Workgroup: oauth
Internet-Draft:
draft-ietf-oauth-transaction-tokens-00
Published: 29 November 2023
Intended Status: Informational
Expires: 1 June 2024
Authors: A. Tulshibagwale
SGNL
G. Fletcher
Capital One
P. Kasselman
Microsoft

Transaction Tokens

Abstract

Transaction Tokens (Txn-Tokens) enable workloads in a trusted domain to ensure that user identity and authorization context of an external programmatic request, such as an API invocation, are preserved and available to all workloads that are invoked as part of processing such a request. Txn-Tokens also enable workloads within the trusted domain to optionally immutably assert to downstream workloads that they were invoked in the call chain of the request.

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 https://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 1 June 2024.

Table of Contents

1. Introduction

Modern computing architectures often use multiple independently running components called workloads. In many cases, external invocations through externally visible interfaces such as APIs result in a number of internal workloads being invoked in order to process the external invocation. These workloads often run in virtually or physically isolated networks. These networks and the workloads running within their perimeter may be compromised by attackers through software supply chain, privileged user compromise or other attacks. Workloads compromised through external attacks, malicious insiders or software errors can cause any or all of the following unauthorized actions:

- Invocations of workloads in the network without any external invocation being present
- Arbitrary user impersonation
- Parameter modification or augmentation

The results of these actions are unauthorised access to resources.

2. Overview

Transaction Tokens (Txn-Token) are a means to mitigate damage from such attacks or spurious invocations. A valid Txn-Token indicates a valid external invocation. They ensure that the identity of the user or a workload that made the external request is preserved throughout subsequent workload invocations. They preserve any context such as:

- Parameters of the original call
- Environmental factors, such as IP address of the original caller
- Any computed context that needs to be preserved in the call chain. This includes information that was not in the original request to the external endpoint.

Cryptographically protected Txn-Tokens ensure that downstream workloads cannot make unauthorized modifications to such information, and cannot make spurious calls without the presence of an external trigger.

2.1. What are Transaction Tokens?

Txn-Tokens are short-lived, signed JWTs [RFC7519] that assert the identity of a user or a workload and assert an authorization context. The authorization context provides information expected to remain

constant during the execution of a call as it passes through multiple workloads.

2.2. Creating Txn-Tokens

2.2.1. Initial Creation

Txn-Tokens are typically created when a workload is invoked using an endpoint that is externally visible, and is authorized using a separate mechanism, such as an OAuth [RFC6749] access token or an OpenID Connect [OpenIdConnect] ID token. This workload then performs an OAuth 2.0 Token Exchange [RFC8693] to obtain a Txn-Token. To do this, it invokes a special Token Service (the Txn-Token Service) and provides context that is sufficient for it to generate a Txn-Token. This context MAY include:

- The external authorization token (e.g., the OAuth access token)
- Parameters that are required to be bound for the duration of this call
- Additional context, such as the incoming IP address, User Agent information, or other context that can help the Txn-Token Service to issue the Txn-Token

The Txn-Token Service responds to a successful invocation by generating a Txn-Token. The calling workload then uses the Txn-Token to authorize its calls to subsequent workloads. Subsequent workloads may obtain Txn-Tokens of their own.

2.2.2. Replacement Txn-Tokens

A service within a call chain may choose to replace the Txn-Token. This can typically happen if the service wants to add to the context of the current Txn-Token

To get a replacement Txn-Token, a service will request a new Txn-Token from the Txn-Token Service and provide the current Txn-Token and other parameters in the request. The Txn-Token service must exercise caution in what kinds of replacement requests it supports so as to not negate the entire value of Txn-Tokens.

2.3. Txn-Token Lifetime

Txn-Tokens are expected to be short-lived (order of minutes, e.g., 5 minutes), and as a result MAY be used only for the expected duration of an external invocation. If the token or other credential presented to the Txn-Token service when requesting a Txn-Token has an expiration time, then the Txn-Token MUST NOT exceed the lifetime of the originally presented token or credential. If a long-running

process such as an batch or offline task is involved, it can use a separate mechanism to perform the external invocation, but the resulting Txn-Token is still short-lived.

2.4. Benefits of Txn-Tokens

Txn-Tokens help prevent spurious invocations by ensuring that a workload receiving an invocation can independently verify the user or workload on whose behalf an external call was made and any context relevant to the processing of the call. Through the presence of additional signatures on the Txn-Token, a workload receiving an invocation can also independently verify that specific workloads were within the path of the call before it was invoked.

