OAuth Working Group	M. Jones
Internet-Draft	A. Nadalin
Intended status: Standards Track	Microsoft
Expires: September 5, 2016	B. Campbell
	J. Bradley
	Ping Identity
	C. Mortimore
	Salesforce
	March 4, 2016

OAuth 2.0 Token Exchange: An STS for the REST of Us

draft-ietf-oauth-token-exchange-04

Abstract

This specification defines a protocol for a lightweight HTTP- and JSON- based Security Token Service (STS) by defining how to request and obtain security tokens from OAuth 2.0 authorization servers, including security tokens employing impersonation and delegation.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

A security token is a set of information that facilitates the sharing of identity and security information in heterogeneous environments or across security domains. Examples of security tokens include JSON Web Tokens (JWTs) [JWT] and SAML Assertions [OASIS.saml-core-2.0-os]. Security tokens are typically signed to achieve integrity and sometimes also encrypted to achieve confidentiality. Security tokens are also sometimes described as Assertions, such as in [RFC7521].

A Security Token Service (STS) is a service capable of validating and issuing security tokens, which enables clients to obtain appropriate access credentials for resources in heterogeneous environments or across security domains. Web Service clients have used WS-Trust [WS-Trust] as the protocol to interact with an STS for token exchange, however WS-Trust is a fairly heavyweight protocol, which uses XML, SOAP, etc. Whereas, the trend in modern Web development has been towards 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 everything necessary to facilitate token exchange 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. 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 familiar to developers accustomed to working with RESTful systems.

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

The entity that makes the request to exchange tokens is considered the client in the context of the token exchange interaction. However, that does not restrict usage of this profile to traditional OAuth clients. An OAuth resource server, for example, might assume the role of the client during token exchange in order to trade an access token, which it received in a protected resource request, for a new token that is appropriate to include in a call to a backend service. The new token might be an access token that is more narrowly scoped for the downstream service or it could be an entirely different kind of token.

The scope of this specification is limited to the definition of a basic request and response protocol for an STS-style token exchange utilizing OAuth 2.0. Although a few new JWT claims are defined that enable delegation semantics to be expressed, the specific 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, such as whether signing and/or encryption of tokens is required; 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.

The security tokens obtained could be used in a number of contexts, the specifics of which are also beyond the scope of this specification.

1.1. Delegation vs. Impersonation Semantics

When principal A impersonates principal B, A is given all the rights that B has within some defined rights context and is indistinguishable from B in that context. Thus, 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.

Delegation semantics are different than impersonation semantics, though the two are closely related. With delegation semantics, principal A still has its own identity separate from B and it is explicitly understood that while B may have delegated some of its rights to A, any actions taken are being taken by A representing B. In a sense, A is an agent for B.

Delegation and impersonation are not inclusive of all situations. When a principal is acting directly on its own behalf, for example, neither delegation nor impersonation are in play. They are, however, the more common semantics operating for token exchange and, as such, are given more direct treatment in this specification.

Delegation semantics are typically expressed in a token by including information about both the primary subject of the token as well as the actor to whom that subject has delegated some of its rights. Such a token is sometimes referred to as a composite token because it is composed of information about multiple subjects. A client can indicate the desire for a composite token by including a want_composite parameter in the request with the value true. Typically, in the request, the subject_token represents the identity of the party on behalf of whom the token is being requested while the actor_token represents the identity of the party to whom the access rights of the issued token are being delegated. A composite token issued by the authorization server will contain information about both parties.

The specifics of representing a composite token and even whether or not such a token will be issued depend on the details of the implementation and the kind of token. The representations of composite tokens that are not JWTs are beyond the scope of this specification. The Section 4.1 request parameter, however, does provide a means for providing information about the desired actor though the representation of a chain of delegation using the JWT act claim.

1.2. 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].

1.3. Terminology

This specification uses the terms "access token type", "authorization server", "client", "client identifier", "resource server", "token endpoint", "token request", and "token response" defined by OAuth 2.0 [RFC6749], and the terms "Claim" and "JWT Claims Set" defined by JSON Web Token (JWT) [JWT].

