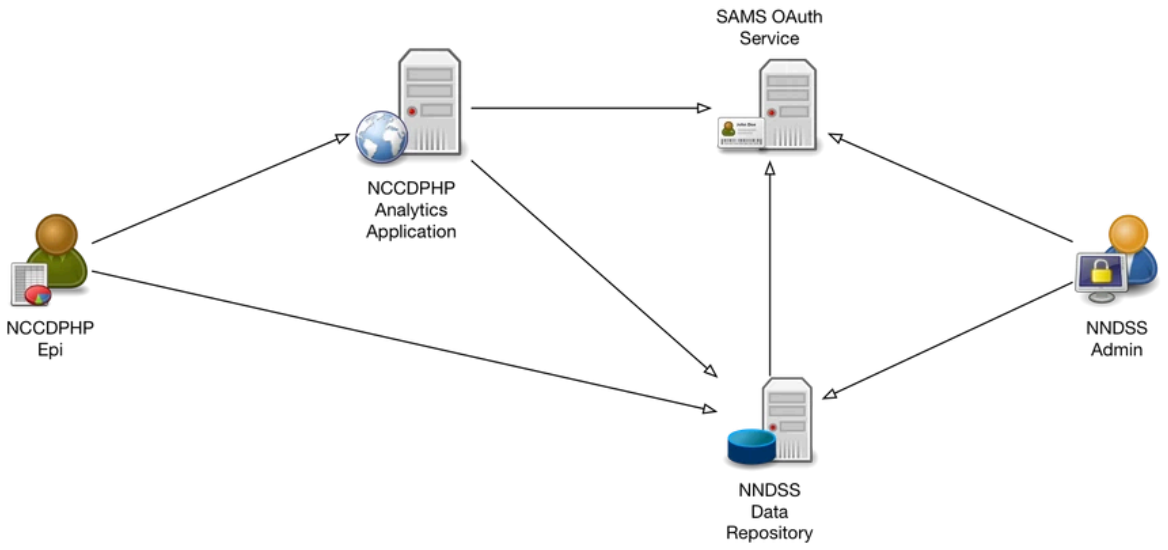
Using OAuth to Secure CDC APIs

Terminology

- **Protected Resource** - An OAuth term for information or functionality that requires permission to access. Examples of protected resources include existing data accessible via an API, or a particular API function such as creating a new repository in a data storage system.
- **Access Token** - a digital credential that is used to prove that a *client* is allowed to access a *protected resource*
- **Resource Server** (a.k.a. the API server) - An OAuth term for the server hosting the protected resources, capable of accepting and responding to *protected resource* requests that include *access tokens*.
- **Client** - An application making *protected resource* requests to a *resource server*. The term client does not imply any particular implementation characteristics (e.g. whether the application executes on a server, a desktop, or other devices).
- **Authorization Server** - An OAuth term for the server issuing access tokens to the *client* after successfully authenticating the *resource owner* and obtaining authorization.
- **Resource Owner** (a.k.a. the end-user) - An OAuth term for an entity capable of granting a *client* access to a *protected resource*. When the resource owner is a person, it is referred to as an end-user.
- **Resource Administrator** - A non-OAuth term for an entity capable of provisioning access to *protected resources* for *resource owners*.

Example



A CDC NCCDPHP epidemiologist is using NNDSS data on food poisoning in a study and was provisioned access to that data by an NNDSS administrator using SAMS. The NNDSS data is accessible via a service API supported by the NNDSS Data Repository system. The epidemiologist is using an analytics application hosted by NCCDPHP and, fortunately, this application can import data from the NNDSS Data Repository API. To facilate the connection from the analytics application to the data repository, the epidemiologist grants the analytics application access to the data using OAuth via an OAuth grant flow built into the analytics application.

In this example:

- The NNDSS food poisoning data is the *protected resource*
- The NNDSS administrator is the *resource administrator*
- The NNDSS data repository is the *resource server*
- The NCCDPHP epidemiologist is the *resource owner*
- The NCCDPHP analytics application is the *client*
- The SAMS OAuth Service is the *authorization server*

Note the counterintuitive use of the term *resource owner*. In OAuth parlance a *resource owner* does not need to "own" the *protected resource* in the traditional sense, instead they need to be able grant a *client* access to the *protected resource*. In this example the NCCDPHP epidemiologist is a *resource owner* from the OAuth perspective even though the protected resource (the NNDSS food poisoning data) belongs to the NNDSS program.