2.5. Txn-Token Issuance and Usage Flows

2.5.1. Basic Flow

Figure 1 shows the basic flow of how Txn-Tokens are used in an a multi-workload environment.

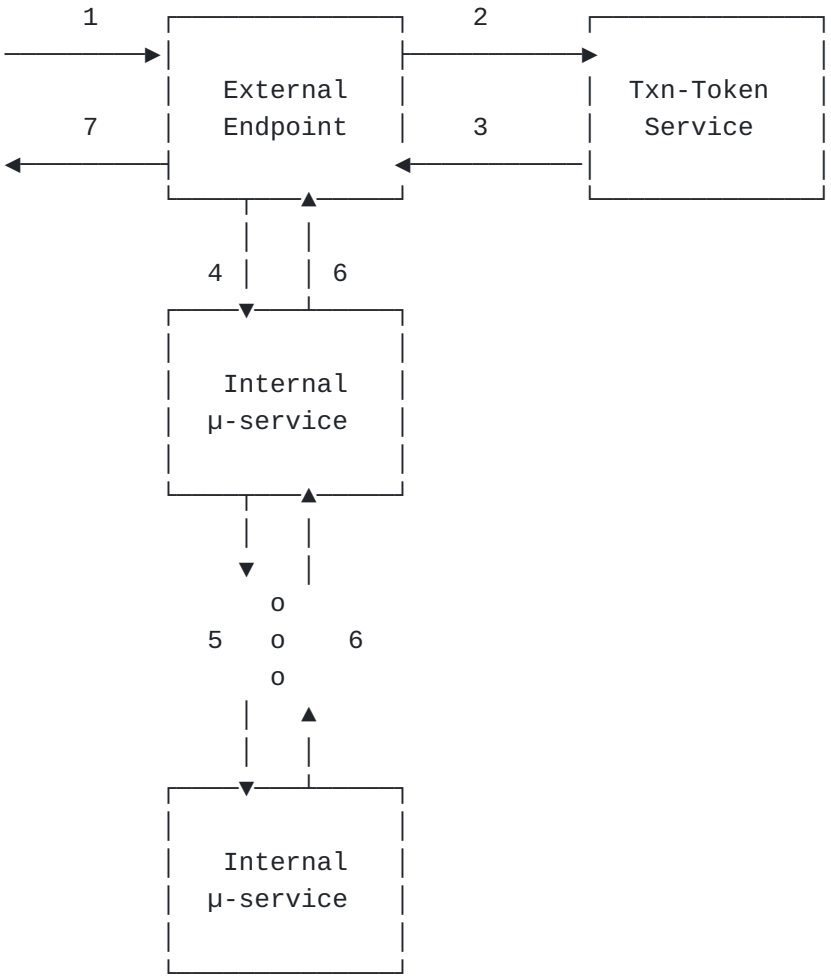

Figure 1: Basic Transaction Tokens Architecture

1. External endpoint is invoked using conventional authorization mechanism such as an OAuth 2.0 Access token

2. External endpoint provides context and incoming authorization (e.g., access token) to the Txn-Token Service

3. Txn-Token Service mints a Txn-Token that provides immutable context for the transaction and returns it to the requester

4. The external endpoint initiates a call to an internal microservice and provides the Txn-Token as authorization

5. Subsequent calls to other internal microservices use the same Txn-Token to authorize calls

6. Responses are provided to callers based on successful authorization by the invoked microservices

7. External client is provided a response to the external invocation

2.5.2. Replacement Txn-Token Flow

An intermediate service may decide to obtain a replacement Txn-Token from the Txn-Token service. That flow is described below in Figure 2

