OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: 26 June 2024

R. Shekh-Yusef
Ernst & Young
D. Hardt
Hell
G. De Marco
Dipartimento per la trasformazione digitale

24 December 2023

JSON Web Token (JWT) Embedded Tokens
draft-yusef-oauth-nested-jwt-08

Abstract

This specification defines a mechanism for embedding tokens into a JWT token. The JWT token and the embedded tokens are issued by one ore more issuers.

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 26 June 2024.

Table of Contents

1. Introduction

JWT is a mechanism that is used to transfer claims between two parties across security domains. There are a number of use cases that need to embed tokens into another JWT token.

This specification defines a mechanism for embedding tokens into a JWT token. The JWT token and the embedded tokens are issued by different issuers.

Such a mechanism allows for a proper auditing trail that allows the Resource Server to identify who accessed what resource and on behalf of whom. In some cases, this allows the service consuming such a token to present some of the information contained in the nested token or claims to the end user in real-time. In addition, in some cases, this allows the Resource Server to apply authorization policies based on who requested the access to the resource and on behalf of whom is the request.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC8174].

2. Use Cases

The following are few use cases that might benefit from such a concept:

2.1. STIR

[RFC8225] defines a PASSporT, which is a JWT, that is used to verify the identity of a caller in an incoming call.

The PASSporT Extension for Diverted Calls draft [STIR] uses a nested PASSporT to deliver the details of an incoming call that get redirected. An authentication service acting for a retargeting entity generates new PASSporT and embeds the original PASSporT inside the new one. When the new target receives the nested PASSporT it will be able to validate the enclosing PASSporT and use the details of the enclosed PASSporT to identify the original target.

In this case, the original JWT is issued by the calling service, and the new enclosing JWT is issued by the retargeting service.

2.2. Network Service Mesh (NSM)

Network Service Mesh [NSM] is a mechanism that maps the concept of a service mesh in Kubernetes to L2/L3 payloads.

NSM GRPS messages may pass through multiple intermediaries, each of which may transform the message. Each intermediary is expected to create its own JWT token, and include a claim that contains the JWT it received with the message it has transformed.

In this case, the original JWT is issued by the entity sending the initial message, and the new enclosing JWT is issued by the intermediate entity.

2.3. Multiple Issuers for same Subject

A JWT may have embedded claims from one or more separate Issuers.

An ID Token may have identity claims from independent issuers such as DOB and a professional accreditation.

2.4. Multiple tokens for multiple resources

A JWT may embeds tokens for different audiences and scopes.

An Authorization Server issues a JWT Token that contains multiple tokens. Each token has a specialized set of attributes and values. The tokens can be used by the client to consume protected resources or to obtain access tokens through a token exchange mechanism, over different domains, releasing the minimum possible number of information, related on the main subject, necessary for the operation.

3. Embedded Tokens Request and Response

This section describes the mechanism to request an existing token or tokens to be embedded in a new token. The mechanism defines a new grant type embedded-tokens for this purpose.

In some cases, the embedding of tokens into a new token could be done locally, if that entity is trusted to so. If this is the case, then this request/response process defined below is not needed, and the trusted entity can then issue the tokens in the format specified in section 4; see the examples provided in sections 3.2.1 or 3.2.2.

3.1. Embedded Tokens Request

When a client receives a token, and it needs to transform or/and enhance the permissions of the token, the client will send a token request to the AS, and include the received token or tokens to be embedded in the newly issued token that contains the transformed token details or permissions.

The Client requests a token by sending an authorization request to the authorization servers token endpoint and include the following parameters in a JSON payload:

grant_type (REQUIRED)
: Carries the new grant type to indicate to the token endpoint that the provided token(s) to be embedded into the newly issued token. The parameter value MUST be urn:ietf:params:oauth:grant-type:embedded-tokens

tokens (REQUIRED)
: An array of objects, where each object represents a token that contains the type claim and either the token claim, or the digest" and jti claims, as defined in section 4 of this specification.

The request MAY include any of the following parameters, defined in Section 2.1 of RFC8693: resource, audience, scope, and requested_token_type.

