Learn OAuth 2.0 (Udemy) Notes

Sources

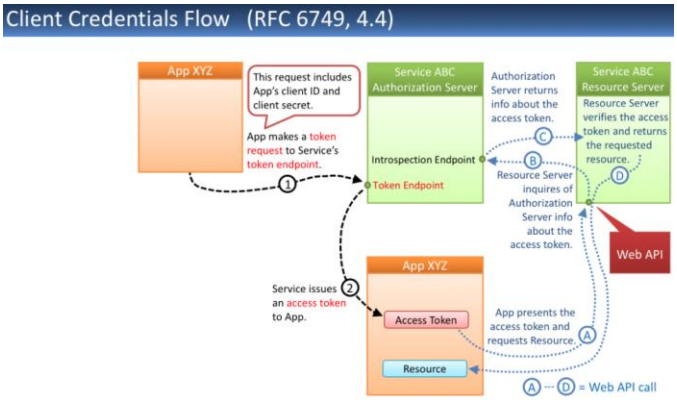
- [Udemy] Learn OAuth 2.0 (Link)

Section 1: Introduction

- OAuth actors work together in an OAuth interaction or flow.
    - Various actors play a specific role in the OAuth interaction to make it secure.
- OAuth tokens are credentials and can be compared to a key to unlock something (e.g. a resource) that is securely stored.
- OAuth endpoints are web services or APIs that are hosted on an OAuth server.
- The components described above can be put together within the OAuth flow, of which there are four types.
- Major technology companies such as Google, LinkedIn, Facebook, and Paypal provide APIs that allow developers to use their resources programmatically.
    - The caveat is that the data is protected by OAuth and is only available to each user separately.

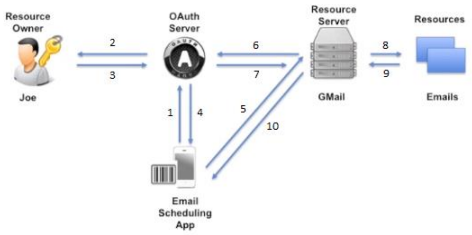
Section 2: OAuth Big Picture

- The resource owner is typically the user, which could be myself in the case where I want to access a resource such as my personal Spotify playlists.
    - A resource server exists in the middle, which provides access to the resources – in the case of Spotify, this would likely be the server that the API endpoints are hosted on.
    - The resource server also provides security and requires credentials provided by the resource owner to permit access.
- 3rd party applications may also want access to the resources through the resource server, and this is popular with 3rd party apps (such as Databasify, for instance).
    - These applications have been developed by a different company than that which offers the resource (e.g. Spotify).
    - One way of allowing a 3rd party app to access the resources, called the password antipattern, would be for the app to send the resource owner's actual password to the resource server – however, doing this would allow the 3rd party app to have unrestricted access to that service, along with others, without the resource owner's knowledge or consent.
    - No password that would allow access to a resource server should ever be provided to a 3rd party app.
- The solution that OAuth 2.0 provides is to have the 3rd party app contact an OAuth server, which sends a login screen to the resources owner so that user can enter their credentials.
    - The OAuth server then validates the password (or provides it to a 3rd party application such as an identity provider).
    - If the password was successfully validated for that resource owner, the OAuth server creates a token that has specific access rights for a subset of data and is only valid for a short period of time.
    - The token is simply a bunch of numbers which are then stored in a database along with a resource owner identifier, and are sent back to the 3rd party app.
    - OAuth 2 is a standard for delegating authorization for accessing resources via HTTP.



- Now when the app goes to access the resource server, it sends both the request for accessing resources along with the token that was generated by the OAuth server.
    - The resource server then checks back with the OAuth server to ensure that the token provided by the 3rd party app is valid (i.e. is it still active, has it been revoked, is it still valid, and what kind of access was granted for the token).

- If the OAuth server determines that the token is valid, it sends back an ok message to the resource server.
- The resource server then has its own interpretation of authorization on top of that.