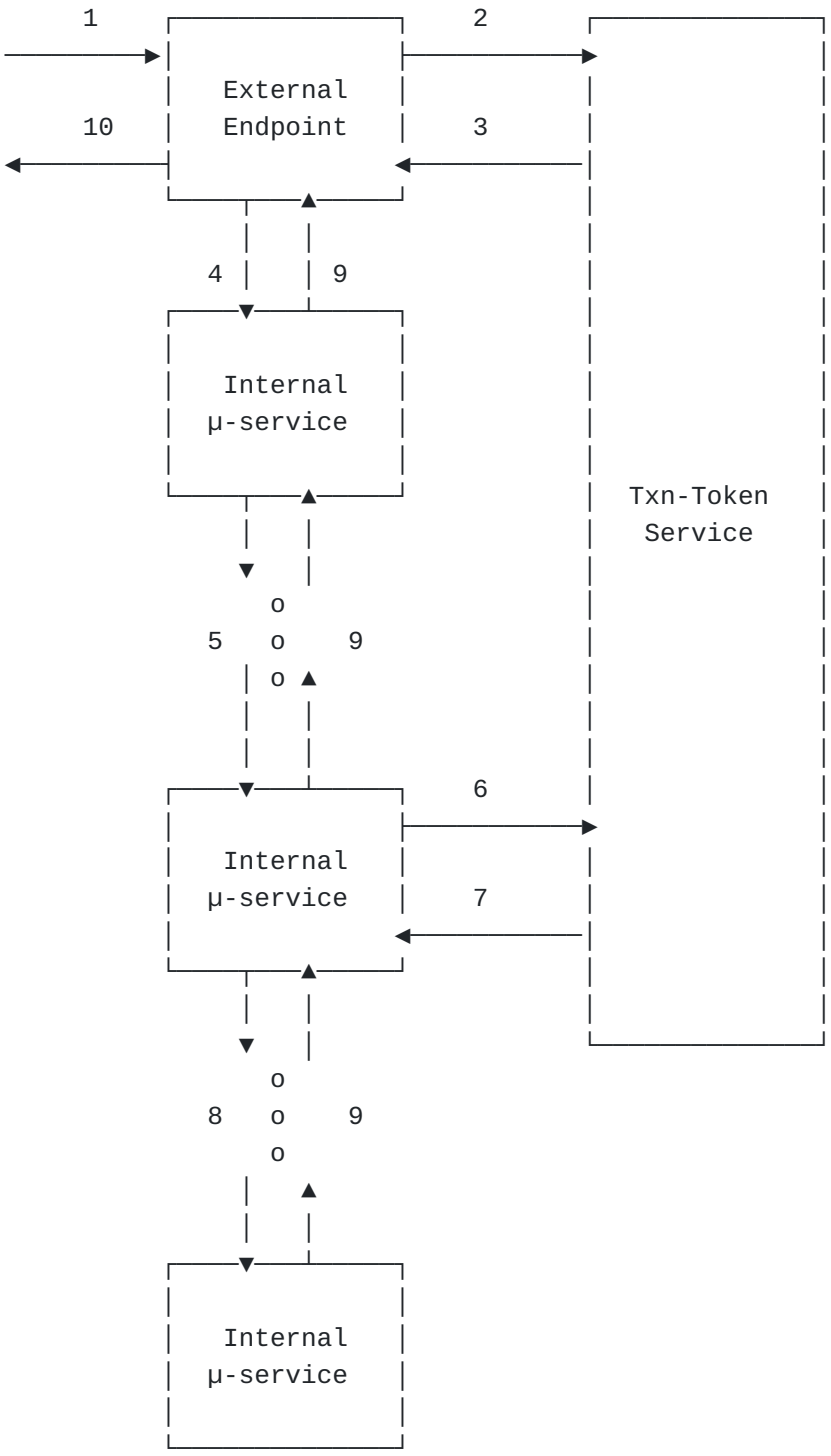

Figure 2: Replacement Txn-Token Flow

In the diagram above, steps 1-5 are the same as in Section 2.5.1

6. An intermediate service determines that it needs to obtain a Replacement Txn-Token. It requests a Replacement Txn-Token from the Txn-Token Service. It passes the incoming Txn-Token in the

request, along with any additional context it needs to send the Txn-Token Service.

7. The Txn-Token Service responds with a replacement Txn-Token

8. The service that requested the Replacement Txn-Token uses that Txn-Token for downstream call authorization

9. Responses are provided to callers based on successful authorization by the invoked microservices

10. External client is provided a response to the external invocation

3. Notational 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.

4. Terminology

Workload: An independent computational unit that can autonomously receive and process invocations, and can generate invocations of other workloads. Examples of workloads include containerized microservices, monolithic services and infrastructure services such as managed databases.

Trust Domain: A virtually or physically separated network, which contains two or more workloads. The workloads within an Trust Domain may be invoked only through published interfaces. A Trust Domain must have an identifier that is used as the aud (audience) value in Txn-Tokens. The format of this identifier is a universal resource identifier. Each Trust Domain has exactly one Txn-Token Service.

External Endpoint: A published interface to an Trust Domain that results in the invocation of a workload within the Trust Domain.

Call Chain: A sequence of invocations that results from the invocation of an external endpoint.

