Stream:	Internet Engineering Task Force (IETF)
RFC:	8693
Category:	Standards Track
Published:	January 2020
ISSN:	2070-1721
Authors:	M. Jones	A. Nadalin	B. Campbell, Ed.	J. Bradley	C. Mortimore
---	---	---	---	---	---
	Microsoft	*Microsoft*	*Ping Identity*	*Yubico*	*Visa*

RFC 8693
OAuth 2.0 Token Exchange

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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8693.

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 Security Assertion Markup Language (SAML) 2.0 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 security tokens provided to it and issuing new security tokens in response, 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. While WS-Trust uses XML and SOAP, the trend in modern Web development has been

towards RESTful (Representational State Transfer) patterns and JSON. The OAuth 2.0 Authorization Framework [RFC6749] and OAuth 2.0 Bearer Tokens [RFC6750] have emerged as popular standards for authorizing third-party applications' access to HTTP and RESTful resources. The conventional OAuth 2.0 interaction involves the exchange of some representation of resource owner authorization for an access token, which has proven to be an extremely useful pattern in practice. However, its input and output are somewhat too constrained as is to fully accommodate a security token exchange framework.

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 that 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 authorization server 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 may 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

One common use case for an STS (as alluded to in the previous section) is to allow a resource server A to make calls to a backend service C on behalf of the requesting user B. Depending on the local site policy and authorization infrastructure, it may be desirable for A to use its own credentials to access C along with an annotation of some form that A is acting on behalf of B ("delegation") or for A to be granted a limited access credential to C but that continues to identify B as the authorized entity ("impersonation"). Delegation and impersonation can be useful concepts in other scenarios involving multiple participants as well.

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 insofar 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 within the context of the rights authorized by the token. A's ability to impersonate B could be limited in scope or time, or even with a one-time-use restriction, whether via the contents of the token or an out-of-band mechanism.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 "Base64url Encoding", "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 [RFC6749].

Client authentication to the authorization server is done using the normal mechanisms provided by OAuth 2.0. Section 2.3.1 of [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 bearer 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. Note that omitting client authentication allows for a compromised token to be leveraged via an STS into other tokens by anyone possessing the compromised token. Thus, client authentication allows for additional authorization checks by the STS as to which entities are permitted to impersonate or receive delegations from other entities.

The client makes a token exchange request to the token endpoint with an extension grant type using the HTTP `POST` method. The following parameters are included in the HTTP request entity-body using the `application/x-www-form-urlencoded` format with a character encoding of UTF-8 as described in Appendix B of [RFC6749].

grant_type
: **REQUIRED**. The value `urn:ietf:params:oauth:grant-type:token-exchange` indicates that a token exchange is being performed.

resource
: **OPTIONAL**. A URI that indicates 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 a URI 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], that **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. See [OAUTH-RESOURCE] for additional background and uses of the `resource` parameter.

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 for the target service. 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], and an OpenID Connect Issuer Identifier [OpenID.Core] are examples of things that might be used as `audience` parameter values. However, `audience` values used with a given authorization server must be unique within that server to ensure that they are properly interpreted as the intended type of value. 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 resource URIs.

scope
: **OPTIONAL**. A list of space-delimited, case-sensitive strings, as defined in Section 3.3 of [RFC6749], 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. The values and associated semantics of scope are service specific and expected to be described in the relevant service documentation.

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 the 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.

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.

In processing the request, the authorization server **MUST** perform the appropriate validation procedures for the indicated token type and, if the actor token is present, also perform the appropriate validation procedures for its indicated token type. The validity criteria and details of any particular token are beyond the scope of this document and are specific to the respective type of token and its content.

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. Furthermore, the exchange is a one-time event and does not create a tight linkage between the input and output tokens, so that (for example) while the expiration time of the output token may be influenced by that of the input token, renewal or extension of the input token is not expected to be reflected in the output token's properties. It may still be appropriate or desirable to propagate token-revocation events. However, doing so is not a general property of the STS protocol and would be specific to a particular implementation, token type, or deployment.