2. Token Exchange Request and Response

2.1. 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 possible. For example, [RFC7523] defines client authentication using JSON Web Tokens (JWTs) [JWT]. 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 token exchange 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:token-exchange indicates that a token exchange is being performed.

resource

OPTIONAL. Indicates the physical location of the target service or resource where the client intends to use the requested security token. This enables the authorization server to apply policy as appropriate for the target, such as determining the type and content of the token to be issued or if and how the token is to be encrypted. In many cases, a client will not have knowledge of the logical organization of the systems with which it interacts and will only know the location of the service where it intends to use the token. The resource parameter allows the client to indicate to the authorization server where it intends to use the issued token by providing the location, typically as an https URL, in the token exchange request in the same form that will be used to access that resource. The authorization server will typically have the capability to map from a resource URI value to an appropriate policy. The value of the resource parameter MUST be an absolute URI, as specified by Section 4.3 of [RFC3986], which MAY include a query component and MUST NOT include a fragment component. Multiple resource parameters may be used to indicate that the issued token is intended to be used at the multiple resources listed.

audience

OPTIONAL. The logical name of the target service where the client intends to use the requested security token. This serves a purpose similar to the resource parameter, but with the client providing a logical name rather than a physical location. Interpretation of the name requires that the value be something that both the client and the authorization server understand. An OAuth client identifier, a SAML entity identifier [OASIS.saml-core-2.0-os], an OpenID Connect Issuer Identifier [OpenID.Core], or a URI are examples of things that might be used as audience parameter values. Multiple audience parameters may be used to indicate that the issued token is intended to be used at the multiple audiences listed. The audience and resource parameters may be used together to indicate multiple target services with a mix of logical names and physical locations.

scope

OPTIONAL. A list of space-delimited, case-sensitive strings that allow the client to specify the desired scope of the requested security token in the context of the service or resource where the token will be used.

requested_token_type

OPTIONAL. An identifier, as described in Section 3, 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. If the requested type is unspecified, the issued token type is at the discretion of the authorization server and may be dictated by knowledge of the requirements of the service or resource indicated by the resource or audience parameter.

subject_token

REQUIRED. A security token that represents the identity of the party on behalf of whom the request is being made. Typically, the subject of this token will be the subject of the security token issued in response to this request.

subject_token_type

REQUIRED. An identifier, as described in Section 3, that indicates the type of the security token in the subject_token parameter. For example, a value of urn:ietf:params:oauth:token-type:jwt, would indicate that the token is a JWT and a value of urn:ietf:params:oauth:token-type:access_token would indicate that the token is an OAuth access token.

actor_token

OPTIONAL. A security token that represents the identity of the party that is authorized to use the requested security token and act on behalf of the subject.

actor_token_type

An identifier, as described in Section 3, that indicates the type of the security token in the actor_token parameter. This is REQUIRED when the actor_token parameter is present in the request but MUST NOT be included otherwise.

want_composite

OPTIONAL. When the value of this parameter is true, it indicates the client's desire for a composite security token to be issued, which contains claims about both the main subject of the token as well as about the party who is authorized to act on behalf of that subject. Note that this parameter only provides a means for the client to indicate its preference. The authorization server is not required to honor the stated preference and the nature of the tokens it issues are ultimately at its discretion.

In the absence of one-time-use or other semantics specific to the token type, the act of performing a token exchange has no impact on the validity of the subject token or actor token.

2.2. Response

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

2.2.1. Successful 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 specified by [RFC7159], 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 token exchange request. The access_token parameter from Section 5.1 of [RFC6749] is used here to carry the requested token, which allows this token exchange protocol to use the existing OAuth 2.0 request and response constructs defined for the token endpoint. The identifier access_token is used for historical reasons and the issued token need not be an OAuth access token.

issued_token_type

REQUIRED. An identifier, as described in Section 3, for the representation of the issued security token. For example, a value of urn:ietf:params:oauth:token-type:access_token indicates that the issued token is an access token and a value of urn:ietf:params:oauth:token-type:jwt indicates that it is a JWT.

token_type

REQUIRED. A case-insensitive value specifying the method of using of the access token issued, as specified in Section 7.1 of [RFC6749]. It provides the client with information about how to utilize the access token to access protected resources. For example, a value of Bearer, as specified 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. Note that the meaning of this parameter is different from the meaning of the issued_token_type parameter, which declares the representation of the issued security token; the term "token type" is typically used with this meaning, as it is in all *_token_type parameters in this specification. If the issued token is not an access token or usable as an access token, then the token_type value N_A is used to indicate that an OAuth 2.0 token_type identifier is not applicable in that context.