The following is an example of this new token request:

```
POST /token
Host: as.example.com
Content-Type: application/json

{
   "grant_type": "urn:ietf:params:oauth:grant-type:embedded-tokens",
   "requested_token_type": "urn:ietf:params:oauth:token-type:access_token"
   "tokens": [
     {
       "type": "urn:ietf:params:oauth:token-type:access_token",
       "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
                 eyJzdWIiOiIyMzQ1Njc4OTAxIiwibmFtZSI6IkFsZXggRG9lIiwia
                 WF0IjoxNTE2MjM5MDIyLCJqdGkiOiJYRkVYYlNDMHhpTXUifQ.
                 wyycbcY9i0U5YldltNhp4WbM-gF3Q1-jtnc-Wvzyxcg"
     }
   ]
}
```

The following is the decoded example embedded token in the above example:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
.
{
  "sub": "2345678901",
  "name": "Alex Doe",
  "iat": 1516239022,
  "jti": "XFEXbSC0xiMu"
}
```

3.2. Embedded Tokens Successful Response

Before issuing the requested token, the authorization server MUST ensure that the request is valid and that the embedded token(s) provided in the subject_token is coming from a trusted and approved entity

In the case that the authorization server validated and approved the request, the authorization server responds to the above request with the standard token exchange response, as defined in section 2.2.1 of [RFC8693].

The type of token issued is as specified in the requested_token_type parameter, if provided. Otherwise, the type of token issued is at the discretion of the authorization server.

3.2.1. Embedded Tokens By Value

The authorization server will typically embed the provided tokens directly into the newly issued token, when there are no concerns around the size of the new token with the embedded tokens.

The following is an example for a JWT token with an embedded token:

```
{
  "alg": "HS256",
  "typ": "JWT",
}
.
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "tokens": [
     {
        "type": "urn:ietf:params:oauth:token-type:access_token",
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
                  eyJzdWIiOiIyMzQ1Njc4OTAxIiwibmFtZSI6IkFsZXggRG9lIiwia
                  WF0IjoxNTE2MjM5MDIyLCJqdGkiOiJYRkVYYlNDMHhpTXUifQ.
                  wyycbcY9i0U5YldltNhp4WbM-gF3Q1-jtnc-Wvzyxcg"
     }
   ]
}
```

3.2.2. Embedded Tokens By Reference

In some cases, embedding the tokens into the JWT directly might cause the token size to become too large. In this case, instead of embedding the tokens, the AS will embed a hash of the token(s) and its associated 'jti' claim(s). This allows the Client to send these tokens directly to the RS with the newly issued JWT, to allow the RS to validate that the tokens are indeed associated with this new issued JWT.

The following is an example for a JWT token with a reference to the token:

```
{
  "alg": "HS256",
  "typ": "JWT",
}
.
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "tokens": [
    {
      "type": "urn:ietf:params:oauth:token-type:access_token:reference",
      "digest":
      {
        "alg": "sha-256",
        "hash": "68e439fd95964da902a8654d47c51d6bc0a7791ea9895173989b263374a9a125
"
      }
      "jti": "XFEXbSC0xiMu"
    }
  ]
}
```

3.3. Embedded Tokens Error Response

If the authorization fails to validate the embedded token, then the authorization server MUST construct an error response, as specified in section 5.2 of RFC6749. The value of the error parameter MUST be invalid_embedded_token error code.

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].

4. JSON Web Token Claims

This section defines the new claims that will be used to represent the embedded tokens. It defines one top-level claim, "tokens" claim, and 3 sub claims under that, ’type’, "token", and "digest" claims.

4.1. "tokens" Claim

The "tokens" claim is an array of objects, where each object represents a token, that contains the type claim and either the token claim, or the digest" and jti claims.

4.1.1. "type" Claim

The "type" claim is used to indicate the type of embedded token, which takes one of the values defined in Section 3, RFC8693.

4.1.2. "token" Claim

The "token" claim is used to hold the details of the embedded token.

4.1.3. "digest" Claim

The "digest" claim is an object that is used to hold the hash of the token, to be used to reference the token instead of embedding it directly in the new JWT.ß

The object contains the following two claims:

4.1.3.1. "alg" Claim

Holds the algorithm used to hash the token. By default this is "sha-256".

4.1.3.2. "hash" Claim

Holds the hash value of the token using the algorithm defined in the "alg" claim.

5. Security Considerations

The entity handling the incoming authorization request MUST validate the token and ensure that it is coming from a trusted entity, before attempting to embed that into a newly issued JWT.

6. IANA Considerations

TODO

7. Acknowledgments

TODO

8. References

8.1. Normative References

[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/info/rfc7519>.

[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>.

8.2. Informative References

[RFC8225] Wendt, C. and J. Peterson, "PASSporT: Personal Assertion Token", RFC 8225, DOI 10.17487/RFC8225, February 2018, <https://www.rfc-editor.org/info/rfc8225>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", October 2018.

[STIR] Peterson, J., "PASSporT Extension for Diverted Calls", October 2018.

Authors' Addresses

Rifaat Shekh-Yusef
Ernst & Young
Ottawa, Ontario, Canada
Email: rifaat.s.ietf@gmail.com

Dick Hardt
Hell
Seattle, Washington, USA
Email: dick.hardt@gmail.com

Giuseppe De Marco
Dipartimento per la trasformazione digitale
Italy
Email: giuseppe.demarco@teamdigitale.governo.it