OAuth Working Group	B. Campbell
Internet-Draft	Ping
Intended status: Standards Track	C. Mortimore
Expires: May 11, 2013	Salesforce
	M. Jones
	Y. Goland
	Microsoft
	November 7, 2012

TOC

Assertion Framework for OAuth 2.0 draft-ietf-oauth-assertions-07

Abstract

This specification provides a framework for the use of assertions with OAuth 2.0 in the form of a new client authentication mechanism and a new authorization grant type. Mechanisms are specified for transporting assertions during interactions with a token endpoint, as well as general processing rules.

The intent of this specification is to provide a common framework for OAuth 2.0 to interwork with other identity systems using assertions, and to provide alternative client authentication mechanisms.

Note that this specification only defines abstract message flows and processing rules. In order to be implementable, companion specifications are necessary to provide the corresponding concrete instantiations.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 11, 2013.

Table of Contents

TOC

1. Introduction

OAuth 2.0 **[RFC6749]** is an authorization framework that enables a third-party application to obtain limited access to a protected HTTP resource. In OAuth, those third-party applications are called clients; they access protected resources by presenting an access token to the HTTP resource. Access tokens are issued to clients by an authorization server with the (sometimes implicit) approval of the resource owner. These access tokens are typically obtained by exchanging an authorization grant, which represents the authorization granted by the resource owner (or by a privileged administrator). Several authorization grant types are defined to support a wide range of client types and user experiences. OAuth also provides an extensibility mechanism for defining additional grant types, which can serve as a bridge between OAuth and other protocol frameworks.

This specification provides a general framework for the use of assertions as authorization grants with OAuth 2.0. It also provides a framework for assertions to be used for client authentication. It provides generic mechanisms for transporting assertions during interactions with an authorization server's token endpoint, as well as general rules for the content and processing of those assertions. The intent is to provide an alternative client authentication mechanism (one that doesn't send client secrets), as well as to facilitate the use of OAuth 2.0 in client-server integration scenarios, where the end-user may not be present.

This specification only defines abstract message flows and processing rules. In order to be implementable, companion specifications are necessary to provide the corresponding concrete instantiations.

Note: The use of assertions for client authentication is orthogonal to and separable from using assertions as an authorization grant. They can be used either in combination or separately. Client assertion authentication is nothing more than an alternative way for a client to authenticate to the token endpoint and must be used in conjunction with some grant type to form a complete and meaningful protocol request. Assertion authorization grants may be used with or without client authentication or identification. Whether or not client authentication is needed in conjunction with an assertion authorization grant, as well as the supported types of client authentication, are policy decisions at the discretion of the authorization server.

2. Terminology

TOC

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]** .

Throughout this document, values are quoted to indicate that they are to be taken literally. When using these values in protocol messages, the quotes must not be used as part of the value.

3. Framework

TOC

An assertion is a package of information that allows identity and security information to be shared across security domains. An assertion typically contains information about a subject or principal, information about the party that issued the assertion and when was it issued, as well as the conditions under which the assertion is to be considered valid, such as when and where it can be used.

The entity that creates and signs the assertion is typically known as the "Issuer" and the entity that consumes the assertion and relies on its information is typically known as the "Relying Party". In the context of this document, the authorization server acts as a relying party.

Assertions used in the protocol exchanges defined by this specification MUST always be protected against tampering using a digital signature or a keyed message digest applied by the issuer. An assertion MAY additionally be encrypted, preventing unauthorized parties from inspecting the content.

Although this document does not define the processes by which the client obtains the assertion (prior to sending it to the authorization server), there are two common patterns described below.

In the first pattern, depicted in **Figure 1**, the client obtains an assertion from a third party entity capable of issuing, renewing, transforming, and validating security tokens. Typically such an entity is known as a "Security Token Service" (STS) or just "Token Service" and a trust relationship (usually manifested in the exchange of some kind of key material) exists between the token service and the relying party. The token service is the assertion issuer; its role is to fulfill requests from clients, which present various credentials, and mint assertions as requested, fill them with appropriate information, and sign them. **WS-Trust** [OASIS.WS-Trust] is one available standard for requesting security tokens (assertions).

