COMP-3704 / 4704
Web Programming II

Lecture 16 - Authentication using OAuth and JWT

Daniel Pittman, Ph.D., CISSP
3/06/2019

Authentication Overview

- Inevitably, when writing a web application, you eventually come to a point where you want to have users!
  - Those users need to have the ability to register for accounts, and login to perform protected operations
    - Like creating reviews of recipes!
- There are many forms of authentication that can be used on a website, one of the more popular ones being **JSON Web Token (JWT)**
- JWT is an open standard (**RFC 7519**) that defines a **compact** and **self-contained** payload format for securely communicating information

What is JWT?

- JWT is a **signed JSON** object that can be used to share information between parties on the Internet
- JWT is both **compact** and **self-contained**:
  - **Compact**: Because of their smaller size, JWTs can be sent through a URL, POST parameter, or inside an HTTP header. Additionally, the smaller size means transmission is fast
  - **Self-contained**: The payload contains all the required information about the user, avoiding the need to query the database more than once
- How is JWT used?
  - Authentication is the most common scenario
    - User authenticates and is given a JWT to use on subsequent requests
    - JWT can be shared between domains, allowing for Single Sign On capability
  - Information Exchange
    - JWT allows for securely exchanging information since the messages are signed
      - You can validate the sender!

JWT information based on overview from:
https://jwt.io/introduction/

JWT Structure - Header

- A JSON Web Token consists of three parts, separated by dots:
  - **Header**
  - **Payload**
  - **Signature**
  - xxxxx.yyyyy.zzzzz
- The **header** consists of two parts, the type of the token (JWT), and the hashing algorithm being used (i.e. HMAC SHA256 or RSA)
  - { "alg": "HS256", "typ": "JWT" }
- The header JSON is Base64Url encoded to form the first part of the JWT

JWT information based on overview from:
https://jwt.io/introduction/

JWT Structure - Payload

- The second part of a JWT is the **payload**, which contains the **claims** that the token is presenting
  - Also Base64Url encoded
- **Claims** are statements about an **entity** (user), and additional metadata
- There are three types of claims: **registered**, **public**, and **private**
- **Registered Claims** are predefined claims which are recommended to be included in a JWT, including:
  - **iss** (issuer)
  - **exp** (expiration time)
  - **sub** (subject)
  - **aud** (audience)

JWT information based on overview from: https://jwt.io/introduction/

JWT Structure - Payload

- **Public Claims** are shared definitions of common attributes that can be used in JWTs
  - They are registered in the IANA JSON Web Token Registry, or are namespaced, to avoid collisions
  - Public claims can be thought of as a **common set of identifiers** (such as name, email address, etc.) that can be used to consistently identify the same information between different JWT providers
- **Private Claims** are custom claims created to share information between parties that agree to use them, and are neither registered or public claims
  - Anything specific to your site that needs to be included in the JWT would be a private claim

JWT information based on overview from:
https://jwt.io/introduction/

JWT Structure - Signature

- The **signature** portion of the JWT is created by taking the encoded **header**, the encoded **payload**, and a **secret**, and then generating a signature using the **algorithm** specified in the **header**
- The signature will be used to verify the sender of the JWT, as well as to verify the message wasn't altered after it was created
- The three Base64 encoded pieces of the message (header, payload, signature) are then combined with dots to create the JWT!
    - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

