Access tokens in the Microsoft identity platform - Microsoft identity platform

By cilwerner

Archived: 2026-04-05 13:35:01 UTC

Access tokens are a type of security token designed for authorization, granting access to specific resources on behalf on an authenticated user. Information in access tokens determines whether a user has the right to access a particular resource, similar to keys unlocking specific doors in a building. These individual pieces of information that make up tokens are called claims. Therefore, they are sensitive credentials and pose a security risk if not handled correctly. Access tokens differ from ID tokens which serve as proof of authentication.

Access tokens enable clients to securely call protected web APIs. Although client applications can receive and use access tokens, they should be treated as opaque strings. The client application shouldn't attempt to validate access tokens. The resource server should validate the access token before accepting it as proof of authorization. The contents of the token are intended only for the API, which means that access tokens must be treated as opaque strings. For validation and debugging purposes *only*, developers can decode JWTs using a site like jwt.ms. Tokens that a Microsoft API receives might not always be a JWT that can be decoded.

Clients should use the token response data that's returned with the access token for details on what's inside it. When the client requests an access token, the Microsoft identity platform also returns some metadata about the access token for the consumption of the application. This information includes the expiry time of the access token and the scopes for which it's valid. This data allows the application to do intelligent caching of access tokens without having to parse the access token itself. This article explains essential information about access tokens, including formats, ownership, lifetimes and how APIs can validate and use the claims inside an access token.

Note

All documentation on this page, except where noted, applies only to tokens issued for registered APIs. It doesn't apply to tokens issued for Microsoft-owned APIs, nor can those tokens be used to validate how the Microsoft identity platform issues tokens for a registered API.

Token formats

There are two versions of access tokens available in the Microsoft identity platform: v1.0 and v2.0. These versions determine the claims that are in the token and make sure that a web API can control the contents of the token.

Web APIs have one of the following versions selected as a default during registration:

- v1.0 for Microsoft Entra-only applications. The following example shows a v1.0 token (the keys are changed and personal information is removed, which prevents token validation):

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSIsImtpZCI6Imk2bEdrM0Z
```

- v2.0 for applications that support consumer accounts. The following example shows a v2.0 token (the keys are changed and personal information is removed, which prevents token validation):

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSJ9.eyJhdWQiOiI2ZTc0MT
```

Set the version for applications by providing the appropriate value to the `requestedAccessTokenVersion` setting in the app manifest. The values of `null` and `1` result in v1.0 tokens, and the value of `2` results in v2.0 tokens.

Token ownership

An access token request involves two parties: the client, who requests the token, and the resource (Web API) that accepts the token. The resource that the token is intended for (its *audience*) is defined in the `aud` claim in a token. Clients use the token but shouldn't understand or attempt to parse it. Resources accept the token.

The Microsoft identity platform supports issuing any token version from any version endpoint. For example, when the value of `requestedAccessTokenVersion` is `2`, a client calling the v1.0 endpoint to get a token for that resource receives a v2.0 access token.

Resources always own their tokens using the `aud` claim and are the only applications that can change their token details.

Token lifetime

The default lifetime of an access token is variable. When issued, the Microsoft identity platform assigns a random value ranging between 60-90 minutes (75 minutes on average) as the default lifetime of an access token. The variation improves service resilience by spreading access token demand over a time, which prevents hourly spikes in traffic to Microsoft Entra ID.

Tenants that don't use Conditional Access have a default access token lifetime of two hours for clients such as Microsoft Teams and Microsoft 365.

Adjust the lifetime of an access token to control how often the client application expires the application session, and how often it requires the user to reauthenticate (either silently or interactively). To override the default access token lifetime variation, use Configurable token lifetime (CTL).

Apply default token lifetime variation to organizations that have Continuous Access Evaluation (CAE) enabled. Apply default token lifetime variation even if the organizations use CTL policies. The default token lifetime for long lived token lifetime ranges from 20 to 28 hours. When the access token expires, the client must use the refresh token to silently acquire a new refresh token and access token.