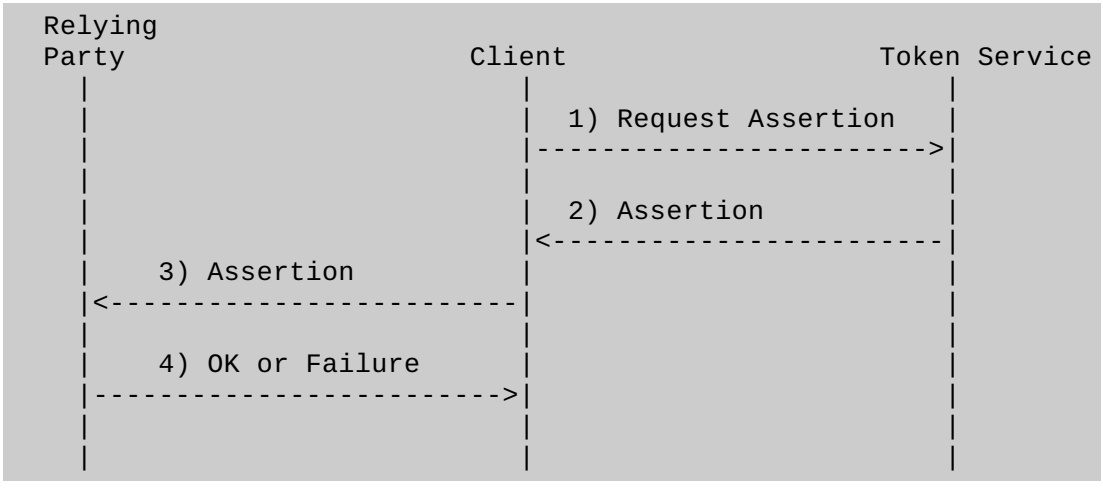

Figure 1: Third Party Created Assertion

In the second pattern, depicted in **Figure 2**, the client creates assertions locally. To sign the assertions, it has to obtain key material: either symmetric keys or asymmetric key pairs. The mechanisms for obtaining this key material are beyond the scope of this specification.

Although assertions are usually used to convey identity and security information, self-issued assertions can also serve a different purpose. They can be used to demonstrate knowledge of some secret, such as a client secret, without actually communicating the secret directly in the transaction. In that case, additional information included in the assertion by the client itself will be of limited value to the relying party and, for this reason, only a bare minimum of information is typically included in such an assertion, such as information about issuing and usage conditions.

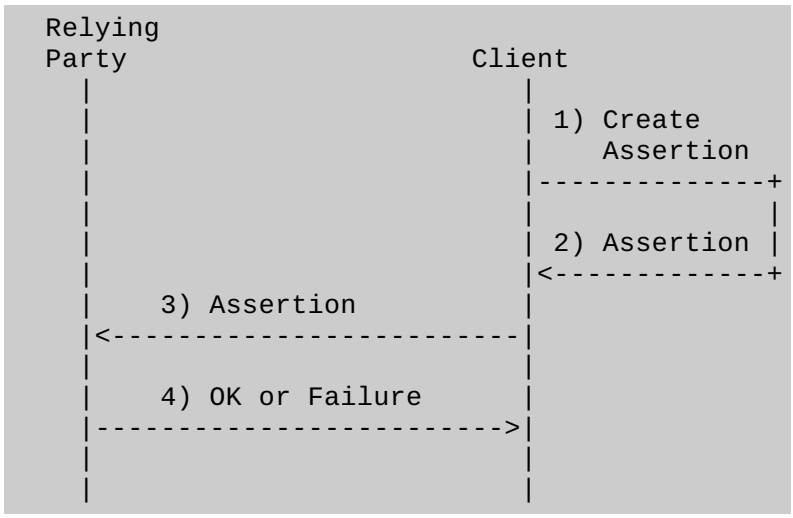

Figure 2: Self-Issued Assertion

Deployments need to determine the appropriate variant to use based on the required level of security, the trust relationship between the entities, and other factors.