High Level Overview of OAuth

1. The client obtains an access token from the authorization server using one of the grant flows described below. In most cases this requires interaction between the client, resource owner and authorization server to grant access to the client.
2. The client makes requests to the resource server for access to protected resources including the access token to prove their right to access the protected resoure
3. The resource server validates the access token to ensure only permitted clients can access protected resources. Depending on the type of access token the resource server may validate the access token directly or defer to the authorization server for validation.

Obtaining an Access Token

Each of the following subsections describes a process, defined by the OAuth 2 specification, that a client can use to obtain an access token. Once a client has an access token it can make requests to a resource server to obtain access to protected resources.

Authorization Code Grant

See: https://tools.ietf.org/html/rfc6749#section-4.1

```
+----------+
| Resource |
|   Owner  |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier      +---------------+
|         -+----(A)-- & Redirection URI ---->|               |
|  User-   |                                 | Authorization |
|  Agent  -+----(B)-- User authenticates --->|     Server    |
|          |                                 |               |
|         -+----(C)-- Authorization Code ---<|               |
+-|----|---+                                 +---------------+
  |    |                                         ^      v
 (A)  (C)                                        |      |
  |    |                                         |      |
  ^    v                                         |      |
+---------+                                      |      |
|         |>---(D)-- Authorization Code ---------'      |
|  Client |          & Redirection URI                  |
|         |                                             |
|         |<---(E)----- Access Token -------------------'
+---------+       (w/ Optional Refresh Token)
```

Note: The lines illustrating steps (A), (B), and (C) are broken into two parts as they pass through the user-agent.

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier.

(D) The client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. The client includes the redirection URI used to obtain the authorization code for verification.

(E) The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step (C). If valid, the authorization server responds back with an access token and, optionally, a refresh token.

Properties of this flow:

- Client is an application that makes requests to the resource server with authorization from the resource owner.
- Client needs to pre-register with the Authorization server (possibly via dynamic client registration protocol)
- Client does not need access to resource owners credentials
- Resource owner interacts with Authorization server directly (e.g. via a browser redirect from Web application or by following a link provided by a desktop client application) to authorize client access to protected resources
- Client is provided with an authorization code initially that it exchanges for an access code for use in resource access requests

Implicit Grant

See: https://tools.ietf.org/html/rfc6749#section-4.2

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier     +---------------+
|         -+----(A)-- & Redirection URI --->|               |
|  User-   |                                | Authorization |
|  Agent  -|----(B)-- User authenticates -->|     Server    |
|          |                                |               |
|          |<---(C)--- Redirection URI ----<|               |
|          |          with Access Token     +---------------+
|          |            in Fragment
|          |                                +----------------+
|          |----(D)--- Redirection URI ---->|   Web-Hosted   |
|          |          without Fragment      |     Client     |
|          |                                |    Resource    |
|     (F)  |<---(E)------- Script ---------<|                |
|          |                                +----------------+
+-|--------+
  |    |
 (A)  (G) Access Token
  |    |
  ^    v
+---------+
|         |
|  Client |
|         |
+---------+
```

Note: The lines illustrating steps (A) and (B) are broken into two parts as they pass through the user-agent.

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.

(D) The user-agent follows the redirection instructions by making a request to the web-hosted client resource (which does not include the fragment per [RFC2616]). The user-agent retains the fragment information locally.

(E) The web-hosted client resource returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the user-agent, and extracting the access token (and other parameters) contained in the fragment.

(F) The user-agent executes the script provided by the web-hosted client resource locally, which extracts the access token.

(G) The user-agent passes the access token to the client.

Properties of this flow:

- Client is an application that makes requests to the resource server with authorization from the resource owner.
- Similar to Authorization Code Grant but omits the use of a client secret and authorization code.
- Suitable for single page Web application where all code is readily accessible (hence nowhere to put client secret).
- Involves additional security considerations since the access token is exchanged in the redirect URI fragment and there is no mechanism to tie an access token to a particular client.

Resource Owner Credentials Grant

See: http://tools.ietf.org/html/rfc6749#section-4.3

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     v
     |    Resource Owner
    (A) Password Credentials
     |
     v
+---------+                                  +---------------+
|         |>--(B)---- Resource Owner ------->|               |
|         |         Password Credentials     | Authorization |
| Client  |                                  |     Server    |
|         |<--(C)---- Access Token ---------<|               |
|         |    (w/ Optional Refresh Token)   |               |
+---------+                                  +---------------+
```

