OAuth Working Group	B. Campbell
Internet-Draft	Ping
Intended status: Standards Track	J. Bradley
Expires: February 12, 2015	Independent
	August 11, 2014

OAuth 2.0 Token Exchange: an STS for the REST of us

draft-campbell-oauth-sts-01

Abstract

An OAuth 2.0 framework for exchanging security tokens enabling authorization servers to act as lightweight HTTP and JSON based security token services.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

A security token service (STS) is a service capable of validating and issuing security tokens, which enables web service clients to obtain appropriate temporary access credentials for resources in heterogeneous environments or across security domains. Clients have historically used WS-Trust [WS-Trust] as the protocol to interact with an STS for token exchange. However WS-Trust is a fairly heavyweight framework which uses XML, SOAP, WS-Security, XML-Signatures, etc. while the trend in modern web development has been towards more lightweight services utilizing RESTful patterns and JSON. The OAuth 2.0 Authorization Framework [RFC6749] and OAuth 2.0 Bearer Tokens [RFC6750] have emerged as popular standards for authorizing and securing access to HTTP and RESTful resources but do not provide all that is needed to support generic STS interactions.

This specification defines a lightweight protocol extending OAuth 2.0 that enables clients to request and obtain security tokens from authorization servers acting in the role of an STS. There is support for enabling one party to act on behalf of another as well as enabling one party to delegate constrained authority to another. Similar to OAuth 2.0, this specification focuses on client developer simplicity and requires only an HTTP client and JSON parser, which are nearly universally available in modern development environments. The STS protocol defined in this specification is not itself RESTful (an STS doesn't lend itself particularly well to a REST approach) but does utilize communication patterns and data formats that should be more palatable to developers accustom to working with RESTful systems.

A new security token request grant type and the associated specific parameters for a security token request to the token endpoint are defined by this specification. A security token response is a normal OAuth 2.0 response from the token endpoint with some additional parameters defined herein to provide information to the client.

The security tokens obtained from an STS could be used in a variety of contexts, the specifics of which are beyond the scope of this document.

The scope of this specification is limited to the definition of a framework and basic wire protocol for an STS style token exchange utilizing OAuth 2.0. The syntax, semantics and security characteristics of the tokens themselves (both those presented to the AS and those obtained by the client) are explicitly out of scope and no requirements are placed on the trust model in which an implementation might be deployed. Additional profiles may provide more detailed requirements around the specific nature of the parties and trust involved,

whether signatures and/or encryption of tokens is required, etc., however, such details will often be policy decisions made with respect to the specific needs of individual deployments and will be configured or implemented accordingly.

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

This specification uses the terms "authorization server" "token endpoint", "access token request", "access token response", and "client" defined by OAuth 2.0 [RFC6749].

1.3. Delegation vs. Impersonation Semantics

When principal A impersonates principal B, A is given all the rights that B has within some defined rights context. Whereas, with delegation semantics, principal A still has its own identity separate from B and it is explicitly understood that while B may have delegated its rights to A, any actions taken are being taken by A and not B. In a sense, A is an agent for B.

Delegation semantics are therefore different than impersonation semantics, with which it is sometimes confused. When principal A impersonates principal B, then in so far as any entity receiving such a token is concerned, they are actually dealing with B. It is true that some members of the identity system might have awareness that impersonation is going on but it is not a requirement. For all intents and purposes, when A is impersonating B, A is B.

A security token with delegation semantics is requested using this framework by including both an on_behalf_of token and an act_as token in the request. The on_behalf_of token represents the identity of the party on behalf of whom the token is being requested while the act_as token represents the identity of the party to whom the access rights of the returned token are being delegated. In this case, the token returned to the client will contain claims about both parties.

A security token with impersonation semantics is requested using this framework by including an on_behalf_of token in the request and omitting the act_as token. The on_behalf_of token represents the identity of the party on behalf of whom the token is being requested the token returned to the client will contain claims about that party.

2. Security Token Request

A client requests a security token by making a token request to the authorization server's token endpoint using the extension grant type mechanism defined in Section 4.5 of OAuth 2.0 [RFC6749].

