DIGITALISERINGSSTYRELSEN

February 11th 2016

OIO IDWS REST profile (V1.0)

Background and purpose

The purpose of this document is to define a web service profile based on REST, where the client is authorized using a SAML Identity Token. The profile is designed to cover similar use cases as the Liberty Basic SOAP Binding [LIB-SOAP] with an equivalent level of security.

The main elements of the profile are:

- Securing a REST invocation from a web service client (WSC) to web service provider (WSP).
- Utilizing TLS for transport layer security (ensuring integrity, confidentiality)
- Authentication and authorization via a SAML token issued by a Security Token Service trusted by the web service provider.
- Optionally using a client certificate for proving Holder-of-key relationships.

The profile is inspired by OAuth 2.0 [RFC6749] which is used in many REST-based use cases.

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 RFC2119.

Assumptions and Design Goals

- The web service client is a rich client or web application.
- The web service client has already obtained a SAML security token from a Security Token Service which the web service provider trusts:
    - The SAML tokens complies to the OIO SAML Identity Token Profile.
    - The SAML token may be encrypted (<EncryptedAssertion>) or in clear text. In case it is encrypted, it is assumed that the STS knows the WSP's public key (for encryption) via an out-of-band mechanism.
    - The SAML token may be a "Bearer" token or "Holder-of-key token" (as defined by the SAML <SubjectConfirmation> element).

    - The SAML token is signed by the STS, and WSP has the STS certificate installed for verification of the signature.
- SAML tokens may exceed the size limits (~ 8KB) usually found in web server implementation for HTTP headers and query parameters[1]. Therefore, it is not a robust option to pass the SAML token via these mechanisms.
- The overhead of processing a SAML assertion on every web service invocation should be avoided.

Main Steps

To complete a scenario, three steps are executed:

1. The web service client requests a SAML token from the STS. This step is outside the scope of this profile and just mentioned for completeness of the scenario[2].
2. The web service client exchanges the SAML token for a short, opaque token called an *access token* using an authorization service (AS) co-located with the WSP. The access token is simply a string representing an access authorization issued to the client.
3. The web service client invokes the WSP (protected resource) using the access token. The WSP is able to verify the access token issued by the authorization service and determine if the invocation is authorized.

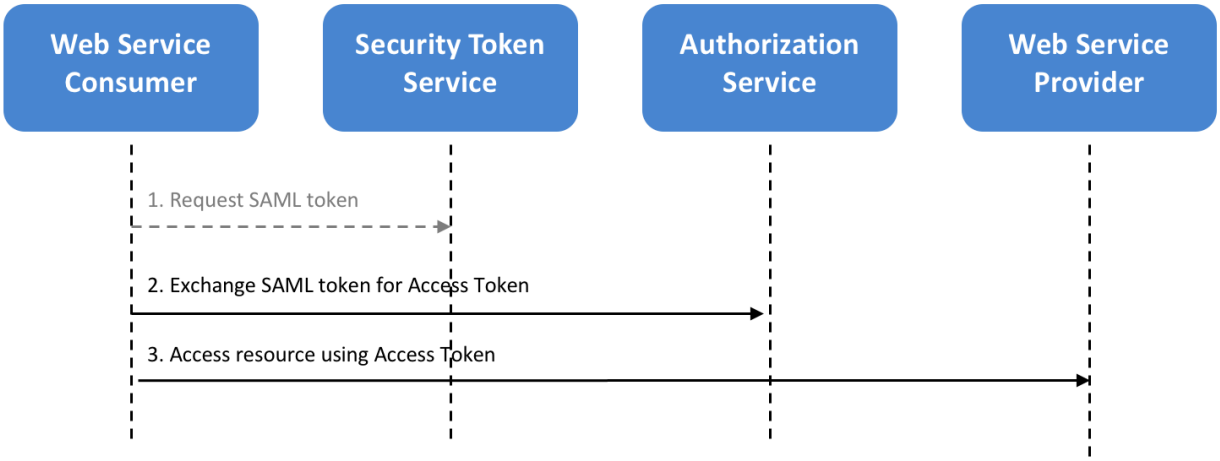
The Authorization Service and the Web Service Provider are closely related and trust each other. For example, they can be part of the same application or they can be more loosely coupled. The main point is that the Authorization Service can issue access tokens understood by the Web Service Provider and which can be used for access control enforcement. The format and internal structure of the access token is private to the implementation – e.g. it could be a reference to shared data structure where the authorizations of the SAML token are stored.

---

1 For example, encrypted Assertions issued by the NemLog-in STS are usually larger than 11KB.

2 In a Danish Context this could be performed by calling the NemLog-in Security Token Service using the WS-Trust protocol (over SOAP). The OIO WS-Trust profile (which NemLog-in follows) can be found here: https://digitaliser.dk/resource/516724

The steps are illustrated below:



Note: after completing all steps, the access token may be re-used by the client for subsequent invocations until it expires – i.e. step 1 and 2 are not repeated.