Transaction Token (Txn-Token): A signed JWT that has a short lifetime, which provides immutable information about the user or

workload, certain parameters of the call and certain contextual attributes of the call.

Authorization Context: A JSON object containing a set of claims that represent the immutable context of a call chain.

Transaction Token Service (Txn-Token Service): A special service within the Trust Domain, which issues Txn-Tokens to requesting workloads. Each Trust Domain has exactly one Txn-Token Service.

5. Txn-Token Format

A Txn-Token is a JSON Web Token [RFC7519] protected by a JSON Web Signature [RFC7515]. The following describes the required values in a Txn-Token:

5.1. JWT Header

In the JWT Header:

- The typ claim MUST be present and MUST have the value txn_token.
- Key rotation of the signing key SHOULD be supported through the use of a kid claim.

Figure 3 is a non-normative example of the JWT Header of a Txn-Token

```
{
    "typ": "txn_token",
    "alg": "RS256",
    "kid": "identifier-to-key"
}
```

Figure 3: Example: Txn-Token Header

5.2. JWT Body

5.2.1. Required Claims

The JWT body MUST have the following claims:

- An iss claim, whose value is a URN [RFC8141] that uniquely identifies the workload or the Txn-Token Service that created the Txn-Token.
- An iat claim, whose value is the time at which the Txn-Token was created.

*An aud claim, whose value is a URN [RFC8141] that uniquely identifies the audience of the Txn-Token. This MUST identify the trust domain in which the Txn-Token is used.

*An exp claim, whose value is the time at which the Txn-Token expires.

*A txn claim, whose value is the unique transaction identifier as defined in Section 2.2 of [RFC8417]. When used in the transaction token, it identifies the entire call chain.

*A sub_id claim, whose value is the unique identifier of the user or workload on whose behalf the call chain is being executed. The format of this claim MAY be a Subject Identifier as specified in [SubjectIdentifiers].

*An azd claim, whose value is a JSON object that contains values that remain constant in the call chain.

5.2.2. Optional Claims

The JWT body MAY have the following claims:

5.2.2.1. Requester Context

The Txn-Token MAY contain an req_ctx claim, whose value is a JSON object the describes the requester context of the transaction. This MAY include the IP address information of the originating user, as well as information about the computational entity that requested the Txn-Token.

The JSON value of the req_ctx claim MAY include any values the Txn-Token Service determines are interesting to downstream services that rely on the Txn-Token. The following claims are defined so that if they are included, they have the following meaning:

*req_ip The IP address of the requester. This MAY be the end-user or a robotic process that requested the Transaction

*authn The authentication method used to idenitfy the requester. Its value is a URN that uniquely identifies the method used.

*req_wl The requesting workload. A URN that uniquely identifies the computational entity that requested the Txn-Token. This entity MUST be within the Trust Domain of the Txn-Token.

5.2.2.2. Purpose

The Txn-Token MAY contain a purp claim, whose value specifies the purpose of the transaction. The format of this claim is a JSON string.

5.2.3. Example

The figure below Figure 4 shows a non-normative example of the JWT body of a Txn-Token:

```
{
    "iss": "https://trust-domain.example/txn-token-service",
    "iat": "1686536226000",
    "aud": "trust-domain.example",
    "exp": "1686536526000",
    "txn": "97053963-771d-49cc-a4e3-20aad399c312",
    "sub_id": {
        "format": "email",
        "email": "user@trust-domain.example"
    },
    "req_ctx": {
      "req_ip": "69.151.72.123", // env context of external call
      "authn": "urn:ietf:rfc:6749", // env context of the external call
      "req_wl": "apigateway.trust-domain.example" // the internal entity
    },
    "purp" : "trade.stocks",
    "azd": {
        "action": "BUY", // parameter of external call
        "ticker": "MSFT", // parameter of external call
        "quantity": "100", // parameter of external call
        "user_level": "vip" // computed value not present in external ca
    }
}
```

Figure 4: Example: Txn-Token Body

6. Txn-Token Service

A Txn-Token Service provides a OAuth 2.0 Token Exchange [RFC8693] endpoint that can respond to Txn-Token issuance requests. The token exchange requests it supports require extra parameters than those defined in the OAuth 2.0 Token Exchange [RFC8693] specification. The unique properties of the Txn-Token requests and responses are described below. The Txn-Token Service MAY optionally support other OAuth 2.0 endpoints and features, but that is not a requirement for it to be a Txn-Token Service.

Each Trust Domain MUST have exactly one Txn-Token Service.

