Santander Global Tech

Azure AD/B2C Access Token Validation

Santander Global Tech

Contents

Santander Global Tech

Version Control

Version	Date	Author	Change
1.4	19/10/2020	C.McMahon	Added validation of permitted signature algorithms

Santander Global Tech

Introduction

A new policy for APIGee is required that will provide Azure AD/B2C Access Token validation and caller identification. This document defines the steps required to validate the token and the mapping of claims.

Basic Concepts

The Access Token (AT) generated by Azure is a JWT and as such is self-contained, that is, the validation can be performed without the need to consult the Azure instance. That said, the validation requires the use of a public key which needs to be obtained dynamically from an endpoint associated with the issuer of the token.

Dynamic key retrieval is required for production environments as Microsoft reserves the right to rotate keys as and when they see fit: https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-signing-key-rollover

The policy should be generic and configurable and allow the use of ATs from different issuers and contemplate ATs corresponding with different OAuth flows:

- User – Personalized flows e.g. Auth Code
- Application – Client Credentials

Policy Input

- The request which contains the "`Authorization: Bearer`" http header with the AT.

Policy Configuration

- A list of valid audience values.
- A list of valid scopes.
- A field indicating the type of token expected: "user" or "application".
- A list of permitted signature algorithms. By default it should be set to "RS256".

Policy Output

A security context with the claims extracted from the received AT.

Issuers and Keys

Issuer Identification and Key Endpoint

A list of valid issuers should be configured at policy level. This list of issuers is used to ensure that only tokens from an approved list of providers are accepted and also to establish their relevant configuration including, in this case, the endpoint for retrieving the public key.

The issuer of the token can be found in the "iss" claim of the JWT body:

{

```
  "iss": "https://azureb2cinstance/43385616-157e-4c02-a610-d83e4868ee39/v2.0/",
  "exp": 1597754774,
  "nbf": 1597751174,
  …
}
```

For each “iss” value an endpoint should be configured to allow public key retrieval and identification, for example:

```
{
  “issuers”: [
    {
      “issuer”: “https://azureb2cinstance/43385616-157e-4c02-a610-d83e4868ee39/v2.0/”
      “issuer_type”:”B2C”
      “jwks_uri”: https://azureb2cinstance/43385616-157e-4c02-a610-
d83e4868ee39/v2.0/b2c_1_on_boarding_custom/discovery/v2.0/keys
    }
}
```

“issuer”:	Unique identifier of the issuer.
“issuer_type”:	Indicates if it is B2C or AD (this is used to determine which claims to extract).
“jwt_uri”:	The URL from where the public keys should be recovered.

Note: If the received token contains an “iss” value which is not present in the configuration it should be rejected.

jws_uri

Calls to the “jws_uri” return the following JSON structure with a “kid” entry for each public key available:

```
{
"keys": [
    {"kid":"X5eXk4xyojNFum1kl2Ytv8dlNP4-
c57dO6QGTVBwaNk","nbf":1493763266,"use":"sig","kty":"RSA","e":"AQAB","n":"tVKUtcx_n9rt5afY_2WFNvU6PlFMggC
atsZ3l4RjKxH0jgdLq6CScb0P3ZGXYbPzXvmmLiWZizpb-h0qup5jznOvOr-
Dhw9908584BSgC83YacjWNqEK3urxhyE2jWjwRm2N95WGgb5mzE5XmZIvkvyXnn7X8dvgFPF5QwIngGsDG8LyHuJWlaDhr_EPLMW4wHvH
0zZCuRMARIJmmqiMy3VD4ftq4nS5s8vJL0pVSrkuNojtokp84AtkADCDU_BUhrc2sIgfnvZ03koCQRoZmWiHu86SuJZYkDFstVTVSR0hi
XudFlfQ2rOhPlpObmku68lXw-7V-P7jwrQRFfQVXw"},
    {“kid”:…}
  ]
}
```