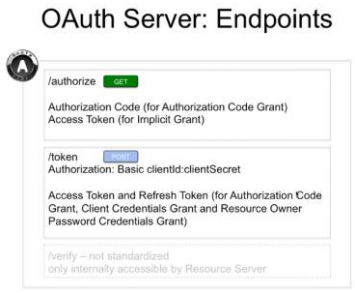
- If everything is ok, the resource server accesses the resource database for the requested information and returns it back to the 3rd party app.
- From the resource owner's (user's) perspective, nothing has really changed since they still need to enter their password once for the OAuth login screen.
  - However, the user may be able to have additional granular control over access to their data through prompts.
- Another advantage from a resource owner/user's perspective is that they can revoke long-standing tokens all at once (e.g. Spotify has this), say if they lose their device or forgot to log out on someone else's device.
- The OAuth API has huge reach within the tech community, and it is used by Google, Amazon, Spotify, etc.
  - OAuth 2.0 has existed since 2015 and has for the most part replaced OAuth 1.0 by this point.

Section 3: OAuth Components

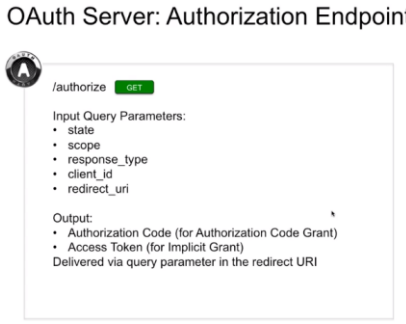
- OAuth is designed for distributed systems, and each actor in the OAuth flow takes on a role that makes sense in the overall OAuth solution.
  - An OAuth provider is an actor and is also known as the OAuth server or authorization server.
  - A resource provider is a set of web APIs, and is also known as the resource server.
  - A resource owner is the user.
  - A client is usually a cloud app or a mobile app.

OAuth Server

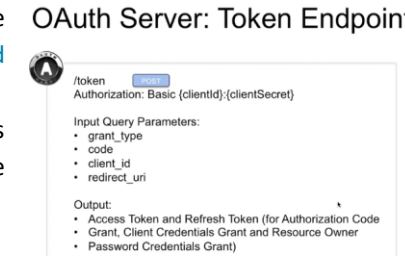
- The OAuth provider is composed of three components to make up the OAuth server.
  - The first component is an authentication component that provides a login page in the web browser for the user to authenticate through, along with the identity provider and identity access management infrastructure that is in the backend. OAuth is the frontend of the identity solution. A database of users is also held, which contains user information and credentials of the users for authenticating the user.
  - The second component is for requesting the authenticated user's consent for the delegation of access rights to the client. This is also called the consent server, and usually pops up after the authentication component has finished its work and the user is authenticated (i.e. logged in) to ask the user if they really want to provide the requested information to the 3rd party app. Both the login and the consent for access to the data happen in the authorization endpoint.
  - The third component is the token management infrastructure that contains another database that stores all the tokens that have been given out from the OAuth provider so that they can be later be requested for validation or even for reauthentication. Stored are token values and attributes – token values being the access and refresh token along with when they were created and how long they are valid for, and token attributes being additional information that is stored with each token.
  - Overall, the OAuth server manages the identity of all other OAuth actors, verifies the identities, creates tokens, and verifies tokens. There are generally two endpoints that the OAuth server provides – an authorization endpoint and a token endpoint.
- The OAuth server provides several endpoints for serving its clients.
  - The authorization endpoint is usually requested via a GET from the browser and allows you to both login and get a consent page. The result of the authorization endpoint is either an authorization code or an access token. These types of information go to the client via redirect.
    - The state parameter is optional and is for dealing with simultaneous requests from the same client. The client can use this random string parameter to match requests with responses.
    - Scope is another optional parameter that specifies the type of resources that are requested by the client.



        - Response type is a mandatory parameter that tells you which type of response you want to receive, such as an authorization code response (i.e. authorization code flow), or a token (i.e. implicit flow).
        - The client ID is a mandatory parameter and is the identification of the client that was registered with the OAuth server by the client prior to the OAuth interaction.
        - The redirect URI is the redirect endpoint which the response from the authorization endpoint is sent to. It is a client URI which can receive the output produced by the OAuth authorization endpoint.
        - The output depends on whether an authorization code for the authorization code grant, or access token for the implicit grant. This depends on the response type that was provided as an input query parameter. The output values are delivered as query parameters in the redirect URI, attached to the end of the URI itself.