expires_in

RECOMMENDED. The validity lifetime, in seconds, of the token issued by the authorization server. Oftentimes the client will not have the inclination or capability to inspect the content of the token and this parameter provides a consistent and token type agnostic indication of how long the token can be expected to be valid. For example, the value 1800 denotes that the token will expire in thirty minutes from the time the response was generated.

scope

OPTIONAL, if the scope of the issued 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 urn:ietf:params:oauth:grant-type:token-exchange grant type requests. Typically, a token exchange is done to exchange one temporary credential (the subject_token sent in the request) for a different temporary credential (the issued token) that can be used in some other context. Issuing a refresh token would add an additional credential that likely has a much longer lifetime, which undermines the typical model of swapping temporary credentials. Profiles or deployments of this specification that do issue refresh tokens SHOULD clearly document the conditions and reasons for doing so.

2.2.2. Error Response

If either the subject_token or actor_token are invalid for any reason, or are unacceptable based on policy, the authorization server MUST construct an error response, as specified in Section 5.2 of [RFC6749]. The value of the error parameter MUST be the invalid_request error code. The authorization server MAY include additional information regarding the reasons for the error using the error_description and/or error_uri parameters. Other error codes may also be used, as appropriate.

2.3. Example Token Exchange

The following example demonstrates a hypothetical token exchange in which an OAuth resource server assumes the role of the client during token exchange in order to trade an access token that it received in a request for a token that it will use to call to a backend service (extra line breaks and indentation in the examples are for display purposes only).

The resource server receives the following request containing an OAuth access token in the Authorization request header, as specified in Section 2.1 of [RFC6750].

```
GET /resource HTTP/1.1
Host: frontend.example.com
Authorization: Bearer accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
```

Figure 1: Protected Resource Request

The resource server assumes the role of the client for the token exchange and the access token from the request above is sent to the authorization server using a request as specified in Section 2.1. The value of the subject_token parameter carries the access token and the value of the subject_token_type parameter indicates that it is an OAuth 2.0 access token. The resource server, acting as the client, uses its identifier and secret to authenticate to the authorization server using the HTTP Basic authentication scheme. The resource parameter indicates the location of the backend service, https://backend.example.com/api, where the issued token will be used.

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Authorization: Basic cnMwODpsb25nLXNlY3VyZS1yYW5kb20tc2VjcmV0
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fbackend.example.com%2Fapi%20
&subject_token=accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
&subject_token_type=
 urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
```

Figure 2: Token Exchange Request

The authorization server validates the client credentials and the subject_token presented in the token exchange request. From the resource parameter, the authorization server is able to determine the appropriate policy to apply to the request and issues a token suitable for use at https://backend.example.com. The access_token parameter of the response contains the new token, which is itself a bearer OAuth access token that is valid for one minute. The token happens to be a JWT; however, its structure and format are opaque to the client so the issued_token_type indicates only that it is an access token.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"eyJhbGciOiJFUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQiOiJo
   dHRwczovL2JhY2tlbmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2FzLmV
   4YW1wbGUuY29tIiwiZXhwIjoxNDQxOTE3NTkzLCJpYXQiOjE0NDE5MTc1MzMsIm
   F6cCI6InJzMDgiLCJzdWIiOiJiY0BleGFtcGxlLmNvbSIsInNjcCI6WyJhcGkiX
   X0.vHJKtJ-zFIN75Tk7qGlmQsWPIvnChb2uSaGwPLvIWl64ts7-vvfwYDaVoXIQ
   e_HkTVdljIzavVlPT60_b_9pDQ",
 "issued_token_type":
     "urn:ietf:params:oauth:token-type:access_token",
 "token_type":"Bearer",
 "expires_in":60
}
```

Figure 3: Token Exchange Response

The resource server can then use the newly acquired access token in making a request to the backend server.

```
GET /api HTTP/1.1
Host: backend.example.com
Authorization: Bearer eyJhbGciOiJFUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQ
   iOiJodHRwczovL2JhY2tlbmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2
   FzLmV4YW1wbGUuY29tIiwiZXhwIjoxNDQxOTE3NTkzLCJpYXQiOjE0NDE5MTc1M
   zMsImF6cCI6InJzMDgiLCJzdWIiOiJiY0BleGFtcGxlLmNvbSIsInNjcCI6WyJh
   cGkiXX0.vHJKtJ-zFIN75Tk7qGlmQsWPIvnChb2uSaGwPLvIWl64ts7-vvfwYDa
   VoXIQe_HkTVdljIzavVlPT60_b_9pDQ
```

Figure 4: Backend Protected Resource Request

Additional examples can be found in Appendix A.