From the perspective of what must be done by the entity presenting the assertion, there are two general types of assertions:

1. Bearer Assertions: Any entity in possession of a bearer assertion (e.g. the bearer) can use it to get access to the associated resources (without demonstrating possession of a cryptographic key). To prevent misuse, bearer assertions need to be protected from disclosure in storage and in transport. A secure communication channel is required between all entities to avoid leaking the assertion to unauthorized parties.
2. Holder-of-Key Assertions: To access the associated resources, the entity presenting the assertion must demonstrate possession of additional cryptographic material. The token service thereby binds a key identifier to the assertion and the client has to demonstrate to the relying party that it knows the key corresponding to that identifier when presenting the assertion. This mechanism provides additional security properties.

The protocol parameters and processing rules defined in this document are intended to support a client presenting a bearer assertion to an authorization server. The use of holder-of-key assertions are not precluded by this document, but additional protocol details would need to be specified.

4. Transporting Assertions

TOC

This section defines HTTP parameters for transporting assertions during interactions with a token endpoint of an OAuth authorization server. Because requests to the token endpoint

result in the transmission of clear-text credentials (in both the HTTP request and response), all requests to the token endpoint MUST use TLS, as mandated in Section 3.2 of **OAuth 2.0** [RFC6749].

TOC

4.1. Using Assertions as Authorization Grants

This section defines the use of assertions as authorization grants, based on the definition provided in Section 4.5 of **OAuth 2.0** [RFC6749]. When using assertions as authorization grants, the client includes the assertion and related information using the following HTTP request parameters:

grant_type
: REQUIRED. The format of the assertion as defined by the authorization server. The value MUST be an absolute URI.

assertion
: REQUIRED. The assertion being used as an authorization grant. Specific serialization of the assertion is defined by profile documents. The serialization MUST be encoded for transport within HTTP forms. It is RECOMMENDED that base64url be used.

scope
: OPTIONAL. The requested scope as described in Section 3.3 of **OAuth 2.0** [RFC6749]. When exchanging assertions for access tokens, the authorization for the token has been previously granted through some out-of-band mechanism. As such, the requested scope MUST be equal or lesser than the scope originally granted to the authorized accessor. If the scope parameter and/or value are omitted, the scope MUST be treated as equal to the scope originally granted to the authorized accessor. The Authorization Server MUST limit the scope of the issued access token to be equal or lesser than the scope originally granted to the authorized accessor.

The following non-normative example demonstrates an assertion being used as an authorization grant (with extra line breaks for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

client_id=s6BhdRkqt3&
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer&
assertion=PHNhbWxwOl...[omitted for brevity]...ZT4
```

An assertion used in this context is generally a short lived representation of the authorization grant and authorization servers SHOULD NOT issue access tokens with a lifetime that exceeds the validity period of the assertion by a significant period. In practice, that will usually mean that refresh tokens are not issued in response to assertion grant requests and access tokens will be issued with a reasonably short lifetime. Clients can refresh an expired access token by requesting a new one using the same assertion, if it is still valid, or with a new assertion.

An IEFT URN for use as the `grant_type` value can be requested using the template in **An IETF URN Sub-Namespace for OAuth** [I-D.ietf-oauth-urn-sub-ns]. A URN of the form urn:ietf:params:oauth:grant_type:* is suggested.

TOC

4.1.1. Error Responses

If an assertion is not valid or has expired, the Authorization Server MUST construct an error response as defined in **OAuth 2.0** [RFC6749]. The value of the `error` parameter MUST be the `invalid_grant` error code. The authorization server MAY include additional information regarding the reasons the assertion was considered invalid using the `error_description` or `error_uri` parameters.

For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error":"invalid_grant",
  "error_description":"Audience validation failed"
}
```