Client authentication to the authorization server is done using the normal mechanisms provided by OAuth 2.0. Section 2.3.1 of The OAuth 2.0 Authorization Framework [RFC6749] defines password-based authentication of the client, however, client authentication is extensible and other mechanisms are allowed. For example, [I-D.ietf-oauth-saml2-bearer] and [I-D.ietf-oauth-jwt-bearer] define client authentication using SAML Assertions and JSON Web Tokens respectively. Other mechanisms, such as TLS client authentication, are also possible. The supported methods of client authentication and whether or not to allow unauthenticated or unidentified clients are deployment decisions that are at the discretion of the authorization server.

The client makes a general security token request to the token endpoint with an extension grant type by including the following parameters using the application/x-www-form-urlencoded format with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type

: REQUIRED. The value urn:ietf:params:oauth:grant-type:security-token-request indicates that it is a security token request.

aud

: REQUIRED. Indicates the location of the service or resource where the client intends to use the requested security token. The value MUST be an absolute URI as defined by Section 4.3 of [RFC3986]. The URI MAY include a query component but MUST NOT include a fragment component. When applicable, the value of this parameter also typically informs the audience restrictions on the returned security token.

scope

: OPTIONAL. A list of space-delimited, case-sensitive strings that allow the client to specify the desired scope of requested security token in the context of the service or resource indicated by the aud parameter.

requested_security_token_type

: OPTIONAL. Identifier for the type of the requested security token. For example, a JWT can be requested with the identifier urn:ietf:params:oauth:token-type:jwt, which is defined in JSON Web Token [I-D.ietf-oauth-json-web-token]. If the requested type is unspecified, the returned token type is at the discretion of the authorization server and may be dictated by knowledge of the requirements of the service or resource whose location is indicated by the aud parameter.

on_behalf_of

: REQUIRED. The value of this request parameter is a security token which represents the identity of the party on behalf of whom the request is being made. Typically the subject of this token will be the primary subject of the security token returned in response to this request.

on_behalf_of_token_type

: REQUIRED. An identifier that indicates the type of the security token sent with the on_behalf_of parameter.

act_as

: OPTIONAL. The value of this request parameter is a security token which represents the identity of the party that is authorized to use the requested security token. When this parameter is present, it indicates that the client wants a token that contains claims about two distinct entities: 1) the entity represented by the token in the on_behalf_of parameter as the primary subject and 2) the entity represented by this token as a party who is authorized to act on behalf of that subject.

act_as_token_type

: REQUIRED when the act_as parameter is present in the request but MUST NOT be included otherwise. The value of this parameter is an identifier that indicates the type of the security token sent with the act_as parameter.

3. Security Token Response

The authorization server responds to a security token request with a normal OAuth 2.0 response from the token endpoint as defined in Section 5 of RFC 6749 [RFC6749]. Additional details and explanation are provided in the following subsections.

3.1. Successful Security Token Response

If the request is valid and meets all policy and other criteria of the authorization server, a successful token response is constructed by adding the following parameters to the entity-body of the HTTP response using the "application/json" media type as defined by [RFC4627] and an HTTP 200 status code. The parameters are serialized into a JavaScript Object Notation (JSON) structure by adding each parameter at the top level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

access_token

REQUIRED. The security token issued by the authorization server in response to the security token request. The access_token parameter from Section 5.1 of RFC 6749 [RFC6749] is used here to carry the requested security token, which allows this token exchange framework to use the existing OAuth 2.0 request and response constructs defined for the token endpoint.

security_token_type

REQUIRED. An identifier for the general type of the returned security token. For example, if the security token is a JWT, this value of the security_token_type is urn:ietf:params:oauth:token-type:jwt.

token_type

REQUIRED. A case insensitive value describing the type of the token issued as discussed in Section 7.1 of RFC 6749 [RFC6749]. Note that this value is different from the value of the security_token_type and provides the client with information about how to utilize the token to access protected resources. For example, a value of Bearer as defined in [RFC6750] indicates that the security token is a bearer token and the client can simply present it as is without any additional proof of eligibility beyond the contents of the token itself. A value of PoP, on the other hand, indicates that use of the token will require demonstrating possession of a cryptographic key associated with the security token ([I-D.ietf-oauth-pop-key-distribution] describes the PoP token type).

expires_in

RECOMMENDED. The validity lifetime, in seconds, of the issued security token. For example, the value 3600 denotes that the token will expire in one hour from the time the response was generated.

scope

OPTIONAL, if the scope of the security token is identical to the scope requested by the client; otherwise, REQUIRED.

refresh_token

NOT RECOMMENDED. Refresh tokens will typically not be issued in response to a urn:ietf:params:oauth:grant-type:security-token-request grant type requests.

