OAuth 2.0 Token Exchange
draft-ietf-oauth-token-exchange-08

Abstract

This specification defines a protocol for an 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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 4, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

1.1. Delegation vs. Impersonation Semantics
1.2. Requirements Notation and Conventions
1.3. Terminology

2. Token Exchange Request and Response

2.1. Request
2.1.1. Relationship Between Resource, Audience and Scope
2.2. Response
2.2.1. Successful Response
2.2.2. Error Response
2.3. Example Token Exchange

3. Token Type Identifiers

4. JSON Web Token Claims and Introspection Response Parameters

4.1. "act" (Actor) Claim
4.2. "scp" (Scopes) Claim
4.3. "cid" (Client Identifier) Claim
4.4. "may_act" (May Act For) Claim

5. IANA Considerations

5.1. OAuth URI Registration
5.1.1. Registry Contents
5.2. OAuth Parameters Registration
5.2.1. Registry Contents
5.3. OAuth Access Token Type Registration
5.3.1. Registry Contents
5.4. JSON Web Token Claims Registration
5.4.1. Registry Contents
5.5. OAuth Token Introspection Response Registration
5.5.1. Registry Contents
5.6. OAuth Extensions Error Registration
5.6.1. Registry Contents

6. Security Considerations

7. References

7.1. Normative References
7.2. Informative References

Appendix A. Additional Token Exchange Examples

A.1. Impersonation Token Exchange Example
A.1.1. Token Exchange Request
A.1.2. Subject Token Claims
A.1.3. Token Exchange Response
A.1.4. Issued Token Claims
A.2. Delegation Token Exchange Example
A.2.1. Token Exchange Request
A.2.2. Subject Token Claims
A.2.3. Actor Token Claims
A.2.4. Token Exchange Response
A.2.5. Issued Token Claims

Appendix B. Acknowledgements
Appendix C. Document History
Authors’ Addresses
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 as the protocol to interact with an STS for token exchange. While WS-Trust uses XML and SOAP, the trend in modern Web development has been towards RESTful patterns and JSON. The OAuth 2.0 Authorization Framework and OAuth 2.0 Bearer Tokens 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 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 needed or if proof-of-possession style tokens will be required or issued; 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. 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. When and if a composite token is issued is at the discretion of the
authorization server and applicable policy and configuration.

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 actor_token request parameter, however, does
provide a means for providing information about the desired actor and the JWT act claim can provide a
representation of a chain of delegation.

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.

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, and the
terms "Claim" and "JWT Claims Set" defined by JSON Web Token (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.

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 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. 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.

**actor_token**

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

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.

### 2.1.1. Relationship Between Resource, Audience and Scope

When requesting a token, the client can indicate the desired target service(s) where it intends to use that token by way of the audience and resource parameters, as well as indicating the desired scope of the requested token using the `scope` parameter. The semantics of such a request are that the client is asking for a token with the requested scope that is usable at all the requested target services. Effectively, the requested access rights of the token are the cartesian product of all the scopes at all the target services.

An authorization server may be unwilling or unable to fulfill any token request but the likelihood of an unfulfillable request is significantly higher when very broad access rights are being solicited. As such, in the absence of specific knowledge about the relationship of systems in a deployment, clients should exercise discretion in the breadth of the access requested, particularly the number of target services. An authorization server can use the `invalid_target` error code, defined in Section 2.2.2, to inform a client that it requested access to too many target services simultaneously.

### 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.

- `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. Often times 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

OPTIONAL. A refresh token will typically not be issued when the the exchange is of one temporary credential (the subject_token) for a different temporary credential (the issued token) for use in some other context. A refresh token can be issued in cases where the client of the token exchange needs the ability to access a resource even when the original credential is no longer valid (e.g. user-not-present or offline scenarios where there is no longer any user entertaining an active session with the client). Profiles or deployments of this specification should clearly document the conditions under which a client should expect a refresh token in response to urn:ietf:params:oauth:grant-type:token-exchange grant type requests.

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.

If the authorization server is unwilling or unable to issue a token for all the target services indicated by the resource or audience parameters, the invalid_target error code MAY be used in the error response.

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 protected resource 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].
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 in the role of 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.

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.

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

Several parameters in this specification utilize an identifier as the value to describe the 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. Token Exchange can work with both tokens issued by other parties and tokens from the given authorization server. For the former the token type identifier indicates the syntax (e.g. JWT or SAML) so the AS can parse it; for the latter it indicates what the AS issued it for (e.g. access_token or refresh_token).

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.

The distinction between an access token and a JWT is subtle. An access token represents a delegated authorization decision, whereas JWT is a token format. An access token can be formatted as a JWT but doesn't necessarily have to be. And a JWT might well be an access token but not all JWTs are access tokens. The intent of this specification is that urn:ietf:params:oauth:token-type:access_token be an indicator that the token is a typical OAuth access token issued by the authorization server in question, opaque to the client, and usable the same manner as any other access token obtained from that authorization server (it could well be a JWT but the client isn't and needn't be aware of that fact). Whereas urn:ietf:params:oauth:token-type:jwt is to indicate specifically that a JWT is being requested or sent (perhaps in a cross-domain use-case where the JWT is used as an authorization grant to obtain an access token from a different authorization server as is facilitated by [RFC7523]).

