Validate Google Id Token

15

6

I'm using ASP.NET Core to serve an API to an Android client. Android signs in as a Google account and passes a JWT, the ID Token, to API as a bearer token. I have the app working, it does pass the auth checks, but I don't think it's validating the token signature.

Per Google's documents, I can call this url to do it: https://www.googleapis.com/oauth2/v3/tokeninfo?id_token=XYZ123, but I can't find the appropriate hooks on the server side to do it. Also according to the Google docs, I can somehow use the Client Access APIs to do it without calling to the server every time.

My configuration code:

```
app.UseJwtBearerAuthentication( new JwtBearerOptions()
{

    Authority = "https://accounts.google.com",
    Audience = "hiddenfromyou.apps.googleusercontent.com",
    TokenValidationParameters = new TokenValidationParameters()
    {
        ValidateAudience = true,
        ValidIssuer = "accounts.google.com"
    },
    RequireHttpsMetadata = false,
    AutomaticAuthenticate = true,
    AutomaticChallenge = false,
});
```

How do I get the JWTBearer middleware to validate the signature? I'm close to giving up on using the MS middleware and rolling my own.

c# google-api asp.net-core google-oauth jwt

edited Feb 23 '17 at 11:37 DaImTo

asked Aug 21 '16 at 6:04 Darthg8r

30

There are a couple of different ways in which you can validate the integrity of the ID token on the server side:

1. "Manually" - constantly download Google's public keys, verify signature and then each and every field, including the `iss` one; the main advantage (albeit a small one in my opinion) I see here is that you can minimize the number of requests sent to Google.
2. "Automatically" - do a GET on Google's endpoint to verify this token `https://www.googleapis.com/oauth2/v3/tokeninfo?id_token={0}`
3. Using a Google API Client Library - like the official one.

Here's how the second one could look:

```
private const string GoogleApiTokenInfoUrl =
"https://www.googleapis.com/oauth2/v3/tokeninfo?id_token={0}";

public ProviderUserDetails GetUserDetails(string providerToken)
{
    var httpClient = new MonitoredHttpClient();
    var requestUri = new Uri(string.Format(GoogleApiTokenInfoUrl, providerToken));

    HttpResponseMessage httpResponseMessage;
    try
    {
        httpResponseMessage = httpClient.GetAsync(requestUri).Result;
    }
    catch (Exception ex)
    {
        return null;
    }

    if (httpResponseMessage.StatusCode != HttpStatusCode.OK)
    {
        return null;
    }

    var response = httpResponseMessage.Content.ReadAsStringAsync().Result;
    var googleApiTokenInfo = JsonConvert.DeserializeObject<GoogleApiTokenInfo>
(response);

    if (!SupportedClientsIds.Contains(googleApiTokenInfo.aud))
    {
        {
            Email = googleApiTokenInfo.email,
            FirstName = googleApiTokenInfo.given_name,
            LastName = googleApiTokenInfo.family_name,
            Locale = googleApiTokenInfo.locale,
            Name = googleApiTokenInfo.name,
            ProviderUserId = googleApiTokenInfo.sub
        };
    }
```

