OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 22, 2016

J. Bradley
Ping Identity
A. Sanso, Ed.
Adobe Systems
H. Tschofenig
July 21, 2015

OAuth 2.0 Security: OAuth Open Redirector
draft-bradley-oauth-open-redirector-02.txt

Abstract

This document gives additional security considerations for OAuth, beyond those in the OAuth 2.0 specification and in the OAuth 2.0 Threat Model and Security Considerations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document gives additional security considerations for OAuth, beyond those in the OAuth 2.0 specification [RFC6749] and in the OAuth 2.0 Threat Model and Security Considerations [RFC6819]. In particular focuses its attention on the risk of abuse the Authorization Server (AS) (Section 1.2) as an open redirector.

It contains the following content:

- o Describes the Authorization Server Error Response as defined in [RFC6749].
- o Describes the risk of abuse the Authorization Server as an open redirector.
- o Gives some mitigation details on how to hinder the risk of open redirector in the ?AS?.

1.1. Notational Conventions

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 RFC 2119 [RFC2119].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

1.2. Terminology

Authorization Server (AS)
: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

Redirection endpoint

Used by the authorization server to return responses containing authorization credentials to the client via the resource owner user-agent.

2. Authorization Server Error Response

The OAuth 2.0 specification [RFC6749] defines the Error Response associated with the Authorization Code Grant flow and the Implicit Grant flow. Both flows use a redirection endpoint where the resource owner’s user agent is directed after the resource owner has completed interacting with the authorization server. The redirection endpoint is also used in the error response scenario. As per RFC6749 Section 4.1.2.1 and 4.2.2.1 [RFC6749] if the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the ?AS? redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found Location: https://client.example.com/ cb?error=access_denied

2.1. Abuse: The Authorization Server As Open Redirector

As described in [RFC6819] an attacker could utilize a user’s trust in an ?AS? to launch a phishing attack. The attack described here though is not mitigated using the countermeasures listed in [RFC6819]. In this scenario the attacker:

- Performs a client registration as per the core specification [RFC6749]. The provided redirection URI is a malicious one e.g. https://attacker.com (namely the one where the victim’s user agent will land without any validation)

- Prepare a forged URI using the assumption that the ?AS? complies with the OAuth 2.0 specification [RFC6749]. In particular with the ?AS? Error Response described in the previous section (Section 2). As an example he can use a wrong or not existing scope e.g.

  https://AUTHORIZATION_SERVER/authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2Fattacker%2Ecom&scope=INVALID_SCOPE

- Attempt the pishing attack trying to have the victim clicking the forged URI prepared on the previous step. Should the attack succeeds the victim’s user agent is redirected to https://attacker.com (all with any user interaction) The HTTP

referer header will be set to the AS domain perhaps allowing manipulation of the user.

2.2. Security Compromise: The Authorization Server As Open Redirector

The attacker can use a redirect error redirection to intercept redirect based protocol messages via the Referer header and URI fragment. In this scenario the attacker:

- o Performs a registration of a malicious client as per the core specification [RFC6749]. The provided redirection URI is a malicious one e.g. https://attacker.com (This URI will capture the fragment and referer header sent as part of the error)

- o Creates a invalid Authentication request URI for the malicious client. As an example he can use a wrong or not existing scope e.g.

  https://AUTHORIZATION_SERVER/authorize?response_type=code&client_id=malicious_client&redirect_uri=https%3A%2F%2Fattacker%2Ecom&scope=INVALID_SCOPE

- o If the AS supports sticky grants (not re-prompting for consent based on a previous grant) a valid authentication request for the user may also be used to trigger a 30x redirect.

- o Performs a OAuth Authorization request using the invalid Authorization request as the redirect_uri. This works if the AS is pattern matching redirect_uri and has a public client that shares the same domain as the AS.

(line breaks for display only)

```
https://AUTHORIZATION_SERVER/authorize?response_type=token
&client_id=good-client&scope=VALID_SCOPE
&redirect_uri=https%3A%2F%2AUTHORIZATION_SERVER%Fauthorize
%3Fresponse_type%3Dcode
%26client_id%3Dattacker-client-id
%26scope%3DINVALID_SCOPE
%26redirect_uri%3Dhttps%253A%252F%252Fattacker.com
```

Figure 1

- o Receive the response redirected to https://attacker.Com

The legitimate OAuth Authorization response will include an access token in the URI fragment.

Most web browsers will append the fragment to the URI sent in the location header of a 302 response if no fragment is included in the location URI.

If the Authorization request is code instead of token, the same technique is used, but the code is leaked by the browser in the referer header rather than the fragment.

This causes the access token from a successful authorization to be leaked across the redirect to the malicious client. This is due to browser behaviour and not because the AS has included any information in the redirect URI other than the error code.

Protocols other than OAuth may be particularly vulnerable to this if they are only verifying the domain of the redirect. Performing exact redirect URI matching in OAuth will protect the AS, but not other protocols.

It should be noted that a legitimate OAuth client registered with a AS might be compromised and used as a redirect target by an attacker, perhaps without the knowledge of the client site. This increases a the attack surface for a ?AS?.

2.3. Mitigation

In order to defend against the attacks described in Section 2.1 and Section 2.2 the ?AS? can either:

- o Respond with an HTTP 400 (Bad Request) status code.

- o Perform a redirect to an intermediate URI under the control of the AS to clear referer information in the browser that may contain security token information. This page SHOULD provide notice to the resource owner that an error occurred, and request permission to redirect them to an external site.

  If redirected, a fragment "#" MUST be appended to the error redirect URI. This prevents the browser from reattaching the fragment from a previous URI to the new location URI.

  Some

When redirecting via 30x a Content Security Policy header SHOULD be added:

```
Content-Security-Policy: referrer origin;
```

Figure 2

When redirecting via a form post the following tag SHOULD be included:

```
<meta name="referrer" content="origin"/>
```

Figure 3

Only newer browsers support these headders, so users with older browsers will be vulnerable to leaking referer information unless a intermediate redirect is used.s

3. Acknowledgements

We would like to thank all the people that partecipated to the discussion, namely Bill Burke, Hans Zandbelt, Justin P. Richer, Phil Hunt, Takahiko Kawasaki, Torsten Lodderstedt, Sergey Beryozkin.

4. Normative References

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

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

[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, <http://www.rfc-editor.org/info/rfc6819>.

Appendix A. Document History

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

-01

- o Added information on HTTP headders to include to set referrer to origin

-00

- o Wrote the first draft.

- o Changed Document name to conform to WG naming convention
- o Added Section on redirect leaking security information
- o Added Terminology section
- o fixed file name
- o cleaned up mitigations a bit

Authors' Addresses

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Antonio Sanso (editor)
Adobe Systems

Email: asanso@adobe.com

Hannes Tschofenig

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2015

B. Campbell
G. Liu
Ping Identity

February 23, 2015

Destination Claim for JSON Web Token
draft-campbell-oauth-dst4jwt-00

Abstract

The Destination Claim for JSON Web Token (JWT) provides a means of indicating the address to which the JWT is sent. The Claim can be used to preventing malicious forwarding or redirection of a JWT to unintended recipients.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

JWT [I-D.ietf-oauth-json-web-token] is a compact, URL-safe means of representing claims to be transferred between two parties. Oftentimes an HTTP 302 redirect or an auto-submitted HTML form, using the user agent as a intermediary, is employed as the method of transfer. The Destination Claim provides a standard way for for the Issuer to indicate the address to which it instructed the user agent to deliver the JWT. The recipient of the JWT can detect and prevent malicious forwarding or redirection to unintended recipients by verifying that the address conveyed by the Destination Claim matches the actual location at which the JWT was received.

While the Destination Claim bears some seeming similarity to the Audience Claim already defined in JWT, the distinction is that the Audience identifies _who_ the JWT is intended for while the Destination identifies _where_ the JWT is sent.

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

This specification uses the terms "JSON Web Token (JWT)", "Issuer" "Claim", "Claim Name", and "Claim Value" as defined in [I-D.ietf-oauth-json-web-token], and the term "user agent" as defined by RFC 7230 [RFC7230].

2. The Destination Claim

The Claim Name of the Destination Claim is "dst" and its Claim Value is a URI [RFC3986] indicating the address to which the JWT is sent. Use of this Claim is OPTIONAL but, if the Claim is present, the recipient MUST check that the URI identifies the location at which the JWT was received. If the JWT is received at a different location than the one conveyed by the value of the "dst" claim, then the JWT MUST be rejected.

3. IANA Considerations

3.1. JSON Web Token Claim Registration

This specification registers the Destination Claim defined herein in the IANA JSON Web Token Claims registry defined in [I-D.ietf-oauth-json-web-token].

3.1.1. Registry Request Contents

- o Claim Name: "dst"
- o Claim Description: Destination
- o Change Controller: IESG
- o Specification Document(s): Section 2 of this document

4. Security Considerations

The Destination Claim defined in Section 2 provides a means to assist in detecting and preventing malicious forwarding or redirection of a JWT to unintended recipients. If, for example, an Issuer can be tricked into sending a JWT to a malicious site (perhaps due to inadequate checking of the target URI combined with Cross-Site Request Forgery) the JWT would be unusable at the legitimate site because the "dst" would contain a URI of the malicious site.

5. References

5.1. Normative References

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-32 (work in progress), December 2014.

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

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

5.2. Informative References

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

Appendix A. Open Issues

- Is there compelling reason to allow the "dst" Claim to accommodate multiple values? A single value is sufficient for the cases envisioned and is certainly simpler.

Appendix B. Document History

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

-00

- Gotta start somewhere...

Authors' Addresses

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

Guoping Liu
Ping Identity

Email: gliu@pingidentity.com

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2016

J. Richer, Ed.
July 3, 2015

OAuth 2.0 Token Introspection
draft-ietf-oauth-introspection-11

Abstract

This specification defines a method for a protected resource to query an OAuth 2.0 authorization server to determine the active state of an OAuth 2.0 token and to determine meta-information about this token. OAuth 2.0 deployments can use this method to convey information about the authorization context of the token from the authorization server to the protected resource.

Requirements Language

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 RFC 2119 [RFC2119].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In OAuth 2.0, the contents of tokens are opaque to clients. This means that the client does not need to know anything about the content or structure of the token itself, if there is any. However, there is still a large amount of metadata that may be attached to a token, such as its current validity, approved scopes, and information about the context in which the token was issued. These pieces of information are often vital to protected resources making authorization decisions based on the tokens being presented. Since OAuth 2.0 [RFC6749] does not define a protocol for the resource server to learn meta-information about a token that is has received from an authorization server, several different approaches have been developed to bridge this gap. These include using structured token formats such as JWT [RFC7519] or proprietary inter-service communication mechanisms (such as shared databases and protected enterprise service buses) that convey token information.