Organizations that use Conditional Access sign-in frequency (SIF) to enforce how frequently sign-ins occur can't override default access token lifetime variation. When organizations use SIF, the time between credential prompts

for a client can range from the sign-in frequency interval to the token lifetime that ranges from 60 - 90 minutes plus the sign-in frequency interval.

Here's an example of how default token lifetime variation works with sign-in frequency. Let's say an organization sets sign-in frequency to occur every hour. When the token has lifetime ranging from 60-90 minutes due to token lifetime variation, the actual sign-in interval occurs anywhere between 1 hour to 2.5 hours.

If a user with a token with a one hour lifetime performs an interactive sign-in at 59 minutes, there's no credential prompt because the sign-in is below the SIF threshold. If a new token has a lifetime of 90 minutes, the user wouldn't see a credential prompt for another hour and a half. During a silent renewal attempt, Microsoft Entra ID requires a credential prompt because the total session length has exceeded the sign-in frequency setting of 1 hour. In this example, the time difference between credential prompts due to the SIF interval and token lifetime variation would be 2.5 hours.

Validate tokens

Not all applications should validate tokens. Only in specific scenarios should applications validate a token:

- Web APIs must validate access tokens sent to them by a client. They must only accept tokens containing one of their AppId URIs as the `aud` claim.
- Web apps must validate ID tokens sent to them by using the user's browser in the hybrid flow, before allowing access to a user's data or establishing a session.

If none of the previously described scenarios apply, there's no need to validate the token. Public clients like native, desktop, or single-page applications don't benefit from validating ID tokens because the application communicates directly with the IDP where SSL protection ensures the ID tokens are valid. They shouldn't validate the access tokens, as they are for the web API to validate, not the client.

APIs and web applications must only validate tokens that have an `aud` claim that matches the application. Other resources may have custom token validation rules. For example, you can't validate tokens for Microsoft Graph according to these rules due to their proprietary format. Validating and accepting tokens meant for another resource is an example of the confused deputy problem.

If the application needs to validate an ID token or an access token, it should first validate the signature of the token and the issuer against the values in the OpenID discovery document.

The Microsoft Entra middleware has built-in capabilities for validating access tokens. See samples to find one in the appropriate language. There are also several third-party open-source libraries available for JWT validation. For more information about authentication libraries and code samples, see the authentication libraries. If your web app or web API is on ASP.NET or ASP.NET Core, use Microsoft.Identity.Web, which handles the validation for you.

v1.0 and v2.0 tokens

- When your web app/API is validating a v1.0 token (`ver` claim ="1.0"), it needs to read the OpenID Connect metadata document from the v1.0 endpoint (`https://login.microsoftonline.com/{example-`

`tenant-id}/.well-known/openid-configuration`), even if the authority configured for your web API is a v2.0 authority.

- When your web app/API is validating a v2.0 token (`ver` claim ="2.0"), it needs to read the OpenID Connect metadata document from the v2.0 endpoint (`https://login.microsoftonline.com/{example-tenant-id}/v2.0/.well-known/openid-configuration`), even if the authority configured for your web API is a v1.0 authority.

The following examples suppose that your application is validating a v2.0 access token (and therefore reference the v2.0 versions of the OIDC metadata documents and keys). Just remove the "/v2.0" in the URL if you validate v1.0 tokens.

Validate the issuer

OpenID Connect Core says "The Issuer Identifier [...] MUST exactly match the value of the iss (issuer) Claim." For applications which use a tenant-specific metadata endpoint (like `https://login.microsoftonline.com/aaaabbbb-0000-cccc-1111-dddd2222eeee/v2.0/.well-known/openid-configuration` or `https://login.microsoftonline.com/contoso.onmicrosoft.com/v2.0/.well-known/openid-configuration`), this is all that is needed.