3. Token Type Identifiers

Several parameters in this specification utilize an identifier as the value to describe the type of token in question. Specifically, they are the requested_token_type, subject_token_type, actor_token_type parameters of the request and the issued_token_type member of the response. Token type identifiers are URIs.

This specification defines the token type identifiers urn:ietf:params:oauth:token-type:access_token and urn:ietf:params:oauth:token-type:refresh_token to indicate that the token is an OAuth 2.0 access token or refresh token, respectively. The value urn:ietf:params:oauth:token-type:jwt defined in Section 9 of [JWT] indicates that the token is a JWT. This specification also defines the token type identifier urn:ietf:params:oauth:token-type:id_token to indicate that the token is an ID Token, as defined in Section 2 of [OpenID.Core]. Other URIs to indicate other token types MAY be used.

4. JSON Web Token Claims

It is useful to have defined mechanisms to express delegation within a token as well as to express authorization to delegate or impersonate. Although the token exchange protocol described herein can be used with any type of token, this section defines claims to express such semantics specifically for JWTs. Similar definitions for other types of tokens are possible but beyond the scope of this specification.

4.1. "act" (Actor) Claim

The act (actor) claim provides a means within a JWT to express that delegation has occurred and identify the acting party to whom authority has been delegated. The act claim value is a JSON object and members in the JSON object are claims that identify the actor. The claims that make up the act claim identify and possibly provide additional information about the actor. For example, the combination of the two claims iss and sub might be necessary to uniquely identify an actor.

However, claims within the act claim pertain only to the identity of the actor and are not relevant to the validity of the containing JWT in the same manner as the top-level claims. Consequently, claims such as exp, nbf, and aud are not meaningful when used within an act claim, and therefore should not be used.

The following example illustrates the act (actor) claim within a JWT Claims Set. The claims of the token itself are about user@example.com while the act claim indicates that admin@example.com is the current actor.

```
{
  "aud":"https://consumer.example.com",
  "iss":"https://issuer.example.com",
  "exp":1443904177,
  "nbf":1443904077,
  "sub":"user@example.com",
  "act":
  {
    "sub":"admin@example.com"
  }
}
```

Figure 5: Actor Claim

A chain of delegation can be expressed by nesting one act claim within another. The outermost act claim represents the current actor while nested act claims represent prior actors. The least recent actor is the most deeply nested.

The following example illustrates nested act (actor) claims within a JWT Claims Set. The claims of the token itself are about user@example.com while the act claim indicates that the system consumer.example.com-web-application is the current actor and admin@example.com was a prior actor. Such a token might come about as the result of the web application receiving a token like the one in the previous example and exchanging it for a new token that lists it as the current actor and that can be used at https://backend.example.com.

```
{
  "aud":"https://backend.example.com",
  "iss":"https://issuer.example.com",
  "exp":1443904100,
  "nbf":1443904000,
  "sub":"user@example.com",
  "act":
  {
    "sub":"consumer.example.com-web-application",
    "iss":"https://issuer.example.net",
    "act":
    {
      "sub":"admin@example.com"
    }
  }
}
```

Figure 6: Nested Actor Claim

4.2. "scp" (Scopes) Claim

The scp claim is an array of strings, each of which represents an OAuth scope granted for the issued security token. Each array entry of the claim value is a scope-token, as defined in Section 3.3 of OAuth 2.0 [RFC6749].

The following example illustrates the scp claim within a JWT Claims Set with four scope-tokens.

```
{
  "aud":"https://consumer.example.com",
  "iss":"https://issuer.example.com",
  "exp":1443904177,
  "nbf":1443904077,
  "sub":"dgaf4mvfs75Fci_FL3heQA",
  "scp":["email","address","profile","phone"]
}
```

Figure 7: Scopes Claim

4.3. "may_act" (May Act For) Claim

The may_act claim makes a statement that one party is authorized to become the actor and act on behalf of another party. The claim value is a JSON object and members in the JSON object are claims that identify the party that is asserted as being eligible to act for the party identified by the JWT containing the claim. The claims that make up the may_act claim identify and possibly provide additional information about the authorized actor. For example, the combination of the two claims iss and sub are sometimes necessary to uniquely identify an authorized actor, while the email claim might be used to provide additional useful information about that party.