TOC

4.2. Using Assertions for Client Authentication

The following section defines the use of assertions as client credentials as an extension of Section 2.3 of **OAuth 2.0** [RFC6749]. When using assertions as client credentials, the client includes the assertion and related information using the following HTTP request parameters:

client_id
: OPTIONAL. The client identifier as described in Section 2.2 of **OAuth 2.0** [RFC6749]. When present, the `client_id` MUST identify the client to the authorization server.

client_assertion_type
: REQUIRED. The format of the assertion as defined by the authorization server. The value MUST be an absolute URI.

client_assertion
: REQUIRED. The assertion being used to authenticate the client. Specific serialization of the assertion is defined by profile documents. The serialization MUST be encoded for transport within HTTP forms. It is RECOMMENDED that base64url be used.

The following non-normative example demonstrates a client authenticating using an assertion during an Access Token Request, as defined in Section 4.1.3 of **OAuth 2.0** [RFC6749] (with extra line breaks for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=i1WsRn1uB1&
client_id=s6BhdRkqt3&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth
%3Aclient-assertion-type%3Asaml2-bearer&
client_assertion=PHNhbW...[omitted for brevity]...ZT
```

Token endpoints can differentiate between assertion based credentials and other client credential types by looking for the presence of the `client_assertion` and `client_assertion_type` parameters, which will only be present when using assertions for client authentication.

An IEFT URN for use as the `client_assertion_type` value may be requested using the template in **An IETF URN Sub-Namespace for OAuth** [I-D.ietf-oauth-urn-sub-ns]. A URN of the form urn:ietf:params:oauth:client-assertion-type:* is suggested.

TOC

4.2.1. Error Responses

If an assertion is invalid for any reason or if more than one client authentication mechanism is used, the Authorization Server MUST construct an error response as defined in **OAuth 2.0** [RFC6749]. The value of the `error` parameter MUST be the `invalid_client` error code. The

authorization server MAY include additional information regarding the reasons the client assertion was considered invalid using the `error_description` or `error_uri` parameters.

For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error":"invalid_client"
  "error_description":"assertion has expired"
}
```

TOC

5. Assertion Content and Processing

This section provides a general content and processing model for the use of assertions in **OAuth 2.0** [RFC6749].

TOC

5.1. Assertion Metamodel

The following are entities and metadata involved in the issuance, exchange, and processing of assertions in OAuth 2.0. These are general terms, abstract from any particular assertion format. Mappings of these terms into specific representations are provided by profiles of this specification.

Issuer
: The unique identifier for the entity that issued the assertion. Generally this is the entity that holds the key material used to generate the assertion. The issuer may be either an OAuth client (when assertions are self-issued) or a third party token service.

Principal
: A unique identifier for the subject of the assertion. When using assertions for client authentication, the Principal SHOULD be the `client_id` of the OAuth client. When using assertions as an authorization grant, the Principal MUST identify an authorized accessor for which the access token is being requested (typically the resource owner, or an authorized delegate).

Audience
: A URI that identifies the party intended to process the assertion. The audience SHOULD be the URL of the Token Endpoint as defined in Section 3.2 of **OAuth 2.0** [RFC6749].

Issued At
: The time at which the assertion was issued. While the serialization may differ by assertion format, this is always expressed in UTC with no time zone component.

Expires At
: The time at which the assertion expires. While the serialization may differ by assertion format, this is always expressed in UTC with no time zone component.

Assertion ID
: A nonce or unique identifier for the assertion. The Assertion ID may be used by implementations requiring message de-duplication for one-time use assertions. Any entity that assigns an identifier MUST ensure that there is negligible probability that that entity or any other entity will accidentally assign the same identifier to a different data object.

