client was issued by a trusted Authorization Server and is valid. The Protected Resource can then examine the contents of the Access Token to determine the authorization that has been granted to the Client.
The following figure1below shows the architecture for OAuth 2.0 with an independent Authorization Server.

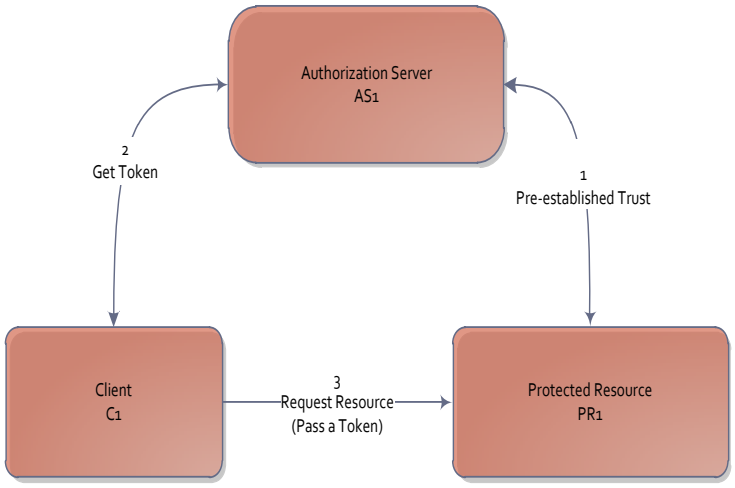

Fig. 1. OAuth 2.0 Architecture

3 Enterprise Integration

It is often required for servers to integrate with each other and exchange protected data. An example of this is the integration with the Microsoft Exchange Server. A third party may want to develop an application to access its users' mail boxes (for archiving or other scenarios that can be monetized). Since the Exchange mailbox is a highly protected resource with high business impact, it may not want to hand its mailbox data to any application with a valid token. In a non-enterprise environment, all you need is a paid account to have access. For example, Amazon may allow access to its listings for anyone who is willing to pay an integration price, while Microsoft Exchange Server wants to consider Tokens for applications that have pre-established trust. Also since Exchange server can host millions of users in the Cloud in a Shared Tenancy [5] model, request for access with valid tokens can easily burden the server.

In our proposal, shown in Figure 2 below, we are working on adding a pre-established trust between the Client (C1) and the Protected Resource (PR1) which can reduce many of the unwanted requests to the Protected Resource (PR1) shown in figure 1 above.

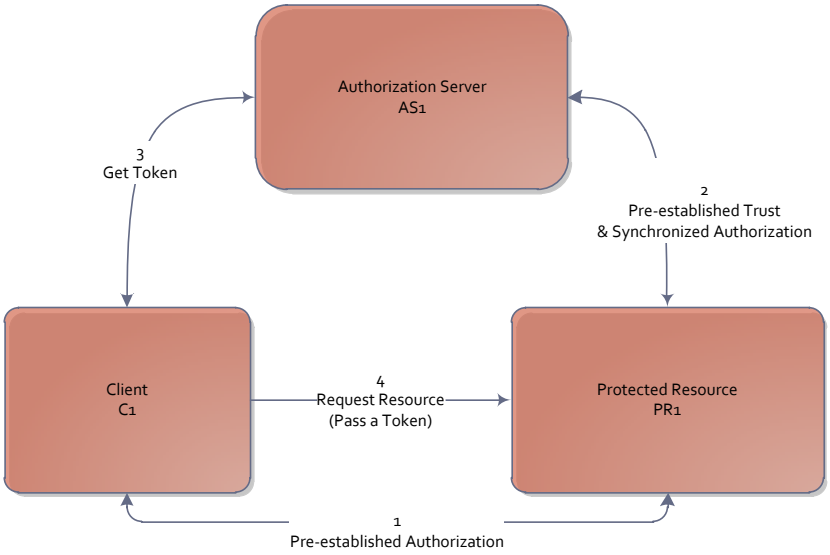

Figure 2. OAuth 2.0 Modified Architecture

In order to do this, we built an Authorization Server and set up an Authorization Table as shown in Table 1 below. This table is replicated on the Authorization Server as well the Protected Resource where Authorization Server can only issue tokens to C1, and C2 and the Protected Resource can only accept tokens issued by C1, and C2.