Microsoft Entra ID has a tenant-independent version of the document available at https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration. This endpoint returns an issuer value `https://login.microsoftonline.com/{tenantid}/v2.0`. Applications may use this tenant-independent endpoint to validate tokens from every tenant with the following modifications:

1. Instead of expecting the issuer claim in the token to exactly match the issuer value from metadata, the application should replace the `{tenantid}` value in the issuer metadata with the tenant id that is the target of the current request, and then check the exact match.

2. The application should use the `issuer` property returned from the keys endpoint to restrict the scope of keys.

    - Keys that have an issuer value like `https://login.microsoftonline.com/{tenantid}/v2.0` may be used with any matching token issuer.
    - Keys that have an issuer value like `https://login.microsoftonline.com/9188040d-6c67-4c5b-b112-36a304b66dad/v2.0` should only be used with exact match.

    Microsoft Entra tenant-independent key endpoint (https://login.microsoftonline.com/common/discovery/v2.0/keys) returns a document like:

```
{
 "keys":[
   {"kty":"RSA","use":"sig","kid":"A1bC2dE3fH4iJ5kL6mN7oP8qR9sT0u","x5t":"A1bC2dE3fH4iJ5kL6mN7oP8qR9sT0u
   {"kty":"RSA","use":"sig","kid":"C2dE3fH4iJ5kL6mN7oP8qR9sT0uV1w","x5t":"C2dE3fH4iJ5kL6mN7oP8qR9sT0uV1w
   {"kty":"RSA","use":"sig","kid":"E3fH4iJ5kL6mN7oP8qR9sT0uV1wX2y","x5t":"E3fH4iJ5kL6mN7oP8qR9sT0uV1wX2y
```

```
  ]
}
```

3. Applications that use a Microsoft Entra tenant id (`tid`) claim as a trust boundary instead of the standard issuer claim should ensure that the tenant-id claim is a guid and that the issuer and tenant id match.

Using tenant-independent metadata is more efficient for applications which accept tokens from many tenants.

Note

With Microsoft Entra tenant-independent metadata, claims should be interpreted within the tenant, as under standard OpenID Connect, claims are interpreted within the issuer. That is, `{"sub":"ABC123","iss":"https://login.microsoftonline.com/aaaabbbb-0000-cccc-1111-dddd2222eeee/v2.0","tid":"aaaabbbb-0000-cccc-1111-dddd2222eeee"}` and `{"sub":"ABC123","iss":"https://login.microsoftonline.com/bbbbcccc-1111-dddd-2222-eeee3333ffff/v2.0","tid":"bbbbcccc-1111-dddd-2222-eeee3333ffff"}` describe different users, even though the `sub` is the same, because claims like `sub` are interpreted within the context of the issuer/tenant.

Validate the signature

A JWT contains three segments separated by the `.` character. The first segment is the **header**, the second is the **body**, and the third is the **signature**. Use the signature segment to evaluate the authenticity of the token.

Microsoft Entra ID issues tokens signed using the industry standard asymmetric encryption algorithms, such as RS256. The header of the JWT contains information about the key and encryption method used to sign the token:

```
{
 "typ": "JWT",
 "alg": "RS256",
 "x5t": "H4iJ5kL6mN7oP8qR9sT0uV1wX2yZ3a",
 "kid": "H4iJ5kL6mN7oP8qR9sT0uV1wX2yZ3a"
}
```

The `alg` claim indicates the algorithm used to sign the token, while the `kid` claim indicates the particular public key that was used to validate the token.

At any given point in time, Microsoft Entra ID may sign an ID token using any one of a certain set of public-private key pairs. Microsoft Entra ID rotates the possible set of keys on a periodic basis, so write the application to handle those key changes automatically. A reasonable frequency to check for updates to the public keys used by Microsoft Entra ID is every 24 hours.