7. Requesting Txn-Tokens

A workload requests a Txn-Token from a Transaction Token Service using OAuth 2.0 Token Exchange [RFC8693]. The request to obtain a Txn-Token using this method is called a Txn-Token Request, and a successful response is called a Txn-Token Response. A Txn-Token Request is a Token Exchange Request, as described in Section 2.1 of [RFC8693] with additional parameters. A Txn-Token Response is a OAuth 2.0 token endpoint response, as described in Section 5 of [RFC6749], where the token_type in the response has the value txn_token.

7.1. Txn-Token Request

A Txn-Token Request is an OAuth 2.0 Token Exchange Request, as described in Section 2.1 of [RFC8693], with an additional parameter in the request. The following parameters are required in the Txn-Token Request by the OAuth 2.0 Token Exchange specification [RFC8693]:

- *The audience value MUST be set to the Trust Domain name
- *The requested_token_type value MUST be urn:ietf:params:oauth:token-type:txn_token
- *The subject_token value MUST be the external token received by the workload that authorized the call
- *The subject_token_type value MUST be present and indicate the type of the authorization token present in the subject_token parameter

The following additional parameter MUST be present in a Txn-Token Request:

- *A parameter named rctx , whose value is a JSON object. This object contains the request context, i.e. any information the Transaction Token Service needs to understand the context of the incoming request

Figure 5 shows a non-normative example of a Txn-Token Request.

```
POST /txn-token-service/token_endpoint HTTP 1.1
Host: txn-token-service.trust-domain.example
Content-Type: application/x-www-form-urlencoded

requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Atxn_toke
&audience=http%3A%2F%2Ftrust-domain.example
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZC...kdXjwhw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_to
&rctx=%7B%22param1%22%3A%22value1%22%2C%22param2%22%3A%22value2%22%2C%22
```

Figure 5: Example: Txn-Token Request

7.2. Txn-Token Response

A successful response to a Txn-Token Request by a Transaction Token Service is called a Txn-Token Response. If the Transaction Token Service responds with an error, the error response is as described in Section 5.2 of [RFC6749]. The following describes required values of a Txn-Token Response:

- The token_type value MUST be set to txn_token
- The access_token value MUST be the Txn-Token
- The response MUST NOT include the values expires_in, refresh_token and scope

Figure 6 shows a non-normative example of a Txn-Token Response.