Key Retrieval

As per “JWT Validation” a public key is required. This key is identified by the “kid” claim included in the JWT header, for example:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "X5eXk4xyojNFum1kl2Ytv8dlNP4-c57dO6QGTVBwaNk"
}
```

The process for key retrieval is as follows:

- If a cached value for the corresponding “kid” is available then use it for token validation.
- Else, recover the configuration corresponding with the “iss” claim. If no entry is found reject the request.
- Else, call the “`jwks_uri`” endpoint and extract the entry corresponding with the “kid” value received in the header. If no entry exists reject the request.

- Else, confirm that the value of “nbf” is before the current time (including a small leeway for clock skew). If it is not reject the request.
- Else, extract the values of the “n” and “e” fields, calculate the public key and store it in the cache and then it for token validation.

JWT Validation

From a standards point of view the JWT should be validated according to https://tools.ietf.org/html/rfc7519#section-7.2 and https://tools.ietf.org/html/rfc7515.

The indications found here have also been taken into consideration: https://docs.microsoft.com/en-us/azure/active-directory/develop/access-tokens

These can be summarized as follows:

- Parse the JWT rejecting tokens that have an incorrect format
- Assure that the “nbf” field contains a value less than the current time
- Assure that the “exp” field contains a value greater than the current time
- Validate the JWT signature using the key recovered as per *Issuers and Keys*

(It is recommended that publically available libraries be used for this validation)

These additional validations should also take place:

- Confirm that one of the values of the “aud” claim matches the audience associated with the API being called (see:
- Audience).
- Confirm that all of the values of the “scp” claim match one of the scopes associated with the API being called (see: *Scopes*).
- Confirm that the “alg” claim matches one of the permitted algorithms in the policy configuration.

Special Notes

If the “nonce” claim is received in the header it should be ignored.

Calculate Public Key

The B2C public endpoint returns “n” (modulus) and “e” (exponent) rather than the public key. It is therefore necessary to generate the public key before using it to validate the signature. The way of doing this depends on the implementation language but these links should serve as a reference:

- https://github.com/rzcoder/node-rsa/issues/43
- https://github.com/dragosgaftoneanu/okta-php-scripts/blob/master/pem-generator/index.php
- https://stackoverflow.com/questions/34403823/verifying-jwt-signed-with-the-rs256-algorithm-using-public-key-in-c-sharp
- https://stackoverflow.com/questions/44664510/difference-between-azure-ad-and-azure-ad-b2c-tokens
- http://www-cs-students.stanford.edu/~tjw/jsbn/

Santander Global Tech

- https://netweblog.wordpress.com/2018/05/29/verify-jwt-with-modulus-and-exponent-php/
- https://medium.com/@software_factotum/validating-rsa-signature-for-a-jws-10229fb46bbf

Audience

The policy should be configured with an audience value. The "aud" claim should contain this value, if not it should be rejected.

Scopes

The policy should be configured with a list of the valid scope values and the token type. The scopes associated with the Access Token should be extracted from one of two claims according to the expected token type. If the required claim is not present or does not match the token type then the token should be rejected.

Token Type: "user"

- **"scp"**

The "scp" claim contains a list of scopes with the following format:

```
"scp": "email openid profile"
```

Token Type: "application"

- **"roles"**

This claim contains an array of scopes with the following format:

```
"roles": [
  "public.api.read"
]
```

All of the values retrieved from the space separated list (scp) or the array of strings (roles) should match of the scopes received as a parameter. If not the JWT should be rejected.

Claims mapping

In order to align with the information returned by the current introspection system this policy should return the following claims extracted or calculated as detailed below:

Output Claim	Origin	Comments
"active"	Calculated	"true" for a valid AT, "false" otherwise. In the case of invalid tokens no other claims are returned
"scope"	AT (JWT)	"scp" or "roles" according to "*Scopes*"
"client_id"	AT (JWT)	If the issuer is of type B2C from the "aud" claim, otherwise from "appid"

“sub”	AT (JWT)	Optional, only returned if the case of user tokens Name of the claim from which to retrieve the value should be configured at API level. The implementation should contemplate the need to extract the value from the first entry in an array.
token_type	Static	“access_token”
exp	AT (JWT)	“exp” for all token and issuer types
“iss”	AT (JWT)	“iss” for all token and issuer types

IMPORTANT: Current requirements do not necessitate additional claims mapping. However it is recommended that the implementation is designed to allow this possibility in the future.

Sample JWTs

Azure B2C

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "X5eXk4xyojNFum1kl2Ytv8dlNP4-c57dO6QGTVBwaNk"
}.{
  "iss": "https://coranbegone.b2clogin.com/43385616-157e-4c02-a610-d83e4868ee39/v2.0/",
  "exp": 1597754774,
  "nbf": 1597751174,
  "aud": "6181399d-652b-4e64-b894-493641aa63f9",
  "given_name": "Christopher",
  "family_name": "McMahon",
  "name": "Christopher McMahon",
  "idp": "google.com",
  "sub": "df738f86-85b6-4806-aa7c-4d3e2dc9ef3d",
  "emails": [
    "christopher.james.mcmahon@gmail.com"
  ],
  "tfp": "B2C_1_google",
  "scp": "adminconsole",
  "azp": "6181399d-652b-4e64-b894-493641aa63f9",
  "ver": "1.0",
  "iat": 1597751174
}.[Signature]
```

