Article

Implementing OAuth for the Apple Ads API

Manage secure access to Ads accounts.

Overview

The Apple Ads Campaign Management API supports OAuth 2. With OAuth 2, users authenticate with credentials in exchange for an access token to make authenticated requests to Apple Ads API. OAuth 2 replaces key and certificate credentials authentication in previous versions of the API.

An immediate advantage of updating the API and using OAuth 2 is the ability to manage access to accounts without requiring or sharing user login and password credentials.

This implementation process guides you through the following steps:

- Invite users with API permissions.
- Generate a private-public key pair.
- Extract a public key from your persisted private key.
- Upload a public key.
- Create a client secret.
- Request an access token.

Invite Users

Account administrators invite users with API permissions using the following process:

1. From Apple Ads, choose Sign In > Advanced and log in as an account administrator.
2. From the Users menu in the top-right corner, select the account to invite users to.
3. Choose Account Settings > User Management.

4. Click Invite Users to invite users to your Apple Ads organization.
5. In the User Details section, enter the user’s first name, last name, and Apple ID.
6. In the User Access and Role section, select an API user role. For non-API roles, see Invite users to your account.
7. Click Send Invite. The invited user receives an email with a secure code. The user signs into the secure Apple URL in the email and inputs the provided code, which activates the user’s account.

Generate a Private Key

API users need to create a private key. If you’re using MacOS or a UNIX-like operating system, OpenSSL works natively. If you’re on a Windows platform, you need to download OpenSSL.

```
openssl ecparam -genkey -name prime256v1 -noout -out private-key.pem
```

Parameter	Description
`-name`	`prime256v1` — the Elliptic Curve Digital Signature Algorithm (ECDSA) filename.
`-out`	The `.pem` filename where you generate and store the key pair.

The generated `private-key.pem` file resembles the following example:

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIKtnxllRY8nbndBQwT9we4pEULtjpW605iwvzLlKcBq4oAoGCCqGSM49
AwEHoUQDQgAEY58v74eQFyLtu5rtCpeU4NggVSUQSOcHhN744t0gWGc/xXkCSusz
LaZriCQnnqq4Vx+IscLFcrjBj+ulZzKlUQ==
-----END EC PRIVATE KEY-----
```

Important

Always keep your private key secure and never share it. If your private key becomes compromised, you need to re-create a private key and client secret, and reupload it to your Apple Ads account.

Extract a Public Key

Use the following command to extract a public key from your persisted private key:

```
openssl ec -in private-key.pem -pubout -out public-key.pem
```

Parameter	Description
`-in`	The private key filename: `private-key.pem`
`-out`	The `ec256-public-key` file where you generate and store the public key.

Open the `public-key.pem` file in a text editor and copy the public key, including the begin and end lines.

Upload a Public Key

Follow these steps to upload your public key:

1. From the Ads UI, choose Account Settings > API. Paste the key created in the above section into the Public Key field.
2. Click Save. A group of credentials displays as a code block above the public key field. Use your `clientId`, `teamId`, and `keyId` to create a client secret.

```
clientId SEARCHADS.aeb3ef5f-0c5a-4f2a-99c8-fca83f25a9
teamId SEARCHADS.hgw3ef3p-0w7a-8a2n-77c8-scv83f25a7
keyId a273d0d3-4d9e-458c-a173-0db8619ca7d7
```

You can make edits to the public key by choosing Account Settings > API > Edit.

3. Copy your private-public key pair into your working directory.

Create a Client Secret

A client secret is a JSON web token (JWT) that you create and sign using your private key. Your client secret authenticates token requests to the authorization server. Only you and the authorization server know the client secret.

The following example is a Python 3 script that generates, encodes, and signs the client secret using your private key. If you decide to create your own JWT using a different programming language and open source library, make sure the library you use supports elliptic curve methods.

Important

Make sure that you copy the private-public key pair into your working directory.

