

# API Pentesting – JWT & Bearer Token



**LinkedIn** : <https://www.linkedin.com/in/mohamed-elsayaad>

**GitHub** : <https://github.com/0xDos>



# What is an API?

هو حلقة وصل ( وسيط ) بين front-end , back-end زي ما الجرسون وسيط بين العميل والطباخ

1. A way of communication between two different applications.
2. Mostly used whenever we find a JavaScript frontend like React/Vue/Angular
3. It's a way for mobile applications to communicate with their external servers

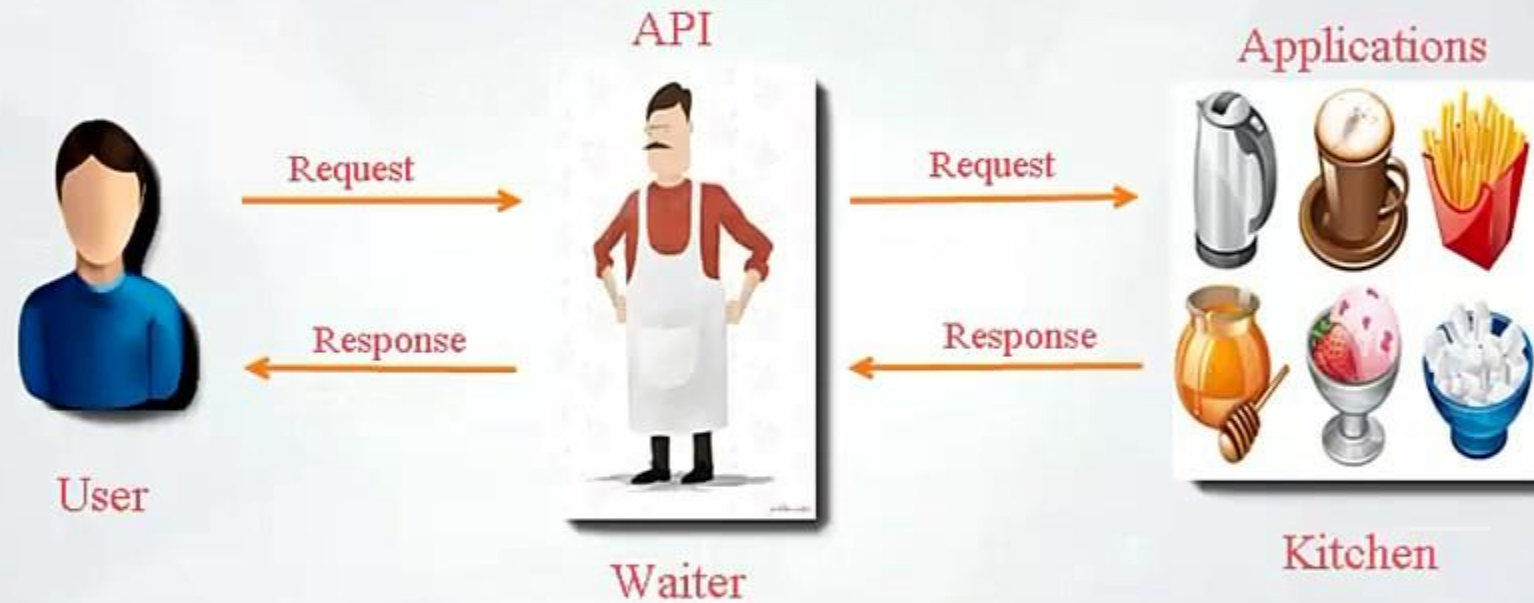
لما بعمل تسجيل دخول علي موقع ما بيكون فيه API Endpoint او نقطه وصول **عن طريقها** front-end بيقدريوصل للبيانات الموجوده في back-end server وتكون معرفه مسبقا للطرفين

## Why is API pentesting important ?

1. It's the side that a user normally doesn't see
2. Developers usually do a sloppy job creating API endpoints which results in either logic problems or actual exploitation

**من أمثله :** مواقع حجز الطيران والفنادق  
بتكون عبارته عن مواقع تحتوي بداخلها علي مجموعه من APIs تسمى **External APIs** باخذها من المواقع الاصلية واطهر المحتوي بتاعهم عندي

عندي موقع وليه تطبيق ال هيربطهم ببعض هو **API**



اكتر استعمالات APIs

1 ( JavaScript Frameworks مثل AngularJS , React , Vue.js , Node.js لان الموقع المنشئ ب Frames دي يعتمد اعتماد كامل علي APIs