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 indicate 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 [RFC8259], 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 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 issued 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 more typically used to mean the structural or syntactical representation of the security token, 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, it is **REQUIRED**.

refresh_token
: **OPTIONAL**. A refresh token will typically not be issued when 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 the request itself is not valid or 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 any target service indicated by the `resource` or `audience` parameters, the `invalid_target` error code **SHOULD** be used in the error response.

The authorization server **MAY** include additional information regarding the reasons for the error using the `error_description` as discussed in Section 5.2 of [RFC6749].

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 the exchange. It trades an access token, which it received in a protected resource request, for a new token that it will use to call to a backend service (extra line breaks and indentation in the examples are for display purposes only).

Figure 1 shows the resource server receiving a protected resource request containing an OAuth access token in the Authorization 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

In Figure 2, the resource server assumes the role of client for the token exchange, and the access token from the request in Figure 1 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.

```
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
&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 shown in Figure 3 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
   4YW1wbGUuY29tIiwiZXhwIjoxNDQxOTE3NTkzLCJpYXQiOjE0NDE5MTc1MzMsIn
   N1YiI6ImJkY0BleGFtcGxlLmNvbSIsInNjb3BlIjoiYXBpIn0.40y3ZgQedw6rx
   f59WlwHDD9jryFOr0_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIWvmDCM
   y5-kdXjwhw",
 "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 as illustrated in Figure 4.

```
GET /api HTTP/1.1
Host: backend.example.com
Authorization: Bearer eyJhbGciOiJFUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQ
   iOiJodHRwczovL2JhY2tlbmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2
   FzLmV4YW1wbGUuY29tIiwiZXhwIjoxNDQxOTE3NTkzLCJpYXQiOjE0NDE5MTc1M
   zMsInN1YiI6ImJkY0BleGFtcGxlLmNvbSIsInNjb3BlIjoiYXBpIn0.40y3ZgQe
   dw6rxf59WlwHDD9jryFOr0_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIW
   vmDCMy5-kdXjwhw
```

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 token in question. Specifically, they are the `requested_token_type`, `subject_token_type`, and `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 2.0) so the authorization server can parse it; for the latter, it indicates what the given authorization server issued it for (e.g., `access_token` or `refresh_token`).

The following token type identifiers are defined by this specification. Other URIs **MAY** be used to indicate other token types.

urn:ietf:params:oauth:token-type:access_token
: Indicates that the token is an OAuth 2.0 access token issued by the given authorization server.

urn:ietf:params:oauth:token-type:refresh_token
: Indicates that the token is an OAuth 2.0 refresh token issued by the given authorization server.

urn:ietf:params:oauth:token-type:id_token
: Indicates that the token is an ID Token as defined in Section 2 of [OpenID.Core].

urn:ietf:params:oauth:token-type:saml1
: Indicates that the token is a base64url-encoded SAML 1.1 [OASIS.saml-core-1.1] assertion.

urn:ietf:params:oauth:token-type:saml2
: Indicates that the token is a base64url-encoded SAML 2.0 [OASIS.saml-core-2.0-os] assertion.

The value `urn:ietf:params:oauth:token-type:jwt`, which is defined in Section 9 of [JWT], indicates that the token is a JWT.

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]).

Note that for tokens that are binary in nature, the URI used for conveying them needs to be associated with the semantics of a base64 or other encoding suitable for usage with HTTP and OAuth.

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 [RFC7662] response. Similar definitions for other types of tokens are possible but beyond the scope of this specification.

Note that the claims not established herein but used in examples and descriptions, such as `iss`, `sub`, `exp`, etc., are defined by [JWT].

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, non-identity claims (e.g., `exp`, `nbf`, and `aud`) are not meaningful when used within an `act` claim and are therefore not used.

Figure 5 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 nested `act` claims serve as a history trail that connects the initial request and subject through the various delegation steps undertaken before reaching the current actor. In this sense, the current actor is considered to include the entire authorization/delegation history, leading naturally to the nested structure described here.

For the purpose of applying access control policy, the consumer of a token **MUST** only consider the token's top-level claims and the party identified as the current actor by the `act` claim. Prior actors identified by any nested `act` claims are informational only and are not to be considered in access control decisions.

The following example in Figure 6 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 <https://service16.example.com> is the current actor and <https://service77.example.com> was a prior actor. Such a token might come about as the result of service16 receiving a token in a call from service77 and exchanging it for a token suitable to call service26 while the authorization server notes the situation in the newly issued token.