```
import os
import datetime as dt
from authlib.jose import jwt
from Crypto.PublicKey import ECC

private_key_file = "private-key.pem"
public_key_file = "public-key.pem"
client_id = "SEARCHADS.9703f56c-10ce-4876-8f59-e78e5e23a152"
team_id = "SEARCHADS.9703f56c-10ce-4876-8f59-e78e5e23a152"
key_id = "d136aa66-0c3b-4bd4-9892-c20e8db024ab"
audience = "https://appleid.apple.com"
alg = "ES256"

# Create the private key if it doesn't already exist.
if os.path.isfile(private_key_file):
    with open(private_key_file, "rt") as file:
        private_key = ECC.import_key(file.read())
else:
    private_key = ECC.generate(curve='P-256')
    with open(private_key_file, 'wt') as file:
        file.write(private_key.export_key(format='PEM'))

# Extract and save the public key.
public_key = private_key.public_key()
if not os.path.isfile(public_key_file):
    with open(public_key_file, 'wt') as file:
        file.write(public_key.export_key(format='PEM'))

# Define the issue timestamp.
issued_at_timestamp = int(dt.datetime.utcnow().timestamp())
# Define the expiration timestamp, which may not exceed 180 days from the issue time
expiration_timestamp = issued_at_timestamp + 86400*180

# Define the JWT headers.
headers = dict()
headers['alg'] = alg
headers['kid'] = key_id
```

```
# Define the JWT payload.
payload = dict()
payload['sub'] = client_id
payload['aud'] = audience
payload['iat'] = issued_at_timestamp
payload['exp'] = expiration_timestamp
payload['iss'] = team_id

# Open the private key.
with open(private_key_file, 'rt') as file:
    private_key = ECC.import_key(file.read())

# Encode the JWT and sign it with the private key.
client_secret = jwt.encode(
    header=headers,
    payload=payload,
    key=private_key.export_key(format='PEM')
).decode('UTF-8')

# Save the client secret to a file.
with open('client_secret.txt', 'w') as output:
     output.write(client_secret)
```

A client secret header describes the type of the token and the hashing algorithm it uses.

Key	Description
`alg`	The algorithm that signs the client secret. The value must be `ES256`.
`kid`	The value is your `keyId` that returns when you upload a public key.

The client secret includes a payload with claims.

Claim	Description
`aud`	The audience for the client secret. The value is: `https://appleid.apple.com`.
`exp`	The UNIX UTC timestamp of when the client secret expires. The value must be greater than the current date and time, and less than 180 days from the `iat` timestamp.
`iss`	The issuer of the client secret. The value is your `teamId`.

Claim	Description
`iat`	The UNIX UTC timestamp of when you create the client secret.
`sub`	The subject the client secret represents. The value is your `clientId`.

The following is an example of an unencoded payload and header encoded into a JWT:

```
// Header
{
"alg": "ES256",
"kid": "d136aa66-0c3b-4bd4-9892-c20e8db024ab"
}
// Payload
{
"iss": "SEARCHADS.9703f56c-10ce-4876-8f59-e78e5e23a152",
"iat": 2234567891,
"exp": 2234567900,
"aud": "https://appleid.apple.com",
"sub": "SEARCHADS.9703f56c-10ce-4876-8f59-e78e5e23a152"
}
```

The following is an example client secret:

```
eyJraWQiOiJiYWNhZWJkYS1lMjE5LTQxZWUtYTkwNy1lMmMyNWIyNGQxYjIiLCJhbGciOiJFUzI1NiJ9.
eyJpc3MiOiJEcmVhbWNvbXBhbnkiLCJhdWQiOiJBdXRoZW50aWNhdG9yIiwiZXhwIjoxNTcxNjcwNjIx
LCJuYmYiOjE1NzE2NjcwMjEsInN1YiI6Im11c3RlciIsImNsaWVudF9pZCI6ImFiY2QxMjM0IiwiYWRt
aW4iOiJ0cnVlIn0.s4C3p9kVNFeRAB5tChatC3ldQX07v9mG7thL7FeEO6cClfNuiaLSgq8f8ymbfO3O
QYW_KuwaA1KYRuoy1JmKk 4DBbYLcz6aoABe0pzI5Z_6wgMzAyqz8pQtwDAcd4Idoi8JdRbtzZce9o-0
nZiFA4hVAXqYwpEYC4UU8ZmJO_z8tY4juHPTV3nDugdtqyNnmAiBoLryOfGNngQZccdY1_QvkXS1y0bg1
a0k8cVVtnq- _93fYJIt9Z64CTvlH3uOeh7uaEv3nIxpXhvhkTySpUmY8e04TO09oTyZijiloByv3KFQ9
2OOJ8L 5N5_CeEc5p9LWjT1pcX8ATamOycZz2Q
```

Request an Access Token

The client credentials code grant authenticates with your credentials in exchange for an access token. To receive an access token, make a POST request from the token endpoint to the authorization server. When an access token expires, use the same process to obtain a new one.