2 ( Mobile Application علشان بيتعامل مع الموقع الخارجي

# XML

- **EX**ensible **M**arkup **L**anguage
- In xml, the terms in the tag don't mean anything
- In HTML it means something
- W3c organization created XML and HTML
- HTML is not extensible (you can't extend it to mean something else)
- XMLs are sent through the internet. Back and forth
- Browsers can understand XML

```
C:\Users\egypt2\Desktop\my youtube apis\shawerma order 1.xml - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window 2
shawerma order 1.xml
1 <shawerma>
2   <size> small </size>
3   <type> chicken </type>
4   <additions>
5     <add> towmea </add>
6     <add> tehena </add>
7     <add> extra cheese </add>
8   </additions>
9 </shawerma>
```

LinkedIn : <https://www.linkedin.com/in/mohamed-elsayaad>

لما بنتعامل مع APIs بنرسل ونستقبل داتا  
الداتا دي بتكون بطريقة كتابه معينه والطريقه دي بتكون اما  
JSON or XML

اما البروتوكول ال بنستخدمه علشان ننقل الداتا دي :

REST : use >> Postman Tool >> JSON

SOAP: use >> Soap UI Tool >> XML

لما بستخدم SOAP بستخدم WSDL

WSDL > Web service description language  
دا standard بمشي عليه وبيكون جواه بعض التاجات ليها معاني زي  
html

Web service هي API متصله بالانترنت

## SOAP API

Common Vulnerabilities:

1. Uses XML ( XML Injection is possible )
2. SQL Injection in parameters
3. Command Injection
4. Actions can be enumerated easily

GitHub : <https://github.com/0xDos>

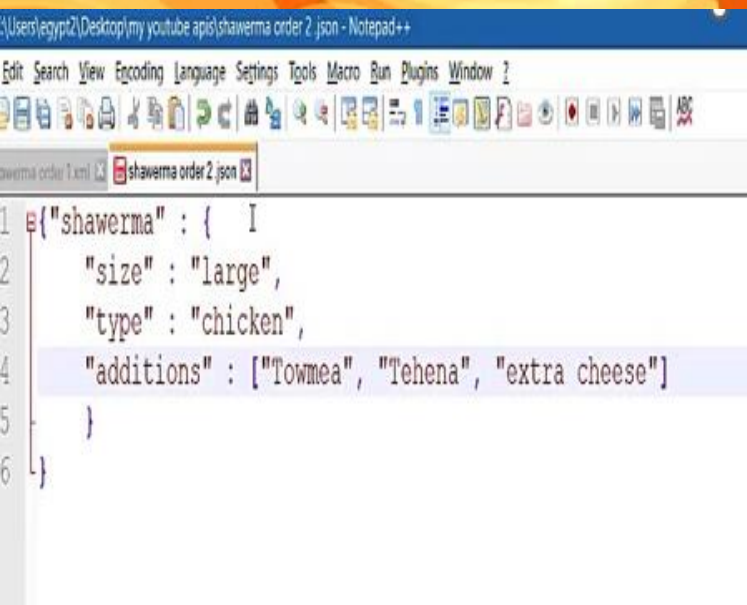


# What is REST?

- Representational State Transfer
- Rest is representational, the actual record is not sent, a representation of the record is sent.
- Rest is Stateless
- IN SOAP, if the program stops (at the server), it breaks down then we have a major issue. REST waits until it works (Stateless)
- REST uses JSON

**JSON** (JavaScript Object Notation) is the most common means of exchanging data using a REST API.

REST API هو الوسيلة الأكثر شيوعاً لتبادل البيانات باستخدام



```
{\"shawerma\": {  
  \"size\": \"large\",  
  \"type\": \"chicken\",  
  \"additions\": [\"Towmea\", \"Tehena\", \"extra cheese\"]  
}}
```

# REST APIs

	XML
Post	Create an order (JSON)
PUT	Erase the order and create another one
Patch	Keep the order and add another one
Get	Receive the order details
Delete	Remove the order

# REST API

## Common Vulnerabilities:

1. Sensitive Data exposure
2. Injections (OS / SQL)
3. Broken Access Control
4. Endpoint's can be enumerated easily based on the REST API Design



# What is postman ?

- Postman is a tool that you can use to test your own REST APIs, or external APIs (FB,Google, etc).
- Postman is used by 6 million developers and more than 200,000 companies to access 130 million APIs every month.
- Postman is a browser