GoogleApiTokenInfo class:

```
public class GoogleApiTokenInfo
{
/// <summary>
/// The Issuer Identifier for the Issuer of the response. Always
https://accounts.google.com or accounts.google.com for Google ID tokens.
/// </summary>
public string iss { get; set; }

/// <summary>
/// Access token hash. Provides validation that the access token is tied to the identity
token. If the ID token is issued with an access token in the server flow, this is always
/// included. This can be used as an alternate mechanism to protect against cross-site
request forgery attacks, but if you follow Step 1 and Step 3 it is not necessary to
verify the
/// access token.
/// </summary>
public string at_hash { get; set; }

/// <summary>
/// Identifies the audience that this ID token is intended for. It must be one of the
OAuth 2.0 client IDs of your application.
/// </summary>
public string aud { get; set; }

/// <summary>
/// An identifier for the user, unique among all Google accounts and never reused. A
Google account can have multiple emails at different points in time, but the sub value
is never
/// changed. Use sub within your application as the unique-identifier key for the user.
```

```
        public string email_verified { get; set; }

        /// <summary>
        /// The client_id of the authorized presenter. This claim is only needed when the party
        requesting the ID token is not the same as the audience of the ID token. This may be the
        /// case at Google for hybrid apps where a web application and Android app have a
        different client_id but share the same project.
        /// </summary>
        public string azp { get; set; }

        /// <summary>
        /// The user's email address. This may not be unique and is not suitable for use as a
        primary key. Provided only if your scope included the string "email".
        /// </summary>
        public string email { get; set; }

        /// <summary>
        /// The time the ID token was issued, represented in Unix time (integer seconds).
        /// </summary>
        public string iat { get; set; }

        /// <summary>
        /// The time the ID token expires, represented in Unix time (integer seconds).
        /// </summary>
        public string exp { get; set; }

        /// <summary>
        /// The user's full name, in a displayable form. Might be provided when:
        /// The request scope included the string "profile"
        /// The ID token is returned from a token refresh
        /// When name claims are present, you can use them to update your app's user records.
        Note that this claim is never guaranteed to be present.
        /// </summary>
        public string name { get; set; }

        /// <summary>
        /// The URL of the user's profile picture. Might be provided when:
        /// The request scope included the string "profile"
        /// The ID token is returned from a token refresh
        /// When picture claims are present, you can use them to update your app's user records.
        Note that this claim is never guaranteed to be present.
        /// </summary>
        public string picture { get; set; }
```

```
public string alg { get; set; }

public string kid { get; set; }
}
```

edited Nov 13 '17 at 7:49

answered Aug 22 '16 at 6:50

Alexandru Marculescu
4,594 6 26 41

Wow I have been a contributor on the Official Google .Net client library since 2012. Are you sure C# doesn't have one yet? Also spamming the tokeninfo is not recommend by Google you should validate the Token_id locally. github.com/google/google-api-dotnet-client – DalmTo Feb 23 '17 at 11:24

apologies @DalmTo, you're right! I've edited my answer – Alexandru Marculescu Apr 13 '17 at 8:09

1 Where is GoogleApiTokenInfo defined? Is this a custom class you've created, or defined in the Google SDK? – Bob Black Nov 11 '17 at 13:56

@BobBlack - I've updated my answer to include it; it's created as per Google's specs – Alexandru Marculescu Nov 13 '17 at 7:50

For those building a Chrome Extension or Chrome App, the `chrome.identity.getAuthToken()` method only provides an `access_token` . Fortunately the endpoint @AlexandruMarculescu suggests in option 2 supports verifying this type of token as well: googleapis.com/oauth2/v3/tokeninfo?access_token={0} – jfbloom22 Apr 6 '18 at 15:25

14

According to this github issue, you can now use `GoogleJsonWebSignature.ValidateAsync` method to validate a Google-signed JWT. Simply pass the `idToken` string to the method.

```
var validPayload = await GoogleJsonWebSignature.ValidateAsync(idToken);
Assert.IsNotNull(validPayload);
```

If it is not a valid token, it will return `null` .

Note that to use this method, you need to install Google.Apis.Auth nuget firsthand.

edited May 7 at 3:32

answered Mar 27 '18 at 7:40

3

Google states in the documentation for openId connect

> For debugging purposes, you can use Google's tokeninfo endpoint. Suppose your ID token's value is XYZ123.

You should not be using that endpoint to validate your JWT.

Validation of an ID token requires several steps:

1. Verify that the ID token is properly signed by the issuer. Google-issued tokens are signed using one of the certificates found at the URI specified in the jwks_uri field of the discovery document.
2. Verify that the value of iss in the ID token is equal to https://accounts.google.com or accounts.google.com.
3. Verify that the value of aud in the ID token is equal to your app's client ID.
4. Verify that the expiry time (exp) of the ID token has not passed.
5. If you passed a hd parameter in the request, verify that the ID token has a hd claim that matches your G Suite hosted domain.

There is an official sample project on how to validate them here. Unfortunately we have not added this to the Google .Net Client library yet. It has been logged as an issue

answered Feb 23 '17 at 11:33
DaImTo
49.7k 12 78 264

2

So, what I found is that as the OpenIDConnect specs have a /.well-known/ url that contains the information that you need to validate a token. This includes access to the public keys for the signature. The JWT middleware forms that .well-known url from the authority, retrieves the information, and proceeds to validate it on it's own.

The short answer to the question is that the validation is already happening in the middleware, there's nothing left to do.

answered Aug 26 '16 at 16:12
Darthg8r
5,677 10 45 81