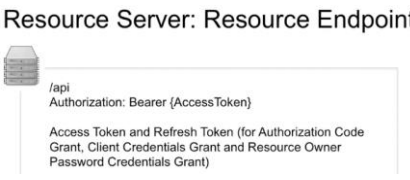
    - The token endpoint requires a POST and is a protected endpoint that requires a Client ID and Client Secret to access it. It is protected by HTTP Basic Access Authentication and will require an Authorization: Basic {clientID}:{clientSecret} header to be provided along with the request. An access token and refresh token are produced and returned to the client. Note that an implicit grant is missing since this grant does not use the token endpoint to obtain a token (i.e. everything is handled on the authorization endpoint for the implicit grant).
        - The grant type input query parameter indicates whether you have authorization code / client credentials or resource owner password credentials.
        - The code input query parameter is only required if the grant type is authorization code and contains the code that was generated by the authorization endpoint previously.
        - Both the client ID and the redirect URI are required parameters.
        - The output is an access token and refresh token for all flows, expect for the implicit grant flow which does not use the token endpoint.



    - While these are standardized token endpoints in terms of the type of service they provide, the URL path is not standardized (e.g. /authorize might be /auth, and /token may be something else).
    - Another endpoint that is not standardized but exists in every implementation of the OAuth server is the verify endpoint, which handles token verification for the resource server. This endpoint exists to tell the resource server whether a token that has been provided to the resource server is valid for authorizing a user. This endpoint is never accessible from the outside since that would allow for certain kinds of attacks, and hence is only accessible internally by the resource server.

Resource Server

- The resource server (or resource provider) holds the resources and makes the protected resources available via the HTTP protocol.
    - The resource may be data or a service and is often offered in the form of a web API, which in turn offers the protected data.
    - The web (REST) API has a URL that the client would go to for the resource endpoint, which would return the protected resource if the client was authorized to receive it – otherwise providing an error message. The protected resource would be returned only if a valid OAuth token (the more common case) or valid credentials (i.e. password) were provided. The endpoint is protected by the Authorization: Bearer {AccessToken}.



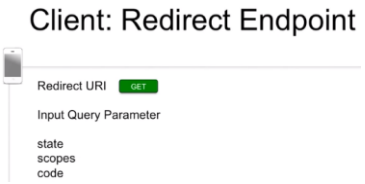
    - To determine whether a client is permitted to access and receive the protected data, the web API would require an OAuth access token to be sent as part of the request.
    - One of the first things that the web API needs to do once a request is received is to check the validity of a provided access token.

Resource Owner

- The resource owner (or user) is the owner of the resources that are hosted on the resource server.
    - An example could be a private playlist access by me, as a resource owner, in the Spotify example.
    - The resource owner owns the resources and holds credentials that allow him to get access to the resource.
    - The resource owner delegates his right to access resources to a 3rd party app, such as a mobile application.

Client

- The client is the fourth actor in the OAuth flow, which is an application that attempts to access a protected resource.
    - The client can be a mobile, web, or a cloud app, and this app gets and holds access tokens and refresh tokens received from the OAuth server and uses them along with subsequent requests to the resource server for resources.
    - The tokens can be reused, which is why the client stores these tokens for their lifetime.
    - The client should never hold the password of the resource owner for access to the resources (i.e. the password antipattern), but rather should only hold tokens in place of a password.
    - The client is identified by a Client ID and a Client Secret, and this identity is provided by the OAuth server by a registration process that the client must go through prior to being able to use OAuth.
    - A redirect endpoint is provided by the client and is the endpoint where we send information from the authorization endpoint back to the client. This endpoint is reachable under the redirect URI.
        - The state, scope, and code created by the authorization endpoint are received by this redirect endpoint.
        - The code will then be used to get an access token from the token endpoint.



- In some scenarios, a software component can have roles of several actors, such as when Spotify is both the OAuth provider and the resource provider.
- It should be noted that OAuth tokens have validity, and whoever holds them is able to access a resource which presents a security risk.