This specification defines a protocol that allows authorized protected resources to query the authorization server to determine the set of metadata for a given token that was presented to them by an OAuth 2.0 client. This metadata includes whether or not the token is currently active (or if it has expired or otherwise been revoked), what rights of access the token carries (usually conveyed through OAuth 2.0 scopes), and the authorization context in which the token was granted (including who authorized the token and which client it was issued to). Token introspection allows a protected resource to query this information regardless of whether or not it is carried in the token itself, allowing this method to be used along with or independently of structured token values. Additionally, a protected resource can use the mechanism described in this specification to introspect the token in a particular authorization decision context and ascertain the relevant metadata about the token to make this authorization decision appropriately.

1.1. Notational Conventions

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 [RFC2119].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

1.2. Terminology

This section defines the terminology used by this specification. This section is a normative portion of this specification, imposing requirements upon implementations.

This specification uses the terms "access token", "authorization endpoint", "authorization grant", "authorization server", "client", "client identifier", "protected resource", "refresh token", "resource owner", "resource server", and "token endpoint" defined by OAuth 2.0 [RFC6749], and the terms "claim names" and "claim values" defined by JSON Web Token (JWT) [RFC7519].

This specification defines the following terms:

Token Introspection
: The act of inquiring about the current state of an OAuth 2.0 token through use of the network protocol defined in this document.

Introspection Endpoint
: The OAuth 2.0 endpoint through which the token introspection operation is accomplished..

2. Introspection Endpoint

The introspection endpoint is an OAuth 2.0 endpoint that takes a parameter representing an OAuth 2.0 token and returns a JSON [RFC7159] document representing the meta information surrounding the token, including whether this token is currently active. The definition of an active token is dependent upon the authorization server, but this is commonly a token that has been issued by this authorization server, is not expired, has not been revoked, and valid for use at the protected resource making the introspection call.

The introspection endpoint MUST be protected by a transport-layer security mechanism as described in Section 4. The means by which the protected resource discovers the location of the introspection endpoint are outside the scope of this specification.

2.1. Introspection Request

The protected resource calls the introspection endpoint using an HTTP POST [RFC7231] request with parameters sent as "application/x-www-form-urlencoded" data as defined in [W3C.REC-html5-20141028]. The protected resource sends a parameter representing the token along with optional parameters representing additional context that is known by the protected resource to aid the authorization server in its response.

token REQUIRED. The string value of the token. For access tokens, this is the "access_token" value returned from the token endpoint defined in OAuth 2.0 [RFC6749] section 5.1. For refresh tokens, this is the "refresh_token" value returned from the token endpoint as defined in OAuth 2.0 [RFC6749] section 5.1. Other token types are outside the scope of this specification.

token_type_hint OPTIONAL. A hint about the type of the token submitted for introspection. The protected resource MAY pass this parameter to help the authorization server to optimize the token lookup. If the server is unable to locate the token using the given hint, it MUST extend its search across all of its supported token types. An authorization server MAY ignore this parameter, particularly if it is able to detect the token type automatically. Values for this field are defined in the OAuth Token Type Hints registry defined in OAuth Token Revocation [RFC7009].

The introspection endpoint MAY accept other OPTIONAL parameters to provide further context to the query. For instance, an authorization server may desire to know the IP address of the client accessing the protected resource to determine if the correct client is likely to be presenting the token. The definition of this or any other parameters

are outside the scope of this specification, to be defined by service documentation or extensions to this specification. If the authorization server is unable to determine the state of the token without additional information, it SHOULD return an introspection response indicating the token is not active as described in Section 2.2.

To prevent token scanning attacks, the endpoint MUST also require some form of authorization to access this endpoint, such as client authentication as described in OAuth 2.0 [RFC6749] or a separate OAuth 2.0 access token such as the bearer token described in OAuth 2.0 Bearer Token Usage [RFC6750]. The methods of managing and validating these authentication credentials are out of scope of this specification.

For example, the following example shows a protected resource calling the token introspection endpoint to query about an OAuth 2.0 bearer token. The protected resource is using a separate OAuth 2.0 bearer token to authorize this call.

Following is a non-normative example request:

```
POST /introspect HTTP/1.1
Host: server.example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Bearer 23410913-abewfq.123483

token=2YotnFZFEjr1zCsicMWpAA
```

In this example, the protected resource uses a client identifier and client secret to authenticate itself to the introspection endpoint as well as send a token type hint.

Following is a non-normative example request:

```
POST /introspect HTTP/1.1
Host: server.example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

token=mF_9.B5f-4.1JqM&token_type_hint=access_token
```

2.2. Introspection Response

The server responds with a JSON object [RFC7159] in "application/ json" format with the following top-level members.

active
: REQUIRED. Boolean indicator of whether or not the presented token is currently active. The specifics of a token’s "active" state will vary depending on the implementation of the authorization server, and the information it keeps about its tokens, but a "true" value return for the "active" property will generally indicate that a given token has been issued by this authorization server, has not been revoked by the resource owner, and is within its given time window of validity (e.g. after its issuance time and before its expiration time). See Section 4 for information on implementation of such checks.

scope
: OPTIONAL. A JSON string containing a space-separated list of scopes associated with this token, in the format described in section 3.3 of OAuth 2.0 [RFC6749].

client_id
: OPTIONAL. Client identifier for the OAuth 2.0 client that requested this token.

username
: OPTIONAL. Human-readable identifier for the resource owner who authorized this token.

token_type
: OPTIONAL. Type of the token as defined in section 5.1 of OAuth 2.0 [RFC6749].

exp
: OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire, as defined in JWT [RFC7519].

iat
: OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token was originally issued, as defined in JWT [RFC7519].

nbf
: OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token is not to be used before, as defined in JWT [RFC7519].

sub
OPTIONAL. Subject of the token, as defined in JWT [RFC7519]. Usually a machine-readable identifier of the resource owner who authorized this token.

aud
OPTIONAL. Service-specific string identifier or list of string identifiers representing the intended audience for this token, as defined in JWT [RFC7519].

iss
OPTIONAL. String representing the issuer of this token, as defined in JWT [RFC7519].

jti
OPTIONAL. String identifier for the token, as defined in JWT [RFC7519].

Specific implementations MAY extend this structure with their own service-specific response names as top-level members of this JSON object. Response names intended to be used across domains MUST be registered in the OAuth Token Introspection Response registry defined in Section 3.1.

The authorization server MAY respond differently to different protected resources making the same request. For instance, an authorization server MAY limit which scopes from a given token are returned for each protected resource to prevent protected resources from learning more about the larger network than is necessary for its operation.

The response MAY be cached by the protected resource to improve performance and reduce load on the introspection endpoint, but at the cost of liveness of the information used by the protected resource. See Section 4 for more information regarding the trade off when the response is cached.

For example, the following response contains a set of information about an active token:

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "active": true,
 "client_id": "l238j323ds-23ij4",
 "username": "jdoe",
 "scope": "read write dolphin",
 "sub": "Z5O3upPC88QrAjx00dis",
 "aud": "https://protected.example.net/resource",
 "iss": "https://server.example.com/",
 "exp": 1419356238,
 "iat": 1419350238,
 "extension_field": "twenty-seven"
}
```

If the introspection call is properly authorized but the token is not active, does not exist on this server, or the protected resource is not allowed to introspect this particular token, the authorization server MUST return an introspection response with the active field set to false. Note that to avoid disclosing too much of the authorization server’s state to a third party, the authorization server SHOULD NOT include any additional information about an inactive token, including why the token is inactive. For example, the response for a token that has been revoked or is otherwise invalid would look like the following:

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "active": false
}
```

2.3. Error Response

If the protected resource uses OAuth 2.0 client credentials to authenticate to the introspection endpoint and its credentials are invalid, the authorization server responds with an HTTP 401 (Unauthorized) as described in section 5.2 of OAuth 2.0 [RFC6749].

If the protected resource uses an OAuth 2.0 bearer token to authorize its call to the introspection endpoint and the token used for authorization does not contain sufficient privileges or is otherwise invalid for this request, the authorization server responds with an HTTP 401 code as described in section 3 of OAuth 2.0 Bearer Token Usage [RFC6750].

Note that a properly formed and authorized query for an inactive or otherwise invalid token (or a token the protected resource is not allowed to know about) is not considered an error response by this specification. In these cases, the authorization server MUST instead respond with an introspection response with the "active" field set to "false" as described in Section 2.2.

3. IANA Considerations

3.1. OAuth Token Introspection Response Registry

This specification establishes the OAuth Token Introspection Response registry.

OAuth registration client metadata names and descriptions are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of names prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register OAuth Token Introspection Response name: example").

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

3.1.1. Registration Template

Name:
The name requested (e.g., "example"). This name is case sensitive. Names that match other registered names in a case

insensitive manner SHOULD NOT be accepted. Names that match claims registered in the JSON Web Token Claims registry established by [RFC7519] SHOULD have comparable definitions and semantics.

Description:
Brief description of the metadata value (e.g., "Example description").

Change controller:
For Standards Track RFCs, state "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s):
Reference to the document(s) that specify the token endpoint authorization method, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

3.1.2. Initial Registry Contents

The initial contents of the OAuth Token Introspection Response registry are:

- Name: "active"
- Description: Token active status
- Change Controller: IESG
- Specification Document(s): Section 2.2 of [[this document]].

- Name: "username"
- Description: User identifier of the resource owner
- Change Controller: IESG
- Specification Document(s): Section 2.2 of [[this document]].

- Name: "client_id"
- Description: Client identifier of the client
- Change Controller: IESG
- Specification Document(s): Section 2.2 of [[this document]].

- Name: "scope"
- Description: Authorized scopes of the token
- Change Controller: IESG
- Specification Document(s): Section 2.2 of [[this document]].

- Name: "token_type"
- Description: Type of the token
- Change Controller: IESG

- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "exp"
- o Description: Expiration timestamp of the token
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "iat"
- o Description: Issuance timestamp of the token
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "nbf"
- o Description: Timestamp which the token is not valid before
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "sub"
- o Description: Subject of the token
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "aud"
- o Description: Audience of the token
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "iss"
- o Description: Issuer of the token
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

- o Name: "jti"
- o Description: Unique identifier of the token
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this document]].

4. Security Considerations

Since there are many different and valid ways to implement an OAuth 2.0 system, there are consequently many ways for an authorization server to determine whether or not a token is currently "active" or not. However, since resource servers using token introspection rely on the authorization server to determine the state of a token, the authorization server MUST perform all applicable checks against a token’s state. For instance:

- If the token can expire, the authorization server MUST determine whether or not the token has expired.
- If the token can be issued before it is able to be used, the authorization server MUST determine whether or not a token’s valid period has started yet.
- If the token can be revoked after it was issued, the authorization server MUST determine whether or not such a revocation has taken place.
- If the token has been signed, the authorization server MUST validate the signature.
- If the token can be used only at certain resource servers, the authorization server MUST determine whether or not the token can be used at the resource server making the introspection call.