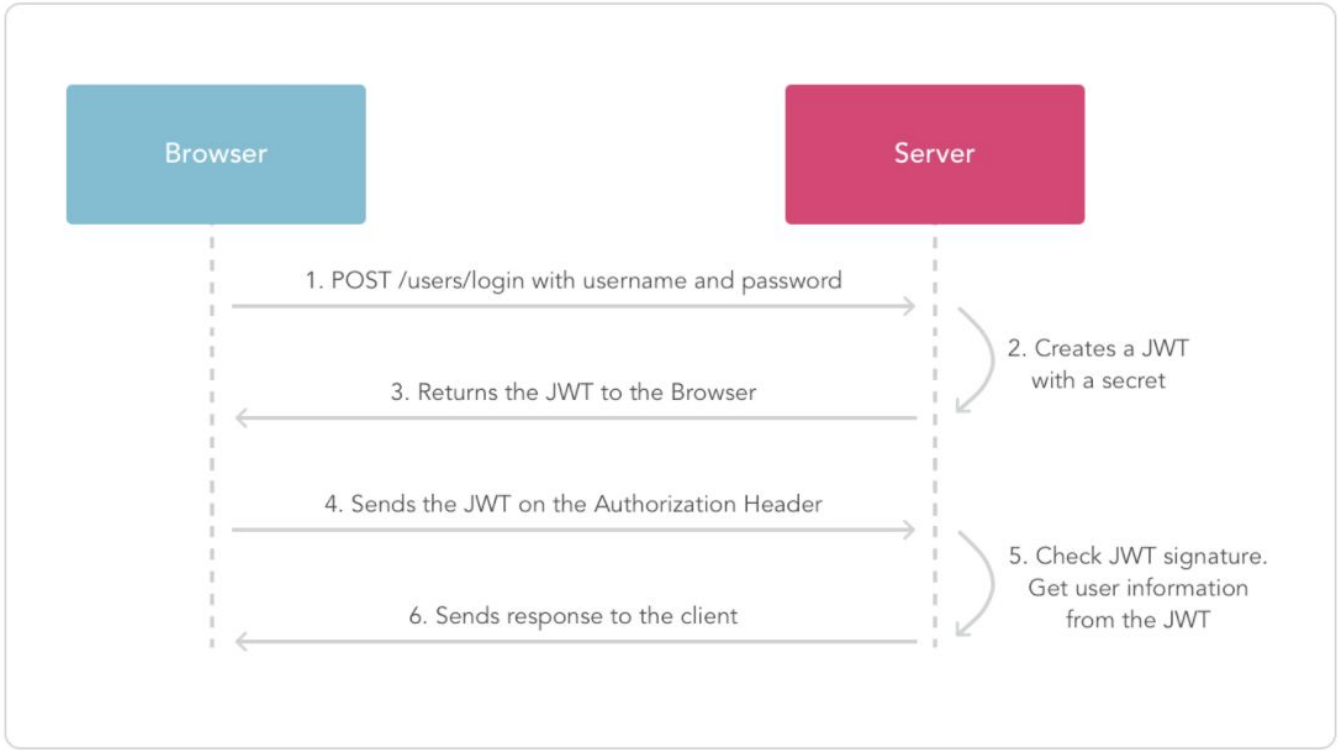
JWT information based on overview from:
https://jwt.io/introduction/

JWT Authentication Flow

- Once a user successfully logs into the system, a JWT is created and returned to the browser
- The token **must** be saved locally, either in local storage or a cookie
  - **No server-side state** is retained regarding the JWT, as is the case in a more traditional session-based approach
    - This allows for stateless APIs, and allows for making requests to multiple services!
- Whenever the user accesses a protected resource, the JWT is sent along, typically in the **Authorization** header using a **Bearer** schema:
  - Authorization: Bearer <token>
- The server will check the validity of the JWT present in the Authorization header, and then allow access
  - All information needed to validate the user is present in the JWT, saving the need to query the database multiple times!

JWT information based on overview from:
https://jwt.io/introduction/

JWT Authentication Flow



JWT information based on overview from:
https://jwt.io/introduction/

OAuth2 Overview

- JWT by itself enables you to securely run a site that maintains its own passwords, and serves as an authentication source for its users
- But what if you want users to login using Google, Facebook, Twitter, etc?
  - Use OAuth!
- OAuth2 is an **authorization framework** that enables applications to obtain **limited** access to users accounts via an external HTTP-based authentication service (such as Facebook, Google, etc…)
- User authentication is **delegated** to the organization hosting the user account, which then **authorizes** third-party applications to access that user's information

JWT information based on overview from:
https://jwt.io/introduction/

OAuth2 Overview

- In OAuth there are four roles involved in the **authentication** and **authorization** workflow:
  - Resource Owner
  - Client
  - Resource Server
  - Authorization Server
- **Resource Owner: User**
  - The resource owner is the **user who authorizes an application to access their account**. The application's access to the user's account is limited to the "scope" of the authorization granted (e.g. read or write access)