## It's not always Remote Code Execution

- Don't expect to get a Remote Code Execution
- Most of your finding's will be broken access control
- API's are mostly forgotten about, thinked about as something no one will see which makes them a prime example for exploitation
- By Exploitation I mean Data leakage (Information a user should not get)



## Abusing RESTful API Design

1. Enumerating API endpoints
  - a. Gobuster/wfuzz/ffuf
  - b. <https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/api/objects.txt>
2. Finding HTTP methods
  - a. `wfuzz -z list,GET-HEAD-POST-TRACE-OPTIONS-PUT -X FUZZ http://example.com/api/post/1/`
3. Attempting to break access control
  - a. Example: Deleting another user's post or being able to edit it

## The devil is in the details

**\*\*Real life example\*\***: Leaking user data with incremental user ID

1. <http://example.com/customers/123> (401)
2. <http://example.com/customers/123/loans> (200)
3. <http://example.com/customers/123/loans/456> (200)

Sometimes the original endpoint is protected, but whatever is built upon it will be completely ignored and therefore exploitable

# Please give me everything

**\*\*Real Life Example\*\*:** Why take something when you can take everything?

1. Enumerating <http://example.com/api/FUZZ> resulted in nothing :(
2. <https://dnsdumpster.com/> shows a <http://317.example.com> live!
3. The Development server has `DEBUG` enabled ?? so what

```
315. ^ ^password_reset/done/$ [name='password_reset_done']
316. ^ ^reset/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$ [name='password_reset_confirm']
317. ^ ^reset/done/$ [name='password_reset_complete']
```

The current URL, `fff`, didn't match any of these.

4. `317` different urls to try and exploit
5. We find unprotected url `importusers.php` which exports all users