```
{
  "aud":"https://service26.example.com",
  "iss":"https://issuer.example.com",
  "exp":1443904100,
  "nbf":1443904000,
  "sub":"user@example.com",
  "act":
  {
    "sub":"https://service16.example.com",
    "act":
    {
      "sub":"https://service77.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 claim of the same name.

4.2. "scope" (Scopes) Claim

The value of the `scope` claim is a JSON string containing a space-separated list of scopes associated with the token, in the format described in Section 3.3 of [RFC6749].

Figure 7 illustrates the `scope` claim within a JWT Claims Set.

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

Figure 7: Scopes Claim

OAuth 2.0 Token Introspection [RFC7662] already defines the `scope` parameter to convey the scopes associated with the token.

4.3. "client_id" (Client Identifier) Claim

The `client_id` claim carries the client identifier of the OAuth 2.0 [RFC6749] client that requested the token.

The following example in Figure 8 illustrates the `client_id` claim within a JWT Claims Set indicating an OAuth 2.0 client with "s6BhdRkqt3" as its identifier.

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

Figure 8: Client Identifier Claim

OAuth 2.0 Token Introspection [RFC7662] already defines the `client_id` parameter as the client identifier for the OAuth 2.0 client that requested the token.

4.4. "may_act" (Authorized Actor) 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 might be used, for example, when a `subject_token` is presented to the token endpoint in a token exchange request and `may_act` claim in the subject token can be used by the authorization server to determine whether the client (or party identified in the `actor_token`) is authorized to engage in the requested delegation or impersonation. 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 are therefore not used.

Figure 9 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 9: Authorized Actor Claim

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

5. Security Considerations

Much of the guidance from Section 10 of [RFC6749], the Security Considerations in The OAuth 2.0 Authorization Framework, is also applicable here. Furthermore, [RFC6819] provides additional security considerations for OAuth, and [OAUTH-SECURITY] has updated security guidance based on deployment experience and new threats that have emerged since OAuth 2.0 was originally published.

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 `scope` claim (in addition to other typical constraints such as a limited token lifetime) is suggested to mitigate potential for such abuse, as it restricts the contexts in which the delegated rights can be exercised.

6. Privacy Considerations

Tokens employed in the context of the functionality described herein may contain privacy-sensitive information and, to prevent disclosure of such information to unintended parties, **MUST** only be transmitted over encrypted channels, such as Transport Layer Security (TLS). In cases where it is desirable to prevent disclosure of certain information to the client, the token **MUST** be encrypted to its intended recipient. Deployments **SHOULD** determine the minimally necessary amount of data and only include such information in issued tokens. In some cases, data minimization may include representing only an anonymous or pseudonymous user.

7. IANA Considerations

7.1. OAuth URI Registration

IANA has registered the following values in the "OAuth URI" subregistry of the "OAuth Parameters" registry [IANA.OAuth.Parameters]. The "OAuth URI" subregistry was established by [RFC6755].

- 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 RFC 8693

- 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 RFC 8693

- 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 RFC 8693

- 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 RFC 8693

- URN: urn:ietf:params:oauth:token-type:saml1
- Common Name: Token type URI for a base64url-encoded SAML 1.1 assertion
- Change Controller: IESG
- Specification Document: Section 3 of RFC 8693

- URN: urn:ietf:params:oauth:token-type:saml2
- Common Name: Token type URI for a base64url-encoded SAML 2.0 assertion
- Change Controller: IESG
- Specification Document: Section 3 of RFC 8693

7.2. OAuth Parameters Registration

IANA has registered the following values in the "OAuth Parameters" subregistry of the "OAuth Parameters" registry [IANA.OAuth.Parameters]. The "OAuth Parameters" subregistry was established by [RFC6749].