Tokens

- Tokens are identifiers that are used in all OAuth flows, and the holder of the token has the access rights that are associated with the token.
    - It is therefore important to keep the access or refresh tokens confidential.
- One way to keep access tokens confidential is to use Transport Layer Security (TLS) for all communication involving tokens.
- Information about the tokens is stored in a database in the OAuth Provider.
    - Tokens are unique long strings of random characters and do not contain any user or identifying data.
- Access tokens are used by the client to access resources, and the client sends the access token to the resource server when requesting to access the protected resource.
    - The validity of the access token is limited to a short period of time (e.g. 24 hours or a month).
    - During the period of validity, the same access token can be used for any number of requests.
    - OAuth access tokens are bearer tokens since the holder of the token has the access rights associated with the token.
    - After the access token is created, the identity of the holder of the token is no longer checked which creates the need to keep access tokens confidential.
- Refresh tokens have a period of validity longer than that of access tokens (e.g. 30 days or 3 months).
    - These tokens can be used to request a new access token after the access token has expired without requiring the resource owner to provide their credentials again.
    - It is considered a shortcut for the authorization code flow, the client credential flow, and for the resource owner password credential flow.
    - Refresh tokens are stored and sent by the client and are sent to the token endpoint of the OAuth server.
    - They are never sent to the resource server, so we don't use refresh tokens in order to access a resource.
- The authorization code is created by the OAuth provider after successfully authenticating the resource owner and is sent back to the client.
    - The consent of the resource owner is provided, which allows resource access to be delegated to the client.

- The code is a confirmation of the successful authentication and the consent by the resource owner.
- The validity of the authorization code is usually limited to a couple of minutes, which is just enough time for the client to use the authorization code to request an access token from the token endpoint.
- The authorization code can only be processed by the token endpoint of the authorization server and it will never be sent to the resource provider.

Credentials

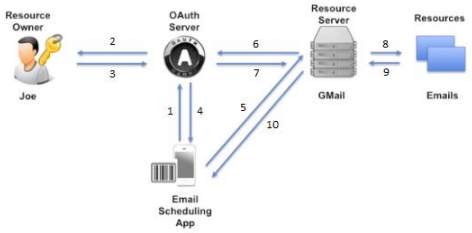
- Resource owner credentials allow the resource owner to get access to his resources.
  - These credentials can be given to the OAuth server, but never to the client.
- Client credentials are referred to as ClientID and ClientSecret.
  - They are used whenever a client accesses the token endpoint, which is protect by HTTP Basic Access Authentication.
  - The ClientID is also a query parameter for calling the authorization endpoint and the token endpoint.
  - ClientID needs to be requested for every new client from the OAuth provider – a process called client registration.
- The access token can also be seen as a credential for accessing resources.
  - Because it is a credential and a bearer token, it must not fall into the wrong hands and needs to be kept secret and protected.
  - It is important to have secure storage of the access token in the client.
- The refresh token cannot access resources, but you can get a new access token by using the refresh token.
  - These tokens should also be stored in a secure manner.
- The authorization code cannot be used to access the resource directly, but if it is fresh and still valid, it can be used to get an access token.

Client Registration

- Client registration is necessary whenever we have a new client which wants to have access to a resource.
  - The OAuth server would be provided with some information about the client before receiving back a ClientID and a Client Secret.
  - When registering a new client, a redirect URI and the required scopes will be provided to the OAuth provider.
- When you get granted client state by the OAuth server, a ClientID and ClientSecret will be provided back.