TOC

5.2. General Assertion Format and Processing Rules

The following are general format and processing rules for the use of assertions in OAuth:

- The assertion MUST contain an Issuer. The Issuer MUST identify the entity that

issued the assertion as recognized by the Authorization Server. If an assertion is self-issued, the Issuer SHOULD be the `client_id`.

- The assertion SHOULD contain a Principal. The Principal MUST identify an authorized accessor for which the access token is being requested (typically the resource owner, or an authorized delegate). When the client is acting on behalf of itself, the Principal SHOULD be the `client_id`.
- The assertion MUST contain an Audience that identifies the Authorization Server as the intended audience. The Authorization Server MUST verify that it is an intended audience for the assertion. The Audience SHOULD be the URL of the Authorization Server's Token Endpoint.
- The assertion MUST contain an Expires At entity that limits the time window during which the assertion can be used. The authorization server MUST verify that the expiration time has not passed, subject to allowable clock skew between systems. The authorization server SHOULD reject assertions with an Expires At attribute value that is unreasonably far in the future.
- The assertion MAY contain an Issued At entity containing the UTC time at which the assertion was issued.
- The assertion MAY contain an Assertion ID. An Authorization Server MAY dictate that Assertion ID is mandatory.
- The Authorization Server MUST validate the assertion's signature to verify the Issuer of the assertion. The algorithm used to validate the signature, and the mechanism for designating the secret used to generate the assertion, are beyond the scope of this specification.

---

6. Specific Assertion Format and Processing Rules

TOC

The following clarifies the format and processing rules defined in **Section 4** and **Section 5** for a number of common use cases:

6.1. Client Authentication

TOC

When a client uses an assertion for authentication, it SHOULD do so according to **Section 4.2**. The following format and processing rules apply:

- The `client_assertion_type` HTTP parameter MUST identify the assertion format being used for authentication.
- The `client_assertion` HTTP parameter MUST contain the serialized assertion in a format indicated by the `client_assertion_type` parameter.
- The Principal SHOULD be the `client_id`.
- The Issuer of the assertion MUST identify the entity that issued the assertion as recognized by the Authorization Server. If the assertion is self-issued, the Issuer SHOULD be the `client_id`.
- The Audience of the assertion MUST identify the Authorization Server and SHOULD be the URL of the Token Endpoint.
- The Authorization Server MUST verify the assertion's signature or keyed message digest to determine the validity of the issuer and the content of the assertion.

The following non-normative example demonstrates a client authentication using an assertion during an Access Token Request, as defined in Section 4.1.3 of **OAuth 2.0** [RFC6749] (with extra line breaks for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=i1WsRn1uB1&
client_id=s6BhdRkqt3&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth
%3Aclient-assertion-type%3Asaml2-bearer&
```

```
client_assertion=PHNhb...[omitted for brevity]...ZT4
```

TOC

6.2. Client Acting on Behalf of Itself

When a client is accessing resources on behalf of itself, it SHOULD do so in a manner analogous to the Client Credentials flow defined in Section 4.4 of **OAuth 2.0** [RFC6749]. This is a special case that combines both the authentication and authorization grant usage patterns. In this case, the interactions with the authorization server SHOULD be treated as using an assertion for Client Authentication according to **Section 4.2**, with the addition of a grant_type parameter. The following format and processing rules apply:

- The grant_type HTTP request parameter MUST be `client_credentials`.
- The `client_assertion_type` HTTP parameter MUST identify the assertion format.
- The `client_assertion` HTTP parameter MUST contain the serialized assertion as in a format indicated by the `client_assertion_type` parameter.
- The Issuer of the assertion MUST identify the entity that issued the assertion as recognized by the Authorization Server. If the assertion is self-issued, the Issuer SHOULD be the `client_id`. If the assertion was issued by a Security Token Service (STS), the Issuer SHOULD identify the STS as recognized by the Authorization Server.
- The Principal SHOULD be the `client_id`.
- The Audience of the assertion MUST identify the Authorization Server and SHOULD be the URL of the Token Endpoint.
- The Authorization Server MUST validate the assertion's signature to verify the Issuer of the assertion.

The following non-normative example demonstrates an assertion being used for a Client Credentials Access Token Request, as defined in Section 4.4.2 of **OAuth 2.0** [RFC6749] (with extra line breaks for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

client_id=s6BhdRkqt3&
grant_type=client_credentials&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth
%3Aclient-assertion-type%3Asaml2-bearer&
client_assertion=PHNhbW...[omitted for brevity]...ZT
```