If an authorization server fails to perform any applicable check, the resource server could make an erroneous security decision based on that response. Note that not all of these checks will be applicable to all OAuth 2.0 deployments and it is up to the authorization server to determine which of these checks (and any other checks) apply.

If left unprotected and un-throttled, the introspection endpoint could present a means for an attacker to poll a series of possible token values, fishing for a valid token. To prevent this, the authorization server MUST require authentication of protected resources that need to access the introspection endpoint and SHOULD require protected resources to be specifically authorized to call the introspection endpoint. The specifics of this authentication credentials are out of scope of this specification, but commonly these credentials could take the form of any valid client authentication mechanism used with the token endpoint, an OAuth 2.0 access token, or other HTTP authorization or authentication mechanism. A single piece of software acting as both a client and a protected resource MAY re-use the same credentials between the token endpoint and the introspection endpoint, though doing so potentially conflates the activities of the client and protected resource portions of the software and the authorization server MAY require separate credentials for each mode.

Since the introspection endpoint takes in OAuth 2.0 tokens as parameters and responds with information used to make authorization decisions, the server MUST support TLS 1.2 RFC 5246 [RFC5246] and MAY support additional transport-layer mechanisms meeting its security requirements. When using TLS, the client or protected resource MUST perform a TLS/SSL server certificate check, as specified in RFC 6125 [RFC6125]. Implementation security considerations can be found in Recommendations for Secure Use of TLS and DTLS [TLS.BCP].

To prevent the values of access tokens from leaking into server-side logs via query parameters, an authorization server offering token introspection MAY disallow the use of HTTP GET on the introspection endpoint and instead require the HTTP POST method to be used at the introspection endpoint.

To avoid disclosing internal server state, an introspection response for an inactive token SHOULD NOT contain any additional claims beyond the required "active" claim (with its value set to "false").

Since a protected resource MAY cache the response of the introspection endpoint, designers of an OAuth 2.0 system using this protocol MUST consider the performance and security trade-offs inherent in caching security information such as this. A less aggressive cache with a short timeout will provide the protected resource with more up to date information (due to it needing to query the introspection endpoint more often) at the cost of increased network traffic and load on the introspection endpoint. A more aggressive cache with a longer duration will minimize network traffic and load on the introspection endpoint, but at the risk of stale information about the token. For example, the token may be revoked while the protected resource is relying on the value of the cached response to make authorization decisions. This creates a window during which a revoked token could be used at the protected resource. Consequently, an acceptable cache validity duration needs to be carefully considered given the concerns and sensitivities of the protected resource being accessed and the likelihood of a token being revoked or invalidated in the interim period. Highly sensitive environments can opt to disable caching entirely on the protected resource to eliminate the risk of stale cached information entirely, again at the cost of increased network traffic and server load. If the response contains the "exp" parameter (expiration), the response MUST NOT be cached beyond the time indicated therein.

An authorization server offering token introspection must be able to understand the token values being presented to it during this call. The exact means by which this happens is an implementation detail and outside the scope of this specification. For unstructured tokens, this could take the form of a simple server-side database query against a data store containing the context information for the token. For structured tokens, this could take the form of the server parsing the token, validating its signature or other protection mechanisms, and returning the information contained in the token back to the protected resource (allowing the protected resource to be unaware of the token’s contents, much like the client). Note that for tokens carrying encrypted information that is needed during the introspection process, the authorization server must be able to decrypt and validate the token to access this information. Also note

that in cases where the authorization server stores no information about the token and has no means of accessing information about the token by parsing the token itself, it can not likely offer an introspection service.

5. Privacy Considerations

The introspection response may contain privacy-sensitive information such as user identifiers for resource owners. When this is the case, measures MUST be taken to prevent disclosure of this information to unintended parties. One method is to transmit user identifiers as opaque service-specific strings, potentially returning different identifiers to each protected resource.

If the protected resource sends additional information about the client’s request to the authorization server (such as the client’s IP address) using an extension of this specification, such information could have additional privacy considerations that the extension should detail. However, the nature and implications of such extensions are outside the scope of this specification.

Omitting privacy-sensitive information from an introspection response is the simplest way of minimizing privacy issues.

6. Acknowledgements

Thanks to the OAuth Working Group and the User Managed Access Working Group for feedback and review of this document, and to the various implementors of both the client and server components of this specification. In particular, the author would like to thank Amanda Anganes, John Bradley, Thomas Broyer, Brian Campbell, George Fletcher, Paul Freemantle, Thomas Hardjono, Eve Maler, Josh Mandel, Steve Moore, Mike Schwartz, Prabath Siriwardena, Sarah Squire, and Hannes Tschofennig.

7. References

7.1. Normative References

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

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

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

[RFC7009] Lodderstedt, T., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, August 2013.

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

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

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

[W3C.REC-html5-20141028]
Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

7.2. Informative References

[TLS.BCP] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of TLS and DTLS", November 2014.

Appendix A. Use with Proof of Posession Tokens

With bearer tokens such as those defined by OAuth 2.0 Bearer Token Usage [RFC6750], the protected resource will have in its possession the entire secret portion of the token for submission to the introspection service. However, for proof-of-possession style tokens, the protected resource will have only a token identifier used during the request, along with the cryptographic signature on the request. The protected resource would be able to submit the token identifier to the authorization server’s token endpoint to obtain the necessary key information needed to validate the signature on the

request. The details of this usage are outside the scope of this specification and will be defined in an extension to this specification.

Appendix B. Document History

[[To be removed by the RFC Editor.]]

-11

- o Minor wording tweaks from IESG review.

-10

- o Added missing 2119 section to terminology.
- o Removed optional HTTP GET at introspection endpoint.
- o Added terminology.
- o Renamed this "a protocol" instead of "web API".
- o Moved JWT to normative reference.
- o Reworded definition of "scope" value.
- o Clarified extensibility of input parameters.
- o Noted that discover is out of scope.
- o Fixed several typos and imprecise references.

-09

- o Updated JOSE, JWT, and OAuth Assertion draft references to final RFC numbers.

-08

- o Added privacy considerations note about extensions.
- o Added acknowledgements (finally).

-07

- o Created a separate IANA registry for introspection responses, importing the values from JWT.

-06

- o Clarified relationship between AS and RS in introduction.
- o Used updated TLS text imported from Dyn-Reg drafts.
- o Clarified definition of active state.
- o Added some advice on caching responses.
- o Added security considerations on active state implementation.
- o Changed user_id to username based on WG feedback.

-05

- o Typo fix.
- o Updated author information.
- o Removed extraneous "linewrap" note from examples.

- 04

- o Removed "resource_id" from request.
- o Added examples.

- 03

- o Updated HTML and HTTP references.
- o Call for registration of parameters in the JWT registry.

- 02

- o Removed SAML pointer.
- o Clarified what an "active" token could be.
- o Explicitly declare introspection request as x-www-form-urlencoded format.
- o Added extended example.
- o Made protected resource authentication a MUST.

- 01

- o Fixed casing and consistent term usage.
- o Incorporated working group comments.
- o Clarified that authorization servers need to be able to understand the token if they’re to introspect it.
- o Various editorial cleanups.

- 00

- o Created initial IETF drafted based on draft-richer-oauth-introspection-06 with no normative changes.

Author’s Address

Justin Richer (editor)

Email: ietf@justin.richer.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 28, 2019

J. Bradley
Ping Identity
P. Hunt
Oracle Corporation
M. Jones
Microsoft
H. Tschofenig
Arm Ltd.
M. Meszaros
GITDA

March 27, 2019

OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution
draft-ietf-oauth-pop-key-distribution-07

Abstract

RFC 6750 specified the bearer token concept for securing access to protected resources. Bearer tokens need to be protected in transit as well as at rest. When a client requests access to a protected resource it hands-over the bearer token to the resource server.

The OAuth 2.0 Proof-of-Possession security concept extends bearer token security and requires the client to demonstrate possession of a key when accessing a protected resource.

Status of This Memo

Table of Contents

1. Introduction

The work on proof-of-possession tokens, an extended token security mechanisms for OAuth 2.0, is motivated in [22]. This document defines the ability for the client request and to obtain PoP tokens from the authorization server. After successfully completing the exchange the client is in possession of a PoP token and the keying material bound to it. Clients that access protected resources then need to demonstrate knowledge of the secret key that is bound to the PoP token.

To best describe the scope of this specification, the OAuth 2.0 protocol exchange sequence is shown in Figure 1. The extension defined in this document piggybacks on the message exchange marked with (C) and (D). To demonstrate possession of the private/secret key to the resource server protocol mechanisms outside the scope of this document are used.

```
+--------+                                +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->|               |
| Client |       (resource, req_cnf)     | Authorization |
|        |                               |     Server    |
|        |<-(D)-- PoP Access Token ------|               |
|        |       (rs_cnf, token_type)    +---------------+
|        |
|        |                               +---------------+
|        |--(E)-- PoP Access Token ----->|               |
|        |   (with proof of private key) |    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 1: Augmented OAuth 2.0 Protocol Flow

In OAuth 2.0 [2] access tokens can be obtained via authorization grants and using refresh tokens. The core OAuth specification defines four authorization grants, see Section 1.3 of [2], and [19] adds an assertion-based authorization grant to that list. The token endpoint, which is described in Section 3.2 of [2], is used with every authorization grant except for the implicit grant type. In the implicit grant type the access token is issued directly.

This specification extends the functionality of the token endpoint, i.e., the protocol exchange between the client and the authorization server, to allow keying material to be bound to an access token. Two types of keying material can be bound to an access token, namely symmetric keys and asymmetric keys. Conveying symmetric keys from the authorization server to the client is described in Section 4.1 and the procedure for dealing with asymmetric keys is described in Section 4.2.

This document describes how the client requests and obtains a PoP access token from the authorization server for use with HTTPS-based

transport. The use of alternative transports, such as Constrained Application Protocol (CoAP), is described in [24].

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [1].

Session Key:

In the context of this specification 'session key' refers to fresh and unique keying material established between the client and the resource server. This session key has a lifetime that corresponds to the lifetime of the access token, is generated by the authorization server and bound to the access token.

This document uses the following abbreviations:

JWA: JSON Web Algorithms[7]

JWT: JSON Web Token[9]

JWS: JSON Web Signature[6]

JWK: JSON Web Key[5]

JWE: JSON Web Encryption[8]

CWT: CBOR Web Token[13]

COSE: CBOR Object Signing and Encryption[14]

3. Processing Instructions

Step (0): As an initial step the client typically determines the resource server it wants to interact with. This may, for example, happen as part of a discovery procedure or via manual configuration.

Step (1): The client starts the OAuth 2.0 protocol interaction based on the selected grant type.

Step (2): When the client interacts with the token endpoint to obtain an access token it MUST use the resource identicator parameter, defined in [16], or the audience parameter, defined in [15], when symmetric PoP tokens are used. For asymmetric PoP tokens the use of resource indicators and audience is optional but

RECOMMENDED. The parameters 'audience' and 'resource' both allow the client to express the location of the target service and the difference between the two is described in [15]. As a summary, 'audience' allows expressing a logical name while 'resource' contains an absolute URI. More details about the 'resource' parameter can be found in [16].

Step (3): The authorization server parses the request from the server and determines the suitable response based on OAuth 2.0 and the PoP token credential procedures.

Note that PoP access tokens may be encoded in a variety of ways:

JWT The access token may be encoded using the JSON Web Token (JWT) format [9]. The proof-of-possession token functionality is described in [10]. A JWT encoded PoP token MUST be protected against modification by either using a digital signature or a keyed message digest, as described in [6]. The JWT may also be encrypted using [8].

CWT [13] defines an alternative token format based on CBOR. The proof-of-possession token functionality is defined in [12]. A CWT encoded PoP token MUST be protected against modification by either using a digital signature or a keyed message digest, as described in [12].

If the access token is only a reference then a look-up by the resource server is needed, as described in the token introspection specification [23].

Note that the OAuth 2.0 framework nor this specification does not mandate a specific PoP token format but using a standardized format will improve interoperability and will lead to better code re-use.

Application layer interactions between the client and the resource server are beyond the scope of this document.

4. Examples

This section provides a number of examples.

4.1. Symmetric Key Transport

4.1.1. Client-to-AS Request

The client starts with a request to the authorization server indicating that it is interested to obtain a token for https://resource.example.com