Section 4: OAuth Flows – Interactions Between the OAuth Components

Interaction of OAuth Components in an OAuth Flow



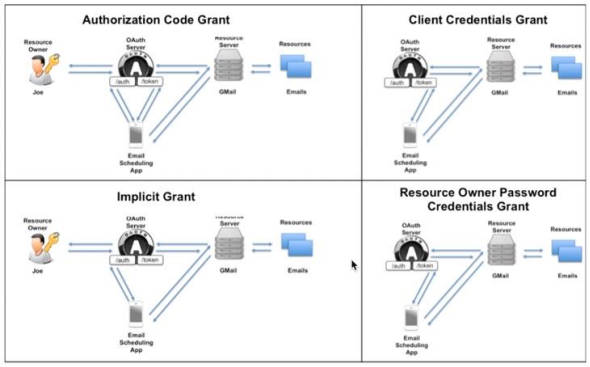
- In the first step, the 3rd party app does not contact the resource server directly but instead contacts the OAuth server **[1]**.
  - The client sends information to the OAuth server about the type of resources that it wants to have access to (e.g. emails for that user).
  - Some metadata is also provided to the OAuth server, such as a state parameter to correlate requests and responses for simultaneous requests. Note that this same state information is sent back in the response so that the client can easily match request to response.
- The OAuth server the receives the request through an authorization endpoint of the web API.
  - This authorization endpoint gives the resource owner the possibility to login and authenticate **[2]**.
  - The login form can be served directly from the OAuth Server, or more typically, is served by redirecting this request to some login service with connections to the identity provider, which renders a login page for the resource owner.
- The resource owner then sees the HTML page on the web browser asking for credentials to be entered, which are then sent back to the OAuth server **[3]**.
  - The OAuth server authenticates the user by contacting the backend server or identity provider.
  - If this is successful, the OAuth server creates an OAuth token (or authorization code), which is a string of randomly generated numbers and letters that are then associated with the identity information of the user in the database. Note that this identity information of the user does not leave the OAuth server.
- The OAuth token is then sent back to the client app that requested it **[4]**.

- The 3rd party application then uses this token and puts it into the header of a request that is sent to the resource server **[5]**.
- Since the resource server does not know whether the token is valid or not, it sends the OAuth token to the OAuth server for confirmation **[6]**.
    - The OAuth server compares the OAuth token string to that which is stored in its database for that ClientID.
    - If the OAuth token matches, the OAuth server sends back the 'ok' to the resource server **[7]**.
- With that confirmation, the resource server grabs the allowed resources from its database and sends it back to the client **[8 – 10]**.

Overview of OAuth Flows

- An excellent guide for determining which OAuth flow to use can be found at:
    - https://auth0.com/docs/authorization/which-oauth-2-0-flow-should-i-use
    - Note that native apps and SPAs should use an Authorization Code Flow with Proof Key for Code Exchange (PKCE) flow.
- For each OAuth flow there is an OAuth grant type.
    - OAuth flows describe the sequences of requests and responses exchanged among the OAuth actors and endpoints.
    - During these interactions, credentials and tokens are requested, delivered, verified, updated, and revoked.
- The underlying philosophy of the OAuth flows is to provide a high degree of security by checking the identity of each of the involved actors.
    - Depending on the requirements of the given user scenario, actors and endpoints can interact according to the rules set by the different flows.
- There are four typical OAuth flows, the first being the authorization code grant.
    - This is both the default and the most secure OAuth flow.
    - A prerequisite for using this flow is that the client has secure storage available for the ClientID and ClientSecret.
- The implicit grant is the second flow type that is used when a client cannot securely store the ClientID, ClientSecret, or OAuth tokens.
    - This is the case, for example, when the client is written in client-side JavaScript.
    - The limitations of this flow are that the generated tokens are short-lived, and that an access token cannot easily be refreshed.
    - Note that the implicit flow is not recommended (see https://blog.postman.com/pkce-oauth-how-to/) and has been replaced by the Authorization Code Flow with Proof Key for Code Exchange (PKCE) flow.

Overview of OAuth Flows



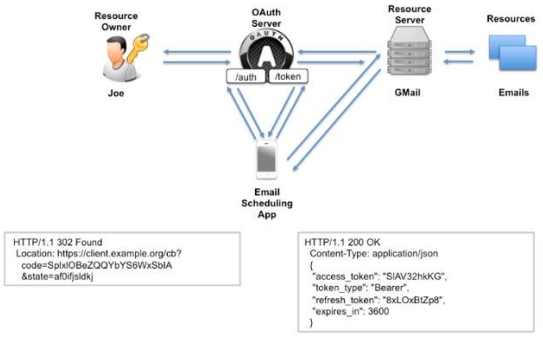
- Client credentials grant is used when the client is also the resource owner.
- The resource owner password credentials grant is used when the resource owner can entrust the password with the client.

Section 5: Authorization Code Flow

Description and Features

- The authorization code flow, also called three-legged OAuth, involves checking the identities of the three involved actors – the OAuth server, resource owner, and client.
- The OAuth server authenticates the resource owner by the username and password that are provided interactively by the resource owner.
- The OAuth server authenticates the client by his ClientID and ClientSecret, which are transmitted by the HTTP header.
    - The HTTP Basic Access Authentication mechanism is used, which reads the credentials from the HTTP field authorization.
- The identity of the OAuth server is checked by the certificate and by the URL.