However, claims within the may_act claim pertain only to the identity of that party and are not relevant to the validity of the containing JWT in the same manner as top level claims. Consequently, claims such as exp, nbf, and aud are not meaningful when used within a may_act claim, and therefore should not be used.

The following example illustrates the may_act claim within a JWT Claims Set. The claims of the token itself are about user@example.com while the may_act claim indicates that admin@example.com is authorized to act on behalf of user@example.com.

```
{
  "aud":"https://consumer.example.com",
  "iss":"https://issuer.example.com",
  "exp":1443904177,
  "nbf":1443904077,
  "sub":"user@example.com",
  "may_act":
  {
    "sub":"admin@example.com"
  }
}
```

Figure 8: May Act For Claim

5. IANA Considerations

5.1. OAuth URI Registration

This specification registers the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

5.1.1. Registry Contents

- URN: urn:ietf:params:oauth:grant-type:token-exchange
- Common Name: Token exchange grant type for OAuth 2.0
- Change controller: IESG
- Specification Document: Section 2.1 of [[this specification]]

- URN: urn:ietf:params:oauth:token-type:access_token
- Common Name: Token type URI for an OAuth 2.0 access token
- Change controller: IESG
- Specification Document: Section 3 of [[this specification]]

- URN: urn:ietf:params:oauth:token-type:refresh_token
- Common Name: Token Type URI for an OAuth 2.0 refresh token
- Change controller: IESG
- Specification Document: Section 3 of [[this specification]]

- URN: urn:ietf:params:oauth:token-type:id_token
- Common Name: Token Type URI for an ID Token
- Change controller: IESG
- Specification Document: Section 3 of [[this specification]]

5.2. OAuth Parameters Registration

This specification registers the following values in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

5.2.1. Registry Contents

- Parameter name: resource
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: audience
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: requested_token_type
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: subject_token
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: subject_token_type
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: actor_token
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: actor_token_type
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: want_composite
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of [[this specification]]

- Parameter name: issued_token_type
- Parameter usage location: token response
- Change controller: IESG
- Specification document(s): Section 2.2.1 of [[this specification]]

5.3. OAuth Access Token Type Registration

This specification registers the following access token type in the IANA "OAuth Access Token Types" registry [IANA.OAuth.Parameters] established by [RFC6749].

5.3.1. Registry Contents

- Type name: N_A
- Additional Token Endpoint Response Parameters: (none)

- HTTP Authentication Scheme(s): (none)
- Change controller: IESG
- Specification document(s): Section 2.2.1 of [[this specification]]

5.4. JSON Web Token Claims Registration

This specification registers the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims] established by [JWT].

5.4.1. Registry Contents

- Claim Name: act
- Claim Description: Actor
- Change Controller: IESG
- Specification Document(s): Section 4.1 of [[this specification]]

- Claim Name: scp
- Claim Description: Scope Values
- Change Controller: IESG
- Specification Document(s): Section 4.2 of [[this specification]]

- Claim Name: may_act
- Claim Description: May Act For
- Change Controller: IESG
- Specification Document(s): Section 4.3 of [[this specification]]

6. Security Considerations

All of the normal security issues that are discussed in [JWT], especially in relationship to comparing URIs and dealing with unrecognized values, also apply here.

In addition, both delegation and impersonation introduce unique security issues. Any time one principal is delegated the rights of another principal, the potential for abuse is a concern. The use of the scp claim is suggested to mitigate potential for such abuse, as it restricts the contexts in which the delegated rights can be exercised.

7. References

7.1. Normative References

[IANA.JWT.Claims] IANA, "JSON Web Token Claims"

[IANA.OAuth.Parameters] IANA, "OAuth Parameters"

[JWT] Jones, M., Bradley, J. and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.

7.2. Informative References

[OASIS.saml-core-2.0-os] Cantor, S., Kemp, J., Philpott, R. and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012.

[RFC7521] Campbell, B., Mortimore, C., Jones, M. and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015.

[RFC7523] Jones, M., Campbell, B. and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015.

[WS-Trust] Nadalin, A., Goodner, M., Gudgin, M., Barbir, A. and H. Granqvist, "WS-Trust 1.4", February 2012.

Appendix A. Additional Token Exchange Examples

Two example token exchanges are provided in the following sections illustrating impersonation and delegation, respectively (with extra line breaks and indentation for display purposes only).