JWT information based on overview from:
https://jwt.io/introduction/

OAuth2 Overview

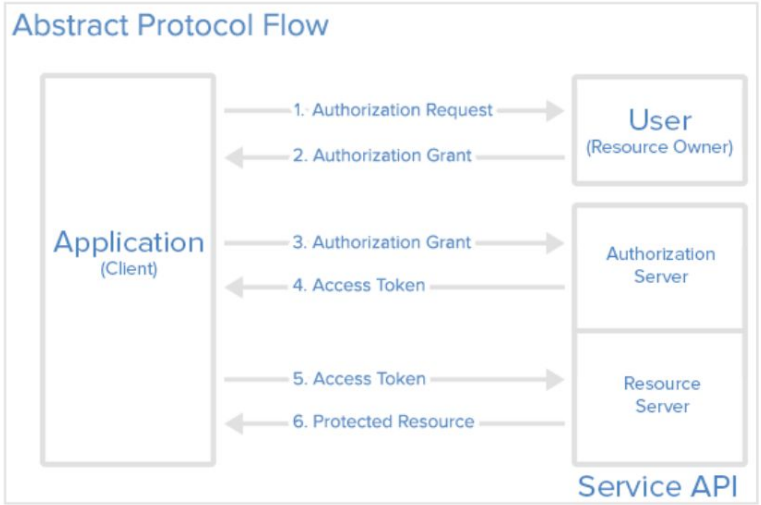
- **Client: Application**
  - The client is the **application that wants to access the user's account**. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API
- **Resource / Authorization Server: API**
  - The resource server **hosts the protected user accounts**, and the authorization server **verifies the identity** of the user then **issues access tokens** to the application
  - From an application developer's point of view, a service's API fulfills both the resource and authorization server roles

JWT information based on overview from:
https://jwt.io/introduction/

OAuth2 Abstract Flow

Generically speaking, the diagram on the right represents how an OAuth2 authorization workflow is processed:

1. The application **requests authorization** to access service resources from the user
2. If the user authorized the request, the application receives an **authorization grant**
3. The application **requests an access token** from the authorization server (API) by presenting authentication of its own identity, and the authorization grant
4. If the application identity is authenticated and the authorization grant is valid, the authorization server (API) **issues an access token** to the application. Authorization is complete.
5. The application **requests the resource** from the resource server (API) and **presents the access token for authentication**
6. If the access token is valid, the resource server (API) **serves the resource** to the application



OAuth2 information based on overview from:
[DigitalOcean Intro to OAuth2]

OAuth2 Application Registration

- In order to use a 3rd party OAuth provider, **you must register** your application with the service
- This is done via the developer portion of the service's website, where you will be required to enter information such as:
  - **Application Name**
  - **Application Website**
  - **Redirect URI or Callback URL**
- The callback URL portion of the registration is used by the service to redirect the user after they authorize or deny your application
  - **It will handle the generated access tokens**

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

OAuth2 Client ID and Client Secret

- Once you have registered your application, the service will issue client credentials in the form of a **client identifier** and **client secret**
- The client identifier is a **publicly exposed string** that is used by service API to identify the application
- The client secret is used to **verify the identity** of the application to the service API when it requests access to a user's account
    - The client secret **must be kept private** between the application and the service API!

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

OAuth2 Authorization Grant

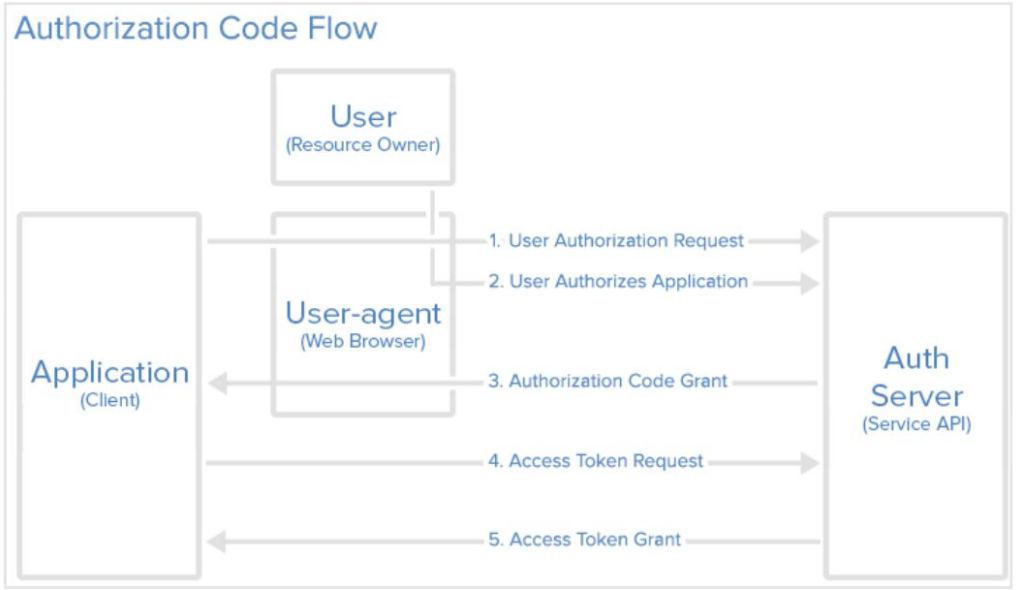
- In an OAuth2 authorization workflow there is a sequence of steps in which an **authorization grant** and **access token** are obtained
- The authorization **grant type** depends on the method used by the application to request authorization, as well as the methods supported by the service API
- There are **four grant types** defined in OAuth2, each of which is useful in different cases:
  - **Authorization Code:** used with server-side Applications
  - **Implicit:** used with Mobile Apps or Web Applications (applications that run on the user's device)
  - **Resource Owner Password Credentials:** used with trusted Applications, such as those owned by the service itself
  - **Client Credentials:** used with Applications API access
- We will cover **Authorization Code** and **Implicit** types in detail