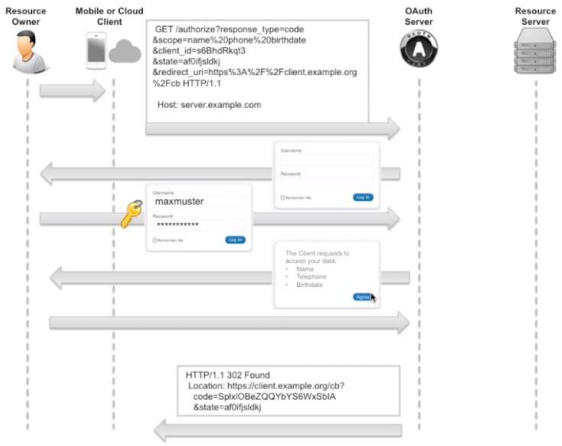
Authorization Code Grant



- This check is performed by the client.

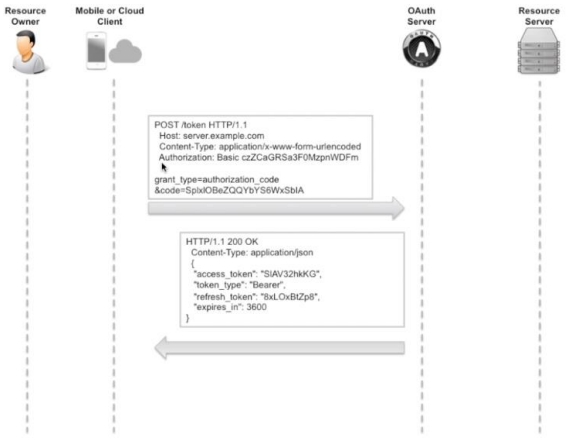
- The authorization code flow is used when secure storage for the token, ClientID, and ClientSecret can be provided by the client.
  - Secure storage can be provided, for example, if the client is implemented as a server-side web application. The data can be stored in a protected database that is not accessible from any outside actors.
  - However, secure storage is not necessarily available to clients written in client-side JavaScript, such as HTML5-based mobile applications. Using the authorization code flow for these types of clients is not recommended.
- The main feature of the authorization code flow is the high level of security since the access token is not flowing through the browser and the username and password of the resource owner is not known to the client.
  - It is also secure because the identities of all three communication partners is assured.
  - This flow also provides convenience to the resource owner since the use of refresh tokens allows access to be granted for longer periods of time without the resource owner having to enter his credentials again to reauthenticate.

Authorization Endpoint



- The authorization code grant is the most complete flow of OAuth, and the other grant types simply derive from it.
  - The first step involves the client requesting an authorization code from the authorization endpoint of the OAuth provider.
  - In the next step, the client sends the authorization code generated in the first step to the token endpoint of the OAuth provider and receives back an access token.
  - The third step involves using the access token that was created to access the resource through the resource endpoint.
- In the GET HTTP request, the ClientID is added along with the other parameters, and ClientID parameter would have been first received after having the client contact the OAuth provider or the resource owner.
  - The redirectURI is used by the OAuth server to send back the results and must be the same URI as was registered when the 3rd party app registered to get its ClientID.
  - The ClientID and the redirectURI must match up with the information stored on the OAuth server, which is a security mechanism.
- After consent has been provided, the OAuth server sends a redirectURI to the mobile or cloud client.
  - Since the there is no way for the OAuth server to directly redirect the app, it instead sends back the redirectURI as an HTTP redirect location parameter inside a Status Code 302 response.