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

{
  "issued_token_type": "urn:ietf:params:oauth:token-type:txn_token",
  "access_token": "eyJCI6IjllciJ9...Qedw6rx"
}
```

Figure 6: Example: Txn-Token Response

7.3. Creating Replacement Txn-Tokens

A workload within a call chain may request the Transaction Token Server to replace a Txn-Token.

Workloads MAY request replacement Txn-Tokens in order to change (add to, remove or modify) the asserted values within a Txn-Token.

The value of the aud claim MUST remain unchanged in a replacement Txn-Token. If the claim req_ctx is present in the original Txn-Token, then it MUST be present unchanged in the replacement Txn-Token.

7.3.1. Txn-Token Service Responsibilities

A Txn-Token Service replacing a Txn-Token must consider that modifying previously asserted values from existing Txn-Tokens can

completely negate the benefits of Txn-Tokens. When issuing replacement Txn-Tokens, a Transaction Token Server therefore:

- MAY enable modifications to asserted values that reduce the scope of permitted actions
- MAY enable additional asserted values
- SHOULD NOT enable modification to asserted values that expand the scope of permitted actions

7.3.2. Replacement Txn-Token Request

To request a replacement Txn-Token, the requester makes a Txn-Token Request as described in Section 7.1 but includes the Txn-Token to be replaced as the value of the subject_token parameter.

7.3.3. Replacement Txn-Token Response

A successful response by the Transaction Token Server to a Replacement Txn-Token Request is a Txn-Token Response as described in Section 7.2

7.4. Mutual Authentication of the Txn-Token Request

A Txn-Token Service MUST ensure that it authenticates any workloads requesting Txn-Tokens. In order to do so:

- It MUST name a limited, pre-configured set of workloads that MAY request Txn-Tokens
- It MUST individually authenticate the requester as being one of the named requesters
- It SHOULD rely on mechanisms, such as [Spiffe] or some other means of performing MTLS [RFC8446], to securely authenticate the requester
- It SHOULD NOT rely on insecure mechanisms, such as long-lived shared secrets to authenticate the requesters

The requesting workload MUST have a pre-configured location for the Transaction Token Service. It SHOULD rely on mechanisms, such as [Spiffe], to securely authenticate the Transaction Token Service before making a Txn-Token Request.

8. Security Considerations

8.1. Txn-Token Lifetime

A Txn-Token is not resistant to replay attacks. A long-lived Txn-Token therefore represents a risk if it is stored in a file, discovered by an attacker, and then replayed. For this reason, a Txn-Token lifetime must be kept short, not exceeding the lifetime of a call-chain. Even for long-running "batch" jobs, a longer lived access token should be used to initiate the request to the batch endpoint. It then obtains short-lived Txn-Tokens that may be used to authorize the call to downstream services in the call-chain.

Because Txn-Tokens are short-lived, the Txn-Token response from the Txn-Token service does not contain the refresh_token field. A Txn-Token cannot be issued by presenting a refresh_token.

The expires_in and scope fields of the OAuth 2.0 Token Exchange specification [RFC8693] are also not used in Txn-Token responses. The expires_in is not required since the issued token has an exp field, which indicates the token lifetime. The scope field is omitted from the request and therefore omitted in the response.

8.2. Sender Constrained Tokens

Although Txn-Tokens are short-lived, they MAY be sender constrained as an additional layer of defence to prevent them from being re-used by a compromised or malicious workload under the control of a hostile actor.

8.3. Access Tokens

When creating Txn-Tokens, the Txn-Token MUST NOT contain the Access Token presented to the external endpoint. If an Access Token is included in a Txn-Token, an attacker may extract the Access Token from the Txn-Token, and replay it to any Resource Server that can accept that Access Token. Txn-Token expiry does not protect against this attack since the Access Token may remain valid even after the Txn-Token has expired.

9. Privacy Considerations

9.1. Obsfucation of Personal Information

Some req_ctx claims may be considered personal information in some jurisdictions and if so their values need to be obsfucated. For example, originating IP address (req_ip) is often considerd personal information and in that case must be protected through some obsfucation method (e.g. SHA256).

10. IANA Considerations

This specification registers the following claims defined in Section Section 5.1 to the OAuth Access Token Types Registry defined in [RFC6749], and the following claims defined in Section Section 5.2.1 in the IANA JSON Web Token Claims Registry defined in [RFC7519]

10.1. OAuth Registry Contents

*Name: txn_token

*Description: JWT of type Transaction Token

*Additional Token Endpoint Response Parameters: none

*HTTP Authentication Schemes: TLS [RFC8446]

*Change Controller: IESG

*Specification Document: Section Section 5.1 of this specificaiton

10.2. JWT Registry Contents

*Claim Name: azd

-Claim Description: The authorization context details

-Change Controller: IESG

-Specification Document: Section Section 5.2.1 of this specification

*Claim Name: req_ctx

-Claim Description: The requester context

-Change Controller: IESG

-Specification Document: Section Section 5.2.2.1 of this specification

*Claim Name: purp

-Claim Description: The purpose of the transaction

-Change Controller: IESG

-Specification Document: Section Section 5.2.2.2 of this specification

11. References

11.1. Normative References

[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/rfc/rfc2119>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

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

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/rfc/rfc8141>.

[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/rfc/rfc8174>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC8417] Hunt, P., Ed., Jones, M., Denniss, W., and M. Ansari, "Security Event Token (SET)", RFC 8417, DOI 10.17487/RFC8417, July 2018, <https://www.rfc-editor.org/rfc/rfc8417>.

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

[SubjectIdentifiers] Backman, A., Scurtescu, M., and P. Jain, "Subject Identifiers for Security Event Tokens", n.d.,

<https://datatracker.ietf.org/doc/html/draft-ietf-secevent-subject-identifiers>.

11.2. Informative References

[Spiffe] Cloud Native Computing Foundation, "Secure Production Identity Framework for Everyone", n.d., <https://spiffe.io/docs/latest/spiffe-about/overview/>.

Acknowledgements

Contributors

Dr. Kelley W. Burgin, PhD.
MITRE Corporation

Email: kburgin@mitre.org

Hannes Tschofenig
Arm Ltd.

Email: Hannes.Tschofenig@arm.com

Evan Gilman
SPIRL

Email: evan@spirl.com

Arndt Schwenkschuster
Microsoft

Email: arndts@microsoft.com

Authors' Addresses

Atul Tulshibagwale
SGNL

Email: atul@sgnl.ai

George Fletcher
Capital One

Email: george.fletcher@capitalone.com

Pieter Kasselman
Microsoft

Email: pieter.kasselman@microsoft.com