```
POST /token HTTP/1.1
Host: authz.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&scope=calendar%20contacts
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&resource=https%3A%2F%2Fresource.example.com
```

Example Request to the Authorization Server

4.1.2. Client-to-AS Response

If the access token request has been successfully verified by the authorization server and the client is authorized to obtain a PoP token for the indicated resource server, the authorization server issues an access token and optionally a refresh token.

Figure 2 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key both encoded in a JSON-based serialization format. The "cnf" parameter contains the RFC 7517 [5] encoded key element.

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

{
  "access_token":"SlAV32hkKG ...
   (remainder of JWT omitted for brevity;
   JWT contains JWK in the cnf claim)",
  "token_type":"pop",
  "expires_in":3600,
  "refresh_token":"8xLOxBtZp8",
  "cnf":{
    {"keys":
     [
       {"kty":"oct",
         "alg":"A128KW",
         "k":"GawgguFyGrWKav7AX4VKUg"
               }
        ]
      }
    }
}
```

Figure 2: Example: Response from the Authorization Server (Symmetric Variant)

Note that the cnf payload in Figure 2 is not encrypted at the application layer since Transport Layer Security is used between the AS and the client and the content of the cnf payload is consumed by the client itself. Alternatively, a JWE could be used to encrypt the key distribution, as shown in Figure 3.

```
{
  "access_token":"SlAV32hkKG ...
   (remainder of JWT omitted for brevity;
   JWT contains JWK in the cnf claim)",
  "token_type":"pop",
  "expires_in":3600,
  "refresh_token":"8xLOxBtZp8",
  "cnf":{
      "jwe":
        "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhDQkMtSFMyNTYifQ.
        (remainder of JWE omitted for brevity)"
      }
      }
}
```


Figure 3: Example: Encrypted Symmmetric Key

The content of the 'access_token' in JWT format contains the 'cnf' (confirmation) claim. The confirmation claim is defined in [10]. The digital signature or the keyed message digest offering integrity protection is not shown in this example but has to be present in a real deployment to mitigate a number of security threats.

The JWK in the key element of the response from the authorization server, as shown in Figure 2, contains the same session key as the JWK inside the access token, as shown in Figure 4. It is, in this example, protected by TLS and transmitted from the authorization server to the client (for processing by the client).


```
{
   "iss": "https://server.example.com",
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "exp": 1311281970,
   "iat": 1311280970,
   "cnf":{
     "jwe":
       "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhDQkMtSFMyNTYifQ.
       (remainder of JWE omitted for brevity)"
     }
}
```


Figure 4: Example: Access Token in JWT Format

Note: When the JWK inside the access token contains a symmetric key it must be confidentiality protected using a JWE to maintain the security goals of the PoP architecture since content is meant for consumption by the selected resource server only. The details are described in [22].

4.2. Asymmetric Key Transport

4.2.1. Client-to-AS Request

This example illustrates the case where an asymmetric key shall be bound to an access token. The client makes the following HTTPS request shown in Figure 5. Extra line breaks are for display purposes only.

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=pop
&req_cnf=eyJhbGciOiJSU0ExXzUi ...
(remainder of JWK omitted for brevity)
```

Figure 5: Example Request to the Authorization Server (Asymmetric Key Variant)

As shown in Figure 6 the content of the 'req_cnf' parameter contains the ECC public key the client would like to associate with the access token (in JSON format).


```
"jwk":{
  "kty": "EC",
  "use": "sig",
  "crv": "P-256",
  "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
  "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
 }
```


Figure 6: Client Providing Public Key to Authorization Server

4.2.2. Client-to-AS Response

If the access token request is valid and authorized, the authorization server issues an access token and optionally a refresh token. The authorization server also places information about the public key used by the client into the access token to create the binding between the two. The new token type "pop" is placed into the 'token_type' parameter.

An example of a successful response is shown in Figure 7.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFE....jr1zCsicMWpAA",
  "token_type":"pop",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Figure 7: Example: Response from the Authorization Server (Asymmetric Variant)

The content of the 'access_token' field contains an encoded JWT, as shown in Figure 8. The digital signature covering the access token offering authenticity and integrity protection is not shown below (but must be present).

```
{
  "iss":"https://authz.example.com",
  "aud":"https://resource.example.com",
  "exp":"1361398824",
  "nbf":"1360189224",
  "cnf":{
     "jwk" : {
       "kty" : "EC",
       "crv" : "P-256",
       "x" : "usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8",
       "y" : "IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4"
     }
   }
 }
```


Figure 8: Example: Access Token Structure (Asymmetric Variant)

Note: In this example there is no need for the authorization server to convey further keying material to the client since the client is already in possession of the private key (as well as the public key).

5. Security Considerations

[22] describes the architecture for the OAuth 2.0 proof-of-possession security architecture, including use cases, threats, and requirements. This requirements describes one solution component of that architecture, namely the mechanism for the client to interact with the authorization server to either obtain a symmetric key from the authorization server, to obtain an asymmetric key pair, or to offer a public key to the authorization. In any case, these keys are then bound to the access token by the authorization server.

To summarize the main security recommendations: A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest. Consequently, the token integrity protection MUST be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key. If the access token contains the symmetric key (see Section 2.2 of [10] for a description about how symmetric keys can be securely conveyed within the access token) this symmetric key MUST be encrypted by the authorization server with a long-term key shared with the resource server.

To deal with token redirect, it is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. Using a single shared secret with multiple

authorization server to simplify key management is NOT RECOMMENDED since the benefit from using the proof-of-possession concept is significantly reduced.

Token replay is also not possible since an eavesdropper will also have to obtain the corresponding private key or shared secret that is bound to the access token. Nevertheless, it is good practice to limit the lifetime of the access token and therefore the lifetime of associated key.

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client will obtain the session key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby making the OAuth 2.0 proof-of-possession security model completely insecure. OAuth 2.0 [2] relies on TLS to offer confidentiality protection and additional protection can be applied using the JWK [5] offered security mechanism, which would add an additional layer of protection on top of TLS for cases where the keying material is conveyed, for example, to a hardware security module. Which version(s) of TLS ought to be implemented will vary over time, and depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [4] is the most recent version. The client MUST validate the TLS certificate chain when making requests to protected resources, including checking the validity of the certificate.

Similarly to the security recommendations for the bearer token specification [17] developers MUST ensure that the ephemeral credentials (i.e., the private key or the session key) is not leaked to third parties. An adversary in possession of the ephemeral credentials bound to the access token will be able to impersonate the client. Be aware that this is a real risk with many smart phone app and Web development environments.

Clients can at any time request a new proof-of-possession capable access token. Using a refresh token to regularly request new access tokens that are bound to fresh and unique keys is important. Keeping the lifetime of the access token short allows the authorization server to use shorter key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens then they SHOULD scope these access tokens to a specific permissions.

6. IANA Considerations

6.1. OAuth Access Token Types

This specification registers the following error in the IANA "OAuth Access Token Types" [25] established by [17].

- o Name: pop
- o Change controller: IESG
- o Specification document(s): [[this specification]]

6.2. OAuth Parameters Registration

This specification registers the following value in the IANA "OAuth Parameters" registry [25] established by [2].

- o Parameter name: cnf_req
- o Parameter usage location: authorization request, token request
- o Change controller: IESG
- o Specification document(s): [[this specification]]

- o Parameter name: cnf
- o Parameter usage location: authorization response, token response
- o Change controller: IESG
- o Specification document(s): [[this specification]]

- o Parameter name: rs_cnf
- o Parameter usage location: token response
- o Change controller: IESG
- o Specification document(s): [[this specification]]

6.3. OAuth Extensions Error Registration

This specification registers the following error in the IANA "OAuth Extensions Error Registry" [25] established by [2].

- o Error name: invalid_token_type
- o Error usage location: implicit grant error response, token error response
- o Related protocol extension: token_type parameter
- o Change controller: IESG
- o Specification document(s): [[this specification]]

7. Acknowledgements

We would like to thank Chuck Mortimore and James Manger for their review comments.

8. References

8.1. Normative References

[1] 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>.

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

[3] 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>.

[4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[5] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

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

[7] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[8] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[9] 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>.

[10] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[11] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/info/rfc7638>.

[12] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-06 (work in progress), February 2019.

[13] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[14] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[15] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", draft-ietf-oauth-token-exchange-16 (work in progress), October 2018.

[16] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", draft-ietf-oauth-resource-indicators-02 (work in progress), January 2019.

8.2. Informative References

[17] 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>.

[18] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[19] 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>.

[20] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/info/rfc7636>.

[21] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[22] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

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

[24] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

[25] IANA, "OAuth Parameters", October 2018.

[26] IANA, "JSON Web Token Claims", June 2018.

Authors' Addresses

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Phil Hunt
Oracle Corporation

Email: phil.hunt@yahoo.com
URI: http://www.indepdentid.com

Michael B. Jones
Microsoft

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

Hannes Tschofenig
Arm Ltd.
Absam 6067
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Mihaly Meszaros
GITDA
Debrecen 4033
Hungary

Email: bakfitty@gmail.com
URI: https://github.com/misi

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 20, 2016

M. Jones
Microsoft
J. Bradley
Ping Identity
H. Tschofenig
ARM Limited

December 18, 2015

Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)
draft-ietf-oauth-proof-of-possession-11

Abstract

This specification defines how to declare in a JSON Web Token (JWT) that the presenter of the JWT possesses a particular proof-of-possession key and that the recipient can cryptographically confirm proof-of-possession of the key by the presenter. Being able to prove possession of a key is also sometimes described as the presenter being a holder-of-key.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This specification defines how a JSON Web Token [JWT] can declare that the presenter of the JWT possesses a particular proof-of-possession (PoP) key and that the recipient can cryptographically confirm proof-of-possession of the key by the presenter. Proof-of-possession of a key is also sometimes described as the presenter being a holder-of-key. The [I-D.ietf-oauth-pop-architecture] specification describes key confirmation, among other confirmation mechanisms. This specification defines how to communicate key confirmation key information in JWTs.

Envision the following two use cases. The first use case employs a symmetric proof-of-possession key and the second use case employs an asymmetric proof-of-possession key.

