Authenticate with a backend server

developers.google.com/identity/sign-in/web/backend-auth

If you use Google Sign-In with an app or site that communicates with a backend server, you might need to identify the currently signed-in user on the server. To do so securely, after a user successfully signs in, send the user's ID token to your server using HTTPS. Then, on the server, verify the integrity of the ID token and use the user information contained in the token to establish a session or create a new account.

Send the ID token to your server

After a user successfully signs in, get the user's ID token:

```
function onSignIn(googleUser) {
  var id_token = googleUser.getAuthResponse().id_token;
  ...
}
```

Then, send the ID token to your server with an HTTPS POST request:

```
var xhr = new XMLHttpRequest();
xhr.open('POST', 'https://yourbackend.example.com/tokensignin');
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xhr.onload = function() {
  console.log('Signed in as: ' + xhr.responseText);
};
xhr.send('idtoken=' + id_token);
```

Verify the integrity of the ID token

After you receive the ID token by HTTPS POST, you must verify the integrity of the token. To verify that the token is valid, ensure that the following criteria are satisfied:

- The ID token is properly signed by Google. Use Google's public keys (available in JWK or PEM format) to verify the token's signature. These keys are regularly rotated; examine the `Cache-Control` header in the response to determine when you should retrieve them again.
- The value of `aud` in the ID token is equal to one of your app's client IDs. This check is necessary to prevent ID tokens issued to a malicious app being used to access data about the same user on your app's backend server.
- The value of `iss` in the ID token is equal to `accounts.google.com` or `https://accounts.google.com`.
- The expiry time (`exp`) of the ID token has not passed.
- If you want to restrict access to only members of your G Suite domain, verify that the ID

token has an `hd` claim that matches your G Suite domain name.

Rather than writing your own code to perform these verification steps, we strongly recommend using a Google API client library for your platform, or calling our `tokeninfo` validation endpoint.

Using a Google API Client Library

Using one of the Google API Client Libraries (e.g. Java, Node.js, PHP, Python) is the recommended way to validate Google ID tokens in a production environment.

Java
To validate an ID token in Java, use the GoogleIdTokenVerifier object. For example:

```
import com.google.api.client.googleapis.auth.oauth2.GoogleIdToken;
import com.google.api.client.googleapis.auth.oauth2.GoogleIdToken.Payload;
import com.google.api.client.googleapis.auth.oauth2.GoogleIdTokenVerifier;

...

GoogleIdTokenVerifier verifier = new GoogleIdTokenVerifier.Builder(transport,
jsonFactory)
    // Specify the CLIENT_ID of the app that accesses the backend:
    .setAudience(Collections.singletonList(CLIENT_ID))
    // Or, if multiple clients access the backend:
    //.setAudience(Arrays.asList(CLIENT_ID_1, CLIENT_ID_2, CLIENT_ID_3))
    .build();

// (Receive idTokenString by HTTPS POST)

GoogleIdToken idToken = verifier.verify(idTokenString);
if (idToken != null) {
  Payload payload = idToken.getPayload();

  // Print user identifier
  String userId = payload.getSubject();
  System.out.println("User ID: " + userId);

  // Get profile information from payload
  String email = payload.getEmail();
  boolean emailVerified = Boolean.valueOf(payload.getEmailVerified());
  String name = (String) payload.get("name");
  String pictureUrl = (String) payload.get("picture");
  String locale = (String) payload.get("locale");
  String familyName = (String) payload.get("family_name");
  String givenName = (String) payload.get("given_name");

  // Use or store profile information
  // ...

} else {
  System.out.println("Invalid ID token.");
}
```

The `GoogleIdTokenVerifier.verify()` method verifies the JWT signature, the `aud` claim, the `iss` claim, and the `exp` claim.

If you want to restrict access to only members of your G Suite domain, also verify the `hd` claim by checking the domain name returned by the `Payload.getHostedDomain()` method.

Node.js

To validate an ID token in Node.js, use the Google Auth Library for Node.js. Install the library:

```
npm install google-auth-library --save
```

Then, call the `verifyIdToken()` function. For example:

```
const {OAuth2Client} = require('google-auth-library');
const client = new OAuth2Client(CLIENT_ID);
async function verify() {
  const ticket = await client.verifyIdToken({
      idToken: token,
      audience: CLIENT_ID,  // Specify the CLIENT_ID of the app that accesses the
backend
      // Or, if multiple clients access the backend:
      //[CLIENT_ID_1, CLIENT_ID_2, CLIENT_ID_3]
  });
  const payload = ticket.getPayload();
  const userid = payload['sub'];
  // If request specified a G Suite domain:
  //const domain = payload['hd'];
}
verify().catch(console.error);
```

The `verifyIdToken` function verifies the JWT signature, the `aud` claim, the `exp` claim, and the `iss` claim.