A.1. Impersonation Token Exchange Example

A.1.1. Token Exchange Request

In the following token exchange request, an anonymous client is requesting a token with impersonation semantics. The client tells the authorization server that it needs a token for use at the target service with the logical name urn:example:cooperation-context.

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&audience=urn%3Aexample%3Acooperation-context
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwc
  zovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI
  uZXhhbXBsZS5uZXQiLCJleHAiOjE0NDE5MTA2MDAsIm5iZiI6MTQ0MTkwOTAwMCwic
  3ViIjoiYmNAZXhhbXBsZS5uZXQiLCJzY3AiOlsib3JkZXJzIiwicHJvZmlsZSIsImh
  pc3RvcnkiXX0.JDe7fZ267iIRXwbFmOugyCt5dmGoy6EeuzNQ3MqDek5cCUlyPhQC6
  cz9laKjK1bnjMQbLJqWix6ZdBI0isjsTA
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
```

Figure 9: Token Exchange Request

A.1.2. Subject Token Claims

The subject_token in the prior request is a JWT and the decoded JWT Claims Set is shown here. The JWT is intended for consumption by the authorization server within a specific time window. The subject of the JWT (bc@example.net) is the party on behalf of whom the new token is being requested.

```
{
  "aud":"https://as.example.com",
  "iss":"https://original-issuer.example.net",
  "exp":1441910600,
  "nbf":1441909000,
  "sub":"bc@example.net",
  "scp":["orders","profile","history"]
}
```

Figure 10: Subject Token Claims

A.1.3. Token Exchange Response

The access_token parameter of the token exchange response shown below contains the new token that the client requested. The other parameters of the response indicate that the token is a bearer access token that expires in an hour.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"eyJhbGciOiJFUzI1NiIsImtpZCI6IjcyIn0.eyJhdWQiOiJ1cm4
   6ZXhhbXBsZTpjb29wZXJhdGlvbi1jb250ZXh0IiwiaXNzIjoiaHR0cHM6Ly9hcy5l
   eGFtcGxlLmNvbSIsImV4cCI6MTQ0MTkxMzYxMCwic3ViIjoiYmNAZXhhbXBsZS5uZ
   XQiLCJzY3AiOlsib3JkZXJzIiwiaGlzdG9yeSIsInByb2ZpbGUiXX0.YQHuLmI1YD
   TugbfEvgGY2gaGBmMyj9BepZSECCBE9j9ogqZv2qx6VQQPrbT1k7vBYGLNMOkkpmm
   JkxZDS0YV7g",
 "issued_token_type":
   "urn:ietf:params:oauth:token-type:access_token",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Figure 11: Token Exchange Response

A.1.4. Issued Token Claims

The decoded JWT Claims Set of the issued token is shown below. The new JWT is issued by the authorization server and intended for consumption by a system entity known by the logical name urn:example:cooperation-context any time before its expiration. The subject (sub) of the JWT is the same as the subject the token used to make the request, which effectively enables the client to impersonate that subject at the system entity known by the logical name of urn:example:cooperation-context by using the token.

```
{
  "aud":"urn:example:cooperation-context",
  "iss":"https://as.example.com",
  "exp":1441913610,
  "sub":"bc@example.net",
  "scp":["orders","history","profile"]
}
```

Figure 12: Issued Token Claims

A.2. Delegation Token Exchange Example

A.2.1. Token Exchange Request

In the following token exchange request, an anonymous client is requesting a token with delegation semantics, which is indicated by the inclusion of the want_composite parameter. The client tells the authorization server that it needs a token for use at the target service with the logical name urn:example:cooperation-context.

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&audience=urn%3Aexample%3Acooperation-context
&want_composite=true
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwc
  zovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI
  uZXhhbXBsZS5uZXQiLCJleHAiOjE0NDE5MTAwNjAsInNjcCI6WyJzdGF0dXMiLCJmZ
  WVkIl0sInN1YiI6InVzZXJAZXhhbXBsZS5uZXQiLCJtYXlfYWN0Ijp7InN1YiI6ImF
  kbWluQGV4YW1wbGUubmV0In19.ut0Ll7wm920VzRvuLGLFoPJLeO5DDElxsax1L_xK
  Um2eooiNSfuif-OGa2382hPyFYnddKIa0wmDhQksW018Rw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
&actor_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczo
  vL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZ
  XhhbXBsZS5uZXQiLCJleHAiOjE0NDE5MTAwNjAsInN1YiI6ImFkbWluQGV4YW1wbGU
  ubmV0In0.7YQ-3zPfhUvzje5oqw8COCvN5uP6NsKik9CVV6cAOf4QKgM-tKfiOwcgZ
  oUuDL2tEs6tqPlcBlMjiSzEjm3yBg
&actor_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
```