Token Endpoint

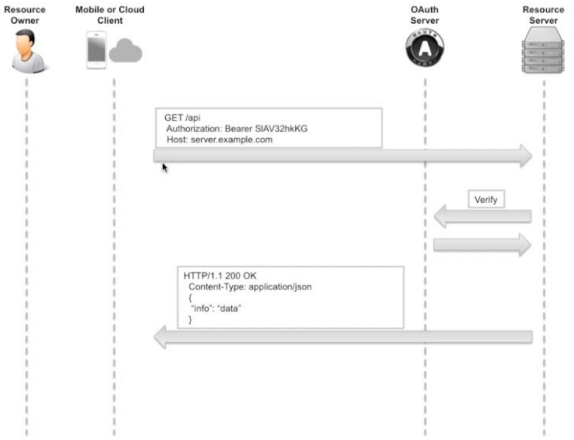


- In the diagram, everything that is happening on the OAuth server is happening on the token endpoint of the server.
- The client sends a request to the token endpoint, which in this case is a POST.
  - Notice that the Content-Type is of form-urlencoded, which would typically come from a web form.
  - It is important to note that these are client credentials that are being sent in the header to the token endpoint, and not the resource owner's credentials.
  - The OAuth server would already have these Authorization: Basic credentials stored in the database and would use these to ensure that the client had been previously registered with the server.
  - The code parameter would be the authorization code that we received in the previous authorization endpoint step.

- The OAuth server would then validate both the Authorization: Basic credentials and the code parameter against what is stored in the database.
    - The length of time that the code is valid is usually around a couple of minutes.
- The response sent back to the client would then be a Status Code 200 'ok' message with the access token, token type, refresh token, and expiry time for the access token.

Resource Access

- The resource endpoint exists on the resource server, so interactions are now from the client directly with the resource server.
    - The bearer authorization token is provided in the header with the request.
- The resource server then verifies that the OAuth token is valid by checking with the OAuth server.
    - If valid, the resource is then provided back in a JSON format to the client.

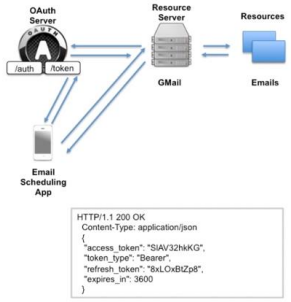


Section 6: Authorization Code Flow – Refresh Tokens

Description and Features

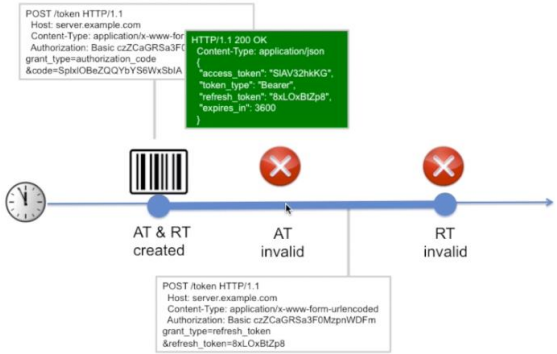
- The refresh flow is the second part of the authorization code flow.
    - This flow is used after we have first received an access and refresh token in the authentication code flow, and the access token has reached the end of its lifetime with the resource owner still requiring access.
- In this case, the refresh token would still be stored on the client side.
    - The client can now go to the OAuth server and refresh the validity of the access token by using the refresh token, which has a longer length of validity than the access token.
    - The response from the OAuth server will provide both a new access token and a new refresh token, with an updated expiry time.
- Only the token endpoint of the OAuth server is used, and the authentication of the resource owner is not needed.

Authorization Code Grant: Refresh



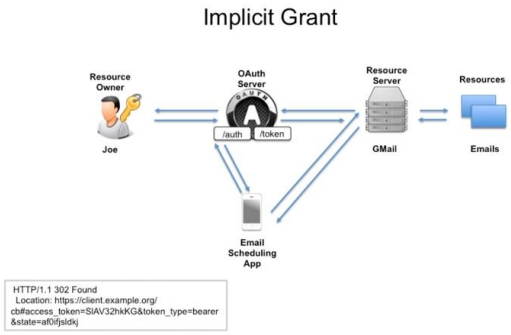
Refreshing Tokens

- Having an access token with a limited lifespan is a security feature to ensure that any lost tokens will not be valid for long.
- A refresh token can help you get a new access token quicker than having to go through resource owner authentication.
    - Note that a refresh token does not allow access to the resources, but only allows a new access token to be obtained.
- A refresh token also eventually expires, but this length of time is much longer than for the access token.
- When getting a new access token with a refresh token from the token endpoint, a POST request will be sent to the token endpoint but this time with a grant type of refresh_token, along with the refresh token itself.
    - Note that we can also access the token endpoint before the access token has expired, although it doesn't really make sense to call the token endpoint before the token has expired.
    - In response, we will get both a new access token and a new refresh token, both of which have new extended expiry times.
- If the token endpoint is called after the refresh token has expired, that call will not be successful, and the authorization code flow will have to begin again from the start.