JWT information based on overview from:
https://jwt.io/introduction/

OAuth2 Grant Type: Authorization Code

1. The **authorization code** grant type is optimized for server-side applications, and maintains the confidentiality of the client access token by not sharing it with the client's browser

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2



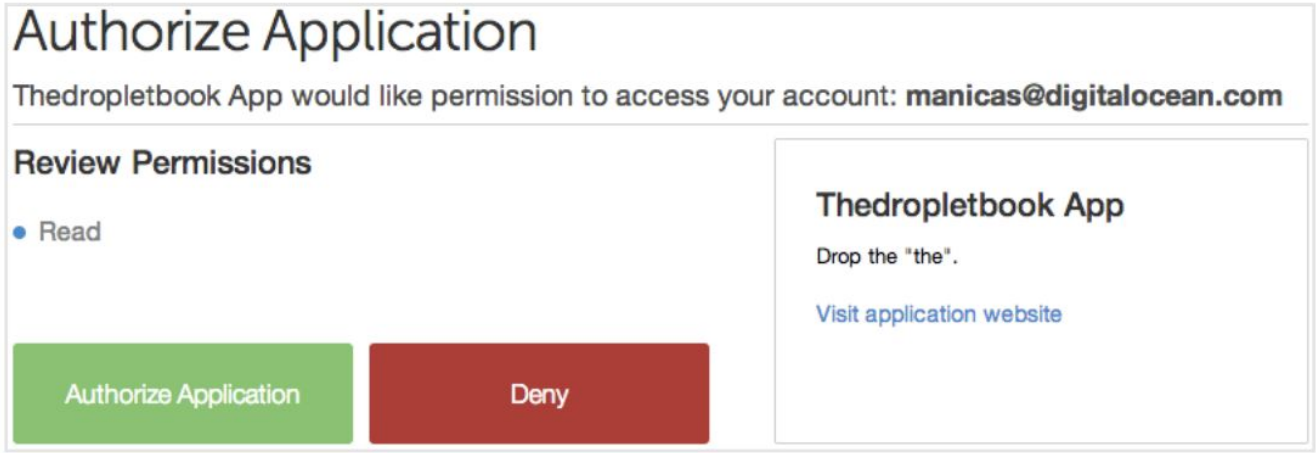
OAuth2 Grant Type: Authorization Code

- **Step 1: Authorization Code Link**
  - First, the user is given an authorization code link that looks like the following:
    - https://cloud.digitalocean.com/v1/oauth/authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read
      - **https://cloud.digitalocean.com/v1/oauth/authorize**: the API authorization endpoint
      - **client_id=client_id**: the application's client ID (how the API identifies the application)
      - **redirect_uri=CALLBACK_URL**: where the service redirects the user-agent after an authorization code is granted
      - **response_type=code**: specifies that your application is requesting an authorization code grant
      - **scope=read**: specifies the level of access that the application is requesting

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

OAuth2 Grant Type: Authorization Code

- **Step 2: User Authorizes Application**
  - When the user clicks the link, they must log into the service to verify their identity, and then either **authorize** or **deny** the application access to their account



OAuth2 information based on overview from:
[DigitalOcean Intro to OAuth2]

OAuth2 Grant Type: Authorization Code

- **Step 3: Application Receives Authorization Code**
  - If the user clicks the Authorize Application button, the service will redirect the browser back to the application redirect URI (specified during client registration), along with an **authorization code**
  - The redirect will look something like this:
    - https://dropletbook.com/callback?code=AUTHORIZATION_CODE
- **Step 4: Application Requests Access Token**
  - The application then requests an access token from the API, passing the authorization code given in step 3, its client secret, and its client Id
    - https://cloud.digitalocean.com/v1/oauth/token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&grant_type=authorization_code&code=AUTHORIZATION_CODE&redirect_uri=CALLBACK_URL