Figure 13: Token Exchange Request

A.2.2. Subject Token Claims

The subject_token in the prior request is a JWT and the decoded JWT Claims Set is shown here. The JWT is intended for consumption by the authorization server before a specific expiration time. The subject of the JWT (user@example.net) is the party on behalf of whom the new token is being requested.

```
{
  "aud":"https://as.example.com",
  "iss":"https://original-issuer.example.net",
  "exp":1441910060,
  "scp":["status","feed"],
  "sub":"user@example.net",
  "may_act":
  {
    "sub":"admin@example.net"
  }
}
```

Figure 14: Subject Token Claims

A.2.3. Actor Token Claims

The actor_token in the prior request is a JWT and the decoded JWT Claims Set is shown here. This JWT is also intended for consumption by the authorization server before a specific expiration time. The subject of the JWT (admin@example.net) is the actor that will wield the security token being requested.

```
{
  "aud":"https://as.example.com",
  "iss":"https://original-issuer.example.net",
  "exp":1441910060,
  "sub":"admin@example.net"
}
```

Figure 15: Actor Token Claims

A.2.4. Token Exchange Response

The access_token parameter of the token exchange response shown below contains the new token that the client requested. The other parameters of the response indicate that the token is a JWT that expires in an hour and that the access token type is not applicable since the issued token is not an access token.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"eyJhbGciOiJFUzI1NiIsImtpZCI6IjcyIn0.eyJhdWQiOiJ1cm4
   6ZXhhbXBsZTpjb29wZXJhdGlvbi1jb250ZXh0IiwiaXNzIjoiaHR0cHM6Ly9hcy5l
   eGFtcGxlLmNvbSIsImV4cCI6MTQ0MTkxMzYxMCwic2NwIjpbInN0YXR1cyIsImZlZ
   WQiXSwic3ViIjoidXNlckBleGFtcGxlLm5ldCIsImFjdCI6eyJzdWIiOiJhZG1pbk
   BleGFtcGxlLm5ldCJ9fQ._qjM7Ij_HcrC78omT4jiZTFJOuzsAj1wPo31ymQS-Suq
   r64S1jCp6pfQR-in_OOAosAGamEg4jyPsht6kMAiYA",
 "issued_token_type":"urn:ietf:params:oauth:token-type:jwt",
 "token_type":"N_A",
 "expires_in":3600
}
```

Figure 16: Token Exchange Response

A.2.5. Issued Token Claims

The decoded JWT Claims Set of the issued token is shown below. The new JWT is issued by the authorization server and intended for consumption by a system entity known by the logical name urn:example:cooperation-context any time before its expiration. The subject (sub) of the JWT is the same as the subject of the subject_token used to make the request. The actor (act) of the JWT is the same as the subject of the actor_token used to make the request. This indicates delegation and identifies admin@example.net as the current actor to whom authority has been delegated to act on behalf of user@example.net.

```
{
  "aud":"urn:example:cooperation-context",
  "iss":"https://as.example.com",
  "exp":1441913610,
  "scp":["status","feed"],
```