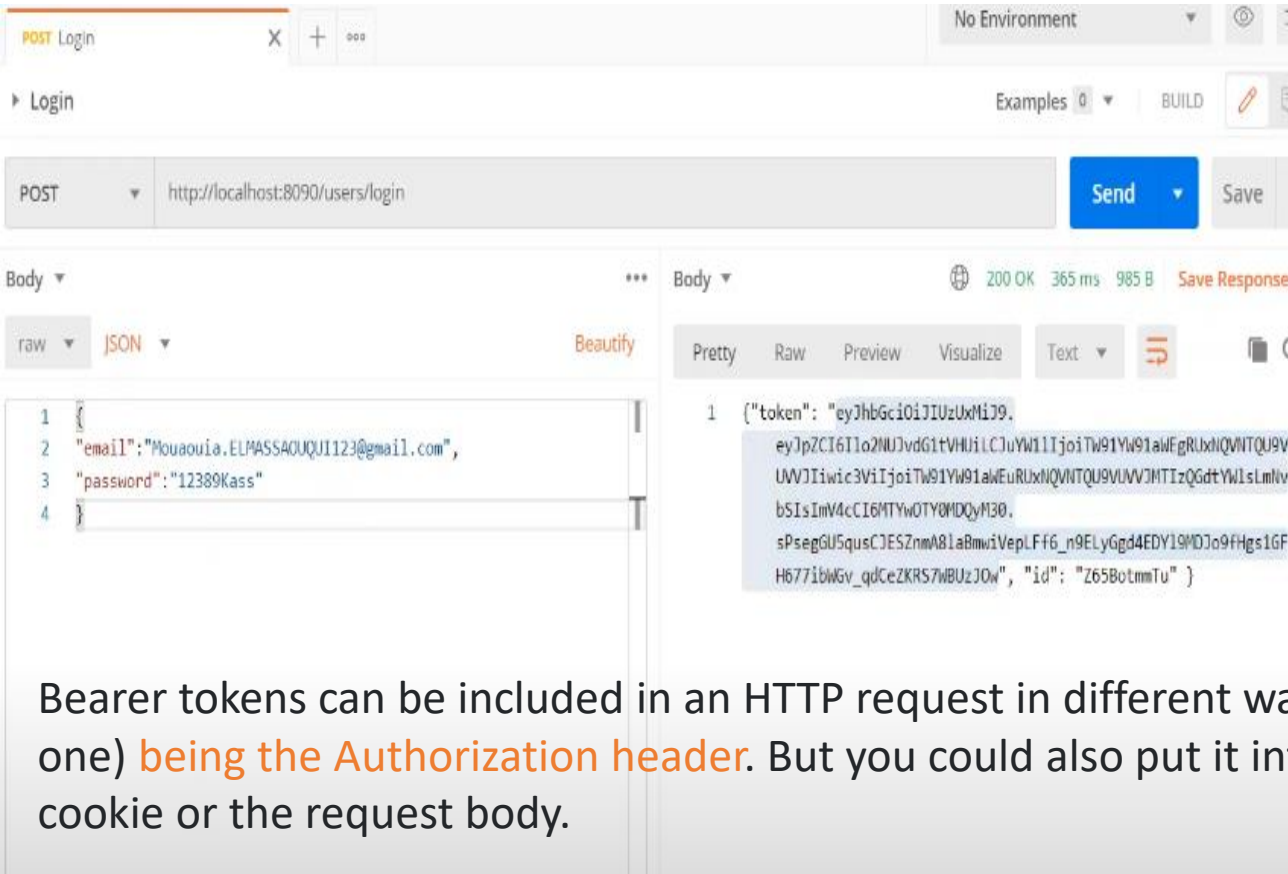


# JWT & Bearer Token Authentication

**JSON Web Token (JWT)** is an encoding standard for tokens that contains a JSON data payload that can be signed and encrypted.

JWT can be used for many things, among those are **bearer tokens**

**Bearer Tokens** are type of access token used with OAuth 2.0.



Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IjZSIkvaG4gRG91IiwiaWF0IjoxNTE2MjM5MDIyfQ.eyJ1IjoxNTE2MjM5MDIyfQ.