- Parameter name: audience
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of RFC 8693

- Parameter name: requested_token_type
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of RFC 8693

- Parameter name: subject_token
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of RFC 8693

- Parameter name: subject_token_type
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of RFC 8693

- Parameter name: actor_token
- Parameter usage location: token request
- Change controller: IESG

- Specification document(s): Section 2.1 of RFC 8693

- Parameter name: actor_token_type
- Parameter usage location: token request
- Change controller: IESG
- Specification document(s): Section 2.1 of RFC 8693

- Parameter name: issued_token_type
- Parameter usage location: token response
- Change controller: IESG
- Specification document(s): Section 2.2.1 of RFC 8693

7.3. OAuth Access Token Type Registration

IANA has registered the following access token type in the "OAuth Access Token Types" subregistry of the "OAuth Parameters" registry [IANA.OAuth.Parameters]. The "OAuth Access Token Types" subregistry was established by [RFC6749].

- 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 RFC 8693

7.4. JSON Web Token Claims Registration

IANA has registered the following Claims in the "JSON Web Token Claims" subregistry of the "JSON Web Token (JWT)" registry [IANA.JWT]. The "JSON Web Token Claims" subregistry was established by [JWT].

- Claim Name: act
- Claim Description: Actor
- Change Controller: IESG
- Specification Document(s): Section 4.1 of RFC 8693

- Claim Name: scope
- Claim Description: Scope Values
- Change Controller: IESG
- Specification Document(s): Section 4.2 of RFC 8693

- Claim Name: client_id
- Claim Description: Client Identifier
- Change Controller: IESG

- Specification Document(s): Section 4.3 of RFC 8693

- Claim Name: may_act
- Claim Description: Authorized Actor - the party that is authorized to become the actor
- Change Controller: IESG
- Specification Document(s): Section 4.4 of RFC 8693

7.5. OAuth Token Introspection Response Registration

IANA has registered the following values in the "OAuth Token Introspection Response" registry of the "OAuth Parameters" registry [IANA.OAuth.Parameters]. The "OAuth Token Introspection Response" registry was established by [RFC7662].

- Name: act
- Description: Actor
- Change Controller: IESG
- Specification Document(s): Section 4.1 of RFC 8693

- Name: may_act
- Description: Authorized Actor - the party that is authorized to become the actor
- Change Controller: IESG
- Specification Document(s): Section 4.4 of RFC 8693

8. References

8.1. Normative References

[IANA.JWT] IANA, "JSON Web Token (JWT)", , <https://www.iana.org/assignments/jwt>.

[IANA.OAuth.Parameters] IANA, "OAuth Parameters", , <https://www.iana.org/assignments/oauth-parameters>.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[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, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC7662]

Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

8.2. Informative References

[OASIS.saml-core-1.1] Maler, E., Mishra, P., and R. Philpott, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1", OASIS Standard oasis-sstc-saml-core-1.1, September 2003, <https://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf>.

[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, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.

[OAUTH-RESOURCE] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", Work in Progress, Internet-Draft, draft-ietf-oauth-resource-indicators-08, 11 September 2019, <https://tools.ietf.org/html/draft-ietf-oauth-resource-indicators-08>.

[OAUTH-SECURITY] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-security-topics-13, 8 July 2019, <https://tools.ietf.org/html/draft-ietf-oauth-security-topics-13>.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/info/rfc6755>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[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, <https://www.rfc-editor.org/info/rfc7521>.

[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, <https://www.rfc-editor.org/info/rfc7523>.

[WS-Trust] Nadalin, A., Ed., Goodner, M., Ed., Gudgin, M., Ed., Barbir, A., Ed., and H. Granqvist, Ed., "WS-Trust 1.4", February 2012, <https://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>.