```
  "sub":"user@example.net",
  "act":
  {
    "sub":"admin@example.net"
  }
}
```

Figure 17: Issued Token Claims

Appendix B. Acknowledgements

This specification was developed within the OAuth Working Group, which includes dozens of active and dedicated participants. It was produced under the chairmanship of Hannes Tschofenig and Derek Atkins with Kathleen Moriarty and Stephen Farrell serving as Security Area Directors. The following individuals contributed ideas, feedback, and wording to this specification:

Caleb Baker, William Denniss, Vladimir Dzhuvinov, Phil Hunt, Jason Keglovitz, Nov Matake, Matt Miller, Matthew Perry, Justin Richer, Rifaat Shekh-Yusef, Scott Tomilson, and Hannes Tschofenig.

Appendix C. Open Issues

The following decisions need to be made and updates to this spec performed:

- Should there be a way to use short names for some common token type identifiers? URIs are necessary in the general case for extensibility and vendor/deployment specific types. But short names like access_token and jwt are aesthetically appealing and slightly more efficient in terms of bytes on the wire and url-encoding. There seemed to be rough consensus in Prague ('No objection to use the proposed mechanism for a default prefix' from https://www.ietf.org/proceedings/93/minutes/minutes-93-oauth) for supporting a shorthand for commonly used types - i.e. when the value does not contain a ":" character, the value would be treated as though urn:ietf:params:oauth:token-type: were prepended to it. So, for example, the value jwt for requested_token_type would be semantically equivalent to urn:ietf:params:oauth:token-type:jwt and the value access_token would be equivalent to urn:ietf:params:oauth:token-type:access_token. However, it was a fairly brief discussion in Prague and it has since been suggested that making participants handle both syntaxes will unnecessarily complicate the supporting code.
- Provide a way to include supplementary claims or information in the request that would/could potentially be included in the issued token. There are real use cases for this but we would need to work through what it would look like.
- Understand and define exactly how the presentation of PoP/non-bearer tokens works. Of course, the specifications defining these kinds of tokens need to do so first before there is much we can do in this specification in this regard.
- It seems there may be cases in which it would be desirable for the authenticated client to be somehow represented in the issued token, sometimes in addition to the actor, which can already be represented using the act claim. Perhaps with a client_id claim?

Appendix D. Document History

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

-04

- Clarified that the "resource" and "audience" request parameters can be used at the same time (via http://www.ietf.org/mail-archive/web/oauth/current/msg15335.html).
- Clarified subject/actor token validity after token exchange and explained a bit more about the recommendation to not issue refresh tokens (via http://www.ietf.org/mail-

archive/web/oauth/current/msg15318.html).

- Updated the examples appendix to use an issuer value that doesn't imply that the client issued and signed the tokens and used "Bearer" and "urn:ietf:params:oauth:token-type:access_token" in one of the responses (via http://www.ietf.org/mail-archive/web/oauth/current/msg15335.html).
- Defined and registered urn:ietf:params:oauth:token-type:id_token, since some use cases perform token exchanges for ID Tokens and no URI to indicate that a token is an ID Token had previously been defined.

-03

- Updated the document editors (adding Campbell, Bradley, and Mortimore).
- Added to the title.
- Added to the abstract and introduction.
- Updated the format of the request to use application/x-www-form-urlencoded request parameters and the response to use the existing token endpoint JSON parameters defined in OAuth 2.0.
- Changed the grant type identifier to urn:ietf:params:oauth:grant-type:token-exchange.
- Added RFC 6755 registration requests for urn:ietf:params:oauth:token-type:refresh_token, urn:ietf:params:oauth:token-type:access_token, and urn:ietf:params:oauth:grant-type:token-exchange.
- Added RFC 6749 registration requests for request/response parameters.
- Removed the Implementation Considerations and the requirement to support JWTs.
- Clarified many aspects of the text.
- Changed on_behalf_of to subject_token, on_behalf_of_token_type to subject_token_type, act_as to actor_token, and act_as_token_type to actor_token_type.
- Added an audience request parameter used to indicate the logical names of the target services at which the client intends to use the requested security token.
- Added a want_composite request parameter used to indicate the desire for a composite token rather than trying to infer it from the presence/absence of token(s) in the request.
- Added a resource request parameter used to indicate the URLs of resources at which the client intends to use the requested security token.
- Specified that multiple audience and resource request parameter values may be used.
- Defined the JWT claim act (actor) to express the current actor or delegation principal.
- Defined the JWT claim may_act to express that one party is authorized to act on behalf of another party.
- Defined the JWT claim scp (scopes) to express OAuth 2.0 scope-token values.
- Added the N_A (not applicable) OAuth Access Token Type definition for use in contexts in which the token exchange syntax requires a token_type value, but in which the token being issued is not an access token.
- Added examples.

-02

- Enabled use of Security Token types other than JWTs for act_as and on_behalf_of request values.
- Referenced the JWT and OAuth Assertions RFCs.

-01

- Updated references.

-00

- Created initial working group draft from draft-jones-oauth-token-exchange-01.

Authors' Addresses

Michael B. Jones
Microsoft

EMail: mbj@microsoft.com
URI: http://self-issued.info/

Anthony Nadalin
Microsoft
EMail: tonynad@microsoft.com

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

John Bradley
Ping Identity
EMail: ve7jtb@ve7jtb.com

Chuck Mortimore
Salesforce
EMail: cmortimore@salesforce.com