Authentication between services

cloud.google.com/api-gateway/docs/authenticate-service-account

In addition to authenticating end user requests, you may want to authenticate services (non-human users) that make requests to your API. This page explains how to use service accounts to provide authentication for humans or services.

Overview

To identify a service that sends requests to your API, you use a service account. The calling service uses the service account's private key to sign a secure JSON Web Token (JWT) and sends the signed JWT in the request to your API.

To implement service account authentication in your API and calling service:

1. Create a service account and key for the calling service to use.
2. Add support for authentication in the API config for your API Gateway service.
3. Add code to the calling service that:
    - Creates a JWT and signs it with the service account's private key.
    - Sends the signed JWT in a request to the API.

API Gateway validates that the claims in the JWT match the configuration in your API config before forwarding the request to your API. API Gateway doesn't check for Cloud Identity permissions that you have granted on the service account.

Prerequisites

This page assumes that you have already:

- Created a Google Cloud project.
- Created an OpenAPI document describing your API.
- Created an API config.

Creating a service account with a key

You need a service account with a private key file that the calling service uses to sign the JWT. If you have more than one service sending requests to your API, you can create one service account to represent all the calling services. If you need to differentiate between the services—for example, they might have different permissions—you can create a service account and key for each calling service.

This section shows how to use the Google Cloud Console and the `gcloud` command-line tool to create the service account and private key file and to assign the service account the **Service Account Token Creator** role. For information on using an API to do this task,

see Creating and managing service accounts.

To create a service account with a key:

Cloud Console gcloud
Create a service account:

1. In the Cloud Console, go to **Create service account**.

   Go to Create service account

2. Select a project.

3. In the **Service account name** field, enter a name. The Cloud Console fills in the **Service account ID** field based on this name.

4. Optional: In the **Service account description** field, enter a description.

5. Click **Create**.

6. Click the **Select a role** field.

   Under **All roles**, select **Service Accounts** > **Service Account Token Creator**.

7. Click **Continue**.

8. Click **Done** to finish creating the service account.

   Do not close your browser window. You will use it in the next procedure.

Create a service account key:

1. In the Cloud Console, click the email address for the service account that you created.
2. Click **Keys**.
3. Click **Add key**, then **Create new key**.
4. Click **Create**. A JSON key file is downloaded to your computer.
5. Click **Close**.

For information on safeguarding the private key, see Best practices for managing credentials.

Configuring your API to support authentication

When you create an API config for your gateway, you specify a service account that your gateway uses to interact with other services. In order to enable service account authentication for services *calling* your gateway, modify the security requirement object and security definitions object in your API config. Following the steps below will enable API Gateway to validate the claims in the signed JWT used by calling services.

1. Add the service account as an issuer in your API config.

```
securityDefinitions:
  DEFINITION_NAME:
    authorizationUrl: ""
    flow: "implicit"
    type: "oauth2"
    x-google-issuer: "SA_EMAIL_ADDRESS"
    x-google-jwks_uri:
"https://www.googleapis.com/robot/v1/metadata/x509/SA_EMAIL_ADDRESS"
```

   - Replace `DEFINITION_NAME` with a string that identifies this security definition. You might want to replace it with the service account name or a name that identifies the calling service.
   - Replace `SA_EMAIL_ADDRESS` with the service account's email address.
   - You can define multiple security definitions in your API config, but each definition must have a different `x-google-issuer`. If you have created separate service accounts for each calling service, you can create a security definition for each service account, for example:

```
securityDefinitions:
  service-1:
    authorizationUrl: ""
    flow: "implicit"
    type: "oauth2"
    x-google-issuer: "service-1@example-project-
12345.iam.gserviceaccount.com"
    x-google-jwks_uri:
"https://www.googleapis.com/robot/v1/metadata/x509/service-1@example-project-
12345.iam.gserviceaccount.com"
  service-2:
    authorizationUrl: ""
    flow: "implicit"
    type: "oauth2"
    x-google-issuer: "service-2@example-project-
12345.iam.gserviceaccount.com"
    x-google-jwks_uri:
"https://www.googleapis.com/robot/v1/metadata/x509/service-2@example-project-
12345.iam.gserviceaccount.com"
```

2. Optionally, add `x-google-audiences` to the `securityDefinitions` section. If you don't add `x-google-audiences`, API Gateway requires that the `"aud"` (audience) claim in the JWT is in the format `https://SERVICE_NAME`, where *SERVICE_NAME* is the name of your API Gateway service, which you have configured in the `host` field of your OpenAPI document.

3. Add a `security` section at either the top level of the file (not indented or nested) to apply to the entire API, or at the method level to apply to a specific method. If you use `security` sections at both the API level and at the method level, the method-level settings override the API-level settings.