Table 1. Authorization Table

Issuer	AppliesTo
C1	https://PR1.com
C2	https://PR2.com
C3	https://PR3.com

During trust establishment (Step 1 in Figure 1 above), the Protected Resource (PR1) sets this table. In return, the Authorization Server will only issue tokens to Issuers in the table. If, for example, C3 comes with a request, it will not be granted a token since it does not have an entry in the table, in other words, C3 needs to be provisioned to be trusted by the protected resource PR1 in order for the Authorization Server to issue tokens.

A second optimization we have designed is the introduction of additional parameters to the Token itself. In this optimization, we added the AppliesTo parameter. When C1 requests a Token from AS1, it will be receiving a token with

AppliesTo parameter addressed to PR1 so that C1 cannot play the Token to any other protected resource. This provides an additional layer of security by ensuring that AS1 is only issuing tokens with pre-established handshake between the client and the protected resource and reduces the number of unwanted attempts to authenticate. For example PR1 can build an interface to allow only clients request with pre-established trust and thus rejecting many of the unwanted claims without putting additional load on the protected resource. In the example above, the token will be addressed to `Https://PR1` and cannot be played to any other server. This ensures that the token was intended for this protected resource. All tokens without the matching AppliesTo parameter will be rejected by the protected resource server PR1. If performance optimization is a high priority, highly trusted application may be allowed to use a wild card in the AppliesTo parameter. In this case, the client can reuse the token to play the token to other protected resources in the enterprise. For example, if a client is accessing SharePoint server, Exchange Server, and SAP server within the same enterprise, it may need to issue a single token from the authorization server with 'AppliesTo = *', this reduces the round trips the client needs to make to the authorization server.

3 Conclusion and Future Work

As a consumer centric authentication protocol, OAuth is light-weight, secure, and simple identity management protocol. With some optimization, it can become ubiquitous model for enterprise adoption. In this paper we have shown an optimization that can significantly reduce unwanted authentication claims and potentially can prevent a DoS type of threat. To better leverage OAuth 2.0 in the enterprise, we proposed two optimizations, one by requiring pre-established authorization table between the client and the protected resource, and the other one by allowing highly trusted clients to play tokens to more than one protected resource within a single enterprise and thus reducing the round trips a client needs to make to the authorization server.

In future work, we plan to show how the two modifications suggested were proven useful using a case study. In order to do so, we want to simulate Exchange Server accepting tokens for clients requesting access to their mailboxes. The clients will have to pre-register trust with Exchange Server. Additional optimization we plan to introduce include caching tokens and reusing them on behalf of other users trusted by the client within the valid lifetime of a token.

References

1. OAuth 1.0, http://oauth.net/core/1.0/#anchor1
2. OAuth, http://tools.ietf.org/html/draft-hardt-oauth-01#page-14
3. OAuth WRAP, http://tools.ietf.org/html/draft-hardt-oauth-01

4. Wang Bin, Huang He Yuan, Liu Xiao Xi, Xy Jing Min: Open Identity Management Framework for SaaS Ecosystem. In: ICEBE 09 Proceedings of the 2009 IEEE International Confernece on e-Businness Engineering.
5. Ye Hu, Johnny Wong, Gabriel Iszlai and Marin Litoiu: Resource Provisioning for Cloud Computing. In: Proceedings of CASCON 2009, November 2009.
6. WS-Trust, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html
7. WS-Federation, http://msdn.microsoft.com/enus/library/bb498017.aspx