TOC

6.3. Client Acting on Behalf of a User

When a client is accessing resources on behalf of a user, it SHOULD be treated as using an assertion as an Authorization Grant according to **Section 4.1**. The following format and processing rules apply:

- The grant_type HTTP request parameter MUST indicate the assertion format.
- The assertion HTTP parameter MUST contain the serialized assertion as in a format indicated by the grant_type parameter.
- The Issuer of the assertion MUST identify the entity that issued the assertion as recognized by the Authorization Server. If the assertion is self-issued, the Issuer SHOULD be the `client_id`. If the assertion was issued by a Security Token Service (STS), the Issuer SHOULD identify the STS as recognized by the Authorization Server.
- The Principal MUST identify an authorized accessor for which the access token is being requested (typically the resource owner, or an authorized delegate).
- The Audience of the assertion MUST identify the Authorization Server and MAY be the URL of the Token Endpoint.
- The Authorization Server MUST validate the assertion's signature to verify the Issuer of the assertion.

The following non-normative example demonstrates a client using an assertion as an Authorization Grant during an Access Token Request, as defined in Section 4.1.3 of **OAuth 2.0** [RFC6749] (with extra line breaks for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

client_id=s6BhdRkqt3&
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer&
assertion=PHNhbWxwOl...[omitted for brevity]...ZT
```

6.4. Client Acting on Behalf of an Anonymous User

TOC

When a client is accessing resources on behalf of an anonymous user, the following format and processing rules apply:

- The grant_type HTTP request parameter MUST indicate the assertion format.
- The assertion HTTP parameter MUST contain the serialized assertion as in a format indicated by the grant_type parameter.
- The Issuer of the assertion MUST identify the entity that issued the assertion as recognized by the Authorization Server. If the assertion is self-issued, the Issuer SHOULD be the `client_id`. If the assertion was issued by a Security Token Service (STS), the Issuer SHOULD identify the STS as recognized by the Authorization Server.
- The Principal SHOULD indicate to the Authorization Server that the client is acting on-behalf of an anonymous user as defined by the Authorization Server. It is implied that authorization is based upon additional criteria, such as additional attributes or claims provided in the assertion. For example, a client may present an assertion from a trusted issuer asserting that the bearer is over 18 via an included claim. In this case, no additional information about the user's identity is included yet all the data needed to issue an access token is present.
- The Audience of the assertion MUST identify the Authorization Server and MAY be the URL of the Token Endpoint.
- The Authorization Server MUST validate the assertion's signature to verify the Issuer of the assertion.

---

7. Security Considerations

TOC

This section discusses security considerations that apply when using assertions with OAuth 2.0 as described in this document. As discussed in **Section 3**, there are two different ways to obtain assertions: either as self-issued or obtained from a third party token service. While the actual interactions for obtaining an assertion are outside the scope of this document, the details are important from a security perspective. **Section 3** discusses the high level architectural aspects. Many of the security considerations discussed in this section are applicable to both the OAuth exchange as well as the client obtaining the assertion.

The remainder of this section focuses on the exchanges that concern presenting an assertion for client authentication and for the authorization grant.

7.1. Forged Assertion

TOC

Threat:

An adversary could forge or alter an assertion in order to obtain an access token (in case of the authorization grant) or to impersonate a client (in case of the client authentication mechanism).

Countermeasures:

To avoid this kind of attack, the entities must assure that proper mechanisms for protecting the integrity of the assertion are employed. This includes the issuer digitally signing the assertion or computing a keyed message digest over the assertion.

7.2. Stolen Assertion

TOC

Threat:

An adversary may be able obtain an assertion (e.g., by eavesdropping) and then reuse it (replay it) at a later point in time.

Countermeasures:

The primary mitigation for this threat is the use of a secure communication channel with server authentication for all network exchanges.

An assertion may also contain several elements to prevent replay attacks. There is, however, a clear tradeoff between reusing an assertion for multiple exchanges and obtaining and creating new fresh assertions.

Authorization Servers and Resource Servers may use a combination of the Assertion ID and Issued At/Expires At attributes for replay protection. Previously processed assertions may be rejected based on the Assertion ID. The addition of the validity window relieves the authorization server from maintaining an infinite state table of processed assertion IDs.

7.3. Unauthorized Disclosure of Personal Information

TOC

Threat:

The ability for other entities to obtain information about an individual, such as authentication information, role in an organization, or other authorization relevant information, raises privacy concerns.

Countermeasures:

To address the threats, two cases need to be differentiated:

First, a third party that did not participate in any of the exchange is prevented from eavesdropping on the content of the assertion by employing confidentiality protection of the exchange using TLS. This ensures that an eavesdropper on the wire is unable to obtain information. However, this does not prevent legitimate protocol entities from obtaining information from an assertion they may not have been allowed to obtain. Some assertion formats allow for the assertion to be encrypted, preventing unauthorized parties from inspecting the content.

Second, an Authorization Server may obtain an assertion that was created by a third party token service and that token service may have placed attributes into the assertion. To mitigate potential privacy problems, prior consent from the resource owner has to be obtained. OAuth itself does not directly provide such capabilities, but this consent approval may be obtained using other identity management protocols, user consent interactions, or in an out-of-band fashion.

For the cases where a third party token service creates assertions to be used for client authentication, privacy concerns are typically lower, since many of these clients are Web servers rather than individual devices operated by humans. If the assertions are used for client authentication of devices or software that can be closely linked to end users, then privacy protection safeguards need to be taken into consideration.

Further guidance on privacy friendly protocol design can be found in **[I-D.iab-privacy-considerations]**.

TOC

8. IANA Considerations

TOC

8.1. assertion Parameter Registration

TOC

The following is the parameter registration request, as defined in The OAuth Parameters Registry of **The OAuth 2.0 Authorization Protocol** [RFC6749], for the `assertion` parameter:

- Parameter name: assertion
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

8.2. client_assertion Parameter Registration

TOC

The following is the parameter registration request, as defined in The OAuth Parameters Registry of **The OAuth 2.0 Authorization Protocol** [RFC6749], for the `client_assertion` parameter:

- Parameter name: `client_assertion`
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

8.3. client_assertion_type Parameter Registration

TOC

The following is the parameter registration request, as defined in The OAuth Parameters Registry of **The OAuth 2.0 Authorization Protocol** [RFC6749], for the `client_assertion_type` parameter:

- Parameter name: `client_assertion_type`
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

9. References

TOC

9.1. Normative References

TOC

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

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

9.2. Informative References

TOC