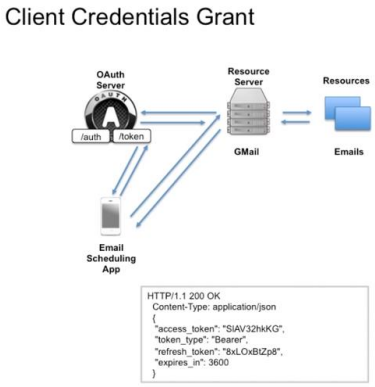
Section 7: Implicit Flow

- The implicit flow is a simplification of the OAuth authentication code flow.
    - It does not use a token endpoint or deliver refresh tokens.
- This flow is for clients that do not have the ability to securely store the refresh token, ClientID, and ClientSecret.
    - This is the case for clients that rely on client-side JavaScript, or for some mobile apps.
- One of the features of the implicit flow is that it is a very simple flow, since only a call to the authorization endpoint is needed.
    - An access token is provided from this authorization endpoint directly without the need for a token endpoint, although a refresh token is not provided.
- As shown in the diagram, the access token is received back directly, along with the token_type and the state.
- The limiting factor of this flow is the short validity of the access token.
    - Because the access token is sent directly without authenticating the client, the access token is usually only valid for a short period of time.
    - This also results in a reduced level of security provided by the implicit grant when compared to the authorization code grant.



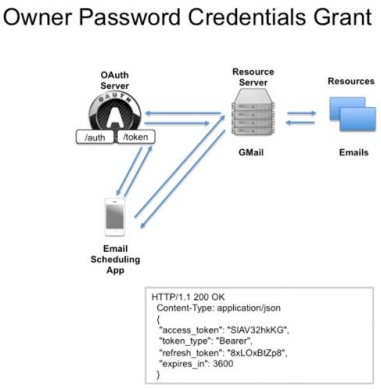
Section 8: Client Credentials Flow

- The client credentials flow only uses the token endpoint of the OAuth server.
    - It does not use the authorization endpoint and does not require the resource owner to authenticate.
- This is a simple flow since the client is identical to the resource owner – they are the same – and as such the resource owner does not need to be separately authenticated since the client is already authenticated by the ClientID and ClientSecret.
    - The client needs to offer secure storage for the ClientID, ClientSecret, access token, and refresh token.
- Features of this flow include a convenient access token renewal since refresh tokens are used, and the simplicity of it since only one call is needed to obtain access and refresh tokens.



Section 9: Resource Owner Password Credentials Flow

- The Resource Owner Password Credentials Flow is a simple flow once again since the only prerequisite is that the resource owner is willing to share their credentials with the client.
    - The client can then use the username and password of the resource owner directly to obtain a token.
- This credential flow should be used when the resource owner trusts the client and is willing to provide their credentials to the client.
    - The pattern of the flow is like that of the password antipattern.
- This trust can be justified only if the resource owner and client are part of the same organization.
    - For example, the client might be the official mobile app of a cloud service.
- It is critical in this type of flow that the client does not store the username and password of the resource owner – it obtains the credentials but does not store them.
    - The credentials should only be used to request an OAuth token.
    - Only the access token and refresh token should be securely stored by the client.



- The features of this flow include convenience since a refresh token is provided for renewal, the simplicity of the flow since only one call needs to be made to get the access and refresh tokens.
    - The limitation is that we have reduced security since there is no guarantee that the client really deletes the password and the resource owner needs to fully trust the client.

Section 10: OAuth vs OpenID Connect

- OpenID Connect standardizes how apps can access the attributes of the resource owner via a token and via a RESTful API, along with how this data is structured and organized.
    - OpenID Connect extends the authorization code flow, introduces new tokens, and standardizes some endpoints.
- It is realized as an extension of OAuth, as a so-called OAuth profile.
    - OAuth profiles are a standardized mechanism to build upon the main OAuth standard.