(A) The resource owner provides the client with its username and password.

(B) The client requests an access token from the authorization server's token endpoint by including the credentials received from the resource owner. When making the request, the client authenticates with the authorization server.

(C) The authorization server authenticates the client and validates the resource owner credentials, and if valid, issues an access token.

Properties of this flow:

- Client is an application that makes requests to the resource server with authorization from the resource owner.
- Client needs to be given the resource owner's credentials
- Suitable for use with trusted applications only
- Defeats the purpose of OAuth to some extent

Client Credentials Grant

See: http://tools.ietf.org/html/rfc6749#section-4.4

```
+---------+                                  +---------------+
|         |                                  |               |
|         |>--(A)- Client Authentication --->| Authorization |
| Client  |                                  |     Server    |
|         |<--(B)---- Access Token ---------<|               |
|         |                                  |               |
+---------+                                  +---------------+
```

(A) The client authenticates with the authorization server and requests an access token from the token endpoint.

(B) The authorization server authenticates the client, and if valid, issues an access token.

- Client is an application that acts on its own behalf
- No resource owner interaction

Types of Access Token

OAuth 2 tokens are typically used as bearer tokens. Anyone in possession of a bearer token can use it to access the associated protected resources. There are two primary variants of bearer tokens in use today: opaque and transparent tokens.

Opaque Tokens

An opaque token is a string of alphanumeric digits that only has meaning to the authorization server that issued it. Use of opaque tokens has the following implications:

- A resource server will need to validate opaque tokens by interacting with the authorization server since the token contains no information (discernable to the resource server) that would enable the resource server to validate it without reference to the authorization server.
- A resource server that needs to access claims (user identity, roles, etc) associated with an opaque token will need to exchange the opaque token for a transparent token first (via interaction with the authorization server).
- An opaque token does not contain potentially sensitive information.
- The need to validate opaque tokens introduces a central control point where tokens can be invalidated when necessary.
- To improve scalability it may be necessary to introduce caching of token validation results at resource servers

Transparent Tokens

A transparent token is a structured string of alphanumeric digits that can be interpreted by clients and resource servers. A transparent token includes cryptographic proof (typically a digital signature) of validity. Use of transparent tokens has the following implications:

- A resource server can validate a transparent token directly without reference to the authorization server that issued it.
- A resource server can obtain the claims associated with a transparent token directly.
- A transparent token may contain potentially sensitive information.
- Token invalidation requires interaction with the authorization server; even though a transparent token can be validated directly by the resource server it is still recommended that tokens are validated regularly, particularly long-lived tokens
- It is possible to encrypt transparent tokens for the resource server so that the client sees the token as an opaque token by the resource server can use it as a transparent token. This does introduce additional complexity in key management etc.

JSON Web Tokens (JWT) are the primary form of transparent access token in use today.

Mapping CDC Use Cases to OAuth Flows

The following subsections describe scenarios where a client (Web application, desktop application, mobile application) accesses a CDC-owned resource server (Web service) via a Web API. In all cases it is assumed that the resource server is hosted within the CDC network (physical or virtual on premise, AWS ACCS or ECPaaS cloud) and that the resource owner has CDC-issued credentials.