```
  +--------------+
  |              |                          +--------------+
  |              |--(3) Presentation of -->|              |
  |              |      JWT w/ Encrypted   |              |
  |  Presenter   |      PoP Key            |              |
  |              |                         |              |
  |              |<-(4) Communication ---->|              |
  |              |      Authenticated by   |              |
  +--------------+      PoP Key            |              |
    ^          ^                          |              |
    |          |                          |              |
   (1) Sym.   (2) JWT w/                  |  Recipient   |
    |  PoP     |  Encrypted               |              |
    |  Key     |  PoP Key                 |              |
    v          |                          |              |
  +--------------+                        |              |
  |              |                         |              |
  |              |                         |              |
  |              |<-(0) Key Exchange for ->|              |
  |   Issuer     |      Key Encryption Key |              |
  |              |                         |              |
  |              |                         |              |
  |              |                         +--------------+
  +--------------+
```

Figure 1: Proof-of-Possession with a Symmetric Key

In the case illustrated in Figure 1, either the presenter generates a symmetric key and privately sends it to the issuer (1) or the issuer generates a symmetric key and privately sends it to the presenter (1). The issuer generates a JWT with an encrypted copy of this symmetric key in the confirmation claim. This symmetric key is

encrypted with a key known only to the issuer and the recipient, which was previously established in step (0). The entire JWT is integrity protected by the issuer. The JWT is then (2) sent to the presenter. Now, the presenter is in possession of the symmetric key as well as the JWT (which includes the confirmation claim). When the presenter (3) presents the JWT to the recipient, it also needs to demonstrate possession of the symmetric key; the presenter, for example, (4) uses the symmetric key in a challenge/response protocol with the recipient. The recipient is then able to verify that it is interacting with the genuine presenter by decrypting the key in the confirmation claim of the JWT. By doing this, the recipient obtains the symmetric key, which it then uses to verify cryptographically protected messages exchanged with the presenter (4). This symmetric key mechanism described above is conceptually similar to the use of Kerberos tickets.

Note that for simplicity, the diagram above and associated text describe the direct use of symmetric keys without the use of derived keys. A more secure practice is to derive the symmetric keys actually used from secrets exchanged, such as the key exchanged in step (0), using a Key Derivation Function (KDF) and use the derived keys, rather than directly using the secrets exchanged.

```
+--------------+
|              |                          +--------------+
|              |--(3) Presentation of -->|              |
|              |      JWT w/ Public       |              |
|  Presenter   |      PoP Key             |              |
|              |                          |              |
|              |<-(4) Communication ---->|              |
|              |      Authenticated by    |              |
+--------------+      PoP Key             |              |
  |          ^                            |              |
  |          |                            |              |
 (1) Public (2) JWT w/                    |  Recipient   |
  |  PoP     |  Public                   |              |
  |  Key     |  PoP Key                  |              |
  v          |                           |              |
+--------------+                          |              |
|              |                          |              |
|              |                          |              |
|              |                          |              |
|    Issuer    |                          |              |
|              |                          |              |
|              |                          |              |
|              |                          +--------------+
+--------------+
```

Figure 2: Proof-of-Possession with an Asymmetric Key

In the case illustrated in Figure 2, the presenter generates a public/private key pair and (1) sends the public key to the issuer, which creates a JWT that contains the public key (or an identifier for it) in the confirmation claim. The entire JWT is integrity protected using a digital signature to protect it against modifications. The JWT is then (2) sent to the presenter. When the presenter (3) presents the JWT to the recipient, it also needs to demonstrate possession of the private key. The presenter, for example, (4) uses the private key in a TLS exchange with the recipient or (4) signs a nonce with the private key. The recipient is able to verify that it is interacting with the genuine presenter by extracting the public key from the confirmation claim of the JWT (after verifying the digital signature of the JWT) and utilizing it with the private key in the TLS exchange or by checking the nonce signature.

In both cases, the JWT may contain other claims that are needed by the application.

1.1. 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 RFC 2119 [RFC2119].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

2. Terminology

This specification uses terms defined in the JSON Web Token [JWT], JSON Web Key [JWK], and JSON Web Encryption [JWE] specifications.

These terms are defined by this specification:

Issuer
: Party that creates the JWT and binds the proof-of-possession key to it.

Presenter
: Party that proves possession of a private key (for asymmetric key cryptography) or secret key (for symmetric key cryptography) to a recipient.

Recipient
Party that receives the JWT containing the proof-of-possession key information from the presenter.

3. Representations for Proof-of-Possession Keys

By including a "cnf" (confirmation) claim in a JWT, the issuer of the JWT declares that the presenter possesses a particular key, and that the recipient can cryptographically confirm that the presenter has possession of that key. The value of the "cnf" claim is a JSON object and the members of that object identify the proof-of-possession key.

The presenter can be identified in one of several ways by the JWT, depending upon the application requirements. If the JWT contains a "sub" (subject) claim [JWT], the presenter is normally the subject identified by the JWT. (In some applications, the subject identifier will be relative to the issuer identified by the "iss" (issuer) claim [JWT].) If the JWT contains no "sub" (subject) claim, the presenter is normally the issuer identified by the JWT using the "iss" (issuer) claim. The case in which the presenter is the subject of the JWT is analogous to SAML 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation usage. At least one of the "sub" and "iss" claims MUST be present in the JWT. Some use cases may require that both be present.

Another means used by some applications to identify the presenter is an explicit claim, such as the "azp" (authorized party) claim defined by OpenID Connect [OpenID.Core]. Ultimately, the means of identifying the presenter is application-specific, as is the means of confirming possession of the key that is communicated.

3.1. Confirmation Claim

The "cnf" (confirmation) claim is used in the JWT to contain members used to identify the proof-of-possession key. Other members of the "cnf" object may be defined because a proof-of-possession key may not be the only means of confirming the authenticity of the token. This is analogous to the SAML 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation element, in which a number of different subject confirmation methods can be included, including proof-of-possession key information.

The set of confirmation members that a JWT must contain to be considered valid is context dependent and is outside the scope of this specification. Specific applications of JWTs will require implementations to understand and process some confirmation members in particular ways. However, in the absence of such requirements,

all confirmation members that are not understood by implementations MUST be ignored.

This specification establishes the IANA "JWT Confirmation Methods" registry for these members in Section 6.2 and registers the members defined by this specification. Other specifications can register other members used for confirmation, including other members for conveying proof-of-possession keys, possibly using different key representations.

The "cnf" claim value MUST represent only a single proof-of-possession key; thus, at most one of the "jwk", "jwe", and "jku" confirmation values defined below may be present. Note that if an application needs to represent multiple proof-of-possession keys in the same JWT, one way for it to achieve this is to use other claim names, in addition to "cnf", to hold the additional proof-of-possession key information. These claims could use the same syntax and semantics as the "cnf" claim. Those claims would be defined by applications or other specifications and could be registered in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims].

3.2. Representation of an Asymmetric Proof-of-Possession Key

When the key held by the presenter is an asymmetric private key, the "jwk" member is a JSON Web Key [JWK] representing the corresponding asymmetric public key. The following example demonstrates such a declaration in the JWT Claims Set of a JWT:


```
{
 "iss": "https://server.example.com",
 "aud": "https://client.example.org",
 "exp": 1361398824,
 "cnf":{
   "jwk":{
     "kty": "EC",
     "use": "sig",
     "crv": "P-256",
     "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
     "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
    }
  }
}
```


The JWK MUST contain the required key members for a JWK of that key type and MAY contain other JWK members, including the "kid" (key ID) member.

The "jwk" member MAY also be used for a JWK representing a symmetric

key, provided that the JWT is encrypted so that the key is not revealed to unintended parties. If the JWT is not encrypted, the symmetric key MUST be encrypted as described below.

3.3. Representation of an Encrypted Symmetric Proof-of-Possession Key

When the key held by the presenter is a symmetric key, the "jwe" member is an encrypted JSON Web Key [JWK] encrypted to a key known to the recipient using the JWE Compact Serialization containing the symmetric key. The rules for encrypting a JWK are found in Section 7 of the JSON Web Key [JWK] specification.

The following example illustrates a symmetric key that could subsequently be encrypted for use in the "jwe" member:

```
{
 "kty": "oct",
 "alg": "HS256",
 "k": "ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE"
}
```

The UTF-8 [RFC3629] encoding of this JWK is used as the JWE Plaintext when encrypting the key.

The following example is a JWE Header that could be used when encrypting this key:

```
{
 "alg": "RSA-OAEP",
 "enc": "A128CBC-HS256"
}
```

The following example JWT Claims Set of a JWT illustrates the use of an encrypted symmetric key as the "jwe" member value:

```
{
 "iss": "https://server.example.com",
 "sub": "24400320",
 "aud": "s6BhdRkqt3",
 "nonce": "n-0S6_WzA2Mj",
 "exp": 1311281970,
 "iat": 1311280970,
 "cnf":{
   "jwe":
     "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhDQkMtSFMyNTYifQ.
     (remainder of JWE omitted for brevity)"
   }
}
```

3.4. Representation of a Key ID for a Proof-of-Possession Key

The proof-of-possession key can also be identified by the use of a Key ID instead of communicating the actual key, provided the recipient is able to obtain the identified key using the Key ID. In this case, the issuer of a JWT declares that the presenter possesses a particular key and that the recipient can cryptographically confirm proof-of-possession of the key by the presenter by including a "cnf" (confirmation) claim in the JWT whose value is a JSON object, with the JSON object containing a "kid" (key ID) member identifying the key.

The following example demonstrates such a declaration in the JWT Claims Set of a JWT:

```
{
 "iss": "https://server.example.com",
 "aud": "https://client.example.org",
 "exp": 1361398824,
 "cnf":{
   "kid": "dfd1aa97-6d8d-4575-a0fe-34b96de2bfad"
  }
}
```

The content of the "kid" value is application specific. For instance, some applications may choose to use a JWK Thumbprint [JWK.Thumbprint] value as the "kid" value.

3.5. Representation of a URL for a Proof-of-Possession Key

The proof-of-possession key can be passed by reference instead of being passed by value. This is done using the "jku" (JWK Set URL) member. Its value is a URI [RFC3986] that refers to a resource for a set of JSON-encoded public keys represented as a JWK Set [JWK], one of which is the proof-of-possession key. If there are multiple keys in the referenced JWK Set document, a "kid" member MUST also be included, with the referenced key’s JWK also containing the same "kid" value.

The protocol used to acquire the resource MUST provide integrity protection. An HTTP GET request to retrieve the JWK Set MUST use Transport Layer Security (TLS) [RFC5246] and the identity of the server MUST be validated, as per Section 6 of RFC 6125 [RFC6125].

The following example demonstrates such a declaration in the JWT Claims Set of a JWT:

```
{
 "iss": "https://server.example.com",
 "sub": "17760704",
 "aud": "https://client.example.org",
 "exp": 1440804813,
 "cnf":{
   "jku": "https://keys.example.net/pop-keys.json",
   "kid": "2015-08-28"
  }
}
```

3.6. Specifics Intentionally Not Specified

Proof-of-possession is typically demonstrated by having the presenter sign a value determined by the recipient using the key possessed by the presenter. This value is sometimes called a "nonce" or a "challenge".

The means of communicating the nonce and the nature of its contents are intentionally not described in this specification, as different protocols will communicate this information in different ways. Likewise, the means of communicating the signed nonce is also not specified, as this is also protocol-specific.

Note that another means of proving possession of the key when it is a symmetric key is to encrypt the key to the recipient. The means of obtaining a key for the recipient is likewise protocol-specific.

For examples using the mechanisms defined in this specification, see [I-D.ietf-oauth-pop-architecture].

4. Security Considerations

All of the security considerations that are discussed in [JWT] also apply here. In addition, proof-of-possession introduces its own unique security issues. Possessing a key is only valuable if it is kept secret. Appropriate means must be used to ensure that unintended parties do not learn private key or symmetric key values.

Applications utilizing proof-of-possession should also utilize audience restriction, as described in Section 4.1.3 of [JWT], as it provides different protections. Proof-of-possession can be used by recipients to reject messages from unauthorized senders. Audience restriction can be used by recipients to reject messages intended for different recipients.

A recipient might not understand the "cnf" claim. Applications that

require the proof-of-possession keys communicated with it to be understood and processed must ensure that the parts of this specification that they use are implemented.

Proof-of-possession via encrypted symmetric secrets is subject to replay attacks. This attack can be avoided when a signed nonce or challenge is used, since the recipient can use a distinct nonce or challenge for each interaction. Replay can also be avoided if a sub-key is derived from a shared secret that is specific to the instance of the PoP demonstration.

Similarly to other information included in a JWT, it is necessary to apply data origin authentication and integrity protection (via a keyed message digest or a digital signature). Data origin authentication ensures that the recipient of the JWT learns about the entity that created the JWT, since this will be important for any policy decisions. Integrity protection prevents an adversary from changing any elements conveyed within the JWT payload. Special care has to be applied when carrying symmetric keys inside the JWT, since those not only require integrity protection, but also confidentiality protection.

5. Privacy Considerations

A proof-of-possession key can be used as a correlation handle if the same key is used with multiple parties. Thus, for privacy reasons, it is recommended that different proof-of-possession keys be used when interacting with different parties.

6. IANA Considerations

The following registration procedure is used for all the registries established by this specification.

Values are registered on a Specification Required [RFC5226] basis after a three-week review period on the oauth-pop-reg-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published. [[Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: oauth-pop-reg-review@ietf.org.]]

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register JWT Confirmation

Method: example"). Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@ietf.org mailing list) for resolution.

Criteria that should be applied by the Designated Experts include determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, evaluating the security properties of the item being registered, and whether the registration makes sense.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly-informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

6.1. JSON Web Token Claims Registration

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

6.1.1. Registry Contents

- o Claim Name: "cnf"
- o Claim Description: Confirmation
- o Change Controller: IESG
- o Specification Document(s): Section 3.1 of [[this document]]

6.2. JWT Confirmation Methods Registry

This specification establishes the IANA "JWT Confirmation Methods" registry for JWT "cnf" member values. The registry records the confirmation method member and a reference to the specification that defines it.

6.2.1. Registration Template

Confirmation Method Value:
The name requested (e.g., "kid"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case-sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Confirmation Method Description:
Brief description of the confirmation method (e.g., "Key Identifier").

Change Controller:
For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

6.2.2. Initial Registry Contents

- Confirmation Method Value: "jwk"
- Confirmation Method Description: JSON Web Key Representing Public Key
- Change Controller: IESG
- Specification Document(s): Section 3.2 of [[this document]]

- Confirmation Method Value: "jwe"
- Confirmation Method Description: Encrypted JSON Web Key
- Change Controller: IESG
- Specification Document(s): Section 3.3 of [[this document]]

- Confirmation Method Value: "kid"
- Confirmation Method Description: Key Identifier
- Change Controller: IESG
- Specification Document(s): Section 3.4 of [[this document]]

- Confirmation Method Value: "jku"
- Confirmation Method Description: JWK Set URL
- Change Controller: IESG
- Specification Document(s): Section 3.5 of [[this document]]

7. References

7.1. Normative References

[IANA.JWT.Claims]
IANA, "JSON Web Token Claims", <http://www.iana.org/assignments/jwt>.

[JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",

RFC 7516, DOI 10.17487/RFC7156, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7157, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7159, May 2015, <http://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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <http://www.rfc-editor.org/info/rfc6125>.

7.2. Informative References

[I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-05 (work

in progress), October 2015.

[JWK.Thumbprint]
Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <http://www.rfc-editor.org/info/rfc7638>.

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

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

Appendix A. Acknowledgements

The authors wish to thank Brian Campbell, Stephen Farrell, Barry Leiba, Kepeng Li, Chris Lonvick, James Manger, Kathleen Moriarty, Justin Richer, and Nat Sakimura for their reviews of the specification.

Appendix B. Document History

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

-11

- o Addressed Sec-Dir review comments by Chris Lonvick and ballot comments by Stephen Farrell.

-10

- o Addressed ballot comments by Barry Leiba.

-09

- o Removed erroneous quotation marks around numeric "exp" claim values in examples.

-08

- o Added security consideration about also utilizing audience restriction.

-07

- o Addressed review comments by Hannes Tschofenig, Kathleen Moriarty, and Justin Richer. Changes were:

- o Clarified that symmetric proof-of-possession keys can be generated by either the presenter or the issuer.

- o Clarified that confirmation members that are not understood must be ignored unless otherwise specified by the application.

-06

- o Added diagrams to the introduction.

-05

- o Addressed review comments by Kepeng Li.

-04

- o Allowed the use of "jwk" for symmetric keys when the JWT is encrypted.

- o Added the "jku" (JWK Set URL) member.

- o Added privacy considerations.

- o Reordered sections so that the "cnf" (confirmation) claim is defined before it is used.

- o Noted that applications can define new claim names, in addition to "cnf", to represent additional proof-of-possession keys, using the same representation as "cnf".

- o Applied wording clarifications suggested by Nat Sakimura.

-03

- o Separated the "jwk" and "jwe" confirmation members; the former represents a public key as a JWK and the latter represents a symmetric key as a JWE encrypted JWK.

- o Changed the title to indicate that a proof-of-possession key is being communicated.

- o Updated language that formerly assumed that the issuer was an OAuth 2.0 authorization server.

- o Described ways that applications can choose to identify the presenter, including use of the "iss", "sub", and "azp" claims.

- o Harmonized the registry language with that used in JWT [RFC 7519].

- o Addressed other issues identified during working group last call.

- o Referenced the JWT and JOSE RFCs.

-02

- o Defined the terms Issuer, Presenter, and Recipient and updated their usage within the document.

- o Added a description of a use case using an asymmetric proof-of-possession key to the introduction.

- o Added the "kid" (key ID) confirmation method.

- o These changes address the open issues identified in the previous draft.

-01

- o Updated references.

-00

- o Created the initial working group draft from draft-jones-oauth-proof-of-possession-02.

Authors' Addresses

Michael B. Jones
Microsoft

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

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 9, 2017

J. Richer, Ed.

J. Bradley
Ping Identity
H. Tschofenig
ARM Limited

August 08, 2016

A Method for Signing HTTP Requests for OAuth
draft-ietf-oauth-signed-http-request-03

Abstract

This document a method for offering data origin authentication and integrity protection of HTTP requests. To convey the relevant data items in the request a JSON-based encapsulation is used and the JSON Web Signature (JWS) technique is re-used. JWS offers integrity protection using symmetric as well as asymmetric cryptography.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In order to prove possession of an access token and its associated key, an OAuth 2.0 client needs to compute some cryptographic function and present the results to the protected resource as a signature. The protected resource then needs to verify the signature and compare that to the expected keys associated with the access token. This is in addition to the normal token protections provided by a bearer token [RFC6750] and transport layer security (TLS).

Furthermore, it is desirable to bind the signature to the HTTP request. Ideally, this should be done without replicating the information already present in the HTTP request more than required. However, many HTTP application frameworks insert extra headers, query

parameters, and otherwise manipulate the HTTP request on its way from the web server into the application code itself. It is the goal of this draft to have a signature protection mechanism that is sufficiently robust against such deployment constraints while still providing sufficient security benefits.

The key required for this signature calculation is distributed via mechanisms described in companion documents (see [I-D.ietf-oauth-pop-key-distribution] and [I-D.ietf-oauth-pop-architecture]). The JSON Web Signature (JWS) specification [RFC7515] is used for computing a digital signature (which uses asymmetric cryptography) or a keyed message digest (in case of symmetric cryptography).

The mechanism described in this document assumes that a client is in possession of an access token and asociated key. That client then creates a JSON object including the access token, signs the JSON object using JWS, and issues an request to a resource server for access to a protected resource using the signed object as its authorization. The protected resource validates the JWS signature and parses the JSON object to obtain token information.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Other terms such as "client", "authorization server", "access token", and "protected resource" are inherited from OAuth 2.0 [RFC6749].

We use the term ’sign’ (or ’signature’) to denote both a keyed message digest and a digital signature operation.

3. Generating a JSON Object from an HTTP Request

This specification uses JSON Web Signatures [RFC7515] to protect the access token and, optionally, parts of the request.

This section describes how to generate a JSON [RFC7159] object from the HTTP request. Each value below is included as a member of the JSON object at the top level.

at REQUIRED. The access token value. This string is assumed to have no particular format or structure and remains opaque to the client.

ts RECOMMENDED. The timestamp. This integer provides replay protection of the signed JSON object. Its value MUST be a number containing an integer value representing number of whole integer seconds from midnight, January 1, 1970 GMT.

m OPTIONAL. The HTTP Method used to make this request. This MUST be the uppercase HTTP verb as a JSON string.

u OPTIONAL. The HTTP URL host component as a JSON string. This MAY include the port separated from the host by a colon in host:port format.

p OPTIONAL. The HTTP URL path component of the request as an HTTP string.

q OPTIONAL. The hashed HTTP URL query parameter map of the request as a two-part JSON array. The first part of this array is a JSON array listing all query parameters that were used in the calculation of the hash in the order that they were added to the hashed value as described below. The second part of this array is a JSON string containing the Base64URL encoded hash itself, calculated as described below.

h OPTIONAL. The hashed HTTP request headers as a two-part JSON array. The first part of this array is a JSON array listing all headers that were used in the calculation of the hash in the order that they were added to the hashed value as described below. The second part of this array is a JSON string containing the Base64URL encoded hash itself, calculated as described below.

b OPTIONAL. The base64URL encoded hash of the HTTP Request body, calculated as the SHA256 of the byte array of the body

All hashes SHALL be calculated using the SHA256 algorithm. [[Note to WG: do we want crypto agility here? If so how do we signal this]]

The JSON object is signed using the algorithm appropriate to the associated access token key, usually communicated as part of key distribution [I-D.ietf-oauth-pop-key-distribution].

3.1. Calculating the query parameter list and hash

To generate the query parameter list and hash, the client creates two data objects: an ordered list of strings to hold the query parameter names and a string buffer to hold the data to be hashed.

The client iterates through all query parameters in whatever order it chooses and for each query parameter it does the following:

1. Adds the name of the query parameter to the end of the list.

2. Percent-encodes the name and value of the parameter as specified in [RFC3986]. Note that if the name and value have already been percent-encoded for transit, they are not re-encoded for this step.

3. Encodes the name and value of the query parameter as "name=value" and appends it to the string buffer separated by the ampersand "&" character.

Repeated parameter names are processed separately with no special handling. Parameters MAY be skipped by the client if they are not required (or desired) to be covered by the signature.

The client then calculates the hash over the resulting string buffer. The list and the hash result are added to a list as the value of the "q" member.

For example, the query parameter set of "b=bar", "a=foo", "c=duck" is concatenated into the string:

b=bar&a=foo&c=duck

When added to the JSON structure using this process, the results are:

"q": [["b", "a", "c"], "u4LgkGUWhP9MsKrEjA4dizIllDXluDku6ZqCeyuR-JY"]

3.2. Calculating the header list and hash

To generate the header list and hash, the client creates two data objects: an ordered list of strings to hold the header names and a string buffer to hold the data to be hashed.

The client iterates through all query parameters in whatever order it chooses and for each query parameter it does the following:

1. Lowercases the header name.

2. Adds the name of the header to the end of the list.

3. Encodes the name and value of the header as "name: value" and appends it to the string buffer separated by a newline "\n" character.

Repeated header names are processed separately with no special handling. Headers MAY be skipped by the client if they are not required (or desired) to be covered by the signature.

The client then calculates the hash over the resulting string buffer. The list and the hash result are added to a list as the value of the "h" member.

For example, the headers "Content-Type: application/json" and "Etag: 742-3u8f34-3r2nvv3" are concatenated into the string:

```
content-type: application/json
etag: 742-3u8f34-3r2nvv3
```

```
"h": [["content-type", "etag"],
  "bZA981YJBrPlIzOvplbu3e7ueREXXr38vSkxIBYOaxI"]
```

4. Sending the signed object

In order to send the signed object to the protected resource, the client includes it in one of the following three places.

4.1. HTTP Authorization header

The client SHOULD send the signed object to the protected resource in the Authorization header. The value of the signed object in JWS compact form is appended to the Authorization header as a PoP value. This is the preferred method. Note that if this method is used, the Authorization header MUST NOT be included in the protected elements of the signed object.

```
GET /resource/foo
Authorization: PoP eyJ....omitted for brevity...
```

4.2. HTTP Form body

If the client is sending the request as a form-encoded HTTP message with parameters in the body, the client MAY send the signed object as part of that form body. The value of the signed object in JWS compact form is sent as the form parameter pop_access_token. Note that if this method is used, the body hash cannot be included in the protected elements of the signed object.

```
POST /resource
Content-type: application/www-form-encoded

pop_access_token=eyJ....omitted for brevity...
```

4.3. HTTP Query parameter

If neither the Authorization header nor the form-encoded body parameter are available to the client, the client MAY send the signed object as a query parameter. The value of the signed object in JWS compact form is sent as the query parameter pop_access_token. Note that if this method is used, the pop_access_token parameter MUST NOT be included in the protected elements of the signed object.