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

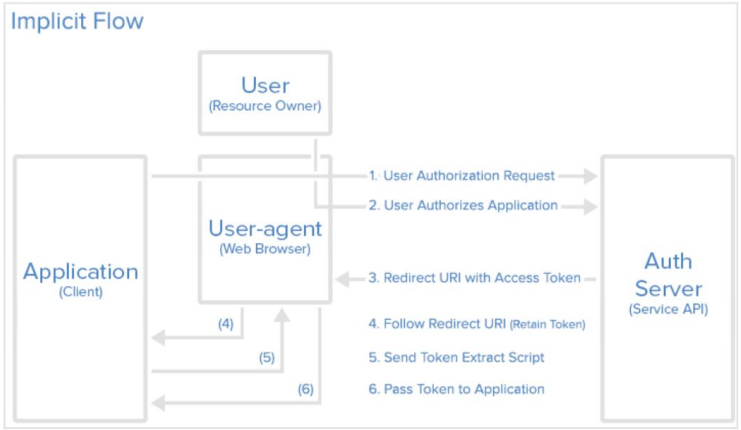
OAuth2 Grant Type: Authorization Code

- **Step 5: Application Receives Access Token**
  - If the authorization is valid, the API will send a response containing the access token (and optionally, a refresh token)
    - {"access_token":"ACCESS_TOKEN","token_type":"bearer","expires_in":2592000,"refresh_token":"REFRESH_TOKEN","scope":"read","uid":100101,"info":{"name":"Mark E. Mark","email":"mark@thefunkybunch.com"}}
- At this point the application is authorized, and has permission to access the user's account with the level of permission specified in the grant request
  - The REFRESH_TOKEN can be used to issue a new access token when the old one is revoked or expires

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

OAuth2 Grant Type: Implicit

- The **implicit grant type** is used for mobile apps and web applications (i.e. web browser applications) where the **client access token** confidentiality cannot be guaranteed
  - The access token is first given to the browser and then forwarded to the application
    - This mean the token is exposed to the user and could be intercepted by another application on the user's device
  - Implicit grant types do not support token refresh



OAuth2 information based on overview from:
[DigitalOcean Intro to OAuth2]

OAuth2 Grant Type: Implicit

- **Step 1: Implicit Authorization Link**
  - With the implicit grant type, the user is presented with an authorization link very similar to the authorization code link, except that it is requesting a **token** instead of a **code:**
    - https://cloud.digitalocean.com/v1/oauth/authorize?**response_type=token**&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read
- **Step 2: User Authorizes Application**
  - When the user clicks the link, they login and authorize/deny the application just like in the authorization code link
- **Step 3: User-agent Receives Access Token with Redirect URI**
  - If the user selects “Authorize Application”, the service redirects the browser to the application redirect URI, and includes a fragment containing the access token:
    - https://dropletbook.com/callback#token=ACCESS_TOKEN

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

OAuth2 Grant Type: Implicit

- **Step 4: User-agent Follows the Redirect URI**
  - The user-agent follows the redirect URI but retains the access token
- **Step 5: Application Sends Access Token Extraction Script**
  - The application returns a webpage that contains a script that can extract the access token from the full redirect URI that the user-agent has retained
- **Step 6: Access Token Passed to Application**
  - The user-agent executes the provided script and passes the extracted access token to the application
- At this point the application is authorized, and has permission to access the user's account with the level of permission specified in the grant request

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

JWT and OAuth2 in MEAN

- As you will see in today's lab, although there is quite a bit of code needed to enable authentication and authorization in our application, **we don't need to manage the low-level details of JWT and OAuth2!**
  - Thanks to the large open-source community supporting Node.js, there are libraries that manage **passport-based JWT and OAuth** authentication strategies
    - Passport is an authentication library for Node.js
- This allows us to focus on validating permissions and managing user accounts, and **not worry** about how to create and sign JSON Web Tokens, or how to execute the authorization workflow for different OAuth providers!

OAuth2 information based on overview from:
DigitalOcean Intro to OAuth2

Let’s get started with today’s lab!

- Today’s lab: Authentication using JWT and OAuth
  - https://canvas.du.edu/courses/93239/assignments/675944
  - In this lab you will update your project to support both local and OAuth authentication using JWT!