```
curl -X POST \
-H 'Host: appleid.apple.com' \
-H 'Content-Type: application/x-www-form-urlencoded' \
'https://appleid.apple.com/auth/oauth2/token?grant_type=client_credentials&
client_id=SEARCHADS.27478e71-3bb0-4588-998c-182e2b405577&client_secret=eyJ0
eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.zI1NiIsImprdSI6Imh0dHBzOi8vYXV0aC5kZXYuYXB
pLnJpY29oL3YxL2Rpc2NvdmVyeS9rZXlzIiwia2lkIjoiMmIyZTgyMTA2NzkxZGM4ZmFkNzgxNW
Q3ZmI1NDRhNjJmNzJjMTZmYSJ9.eyJpc3MiOiJodHRwczovL2F1dGguZGV2LmFwaS5yaWNvaC8iL
CJhdWQiOiJodHRwczovL2lwcy5kZXYuYXBpLnJpY29oLyIsImlhdCI6MTQ5MDg1Mjc0MSwiZXhwI
joxNDkwODU2MzQxLCJjbGllbnRfaWQiOiI4ODQwMWU1MS05MzliLTQ3NzktYjdmNy03YzlmNGIzZj
kyYzAiLCJzY29wZSI6Imh0dHBzOi8vaXBzLmRldi5hcGkucmljb2gvdjEiLCJyaWNvaF9tc3Mi
OnsibWVkaWEiOnsicXVvdGEiOjEwLCJ0aHJvdHRsZSI6eyJ2YWx1ZSI6MCwid2luZG93IjowfX1
9fQ.jVq_c_cTzgsLipkJKBjAHzm8KDehW4rFA1Yg0EQRmqWmBDlEKtpRpDHZeF6ZSQfNH2OlrBW
FBiVDV9Th091QFEYrZETZ1IE1koAO14oj4kf8TCmhiG_CtJagvctvloW1wAdgMB1_Eubz9a8oim
cODqL7_uTmA5jKFx3ez9uoqQrEKZ51g665jSI6NlyeLtj4LrxpI9jZ4zTx1yqqjQx0doYQjBPhOB
06Z5bdiVyhJDRpE8ksRCC3kDPS2nsvDAal28sMgyeP8sPvfKvp5sa2UsH78WJmTzeZWcJfX2C2ba3
xwRMB5LaaVrQZlhj9xjum0MfDpIS1hJI6p5CHZ8w&scope=searchadsorg'
```

Request header	Description
`Host`	**Required**. `appleid.apple.com`
`Content-Type`	**Required**. `application/x-www-form-urlencoded`

Request parameter	Type	Description
`client_id`	String	**Required**. You receive your `clientId` when you upload a public key.
`client_secret`	String	**Required**. The client secret is a JWT that you create and sign with your private key.
`grant_type`	String	**Required**. The method to request authorization and get an access token. The value is `client_credentials`.
`scope`	String	**Required**. Defines the access permissions you request from the user, and limits the authorization of the access token you receive. The value is `searchadsorg`.

After accepting the credentials, the authorization server returns an access token.

```
{
"access_token":"eyJhbGciOiJkaXIiLCJlbmMiOiJBMjU2R0NNIiwia2lkIjpudWxsfQ
..lXm332TFi0u2E9YZ.bVVBvsjcavoQbBnQVeDiqEzmUIlaH9zLKY6rl36A_TD8wvgvWxp
yBXMQuhs-qWG_dxQ5nfuJEIxOp8bIndfLE_4a3AiYtW0BsppO3vkWxMe0HWnzglkFbKUHU
3PaJbLHpimmnLvQr44wUAeNcv1LmUPaSWT4pfaBzv3dMe3PNHJJCLVLfzNlWTmPxViIivQ
t3xyiQ9laBO6qIQiKs9zX7KE3holGpJ-Wvo39U6ZmGs7uK9BoNBPaFtd_q914mb9ChHAKc
QaxF3Gadtu_Z5rYFg.vD0iQuRwHGYVnDy27qexCw",
"token_type": "Bearer",
"expires_in": 3600,
"scope": "searchadsorg"
}
```

Response parameter	**Type**	**Description**
`access _token`	String	Your `access_token` is a requirement to make calls to Apple Ads Campaign Management API endpoints. See Calling the Apple Ads API. Your `access_token` is valid for the number of seconds that `expires_in` specifies.
`token_type`	String	The type of access token. The value is always `Bearer`.
`expires_in`	Integer	The token lifetime (TTL) of one hour (3600 seconds).
`scope`	String	Defines the access permissions you request from the user, and limits the authorization of the access token you receive.

See Also

Essentials

Calling the Apple Ads API

Pass your access token in the authorization header of HTTP requests.

Using Apple Ads API Functionality

Call endpoints using CRUD methods.