   ```
   security:
     - DEFINITION_NAME: []
   ```

   - Replace `DEFINITION_NAME` with the name that you used in the `securityDefinitions` section.
   - If you have more than one definition in the `securityDefinitions` section, add them in the `security` section, for example:

     ```
     security:
       - service-1: []
       - service-2: []
     ```

4. Deploy your updated API config.

Before API Gateway forwards a request to your API, API Gateway verifies:

- The signature of the JWT by using the public key, which is located at the URI specified in the `x-google-jwks_uri` field in your API config.
- That the `"iss"` (issuer) claim in the JWT matches the value specified in the `x-google-issuer` field.
- That the `"aud"` (audience) claim in the JWT contains your API Gateway service name or matches one of the values that you specified in the `x-google-audiences` field.
- That the token isn't expired by using the `"exp"` (expiration time) claim.

For more information about `x-google-issuer`, `x-google-jwks_uri`, and `x-google-audiences`, see OpenAPI extensions.

Making an authenticated request to an API Gateway API

To make an authenticated request, the calling service sends a JWT signed by the service account that you specified in the API config. The calling service must:

1. Create a JWT and sign it with the service account's private key.
2. Send the signed JWT in a request to the API.

The following sample code demonstrates this process for select languages. To make an authenticated request in other languages, reference jwt.io for a list of supported libraries.

1. In the calling service, add the following function and pass it the following parameters:
   JavaPythonGo
   - `saKeyfile` : The full path to the service account's private key file.
   - `saEmail` : The service account's email address.
   - `audience` : If you added the `x-google-audiences` field to your API config, set `audience` to one of the values that you specified for `x-google-audiences` . Otherwise, set `audience` to `https://SERVICE_NAME` , where `SERVICE_NAME` is your API Gateway service name.
   - `expiryLength` : The JWT expiration time, in seconds.

   The function creates a JWT, signs it by using the private key file, and returns the signed JWT.

   JavaPythonGo

```
// generateJWT creates a signed JSON Web Token using a Google API Service
Account.
func generateJWT(saKeyfile, saEmail, audience string, expiryLength int64)
(string, error) {
        now := time.Now().Unix()
        // Build the JWT payload.
        jwt := &jws.ClaimSet{
                Iat: now,
                // expires after 'expiryLength' seconds.
                Exp: now + expiryLength,
                // Iss must match 'issuer' in the security configuration in
your
                // swagger spec (e.g. service account email). It can be any
string.
                Iss: saEmail,
                // Aud must be either your Endpoints service name, or match
the value
                // specified as the 'x-google-audience' in the OpenAPI
document.
                Aud: audience,
                // Sub and Email should match the service account's email
address.
                Sub:           saEmail,
                PrivateClaims: map[string]interface{}{"email": saEmail},
        }
        jwsHeader := &jws.Header{
                Algorithm: "RS256",
                Typ:       "JWT",
        }
        // Extract the RSA private key from the service account keyfile.
        sa, err := ioutil.ReadFile(saKeyfile)
        if err != nil {
                return "", fmt.Errorf("Could not read service account file:
%v", err)
        }
        conf, err := google.JWTConfigFromJSON(sa)
        if err != nil {
                return "", fmt.Errorf("Could not parse service account JSON:
%v", err)
        }
        block, _ := pem.Decode(conf.PrivateKey)
        parsedKey, err := x509.ParsePKCS8PrivateKey(block.Bytes)
        if err != nil {
                return "", fmt.Errorf("private key parse error: %v", err)
        }
        rsaKey, ok := parsedKey.(*rsa.PrivateKey)
        // Sign the JWT with the service account's private key.
        if !ok {
                return "", errors.New("private key failed rsa.PrivateKey type
assertion")
        }
        return jws.Encode(jwsHeader, jwt, rsaKey)
}
```

2. In the calling service, add the following function to send the signed JWT in the `Authorization: Bearer` header in the request to the API:
   Java Python Go

```
// makeJWTRequest sends an authorized request to your deployed endpoint.
func makeJWTRequest(signedJWT, url string) (string, error) {
        client := &http.Client{
                Timeout: 10 * time.Second,
        }        req, err := http.NewRequest("GET", url, nil)
        if err != nil {
                return "", fmt.Errorf("failed to create HTTP request: %v",
err)
        }
        req.Header.Add("Authorization", "Bearer "+signedJWT)
        req.Header.Add("content-type", "application/json")        response,
err := client.Do(req)
        if err != nil {
                return "", fmt.Errorf("HTTP request failed: %v", err)
        }
        defer response.Body.Close()
        responseData, err := ioutil.ReadAll(response.Body)
        if err != nil {
                return "", fmt.Errorf("failed to parse HTTP response: %v",
err)
        }
        return string(responseData), nil
}
```

When you send a request by using a JWT, for security reasons, we recommend that you put the authentication token in the `Authorization: Bearer` header. For example:

```
curl --request POST \
  --header "Authorization: Bearer ${TOKEN}" \
  "${GATEWAY_URL}/echo"
```

where `GATEWAY_URL` and `TOKEN` are environment variables containing your deployed gateway URL and authentication token, respectively.

Receiving authenticated results in your API

API Gateway usually forwards all headers it receives. However, it overrides the original `Authorization` header when the backend address is specified by `x-google-backend` in the API config.

API Gateway will send the authentication result in the `X-Apigateway-Api-Userinfo` to the backend API. It is recommended to use this header instead of the original `Authorization` header. This header is `base64url` encoded and contains the JWT payload.

What's next

Managing API access