[I-D.iab-privacy-considerations] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols," draft-iab-privacy-considerations-03 (work in progress), July 2012 (TXT).

[I-D.ietf-oauth-urn-sub-ns] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth," draft-ietf-oauth-urn-sub-ns-06 (work in progress), July 2012 (TXT).

[OASIS.WS-Trust] Nadalin, A., Ed., Goodner, M., Ed., Gudgin, M., Ed., Barbir, A., Ed., and H. Granqvist, Ed., "WS-Trust," Feb 2009.

Appendix A. Acknowledgements

TOC

The authors wish to thank the following people that have influenced or contributed this specification: Paul Madsen, Eric Sachs, Jian Cai, Tony Nadalin, Hannes Tschofenig, the authors of the OAuth WRAP specification, and the members of the OAuth working group.

Appendix B. Document History

TOC

[[to be removed by RFC editor before publication as an RFC]]

draft-ietf-oauth-assertions-07

- Reference RFC 6749.
- Remove extraneous word per http://www.ietf.org/mail-archive/web/oauth/current/msg10029.html

draft-ietf-oauth-assertions-06

- Add more text to intro explaining that an assertion grant type can be used with or without client authentication/identification and that client assertion authentication is nothing more than an alternative way for a client to authenticate to the token endpoint

draft-ietf-oauth-assertions-05

- Non-normative editorial cleanups

draft-ietf-oauth-assertions-04

- Updated document to incorporate the review comments from the shepherd - thread and alternative draft at http://www.ietf.org/mail-archive/web/oauth/current/msg09437.html
- Added reference to draft-ietf-oauth-urn-sub-ns and include suggestions on urn:ietf:params:oauth:[grant-type|client-assertion-type]:* URNs

draft-ietf-oauth-assertions-03

- updated reference to draft-ietf-oauth-v2 from -25 to -26

draft-ietf-oauth-assertions-02

- Added text about limited lifetime ATs and RTs per http://www.ietf.org/mail-archive/web/oauth/current/msg08298.html.
- Changed the line breaks in some examples to avoid awkward rendering to text format. Also removed encoded '=' padding from a few examples because both known derivative specs, SAML and JWT, omit the padding char in serialization/encoding.
- Remove section 7 on error responses and move that (somewhat modified) content into subsections of section 4 broken up by authn/authz per http://www.ietf.org/mail-archive/web/oauth/current/msg08735.html.
- Rework the text about "MUST validate ... in order to establish a mapping between ..." per http://www.ietf.org/mail-archive/web/oauth/current/msg08872.html and http://www.ietf.org/mail-archive/web/oauth/current/msg08749.html.
- Change "The Principal MUST identify an authorized accessor. If the assertion is self-issued, the Principal SHOULD be the client_id" in 6.1 per http://www.ietf.org/mail-archive/web/oauth/current/msg08873.html.
- Update reference in 4.1 to point to 2.3 (rather than 3.2) of oauth-v2 (rather than self) http://www.ietf.org/mail-archive/web/oauth/current/msg08874.html.
- Move the "Section 3 of" out of the xref to hopefully fix the link in 4.1 and remove the client_id bullet from 4.2 per http://www.ietf.org/mail-archive/web/oauth/current/msg08875.html.
- Add ref to Section 3.3 of oauth-v2 for scope definition and remove some then redundant text per http://www.ietf.org/mail-archive/web/oauth/current/msg08890.html.

- Change "The following format and processing rules SHOULD be applied" to "The following format and processing rules apply" in sections 6.x to remove conflicting normative qualification of other normative statements per http://www.ietf.org/mail-archive/web/oauth/current/msg08892.html.
- Add text the client_id must id the client to 4.1 and remove similar text from other places per http://www.ietf.org/mail-archive/web/oauth/current/msg08893.html.
- Remove the MUST from the text prior to the HTTP parameter definitions per http://www.ietf.org/mail-archive/web/oauth/current/msg08920.html.
- Updated examples to use grant_type and client_assertion_type values from the OAuth SAML Assertion Profiles spec.

Authors' Addresses

TOC

Brian Campbell
Ping Identity Corp.
Email: brian.d.campbell@gmail.com

Chuck Mortimore
Salesforce.com
Email: cmortimore@salesforce.com

Michael B. Jones
Microsoft
Email: mbj@microsoft.com

Yaron Y. Goland
Microsoft
Email: yarong@microsoft.com