Session 2: Zero Trust API Access based on OAuth

James Lee | Security Solutions Architect | F5
OAuth 2.0
What is OAuth 2.0?

AN OPEN STANDARD FOR AUTHORIZATION

- OAuth 2.0 is a delegation protocol.
- It allows a user to authorize access to a resource without sharing their credentials.
- The user who owns a resource can allow a 3rd party application to access the user’s resource on their behalf in the OAuth framework.
- It is the most widely supported authorization/authentication(with OIDC extension) framework.
Types of API

**External APIs**
- Authorized clients connecting from external. Sometimes indicates the API calls from internal apps to external 3rd party APIs.

**Partner APIs**
- Available to selected and authorized external developers or API consumers.

**Internal APIs**
- Use within the enterprise to connect systems and data within the business.
- Normally, it represents the App-to-App API calls or API calls from developers.
Layered Defense for API Security - Gartner
Zero Trust for User-to-App
API Security
Gartner’s API G/W design Architecture

Figure 3: Enterprise Architecture of API Gateways

Source: Gartner
777062_C
Two-tier API G/W Design

Coarse-grained API Access Control
- Eliminate the weak cryptography algorithm
- Authenticate the user and enforce basic control. (e.g. Control HTTP methods based on group info. / Deny access to the admin API endpoints from external groups.)
- Enable mTLS as an additional authentication methods for partners.
- Blacklist for suspicious JWT list.
- Enforce OpenAPI spec.

Enterprise API G/W

Departmental API G/W

Fine-grained API Access Control
- Enforce the object ID comparison for the specific API endpoints. (Preventing BOLA)
- Checking detailed user’s privileges and enforcing the micro control.
- API-specific control policy can be enforced. (e.g. Max file-size limit for the file-uploading API endpoint.)
Zero Trust API Security Demo for User-to-App API Access

Secure Token Translation
Opaque Token (By-reference Token)

- Opaque Token doesn't contain any information about the grant
- The Client or Resource Server has to ask the Authorization Server every time it wants to know the validity or scope data for an Access Token
- This allows tokens to be revoked before they expire

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"bearer",
    "expires_in":3600,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
1. User delegates the access to the 3rd party S/W (OAuth client). In this example, 3rd party S/W is ‘Postman’.

2. Postman a ‘authorize’ the OAuth AS.

3. The OAuth AS authenticates the user.

4. The OAuth AS redirects the user back to the OAuth Client app with ‘authorization code’.

5. Postman(OAuth client) exchanges the authorization code to the access token (Opaque token).

6. Postman(OAuth client) initiates the API access to the OAuth resource server with the opaque token.

7. Token validation & userinfo extraction at the API G/W level

8. If the token is valid, OAuth RS(API G/W) forwards the request to API endpoints.

9. Token validation & userinfo extraction at the code level

OAuth Token Flow – Opaque token
JWT Token (By-Value Token)

**Encoded**
PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFiZXI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMekKF2QT4fwpMeJf36P0k6yJV_adQssw5c

**Decoded**
EDIT THE PAYLOAD AND SECRET

**HEADER: ALGORITHMS & TOKEN TYPE**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

**PAYLOAD: DATA**

```
{
  "sub": "1234567898",
  "name": "John Doe",
  "iat": 1516239022
}
```

**VERIFY SIGNATURE**

```
HMACSHA256(
  base64UrlEncode(header) + "," +
  base64UrlEncode(payload),
  your-256-bit-secret
) □ secret base64 encoded
```
OAuth Token Flow – JWT

1. User delegates the access to the 3rd party S/W (OAuth client). In this example, 3rd party S/W is ‘Postman’.

2. Postman access to the ‘authorize’ endpoint of the OAuth AS.

3. The OAuth AS authenticates the user.

4. The OAuth AS redirects the user back to the OAuth Client app with ‘authorization code’.

5. Postman (OAuth client) exchanges the authorization code to the access token (JWT).

6. Postman (OAuth client) initiates the API access to the OAuth resource server with the JWT.

7. Token validation and forwards the request to API endpoints.

8. Token validation & extract the data from the token directly.
### Opaque Token VS Transparent Token

<table>
<thead>
<tr>
<th>Opaque Token</th>
<th>Transparent Token (JWT Token)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Also called as ‘By-Reference’ token.</td>
<td>• Also called as ‘By-Value’ token.</td>
</tr>
<tr>
<td>• Randomly generated identifier.</td>
<td>• It contains real user data.</td>
</tr>
<tr>
<td>• Proprietary format.</td>
<td>• Anyone can inspect the content.</td>
</tr>
<tr>
<td>• Require validation from an ‘authorization server’.</td>
<td>• Self-validation.</td>
</tr>
<tr>
<td>• Can be revoked.</td>
<td></td>
</tr>
<tr>
<td>• Does not contain any real user data.</td>
<td></td>
</tr>
</tbody>
</table>

More secure but less scalable. Aligned with Zero trust Concept [*Never trust, Always verify*]

More scalable, faster but less secure.
1. User delegates the access to the 3rd party S/W (OAuth client). In this example, 3rd party S/W is ‘Postman’.

2. Postman access to the ‘authorize’ endpoint of the OAuth AS.

3. The OAuth AS authenticates the user.

4. The OAuth AS redirects the user back to the OAuth Client app with ‘authorization code’.

5. Postman (OAuth client) exchanges the authorization code to the access token (Opaque token).

6. Postman (OAuth client) initiates the API access to the OAuth resource server with the opaque token.

7. OAuth resource server (API G/W) checks the validity of the token with OAuth AS.

8. If the token is valid, OAuth RS (API G/W) extracts the required userinfo from the OAuth AS, and generates a new JWT for internal consumption only.

9. Code (API endpoint) can consume the data from the internal JWT.
Secure Token Translation – Token revocation

1. Postman (OAuth client) initiates the API access to the OAuth resource server with the opaque token (compromised token).

2. OAuth resource server (API G/W) checks the validity of the token with OAuth AS.

3. Security team easily can revoke the compromised token in the OAuth AS, because all token needs to be validated in the OAuth AS.

Opaque Token (OAuth access token)