Details of step 2: Requesting an access token

The client begins by base64 encoding the SAML token (either <Assertion> or <EncryptedAssertion> element) and POSTING the result to the authorization service endpoint[3]:

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

saml-token=MIIGNDCCBRygAwIBAgIETBJt6DANBgkqhkiG9w0BAQsFAD...
```

The client MUST use TLS 1.1 or higher. If the SAML token is a Holder-of-key token, the client MUST use TLS with client authentication.

Authorization Service Processing rules

The Authorization Service validates the SAML token using normal SAML processing rules including (but not limited to):

---

[3] Note that RFC7522 describes a ”SAML 2.0 profile for OAuth 2.0 Client Authentication and Authorization Grants” which is similar but does not handle Holder-of-key Assertions (only Bearer). Therefore, it has been disregarded in this profile.

- Decrypting the Assertion (if encrypted)
- Validating that the Assertion was signed by a trusted STS
- Validating signature value and digests
- Validating that the Assertion is valid (XML wellformed and not expired)
- Validating that the AudienceRestriction element identifies the WSP.

Note: If the Assertion has "Holder-of-key" confirmation type, the Authorization Server MUST verify that the client has used TLS with client authentication AND the client's TLS certificate is equal to the certificate included in the Assertion <SubjectConfirmation> element as "Holder-of-key". This ensures that only a client with the corresponding private key can present the Assertion (i.e. it is bound to the client).

If any of the token validations fail, the Authorization Server MUST reject the request with an appropriate HTTP error code (401) and a token MUST NOT be returned. Instead, an error code and error message SHOULD be returned as described below:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer error="invalid_token",
                          error_description="The SAML token is expired"
```

If the validation passes, an HTTP response with a JSON structure like the following shall be returned:

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

{
"access_token":"7Fjfp0ZBr1H8shJgaJs97Jb",
"token_type":"Bearer",
"expires_in":3600
}
```

Parameters:

access_token

> REQUIRED. The access token issued by the authorization server. This is simply an opaque string and it MUST contain at least 64 bits of entropy to prevent guessing.

token_type

> REQUIRED. The type of the token issued MUST be “Bearer” or "Holder-of-key".[4] This SHOULD correspond to the type of the original SAML token.

expires_in

> REQUIRED. The lifetime in seconds of the access token. For example, the value 3600 denotes that the access token will expire in one hour from the time the response was generated.

Note: for security reasons, the Authorization server SHOULD limit the validity period (e.g. less than one hour) when issuing Bearer tokens. If the access token expires but the corresponding SAML token is not expired, the SAML token can be used to request a new access token without contacting the STS (i.e. just performing step 2 and 3 in the flow).

---

[4] OAuth does not seem to have a token type that mimics SAML’s concept of “Holder-of-key” exactly. Mac-tokens [OAuth-mac] have been considered for this profile but not found to fulfil the need since a symmetric key must be exchanged out-of-band and OIO IDWS uses asymmetric keys for Holder-of-key.

Details of step 3: Using an access token

The access token is used in a normal HTTP REST operation by passing the token as shown below.

In case the client has received multiple SAML tokens for the same WSP to be used in different contexts, the client MUST select the appropriate access token for the current invocation context.

Presenting "Bearer" access tokens

The client MUST use TLS 1.1 or higher.

The client MUST use the "Bearer" token type defined in [RFC6750] and follow requirements in this specification.

The client MUST pass the token in an Authorization header (the other options in RFC6750 are not allowed):

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Bearer 7Fjfp0ZBr1H8shJgaJs97Jb
```

Presenting "Holder-of-key" access tokens

The client MUST use TLS 1.1 or higher with client authentication.

The client MUST use the "Holder-of-key" token type defined below but otherwise follow [RFC6750].

The client MUST pass the token in an Authorization header (the other options in RFC6750 are not allowed):

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Holder-of-key 7Fjfp0ZBr1H8shJgaJs97Jb
```

WSP processing rules

The WSP MUST validate the access token on every request:

- The token is known.
- Not expired.

- Of correct type (Bearer vs. Holder-of-key)
- For Holder-of-key tokens the WSP MUST check that the WSC TLS client certificate is the same as referenced in the original SAML token's <SubjectConfirmation> "Holder-of-key" element. This in effect makes the access token a Holder-of-key token.

All rejected access requests SHOULD be logged by the WSP.

The WSP SHOULD consider the risk of replay attacks and implement appropriate countermeasures if necessary.

Error handling

If the WSP does not recognize the access token, if is expired, if the token does not authorize the current operation or if any other authorization fails, an appropriate HTTP error code (e.g. 400, 401, 403, or 405) SHOULD be returned along with error codes detailing the reason.

The server SHOULD respond to errors as described in RFC6750 e.g.:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate:  Bearer realm="example",
                   error="invalid_token",
                   error_description="The access token is expired"
```

The defined error codes are “invalid_request”, “invalid_token” and “insufficient_scope”.

References

RFC6750	"The OAuth 2.0 Authorization Framework: Bearer Token Usage", IETF. http://tools.ietf.org/html/rfc6750
OAuth-mac	"OAuth 2.0 Message Authentication Code (MAC) Tokens", IETF. http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-05
RFC6749	"The OAuth 2.0 Authorization Framework", IETF. https://tools.ietf.org/html/rfc6749
LIB-SOAP	"Liberty Basic SOAP Binding 1.0". https://digitaliser.dk/resource/414852