## When should you use JWT ?

- JWT ARE USEFUL :
  - AUTHORISATION
  - INFORMATION EXCHANGE

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```

HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    your-256-bit-secret
) ☐ secret base64 encoded

```

# J WHAT??

## JSON Web Token (JWT)

1. I love these things, if you are able to exploit them you should have the ability to impersonate any other user

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90aWNlIjoiYXNjaW91IiwiaWF0IjoxNDU0MjE0ODU5fQ.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90aWNlIjoiYXNjaW91IiwiaWF0IjoxNDU0MjE0ODU5fQ.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90aWNlIjoiYXNjaW91IiwiaWF0IjoxNDU0MjE0ODU5fQ`

2. You can use <https://jwt.io> to decode the JWT token and find interesting information
3. Lot's of attacks can be applied to JWT tokens and found here

[https://github.com/ticarpi/jwt\\_tool](https://github.com/ticarpi/jwt_tool)

4. JWT tokens depends on a secret key, if a Development server leaks it, its game over



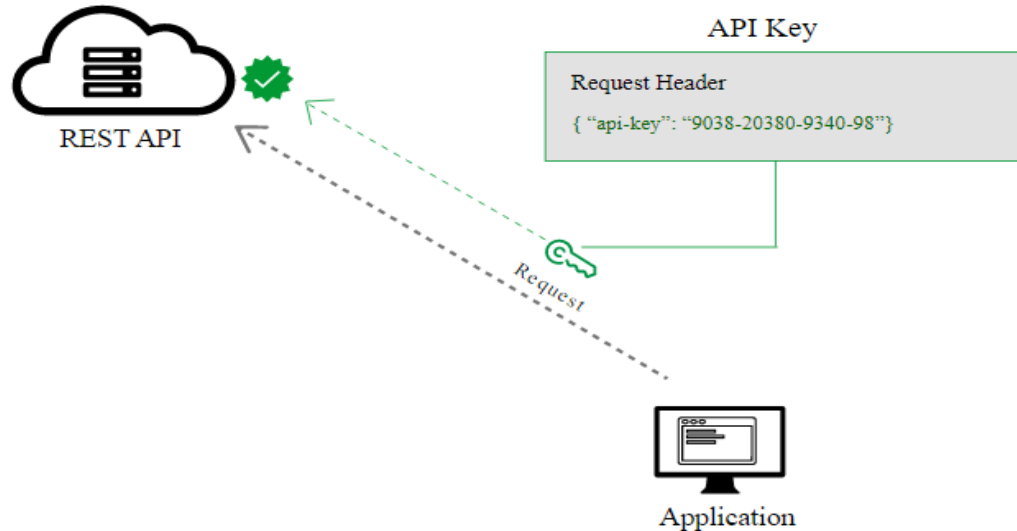
## Different types of authorization

There are several methods for authorization. The following are various types of API authorization you might encounter:

- API keys
- Basic Auth
- HMAC
- OAuth

### API keys

Most APIs require you to sign up for an API key in order to use the API. The API key is a long string that you usually include either in the request URL or request header. The API key mainly functions as a way to identify the person making the API call (authenticating you to use the API). The API key might also be associated with a specific app that you register.



### OAuth 2.0

One popular method for authenticating and authorizing users is OAuth 2.0. This approach relies on an authentication server to communicate with the API server to grant access. You often see OAuth 2.0 when you're using a site and are prompted to log in using a service like Twitter, Google, or Facebook.