4. JSON Web Token Claims and Introspection Response Parameters

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 and in an OAuth 2.0 Token Introspection response. 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**

When included as a top-level member of an OAuth token introspection response, act has the same semantics and format as the the claim of the same name.
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.

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

```json
{
  "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

OAuth 2.0 Token Introspection defines the scope parameter to convey the scopes associated with the token.

4.3. "cid" (Client Identifier) Claim

The cid claim carries the client identifier of the OAuth 2.0 client that requested the token.

The following example illustrates the cid claim within a JWT Claims Set indicating an OAuth 2.0 client with "s6BhdRkqt3" as its identifier.

```json
{
  "aud":"https://consumer.example.com",
  "iss":"https://issuer.example.com",
  "exp":1443904177,
  "sub":"user@example.com",
  "cid":"s6BhdRkqt3"
}
```

Figure 8: Client Identifier Claim

OAuth 2.0 Token Introspection defines the client_id parameter as the client identifier for the OAuth 2.0 client that requested the token.

4.4. "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.

```json
{
  "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 9: May Act For Claim**

When included as a top-level member of an OAuth token introspection response, may_act has the same semantics and format as the the claim of the same name.

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]]

- 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
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
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: cid
- Claim Description: Client Identifier
- Change Controller: IESG
- Specification Document(s): Section 4.3 of [[this specification]]

- Claim Name: may_act
- Claim Description: May Act For
- Change Controller: IESG
- Specification Document(s): Section 4.4 of [[this specification]]

5.5. OAuth Token Introspection Response Registration

This specification registers the following values in the IANA "OAuth Token Introspection Response" registry [IANA.OAuth.Parameters] established by [RFC7662].

5.5.1. Registry Contents

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

- Claim Name: may_act
- Claim Description: May Act For
- Change Controller: IESG
- Specification Document(s): Section 4.4 of [[this specification]]

5.6. OAuth Extensions Error Registration

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

5.6.1. Registry Contents

- Error Name: invalid_target
- Error Usage Location: token error response
- Related Protocol Extension: OAuth 2.0 Token Exchange
- Change Controller: IETF
- Specification Document(s): Section 2.2.2 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"

7.2. Informative References

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and C. Mortimore, "OpenID Connect Core 1.0", August 2015.

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, a 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.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcm1uaW5hbC5wbi5jeGZhcmlTaWRvZ2h0c2l0ZDwv
...}

Figure 10: 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 11: 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.eyJhdWQiOiJ1cmVodW5lc3MiLCJpZCI6IjEyMDg4NSIsInN1YiI6IjEyMDg4NSIsImlzaCI6Im9uZzo6
..."}
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 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 13: Issued Token Claims

A.2. Delegation Token Exchange Example

A.2.1. Token Exchange Request

In the following token exchange request, a client is requesting a token and providing both a subject_token and an actor_token. 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. Policy at the authorization server dictates that the issued token be a composite.

```
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.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUubmV0IiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIubmV0Iiwic3ViIjoiYmNAZXhhbXBsZS5uZXQiLCJzY3AiOlsi3JkZXJzIiwibG9yeSI6IlsInByb2ZpbGUXX0.YQHuLm1YD
TugblEvGf9oZS8uMgaGBmMyj9BepZSECCBE9j9ogqZv2qx6VQQPrbT1k7vBYGLNMOkkpm
JkxZDS0YV7g",
"issued_token_type":
"urn:ietf:params:oauth:token-type:access_token",
"token_type":"Bearer",
"expires_in":3600
```

Figure 12: Token Exchange Response
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"
  }
}
```

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"
}
```

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...
}
```
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.

```json
{
    "aud":"urn:example:cooperation-context",
    "iss":"https://as.example.com",
    "exp":1441913610,
    "scp":null,
    "sub":"user@example.net",
    "act":{
        "sub":"admin@example.net"
    }
}
```

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:


Appendix C. Document History

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

- Use the bibxml reference for OpenID.Core rather than defining it inline.
- Added editor role for Campbell.
• Minor clarification of the text for actor_token.

-07
• Fixed typo (desecration -> discretion).
• Added an explanation of the relationship between scope, audience and resource in the request and added an "invalid_target" error code enabling the AS to tell the client that the requested audiences/resources were too broad.

-06
• Drop "An STS for the REST of Us" from the title.
• Drop "heavyweight" and "lightweight" from the abstract and introduction.
• Clarifications on the language around xxxxxxx_token_type.
• Remove the want_composite parameter.
• Add a short mention of proof-of-possession style tokens to the introduction and remove the respective open issue.

-05
• Defined the JWT claim cid to express the OAuth 2.0 client identifier of the client that requested the token.
• Defined and requested registration for act and may_act as Token introspection response parameters (in addition to being JWT claims).
• Loosen up the language about refresh_token in the response to OPTIONAL from NOT RECOMMENDED based on feedback from real world deployment experience.
• Add clarifying text about the distinction between JWT and access token URIs.
• Close out (remove) some of the Open Issues bullets that have been resolved.

-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 6749 registration requests for request/response parameters.
• Removed the Implementation Considerations and the requirement to support JWEs.
• 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 (editor)
Ping Identity
EMail: brian.d.campbell@gmail.com

John Bradley
Ping Identity
EMail: ve7jtb@ve7jtb.com

Chuck Mortimore
Salesforce
EMail: cmortimore@salesforce.com