Acquire the signing key data necessary to validate the signature by using the OpenID Connect metadata document located at:

```
https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration
```

Tip

Try this in a browser: URL

The following information describes the metadata document:

- Is a JSON object that contains several useful pieces of information, such as the location of the various endpoints required for doing OpenID Connect authentication.
- Includes a `jwks_uri`, which gives the location of the set of public keys that correspond to the private keys used to sign tokens. The JSON Web Key (JWK) located at the `jwks_uri` contains all of the public key information in use at that particular moment in time. RFC 7517 describes the JWK format. The application can use the `kid` claim in the JWT header to select the public key, from this document, which corresponds to the private key that has been used to sign a particular token. It can then do signature validation using the correct public key and the indicated algorithm.

Note

Use the `kid` claim to validate the token. Though v1.0 tokens contain both the `x5t` and `kid` claims, v2.0 tokens contain only the `kid` claim.

Doing signature validation is outside the scope of this document. There are many open-source libraries available for helping with signature validation if necessary. However, the Microsoft identity platform has one token signing extension to the standards, which are custom signing keys.

If the application has custom signing keys as a result of using the claims-mapping feature, append an `appid` query parameter that contains the application ID. For validation, use `jwks_uri` that points to the signing key information of the application. For example: `https://login.microsoftonline.com/{tenant}/.well-known/openid-configuration?appid=00001111-aaaa-2222-bbbb-3333cccc4444` contains a `jwks_uri` of `https://login.microsoftonline.com/{tenant}/discovery/keys?appid=00001111-aaaa-2222-bbbb-3333cccc4444`.

Validate the issuer

Web apps validating ID tokens, and web APIs validating access tokens need to validate the issuer of the token (`iss` claim) against:

1. the issuer available in the OpenID connect metadata document associated with the application configuration (authority). The metadata document to verify against depends on:
    - the version of the token
    - the accounts supported by your application.
2. the tenant ID (`tid` claim) of the token,
3. the issuer of the signing key.

Single tenant applications

OpenID Connect Core says "The Issuer Identifier [...] MUST exactly match the value of the `iss` (issuer) Claim." For applications that use a tenant-specific metadata endpoint, such as `https://login.microsoftonline.com/{example-tenant-id}/v2.0/.well-known/openid-configuration` or `https://login.microsoftonline.com/contoso.onmicrosoft.com/v2.0/.well-known/openid-configuration`.

Single tenant applications are applications that support:

- Accounts in one organizational directory (**example-tenant-id** only): `https://login.microsoftonline.com/{example-tenant-id}`
- Personal Microsoft accounts only: `https://login.microsoftonline.com/consumers` (**consumers** being a nickname for the tenant 9188040d-6c67-4c5b-b112-36a304b66dad)

Multitenant applications

Microsoft Entra ID also supports multitenant applications. These applications support:

- Accounts in any organizational directory (any Microsoft Entra directory): `https://login.microsoftonline.com/organizations`
- Accounts in any organizational directory (any Microsoft Entra directory) and personal Microsoft accounts (for example, Skype, XBox): `https://login.microsoftonline.com/common`

For these applications, Microsoft Entra ID exposes tenant-independent versions of the OIDC document at `https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration` and `https://login.microsoftonline.com/organizations/v2.0/.well-known/openid-configuration` respectively. These endpoints return an issuer value, which is a template parametrized by the `tenantid`: `https://login.microsoftonline.com/{tenantid}/v2.0`. Applications may use these tenant-independent endpoints to validate tokens from every tenant with the following stipulations:

- Validate the signing key issuer
- Instead of expecting the issuer claim in the token to exactly match the issuer value from metadata, the application should replace the `{tenantid}` value in the issuer metadata with the tenant ID that is the target of the current request, and then check the exact match (`tid` claim of the token).
- Validate that the `tid` claim is a GUID and the `iss` claim is of the form `https://login.microsoftonline.com/{tid}/v2.0` where `{tid}` is the exact `tid` claim. This validation ties the tenant back to the issuer and back to the scope of the signing key creating a chain of trust.
- Use `tid` claim when they locate data associated with the subject of the claim. In other words, the `tid` claim must be part of the key used to access the user's data.