3.2. Error Response

If either the on_behalf_of or act_as tokens are invalid for any reason, or are unacceptable based on policy, the authorization server MUST construct an error response as defined in Section 5.2 of OAuth 2.0 [RFC6749] The value of the error parameter MUST be the invalid_grant error code. The authorization server MAY include additional information regarding the reasons for the error using the error_description or error_uri parameters.

4. Examples

[[TODO: at least two examples, with and without act_as, showing a request/response exchange and

including some relevant internal details of the tokens involved]]

5. IANA Considerations

[[TODO]] The urn:ietf:params:oauth:grant-type:security-token-request Grant Type is to be registered in the IANA urn:ietf:params:oauth registry established in [RFC6755].

[[TODO]] Other parameters like requested_security_token_type, on_behalf_of, on_behalf_of_token_type, act_as, etc. need to be registered in the appropriate registries. The aud parameter needs to be registered too but that may well get done in [I-D.ietf-oauth-pop-key-distribution] and aud may/does have wider applicability so perhaps deserves it's own little spec?

6. Security Considerations

[[TODO]]

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

7.2. Informative References

[I-D.ietf-oauth-json-web-token] Jones, M., Bradley, J. and N. Sakimura, "JSON Web Token (JWT)", Internet-Draft draft-ietf-oauth-json-web-token-25, July 2014.

[I-D.ietf-oauth-jwt-bearer] Jones, M., Campbell, B. and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", Internet-Draft draft-ietf-oauth-jwt-bearer-10, July 2014.

[I-D.ietf-oauth-pop-key-distribution] Bradley, J., Hunt, P., Jones, M. and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", Internet-Draft draft-ietf-oauth-pop-key-distribution-00, July 2014.

[I-D.ietf-oauth-saml2-bearer] Campbell, B., Mortimore, C. and M. Jones, "SAML 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants", Internet-Draft draft-ietf-oauth-saml2-bearer-21, July 2014.

[I-D.jones-oauth-token-exchange] Jones, M., Nadalin, A. and C. Baker, "OAuth 2.0 Token Exchange", Internet-Draft draft-jones-oauth-token-exchange-01, July 2014.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., Medeiros, B. and C. Mortimore, "OpenID Connect Core 1.0", February 2014.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, October 2012.

[WS-Trust] Nadalin, A., Goodner, M., Gudgin, M., Barbir, A. and H. Granqvist, "WS-Trust 1.4 (incorporating Approved Errata 01)", February 2012.

Appendix A. Open Issues

[Some] decisions to be made with potential associated draft updates:

- Are the constructs for expressing delegation and impersonation the 'right' ones? Do they provide sufficient flexibility while being reasonably understandable and implementable?
- Does it really make sense to use the act_as and on_behalf_of terms? They come with some baggage.
- More guidance on what delegation should look like in the returned token? I.e. refer to azp in [OpenID.Core]? Or something else?
- Do we need to codify if/how the identity of the client end up in returned token? Should it be an AS decision? A special case of delegation/act_as? Something else?
- Does the error response need a way to convey additional information beyond what OAuth 2.0 provides?
- Exactly how the presentation of PoP or other non-bearer tokens works. Should a challenge-response mechanism be considered rather than trying to stuff the whole PoP into a single request?

Appendix B. Acknowledgements

The author wishes to thank Michael Jones for bringing forth the concept of OAuth 2.0 Token Exchange with [I-D.jones-oauth-token-exchange]. This draft borrows heavily from Jones' work while striving to provide a syntax that is more consistent with OAuth 2.0 [RFC6749], which will hopefully be more familiar to developers and easier to understand and implement.

The author also wishes to thank John Bradley for his endless patience and willingness to share his expertise.

Appendix C. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-01

- Add Bradley as an author.

-00

- Gotta start somewhere...

Authors' Addresses

Brian Campbell
Ping Identity
EMail: brian.d.campbell@gmail.com

John Bradley
Independent
EMail: ve7jtb@ve7jtb.com