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 (delegation is impossible with only a `subject_token` and no `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`.

```
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
  3ViIjoiYmRjQGV4YW1wbGUubmV0Iiwic2NvcGUiOiJvcmRlcnMgcHJvZmlsZSBoaXN
  0b3J5In0.PRBg-jXn4cJuj1gmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZ
  hF64pbTtfjy4VXFVBDaQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
```

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 (`bdc@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":"bdc@example.net",
  "scope":"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.eyJhdWQiOiJ1cm4
   6ZXhhbXBsZTpjb29wZXJhdGlvbi1jb250ZXh0IiwiaXNzIjoiaHR0cHM6Ly9hcy5l
   eGFtcGxlLmNvbSIsImV4cCI6MTQ0MTkxMzYxMCwic3ViIjoiYmRjQGV4YW1wbGUub
   mV0Iiwic2NvcGUiOiJvcmRlcnMgcHJvZmlsZSBoaXN0b3J5In0.rMdWpSGNACTvnF
   uOL74sYZ6MVuld2Z2WkGLmQeR9ztj6w2OXraQlkJmGjyiCq24kcB7AI2VqVxl3wSW
   nVKh85A",
 "issued_token_type":
   "urn:ietf:params:oauth:token-type:access_token",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Figure 12: 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":"bdc@example.net",
  "scope":"orders profile history"
}
```

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.eyJhdWQiOiJodHRwc
  zovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI
  uZXhhbXBsZS5uZXQiLCJleHAiOjE0NDE5MTAwNjAsInNjb3BlIjoic3RhdHVzIGZlZ
  WQiLCJzdWIiOiJ1c2VyQGV4YW1wbGUubmV0IiwibWF5X2FjdCI6eyJzdWIiOiJhZG1
  pbkBleGFtcGxlLm5ldCJ9fQ.4rPRSWihQbpMIgAmAoqaJojAxj-p2X8_fAtAGTXrvM
  xU-eEZHnXqY0_AOZgLdxw5DyLzua8H_I10MCcckF-Q_g
&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 14: 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,
  "scope":"status feed",
  "sub":"user@example.net",
  "may_act":
  {
    "sub":"admin@example.net"
  }
}
```

Figure 15: 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 16: 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
   eGFtcGxlLmNvbSIsImV4cCI6MTQ0MTkxMzYxMCwic2NvcGUiOiJzdGF0dXMgZmVlZ
   CIsInN1YiI6InVzZXJAZXhhbXBsZS5uZXQiLCJhY3QiOnsic3ViIjoiYWRtaW5AZX
   hhbXBsZS5uZXQifX0.3paKl9UySKYB5ng6_cUtQ2qlO8Rc_y7Mea7IwEXTcYbNdwG
   9-G1EKCFe5fW3H0hwX-MSZ49Wpcb1SiAZaOQBtw",
 "issued_token_type":"urn:ietf:params:oauth:token-type:jwt",
 "token_type":"N_A",
 "expires_in":3600
}
```

Figure 17: 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,
   "scope":"status feed",
   "sub":"user@example.net",
   "act":
   {
     "sub":"admin@example.net"
   }
 }
```

Figure 18: Issued Token Claims

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, Derek Atkins, and Rifaat Shekh-Yusef, with Kathleen Moriarty, Stephen Farrell, Eric Rescorla, Roman Danyliw, and Benjamin Kaduk serving as Security Area Directors.

The following individuals contributed ideas, feedback, and wording to this specification: Caleb Baker, Vittorio Bertocci, Mike Brown, Thomas Broyer, Roman Danyliw, William Denniss, Vladimir Dzhuvinov, Eric Fazendin, Phil Hunt, Benjamin Kaduk, Jason Keglovitz, Torsten Lodderstedt, Barry Leiba, Adam Lewis, James Manger, Nov Matake, Matt Miller, Hilarie Orman, Matthew Perry, Eric Rescorla, Justin Richer, Adam Roach, Rifaat Shekh-Yusef, Scott Tomilson, and Hannes Tschofenig.

Authors' Addresses

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: https://self-issued.info/

Anthony Nadalin
Microsoft
Email: tonynad@microsoft.com

Brian Campbell (EDITOR)
Ping Identity
Email: brian.d.campbell@gmail.com

John Bradley
Yubico
Email: ve7jtb@ve7jtb.com

Chuck Mortimore
Visa
Email: chuck.mortimore@visa.com