API Research Document

By Eric Spiteri

Eric Spiteri 5.2 Interactive

Table of Contents

Part 1: RESTful APIs

What is a RESTful API?,

A RESTful API (Representational State Transfer) is an architecture used for designing networked applications. RESTful APIs allow for effective communication between the client-side and the server-side of the system using standard HTTP methods (GET, POST, etc).

Core Principles of Restful APIs:

-Focus on client-server architecture.

This means that the client (front-end) sends HTTP requests and the server processes them and sends back HTTP responses. This separation of operations between front-end and back-end is crucial for them to be developed independently, perhaps by two different employees each with their own specialisation.

-Statelessness

Statelessness ensures that all session information is stored in the client side and not the back-end, thus each request must contain all the information needed to understand and process each request.

-Cacheablity

Cacheability defines whether a response from the server-side is cacheable or non-cacheable. This significantly increases the performance of both ends by reducing client-server interactions, as cached information stores responses and requests that are made frequently, improving performance and efficiency.

How Making a Request to an API Works:

1.) Client makes an HTTP request to the server, containing the URL, HTTP method, headers and an optional body. (usually in JSON format)
2.) Server processes the request.
3.) Server returns an HTTP response, containing status codes, headers and an optional body (usually in JSON format).

Common HTTP methods:

-GET :

Retrieve data from the server, it is used to request any of the following resources:

-A webpage or HTML file

-Media such as images or video

-A JSON Document

-CSS files

-An XML file

The GET request is a safe operation, meaning it does not change any resource in the server.

-POST : Create data for the server, this is used for further processing. The data sent is typically in these formats:

-input fields in online forms.

-XML or JSON data

-Text data from query parameters.

Note that this is not a safe operation as it changes data in the server and can cause potential side effects to the state of the server when executed. Moreover, this operation is not necessarily idempotent, meaning that if it is executed more than once, it might return a different result.

-PUT : Update or Replace existing data, this means that the PUT request always includes a completely new Payload to be saved to the server, similar to the POST request, but obviously this is meant to be a replacement. Moreover, an existing URL must exist for a PUT request to work, otherwise a new resource will be created.

PUT operations are unsafe but idempotent because they are changing the resources of the server while at the same time those changes will always leave the same result after each execution.

-PATCH : Partially update existing data. Sometimes object representations get very large and so completely replacing the resource with a PUT request would be a waste of resources.

-DELETE : Remove data. Like PUT operations, it is considered highly unsafe and idempotent.

QueryString Parameters and Request Body

Query String Parameters: These are parameters embedded in the URL which are used for filtering and sorting (e.g: GET /users**?**age=21)

Request Body: This is the actual data sent to the server, usually in a GET, PUT or PATCH request.

Example {

“username”:”ericspiteri1410”

“password”: “JagerTh@rn”
}

Part 2: Authorisation: (0Auth 2.0)

0Auth2.0 is an open authorisation framework which allows applications to access user data without exposing sensitive user information, meaning it provides secure access to web resources.

Some of the benefits of 0Auth 2.0 include avoiding sharing user credentials with third party applications as well as supporting multiple client types (e.g mobile, desktop etc).

-Access Tokens:

Access tokens are temporary credentials in requests to server APIs. Instead of sending user credentials (e.g email and password), an app or website uses these in API requests. Note that these contain expiration dates for security purposes, moreover it is imperative that the tokens are never stored in the front-end, also for security.

-Scopes:

A key component in tokens are scopes, scopes define precisely what the user can access. Example “read:user” “write:posts”. If the token does not contain what is defined in the scope, the server returns an “ERROR 403: FORBIDDEN” message.

This also has the advantage of being easily documentable for developers.

-Client ID and Client Secret:

The client ID and Client Secret are unique credentials of an application used to help identify and authenticate it when requesting access to sensitive resources.

A client ID is a public identifier for an application. It is used to identify precisely which application is making a request to the 0Auth provider.

A Client Secret is a highly confidential key used by the application to authenticate itself with the 0Auth provider, confirming that the request is coming from a trusted source.

One should NEVER expose this in the front-end or in repositories, as this means that a custom application could be quickly developed with the client secret to access sensitive resources in the API.

Part 3: Security Considerations:

What is OWASP API Security Top 10?

OWASP API Security Top 10 is a list of the most serious security threats facing APIs, created by the Open Web Application Security Project (OWASP). It aims to help developers deal with common security vulnerabilities in APIs.

The following is a list of these security vulnerabilities in order of their seriousness.

Rank	Threat	Description
1	Broken Object Level Authorisation	Improper access control allowing users to access others' data.
2	Broken Authentication	Weak authentication mechanisms that can be bypassed.
3	Broken Object Property Level Authorisation	Exposing or modifying sensitive properties within objects.
4	Unrestricted Resource Consumption	API misuse leading to high resource usage (DOS attacks).
5	Broken Function Level Authorisation	Failure to enforce authorisation on API functions.
6	Unrestricted Access to Sensitive Business Flows	Business logic flaws that can be abused.
7	Server Side Request Forgery	API requests can be manipulated to target internal systems.
8	Security Misconfiguration	Poor security settings leading to vulnerabilities.
9	Improper Inventory Management	Lack of proper API versioning or exposed endpoints.

10	Unsafe Consumption of APIs	Trusting external APIs without proper validation.

My plan for my API

I will base my RESTful API on the project for PHP and Databases. The project will consist of a property booking website with various back-end features including creating account, logging in, viewing properties and booking properties. Obviously in order for the front-end to update the HTML based on the back-end information, a RESTful API is needed to facilitate client-server communication. I also plan to include the google-maps API endpoint for the taxi booking page.

0Auth 2.0 will be essential for this booking website as we do not want user's personal information to be compromised.

To implement OAuth 2.0 in my booking website, I will do the following procedure:

1.) **User Authentication:** User logs in via the OAuth Server
2.) **Authorization Code Grant:** My API redirects the user to the 0Auth server with the client ID and requested scopes
3.) **Token Exchange:** Once the user authorizes, the server returns an authorization code.
4.) **Access Token Retrieval:** My API exchanges the code for an access token using client secret.
5.) **Access Protected Resources:** My API stores the token and uses it to access protected routes.

The same procedure will be applied to different kind of requests, but with changed scopes. (e.g read:Booking).

To implement OWASP API security features I will tackle the Broken Object Level Authorisation and Security Misconfiguration issues.

Broken Object Level Authorisation:

I will tackle this security risk by implementing strict authorisation checks for each JSON object.

I will also validate the user's right's before performing any action

Security Misconfiguration:

Disable Unnecessary HTTP methods in headers

Hide sensitive information in error messages

References

https://owasp.org/API-Security/editions/2023/en/0x11-t10/

https://www.moesif.com/blog/technical/api-design/REST-API-Design-Best-Practices-for-Parameters-and-Query-String-Usage/

https://blog.postman.com/rest-api-examples/

https://oauth.net/2/scope/

https://auth0.com/docs/get-started/authentication-and-authorization-flow/client-credentials-flow