CDC Client Acting on Behalf of CDC User

Resource Owner: CDC staff or contractor with CDC credentials

Resource Server: CDC service providing functionality of interest

Client: CDC-owned end-user oriented application (Web, desktop or mobile)

Authorization Server: CDC SAMS OAuth Service

Grant Type: Authorization Code Grant flow

Access Token Type: JWT for Web and desktop clients; opaque token for mobile applications that operate off the CDC network

CDC Client Acting on Behalf of External User

Resource Owner: External partner with SAMS credentials

Resource Server: CDC service providing functionality of interest

Client: CDC-owned end-user oriented application (Web, desktop or mobile)

Authorization Server: CDC SAMS OAuth Service

Grant Type: Authorization Code Grant flow

Access Token Type: JWT for CDC-hosted Web clients; opaque token for Web, desktop and mobile applications that operate off the CDC network

External Client Acting on Behalf of External User

Resource Owner: External partner with SAMS credentials

Resource Server: CDC service providing functionality of interest

Client: Partner-owned end-user oriented application (Web, desktop or mobile)

Authorization Server: CDC SAMS OAuth Service

Grant Type: Authorization Code Grant flow

Access Token Type: opaque token for Web, desktop and mobile applications

Autonomous CDC Client

Resource Owner: N/A

Resource Server: CDC service providing functionality of interest

Client: CDC-owned autonomous server application

Authorization Server: CDC SAMS OAuth Service

Grant Type: Resource Owner Credentials Grant flow

Access Token Type: JWT

Autonomous External Client

Resource Owner: N/A

Resource Server: CDC service providing functionality of interest

Client: Partner-owned autonomous server application

Authorization Server: CDC SAMS OAuth Service

Grant Type: Resource Owner Credentials Grant flow

Access Token Type: opaque

Authenticating Chained Protected Resource Requests

A resource server will sometimes need to access protected resources hosted on other resource servers to complete its work, these are referred to as chained protected resource requests. When this happens it is often important to maintain the security context of the original protected resource request. Use of bearer access tokens facilitates this since, as described above, anyone in possession of a bearer token can use it to make protected resource requests. The original resource server can include the same access token in chained protected resource requests; the downstream resource servers can validate the token and use it for accounting/attribution purposes.

For efficiency, it is recommended that opaque tokens provided with the original protected resource request are exchanged for JWTs prior to making chained protected resource requests.

CDC OAuth Infrastructure

CDC's Secure Access Management Service (SAMS) incorporates an OAuth 2 Authorization Server that can be used to secure CDC APIs. SAMS identity services support both CDC staff, external partners, and system accounts for autonomous applications.

Full details of SAMS support for OAuth is available from the SAMS team.

Registering Clients and Resource Servers

Both Client and API developers will need to submit a "SAMS Application Configuration Worksheet" to the SAMS team in order to have their applications and services registered with SAMS. The worksheet includes program and organizational information in addition to specific OAuth information such as client call-back URLs and service API URLs. Once registered, the SAMS team will provide any required information including authorization server endpoints (for user information, token issuance, token renewal and token validation) and, for clients, client credentials (client id and secret).

Implementing OAuth Grant Flows

It is recommended that clients and resource servers use existing OAuth libraries rather than re-implementing OAuth flows and token processing from scratch. RFCs 6749 and 6819 include extensive security considerations important to implementing OAuth in a secure manner, use of an existing, well regarded, OAuth library is a good first step to ensuring that OAuth is implemented in a secure fashion. A list of OAuth implementations is maintained on the OAuth Web site.

Authorizing Resource Requests

Authorization decisions can be based upon one or more of the following:

- Resource owner attributes
- Resource owner roles
- Access token scopes

It is the responsibility of each service owner to develop a suitable authorization approach and to work with the SAMS team to ensure that the metadata needed for authorization decisions is available via the appropriate OAuth flows.

References

The OAuth 2.0 Authorization Framework: https://tools.ietf.org/html/rfc6749

A Guide to OAuth 2.0 Grants: https://alexbilbie.com/guide-to-oauth-2-grants/

Diagrams And Movies Of All The OAuth 2.0 Flows: https://medium.com/@darutk/diagrams-and-movies-of-all-the-oauth-2-0-flows-194f3c3ade85

JSON Web Token (JWT): https://tools.ietf.org/html/rfc7519