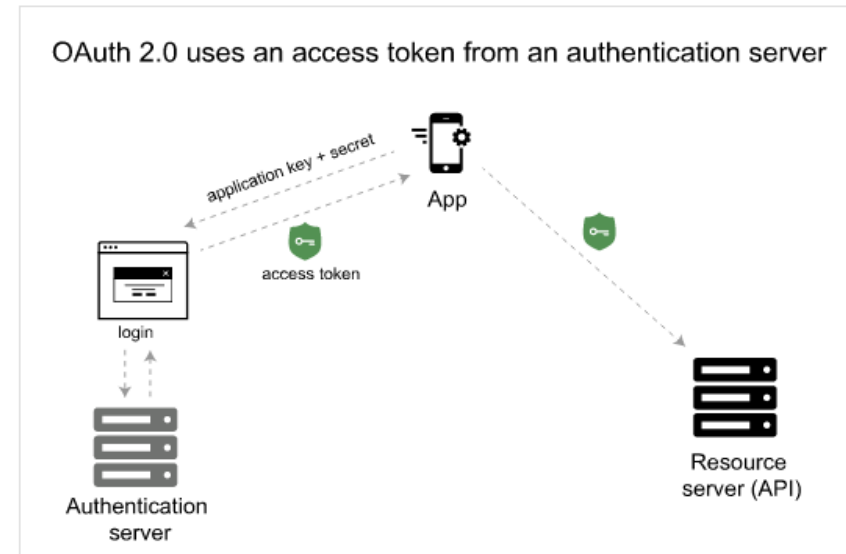
*OAuth login window*

There are a few varieties of OAuth — namely, “one-legged OAuth” and “three-legged OAuth.” One-legged OAuth is used when you don't have sensitive data to secure. This might be the case if you're just retrieving general, read-only information.

In contrast, three-legged OAuth is used when you need to protect sensitive data. Three groups are interacting in this scenario:

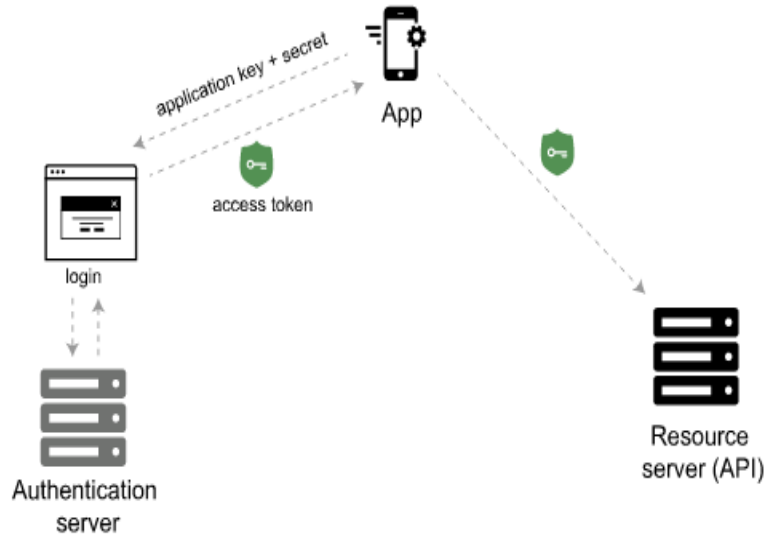
- The authentication server
- The resource server (API server)
- The user or app

Here's the basic workflow of OAuth 2.0:



*OAuth authentication*

OAuth 2.0 uses an access token from an authentication server



OAuth authentication

First, the consumer application sends over an application key and secret to a login page at the authentication server. If authenticated, the authentication server responds to the user with an access token.

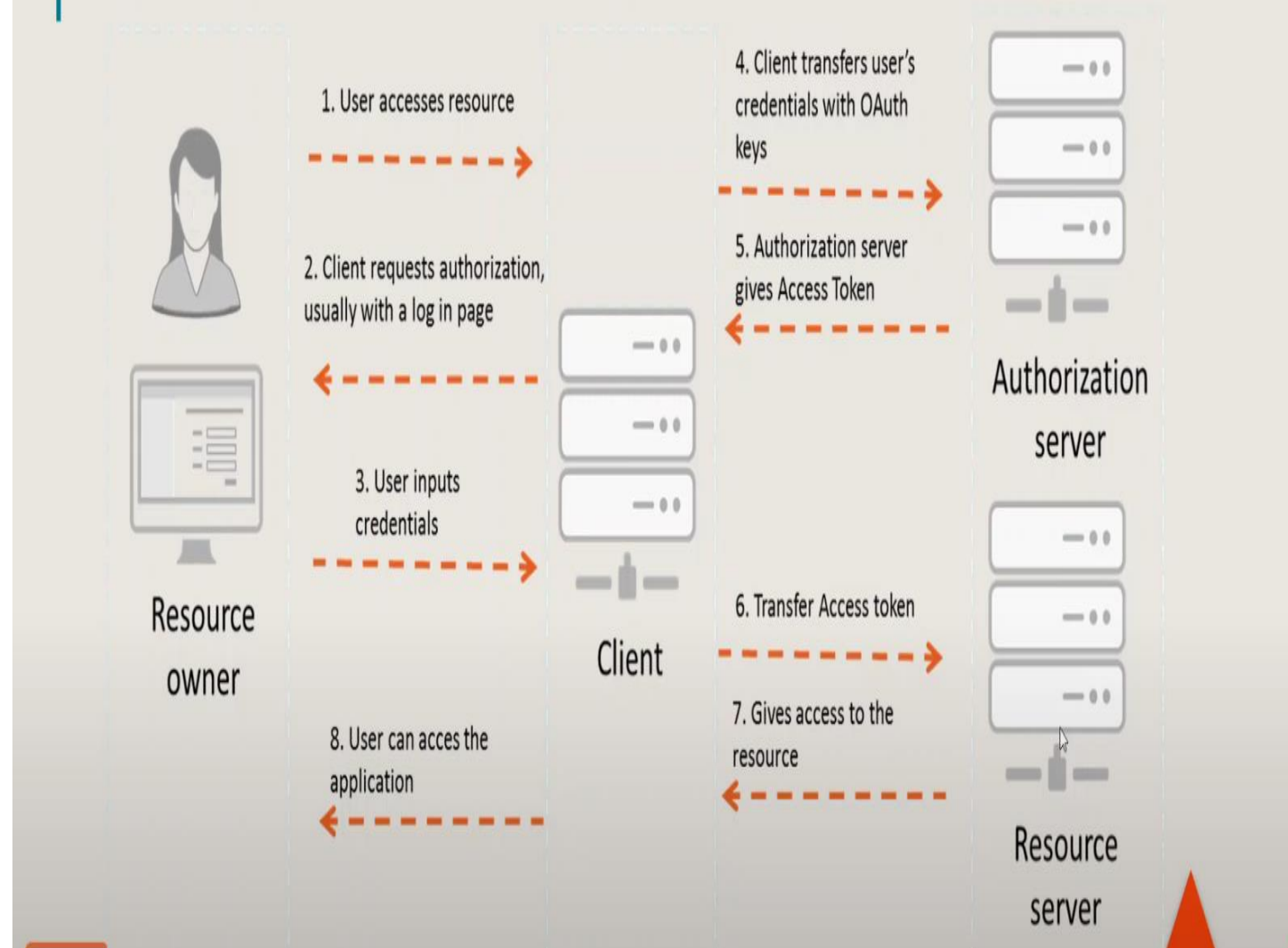
The access token is packaged into a query parameter in a response redirect (302) to the request. The redirect points the user's request back to the resource server (the API server).

The user then makes a request to the resource server (API server). The access token gets added to the header of the API request with the word **Bearer** followed by the token string. The API server checks the access token in the user's request and decides whether to authenticate the user.

Access tokens not only provide authentication for the requester but also define the permissions of how the user can use the API. Additionally, access tokens usually expire after a period of time and require the user to log in again. For more information about OAuth 2.0, see these resources:

LinkedIn : <https://www.linkedin.com/in/mohamed-elsayaad>

## BEARER TOKEN AUTHENTICATION JWT



GitHub : <https://github.com/OxDos>