Validate the signing key issuer

Applications using the v2.0 tenant-independent metadata need to validate the signing key issuer.

Keys document and signing key issuer

As discussed, from the OpenID Connect document, your application accesses the keys used to sign the tokens. It gets the corresponding keys document by accessing the URL exposed in the **jwks_uri** property of the OpenIdConnect document.

```
  "jwks_uri": "https://login.microsoftonline.com/{example-tenant-id}/discovery/v2.0/keys",
```

The `{example-tenant-id}` value can be replaced by a GUID, a domain name, or **common**, **organizations, and **consumers**.

The `keys` documents exposed by Azure AD v2.0 contains, for each key, the issuer that uses this signing key. For instance, the tenant-independent "common" key endpoint `https://login.microsoftonline.com/common/discovery/v2.0/keys` returns a document like:

```
 {
  "keys":[
    {"kty":"RSA","use":"sig","kid":"A1bC2dE3fH4iJ5kL6mN7oP8qR9sT0u","x5t":"A1bC2dE3fH4iJ5kL6mN7oP8qR9sT0u","n":"
    {"kty":"RSA","use":"sig","kid":"C2dE3fH4iJ5kL6mN7oP8qR9sT0uV1w","x5t":"C2dE3fH4iJ5kL6mN7oP8qR9sT0uV1w","n":"
    {"kty":"RSA","use":"sig","kid":"E3fH4iJ5kL6mN7oP8qR9sT0uV1wX2y","x5t":"E3fH4iJ5kL6mN7oP8qR9sT0uV1wX2y","n":"
  ]
}
```

Validation of the signing key issuer

The application should use the `issuer` property of the keys document, associated with the key used to sign the token, in order to restrict the scope of keys:

- Keys that have an issuer value with a GUID like `https://login.microsoftonline.com/9188040d-6c67-4c5b-b112-36a304b66dad/v2.0` should only be used when the `iss` claim in the token matches the value exactly.
- Keys that have a templated issuer value like `https://login.microsoftonline.com/{tenantid}/v2.0` should only be used when the `iss` claim in the token matches this value after substituting the `tid` claim in the token for the `{tenantid}` placeholder.

Using tenant-independent metadata is more efficient for applications that accept tokens from many tenants.

Note

With Microsoft Entra tenant-independent metadata, claims should be interpreted within the tenant, as under standard OpenID Connect, claims are interpreted within the issuer. That is, `{"sub":"ABC123","iss":"https://login.microsoftonline.com/{example-tenant-id}/v2.0","tid":"{example-tenant-id}"}` and `{"sub":"ABC123","iss":"https://login.microsoftonline.com/{another-tenand-id}/v2.0","tid":"{another-tenant-id}"}` describe different users, even though the `sub` is the same, because claims like `sub` are interpreted within the context of the issuer/tenant.

Recap

Here's some pseudo code that recapitulates how to validate issuer and signing key issuer:

1. Fetch keys from configured metadata URL
2. Check token if signed with one of the published keys, fail if not
3. Identify key in the metadata based on the kid header. Check the "issuer" property attached to the key in the metadata document:

```
 var issuer = metadata["kid"].issuer;
if (issuer.contains("{tenantId}", CaseInvariant)) issuer = issuer.Replace("{tenantid}", token["tid"], Cas
if (issuer != token["iss"]) throw validationException;
if (configuration.allowedIssuer != "*" && configuration.allowedIssuer != issuer) throw validationExceptio
var issUri = new Uri(token["iss"]);
if (issUri.Segments.Count < 1) throw validationException;
if (issUri.Segments[1] != token["tid"]) throw validationException;
```

Related content

- Access token claims reference
- Secure applications and APIs by validating claims