If you want to restrict access to only members of your G Suite domain, also verify the `hd` claim matches your G Suite domain name.

PHP

To validate an ID token in PHP, use the Google API Client Library for PHP. Install the library (for example, using Composer):

```
composer require google/apiclient
```

Then, call the `verifyIdToken()` function. For example:

```
require_once 'vendor/autoload.php';

// Get $id_token via HTTPS POST.

$client = new Google_Client(['client_id' => $CLIENT_ID]);  // Specify the CLIENT_ID of
the app that accesses the backend
$payload = $client->verifyIdToken($id_token);
if ($payload) {
  $userid = $payload['sub'];
  // If request specified a G Suite domain:
  //$domain = $payload['hd'];
} else {
  // Invalid ID token
}
```

The `verifyIdToken` function verifies the JWT signature, the `aud` claim, the `exp` claim, and the `iss` claim.

If you want to restrict access to only members of your G Suite domain, also verify the `hd` claim matches your G Suite domain name.

Python

To validate an ID token in Python, use the verify_oauth2_token function. For example:

```
from google.oauth2 import id_token
from google.auth.transport import requests

# (Receive token by HTTPS POST)
# ...

try:
    # Specify the CLIENT_ID of the app that accesses the backend:
    idinfo = id_token.verify_oauth2_token(token, requests.Request(), CLIENT_ID)

    # Or, if multiple clients access the backend server:
    # idinfo = id_token.verify_oauth2_token(token, requests.Request())
    # if idinfo['aud'] not in [CLIENT_ID_1, CLIENT_ID_2, CLIENT_ID_3]:
    #     raise ValueError('Could not verify audience.')

    if idinfo['iss'] not in ['accounts.google.com', 'https://accounts.google.com']:
        raise ValueError('Wrong issuer.')

    # If auth request is from a G Suite domain:
    # if idinfo['hd'] != GSUITE_DOMAIN_NAME:
    #     raise ValueError('Wrong hosted domain.')

    # ID token is valid. Get the user's Google Account ID from the decoded token.
    userid = idinfo['sub']
except ValueError:
    # Invalid token
    pass
```

The `verify_oauth2_token` function verifies the JWT signature, the `aud` claim, and the `exp` claim. You must also verify the `iss` claim and the `hd` claim (if applicable) by examining the object that `verify_oauth2_token` returns. If multiple clients access the backend server, also manually verify the `aud` claim.

Calling the tokeninfo endpoint

An easy way to validate an ID token for debugging and low-volume use is to use the `tokeninfo` endpoint. Calling this endpoint involves an additional network request that does most of the validation for you, but introduces some latency and the potential for network errors.

To validate an ID token using the `tokeninfo` endpoint, make an HTTPS POST or GET request to the endpoint, and pass your ID token in the `id_token` parameter. For example, to validate the token "XYZ123", make the following GET request:

```
https://www.googleapis.com/oauth2/v3/tokeninfo?id_token=XYZ123
```

If the token is properly signed and the `iss` and `exp` claims have the expected values, you will get a HTTP 200 response, where the body contains the JSON-formatted ID token claims. Here's an example response:

```
{
 // These six fields are included in all Google ID Tokens.
 "iss": "https://accounts.google.com",
 "sub": "110169484474386276334",
 "azp": "1008719970978-hb24n2dstb40o45d4feuo2ukqmcc6381.apps.googleusercontent.com",
 "aud": "1008719970978-hb24n2dstb40o45d4feuo2ukqmcc6381.apps.googleusercontent.com",
 "iat": "1433978353",
 "exp": "1433981953",

 // These seven fields are only included when the user has granted the "profile" and
 // "email" OAuth scopes to the application.
 "email": "testuser@gmail.com",
 "email_verified": "true",
 "name" : "Test User",
 "picture": "https://lh4.googleusercontent.com/-
kYgzyAWpZzJ/ABCDEFGHI/AAAJKLMNOP/tIXL9Ir44LE/s99-c/photo.jpg",
 "given_name": "Test",
 "family_name": "User",
 "locale": "en"
}
```

If you are a G Suite customer, you might also be interested in the `hd` claim, which indicates the hosted domain of the user. This can be used to restrict access to a resource to only members of certain domains. The absence of this claim indicates that the user does not belong to a G Suite hosted domain.

Create an account or session

After you have verified the token, check if the user is already in your user database. If so, establish an authenticated session for the user. If the user isn't yet in your user database, create a new user record from the information in the ID token payload, and establish a session for the user. You can prompt the user for any additional profile information you require when you detect a newly created user in your app.