```
GET /resource?pop_access_token=eyJ....
```

5. Validating the request

Just like with a bearer token [RFC6750], while the access token value included in the signed object is opaque to the client, it MUST be understood by the protected resource in order to fulfill the request. Also like a bearer token, the protected resource traditionally has several methods at its disposal for understanding the access token. It can look up the token locally (such as in a database), it can parse a structured token (such as JWT [RFC7519]), or it can use a service to look up token information (such as introspection [RFC7662]). Whatever method is used to look up token information, the protected resource MUST have access to the key associated with the access token, as this key is required to validate the signature of the incoming request. Validation of the signature is done using normal JWS validation for the signature and key type.

Additionally, in order to trust any of the hashed components of the HTTP request, the protected resource MUST re-create and verify a hash for each component as described below. This process is a mirror of the process used to create the hashes in the first place, with a mind toward the fact that order may have changed and that elements may have been added or deleted. The protected resource MUST similarly compare the replicated values included in various JSON fields with the corresponding actual values from the request. Failure to do so will allow an attacker to modify the underlying request while at the same time having the application layer verify the signature correctly.

5.1. Validating the query parameter list and hash

The client has at its disposal a map that indexes the query parameter names to the values given. The client creates a string buffer for calculating the hash. The client then iterates through the "list" portion of the "p" parameter. For each item in the list (in the order of the list) it does the following:

1. Fetch the value of the parameter from the HTTP request query parameter map. If a parameter is found in the list of signed parameters but not in the map, the validation fails.

2. Percent-encodes the name and value of the parameter as specified in [RFC3986]. Note that if the name and value have already been percent-encoded for transit, they are not re-encoded for this step.

3. Encode the parameter as "name=value" and concatenate it to the end of the string buffer, separated by an ampersand character.

The client calculates the hash of the string buffer and base64url encodes it. The protected resource compares that string to the string passed in as the hash. If the two match, the hash validates, and all named parameters and their values are considered covered by the signature.

There MAY be additional query parameters that are not listed in the list and are therefore not covered by the signature. The client MUST decide whether or not to accept a request with these uncovered parameters.

5.2. Validating the header list and hash

The client has at its disposal a map that indexes the header names to the values given. The client creates a string buffer for calculating the hash. The client then iterates through the "list" portion of the "h" parameter. For each item in the list (in the order of the list) it does the following:

1. Fetch the value of the header from the HTTP request header map. If a header is found in the list of signed parameters but not in the map, the validation fails.

2. Encode the parameter as "name: value" and concatenate it to the end of the string buffer, separated by a newline character.

The client calculates the hash of the string buffer and base64url encodes it. The protected resource compares that string to the string passed in as the hash. If the two match, the hash validates, and all named headers and their values are considered covered by the signature.

There MAY be additional headers that are not listed in the list and are therefore not covered by the signature. The client MUST decide whether or not to accept a request with these uncovered headers.

6. IANA Considerations

6.1. The 'pop' OAuth Access Token Type

Section 11.1 of [RFC6749] defines the OAuth Access Token Type Registry and this document adds another token type to this registry.

Type name: pop

Additional Token Endpoint Response Parameters: (none)

HTTP Authentication Scheme(s): Proof-of-possession access token for use with OAuth 2.0

Change controller: IETF

Specification document(s): [[this document]]

6.2. JSON Web Signature and Encryption Type Values Registration

This specification registers the "pop" type value in the IANA JSON Web Signature and Encryption Type Values registry [RFC7515]:

- "typ" Header Parameter Value: "pop"
- Abbreviation for MIME Type: None
- Change Controller: IETF
- Specification Document(s): [[this document]]

7. Security Considerations

7.1. Offering Confidentiality Protection for Access to Protected Resources

This specification can be used with and without Transport Layer Security (TLS).

Without TLS this protocol provides a mechanism for verifying the integrity of requests, it provides no confidentiality protection. Consequently, eavesdroppers will have full access to communication content and any further messages exchanged between the client and the resource server. This could be problematic when data is exchanged that requires care, such as personal data.

When TLS is used then confidentiality of the transmission can be ensured between endpoints, including both the request and the

response. The use of TLS in combination with the signed HTTP request mechanism is highly recommended to ensure the confidentiality of the data returned from the protected resource.

7.2. Plaintext Storage of Credentials

The mechanism described in this document works in a similar way to many three-party authentication and key exchange mechanisms. In order to compute the signature over the HTTP request, the client must have access to a key bound to the access token in plaintext form. If an attacker were to gain access to these stored secrets at the client or (in case of symmetric keys) at the resource server they would be able to perform any action on behalf of any client just as if they had stolen a bearer token.

It is therefore paramount to the security of the protocol that the private keys associated with the access tokens are protected from unauthorized access.

7.3. Entropy of Keys

Unless TLS is used between the client and the resource server, eavesdroppers will have full access to requests sent by the client. They will thus be able to mount off-line brute-force attacks to attempt recovery of the session key or private key used to compute the keyed message digest or digital signature, respectively.

This specification assumes that the key used herein has been distributed via other mechanisms, such as [I-D.ietf-oauth-pop-key-distribution]. Hence, it is the responsibility of the authorization server and or the client to be careful when generating fresh and unique keys with sufficient entropy to resist such attacks for at least the length of time that the session keys (and the access tokens) are valid.

For example, if the key bound to the access token is valid for one day, authorization servers must ensure that it is not possible to mount a brute force attack that recovers that key in less than one day. Of course, servers are urged to err on the side of caution, and use the longest key length possible within reason.

7.4. Denial of Service

This specification includes a number of features which may make resource exhaustion attacks against resource servers possible. For example, a resource server may need to process the incoming request, verify the access token, perform signature verification, and might (in certain circumstances) have to consult back-end databases or the

authorization server before granting access to the protected resource. Many of these actions are shared with bearer tokens, but the additional cryptographic overhead of validating the signed request needs to be taken into consideration with deployment of this specification.

An attacker may exploit this to perform a denial of service attack by sending a large number of invalid requests to the server. The computational overhead of verifying the keyed message digest alone is not likely sufficient to mount a denial of service attack. To help combat this, it is RECOMMENDED that the protected resource validate the access token (contained in the "at" member of the signed structure) before performing any cryptographic verification calculations.

7.5. Validating the integrity of HTTP message

This specification provides flexibility for selectively validating the integrity of the HTTP request, including header fields, query parameters, and message bodies. Since all components of the HTTP request are only optionally validated by this method, and even some components may be validated only in part (e.g., some headers but not others) it is up to protected resource developers to verify that any vital parameters in a request are actually covered by the signature. Failure to do so could allow an attacker to inject vital parameters or headers into the request, ouside of the protection of the signature.

The application verifying this signature MUST NOT assume that any particular parameter is appropriately covered by the signature unless it is included in the signed structure and the hash is verified. Any applications that are sensitive of header or query parameter order MUST verify the order of the parameters on their own. The application MUST also compare the values in the JSON container with the actual parameters received with the HTTP request (using a direct comparison or a hash calculation, as appropriate). Failure to make this comparison will render the signature mechanism useless for protecting these elements.

The behavior of repeated query parameters or repeated HTTP headers is undefined by this specification. If a header or query parameter is repeated on either the outgoing request from the client or the incoming request to the protected resource, that query parameter or header name MUST NOT be covered by the hash and signature.

This specification records the order in which query parameters and headers are hashed, but it does not guarantee that order is preserved between the client and protected resource. If the order of

parameters or headers are significant to the underlying application, it MUST confirm their order on its own, apart from the signature and HTTP message validation.

8. Privacy Considerations

This specification addresses machine to machine communications and raises no privacy considerations beyond existing OAuth transactions.

9. Acknowledgements

The authors thank the OAuth Working Group for input into this work.

10. Normative References

[I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[I-D.ietf-oauth-pop-key-distribution]
Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", draft-ietf-oauth-pop-key-distribution-02 (work in progress), October 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://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, <http://www.rfc-editor.org/info/rfc3986>.

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

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

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

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

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

Authors' Addresses

Justin Richer (editor)

Email: ietf@justin.richer.org

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 21, 2020

M. Jones
A. Nadalin
Microsoft
B. Campbell, Ed.
Ping Identity
J. Bradley
Yubico
C. Mortimore
Salesforce
July 20, 2019

OAuth 2.0 Token Exchange
draft-ietf-oauth-token-exchange-19

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

Copyright Notice

Table of Contents

1. Introduction

A security token is a set of information that facilitates the sharing of identity and security information in heterogeneous environments or across security domains. Examples of security tokens include JSON Web Tokens (JWTs) [JWT] and SAML 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 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 [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], which MAY include a query component and MUST NOT include a fragment component. Multiple "resource" parameters may be used to indicate that the issued token is intended to be used at the multiple resources listed. See [I-D.ietf-oauth-resource-indicators] 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], 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 indicating the desired scope of the requested token using the "scope" parameter. The semantics of such a request are that the client is asking for a token with the requested scope that is usable at all the requested target services. Effectively, the requested access rights of the token are the cartesian product of all the scopes at all the target services.

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

2.2. Response

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

2.2.1. Successful Response

If the request is valid and meets all policy and other criteria of the authorization server, a successful token response is constructed by adding the following parameters to the entity-body of the HTTP

response using the "application/json" media type, as specified by [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 with the aforementioned meaning as 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, 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", "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 which 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 therefore are 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 therefore are 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 [I-D.ietf-oauth-security-topics] 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

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

7.1.1. Registry Contents

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

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

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

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

- 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 [[this specification]]

- 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 [[this specification]]

7.2. OAuth Parameters Registration

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

7.2.1. Registry Contents

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

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

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

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

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

- Specification document(s): Section 2.1 of [[this specification]]

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

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

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

7.3. OAuth Access Token Type Registration

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

7.3.1. Registry Contents

- Type name: N_A
- Additional Token Endpoint Response Parameters: (none)
- HTTP Authentication Scheme(s): (none)
- Change controller: IESG
- Specification document(s): Section 2.2.1 of [[this specification]]

7.4. JSON Web Token Claims Registration

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

7.4.1. Registry Contents

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

- Claim Name: "scope"
- Claim Description: Scope Values
- Change Controller: IESG

- Specification Document(s): Section 4.2 of [[this specification]]

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

- 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 [[this specification]]

7.5. OAuth Token Introspection Response Registration

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

7.5.1. Registry Contents

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

- Claim Name: "may_act"
- Claim Description: Authorized Actor - the party that is authorized to become the actor
- Change Controller: IESG
- Specification Document(s): Section 4.4 of [[this specification]]

7.6. OAuth Extensions Error Registration

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

7.6.1. Registry Contents

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

8. References

8.1. Normative References

[IANA.JWT.Claims]
IANA, "JSON Web Token Claims",
<http://www.iana.org/assignments/jwt>.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters",
<http://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,
<http://tools.ietf.org/html/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

[I-D.ietf-oauth-resource-indicators]
Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", draft-ietf-oauth-resource-indicators-02 (work in progress), January 2019.

[I-D.ietf-oauth-security-topics]
Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", draft-ietf-oauth-security-topics-13 (work in progress), July 2019.

[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.

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

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <http://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., Goodner, M., Gudgin, M., Barbir, A., and H. Granqvist, "WS-Trust 1.4", February 2012, <http://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 (with only a "subject_token" and no "actor_token", delegation is impossible). 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

Appendix B. Acknowledgements

This specification was developed within the OAuth Working Group, which includes dozens of active and dedicated participants. It was produced under the chairmanship of Hannes Tschofenig, 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.

Appendix C. Document History

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

-19

- o Fix-up changes introduced in -18.
- o Fix invalid JSON in the Nested Actor Claim example.
- o Reference figure numbers in text when introducing the examples in Section 2 and 4.
- o Editorial updates from additional IESG evaluation comments.
- o Add an informational reference to ietf-oauth-resource-indicators
- o Update ietf-oauth-security-topics ref to 13

-18

- o Editorial updates based on a few more IESG evaluation comments.

-17

- o Editorial improvements and example fixes resulting from IESG evaluation comments.
- o Added a pointer to RFC6749's Appendix B. on the "Use of application/x-www-form-urlencoded Media Type" as a way of providing a normative citation (by reference) for the media type.
- o Strengthened some of the wording in the privacy considerations to bring it inline with RFC 7519 Sec. 12 and RFC 6749 Sec. 10.8.

-16

- o Fixed typo and added an AD to Acknowledgements.

-15

- o Updated the nested actor claim example to (hopefully) be more straightforward.
- o Reworked Privacy Considerations to say to use TLS in transit, minimize the amount of information in the token, and encrypt the token if disclosure of its information to the client is a concern per https://mailarchive.ietf.org/arch/msg/secdir/KJhx4aq_U5uk3k6zpYP-CEHbpVM
- o Moved the Security and Privacy Considerations sections to before the IANA Considerations.

-14

- o Added text in Section 4.1 about the "act" claim stating that only the top-level claims and the current actor are to be considered in applying access control decisions.

-13

- o Updated the claim name and value syntax for scope to be consistent with the treatment of scope in RFC 7662 OAuth 2.0 Token Introspection.
- o Updated the client identifier claim name to be consistent with the treatment of client id in RFC 7662 OAuth 2.0 Token Introspection.

-12

- o Updated to use the boilerplate from RFC 8174.

-11

- o Added new WG chair and AD to the Acknowledgements.
- o Applied clarifications suggested during AD review by EKR.

-10

- o Defined token type URIs for base64url-encoded SAML 1.1 and SAML 2.0 assertions.
- o Applied editorial fixes.

-09

- o Changed "security tokens obtained could be used in a number of contexts" to "security tokens obtained may be used in a number of contexts" per a WGLC suggestion.

- o Clarified that the validity of the subject or actor token have no impact on the validity of the issued token after the exchange has occurred per a WGLC comment.
- o Changed use of invalid_target error code to a SHOULD per a WGLC comment.
- o Clarified text about non-identity claims within the "act" claim being meaningless per a WGLC comment.
- o Added brief Privacy Considerations section per WGLC comments.

-08

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

-07

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

-06

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

-05

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

-04

- Clarified that the "resource" and "audience" request parameters can be used at the same time (via http://www.ietf.org/mail-archive/web/oauth/current/msg15335.html).
- Clarified subject/actor token validity after token exchange and explained a bit more about the recommendation to not issue refresh tokens (via http://www.ietf.org/mail-archive/web/oauth/current/msg15318.html).
- Updated the examples appendix to use an issuer value that doesn't imply that the client issued and signed the tokens and used "Bearer" and "urn:ietf:params:oauth:token-type:access_token" in one of the responses (via http://www.ietf.org/mail-archive/web/oauth/current/msg15335.html).
- Defined and registered urn:ietf:params:oauth:token-type:id_token, since some use cases perform token exchanges for ID Tokens and no URI to indicate that a token is an ID Token had previously been defined.

-03

- Updated the document editors (adding Campbell, Bradley, and Mortimore).
- Added to the title.
- Added to the abstract and introduction.
- Updated the format of the request to use application/x-www-form-urlencoded request parameters and the response to use the existing token endpoint JSON parameters defined in OAuth 2.0.
- Changed the grant type identifier to urn:ietf:params:oauth:grant-type:token-exchange.
- Added RFC 6755 registration requests for urn:ietf:params:oauth:token-type:refresh_token, urn:ietf:params:oauth:token-type:access_token, and urn:ietf:params:oauth:grant-type:token-exchange.
- Added RFC 6749 registration requests for request/response parameters.
- Removed the Implementation Considerations and the requirement to support JWTs.
- Clarified many aspects of the text.
- Changed "on_behalf_of" to "subject_token", "on_behalf_of_token_type" to "subject_token_type", "act_as" to "actor_token", and "act_as_token_type" to "actor_token_type".
- Added an "audience" request parameter used to indicate the logical names of the target services at which the client intends to use the requested security token.
- Added a "want_composite" request parameter used to indicate the desire for a composite token rather than trying to infer it from the presence/absence of token(s) in the request.

- o Added a "resource" request parameter used to indicate the URLs of resources at which the client intends to use the requested security token.
- o Specified that multiple "audience" and "resource" request parameter values may be used.
- o Defined the JWT claim "act" (actor) to express the current actor or delegation principal.
- o Defined the JWT claim "may_act" to express that one party is authorized to act on behalf of another party.
- o Defined the JWT claim "scp" (scopes) to express OAuth 2.0 scope-token values.
- o Added the "N_A" (not applicable) OAuth Access Token Type definition for use in contexts in which the token exchange syntax requires a "token_type" value, but in which the token being issued is not an access token.
- o Added examples.

-02

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

-01

- o Updated references.

-00

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

Authors' Addresses

Michael B. Jones
Microsoft

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

Anthony Nadalin
Microsoft

Email: tonynad@microsoft.com

Brian Campbell (editor)
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Yubico

Email: ve7jtb@ve7jtb.com

Chuck Mortimore
Salesforce

Email: cmortimore@salesforce.com