Azure AD

```
{
  "typ": "JWT",
  "nonce": "mfPDC_xObKcUO3L79XlHz3kglMuuPqx3x162WEdDjuc",
  "alg": "RS256",
  "x5t": "huN95IvPfehq34GzBDZ1GXGirnM",
  "kid": "huN95IvPfehq34GzBDZ1GXGirnM"
}.{
  "aud": "00000003-0000-0000-c000-000000000000",
  "iss": "https://sts.windows.net/5f348a75-4db6-4b83-9268-c781e497d12d/",
  "iat": 1597751152,
  "nbf": 1597751152,
  "exp": 1597755052,
  "acct": 0,
  "acr": "1",
  "aio":
"AUQAu/8QAAAAomIlSl9RCzs4rDTkGY/XHNpSq4cnsc1KCta0+67HoIeACS6kjgc5CE2vANCawB5CXuDroe152xkbAmv8scYg2g==",
  "amr": [
    "pwd",
    "mfa"
```

```
  ],
  "app_displayname": "openidtest",
  "appid": "ff81a293-7406-4438-a888-0cf53d861421",
  "appidacr": "1",
  "family_name": "Admin",
  "given_name": "Adrian",
  "ipaddr": "193.127.195.2",
  "name": "Adrian Admin",
  "oid": "06efd6a6-827c-4ed5-8806-80d8cd347bf5",
  "platf": "3",
  "puid": "10032000ACB84D40",
  "scp": "email openid profile",
  "sub": "RGT08UeGxyjI4-y8xWcBjHjjt5aOWjjdEuKdhiaEQxs",
  "tenant_region_scope": "EU",
  "tid": "5f348a75-4db6-4b83-9268-c781e497d12d",
  "unique_name": "adrianadmin@christophermcmahonyandex.onmicrosoft.com",
  "upn": "adrianadmin@christophermcmahonyandex.onmicrosoft.com",
  "uti": "AdmnKDY4oEyI9yUOmIIiAA",
  "ver": "1.0",
  "wids": [
    "be2f45a1-457d-42af-a067-6ec1fa63bc45",
    "e8cef6f1-e4bd-4ea8-bc07-4b8d950f4477"
  ],
  "xms_st": {
    "sub": "KHwqFmVk7tT_h9TFJzih4gF858ic2LhdlYc7FnFGKfM"
  },
  "xms_tcdt": 1586255735
}.[Signature]
```


Azure AD – Client Credentials


```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "huN95IvPfehq34GzBDZ1GXGirnM"
}.{
  "aud": "6181399d-652b-4e64-b894-493641aa63f9",
  "iss": "https://login.microsoftonline.com/43385616-157e-4c02-a610-d83e4868ee39/v2.0",
  "iat": 1597842173,
  "nbf": 1597842173,
  "exp": 1597846073,
  "aio": "E2BgYBBSO/zopumh+PvdWaqXV3z4AQA=",
  "azp": "6181399d-652b-4e64-b894-493641aa63f9",
  "azpacr": "1",
  "oid": "3ef949b6-2f29-4d6b-99e2-fb473ba43751",
  "roles": [
    "public.api.read"
  ],
  "sub": "3ef949b6-2f29-4d6b-99e2-fb473ba43751",
  "tid": "43385616-157e-4c02-a610-d83e4868ee39",
  "uti": "_vtH2gmoz0y46K-U72NdAA",
  "ver": "2.0"
}.[Signature]
```