Web Security Practices

Ali-Akbor

**Security Practices May Varies**

1. May varies from application to application
2. May varies from developer to developer
3. May varies from environment to environment
4. May varies from use case to use case

**But we have to know**

1. The best practices
2. Know about the security layers
3. Security placement

**API END POINT**

Request

***Security Layerr***

**API Key Verification**

**User Agent Verification**

**CSRF Protection**

**Rate Limiting**

**HTTP Secure Headers**

**Response**

**Parameter Pollution**

**Controller**

**Payload Validation**

**Data Sanitaization**

**Response Validation**

**Authentication/Authoraization**

**Output Validation**

**Output Header:**

* Provide proper http response status code.
* Provide proper content type, file type if any.
* Provide cache status if any.
* Authenticationtoken should provide via response header.
* Only string data is allowed for response header.
* Provide content length if any.
* Provide response date and time.
* Follow request-response model described before.

**Output Body:**

* Avoid providing response status, code, message via response body
* Use JSON best practices for JSON response body.
* For single result, can use String, Boolean directly.
* Provide proper JSON encode-decode before writing JSON Body.
* Follow discussion on JSON described before.

**Request Rate limit- Throttling**

We need to make sure our APIs are running as efficiently as possible. Otherwise, everyone using your database

will suffer from slow performance. Performance isn’t the only reason to limit API requests, either. API limiting, which also known

as rate is limiting, is an essential component of Internet security, as DoS attacks can tank a server with unlimited API requests.

Rate limiting also helps make your API scalable. If your API blows up in popularity, there can be unexpected spikes in traffic,

causing severe lag time.

|  |  |  |  |
| --- | --- | --- | --- |
| Language | Platform | Library name | Library Sources |
| **C#** | **ASP.NET** | **WebApiThrottle,MvcThrorrle** | **Nuget package manager** |
| **PHP** | **Laravel** | **Laravel Karnel Default** | **Packagist** |
| **JS** | **Node/Express.JS** | **Express-rate-limit** | **NPM** |

**CSRF/XSRF Protection**

Cross-site request forgery attacks (CSRF or XSRF for short) are used to send malicious requests from an authenticated user to a web application.

* Use request-response header to pass CSRF token
* CSRF token should be unique for every session
* For self API CSRF token works well.

|  |  |  |  |
| --- | --- | --- | --- |
| Language | Platform | Library name | Library Sources |
| **C#** | **ASP.NET** | **AntiCSRF** | **Nuget package manager** |
| **PHP** | **Laravel** | **Laravel Default** | **Packagist** |
| **JS** | **Node/Express.JS** | **npm i CSRF** | **NPM** |

**User Agent Protection**

User agent is a request header property, describe client identity like operating system, browser details, device details etc. Moreover every web crawler like Google crawler, Facebook crawler has specific user-agent name.

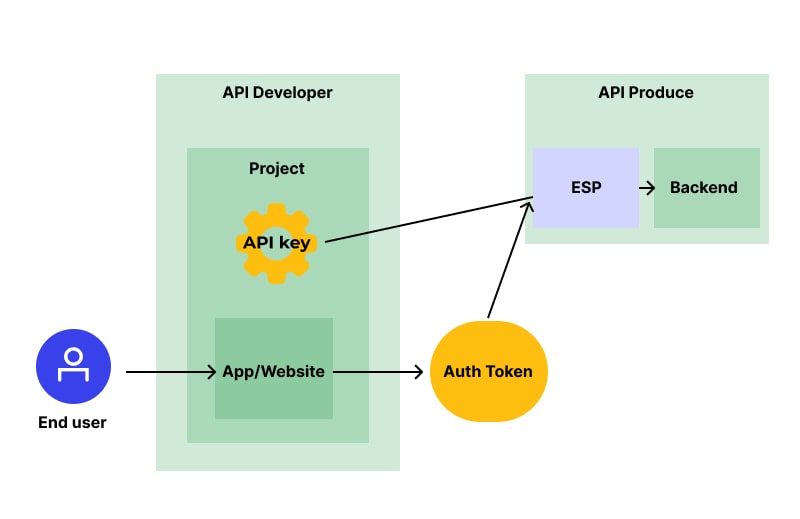
* Using user agent we can prevent REST API from search engine indexing, social media sharing.
* Can stop subspecies request from who is hiding his identity.
* We can add user agent along with REST API usage history.
* We can add device/OS usage restriction.

|  |  |
| --- | --- |
| Platform | Example User Agent Like |
| Android web browser | Mozila/5.0(Linux; Android 6.0.1: Redmi Note 5 Build/RB3N5C; wv) Apple webKit/537.36(KHTML, like Gecko) Version/4.0 Chrome/68.0.3440.91 Mobile Safari/537.36 |
| IOS web browser | Mozila/5.0(iPhone; CPU iPone OS 12\_3\_1 like Mac OS X) Apple webKit/605.36(KHTML, like Gecko) Version/12.1.1 Mobile/15E148 Safari/604.1 |
| Windows | Mozila/5.0(Windows NT 10.0; Win64; x64) Apple webKit/537.36(KHTML, like Gecko) Chrome/74.0.3729.169 Mobile Safari/537.36 |
| Mac | Mozila/5.0(Macintosh; Intel Mac OS X 10\_11\_6) Apple webKit/605.1.15(KHTML, like Gecko) Version/12.1.12 Safari/605.1.15 |
| Google BOT | Mozila/5.0(compatible; Googlebot/2.1; +http://www.google.com/bot.html) |
| Facebook BOT | Facebookecternalhit/1.0(+http://www.facebook.com/externalhit\_uatext.php) |

**API Key**

* This is the most straightforward method and the easiest way for auth
* With this method, the sender places a **username:password/ ID / Keys** into the request header.
* The credentials are encoded and decode to ensure safe transmission.
* This method does not require cookies, session IDs, login pages, and other such specialty solutions

**Authorizarion: Basic bG9sOnNIY3VyzQ==**



**Bearer Authentication/ Auth 2.0**

Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens, passes through request-response header. In General JSON Web Tokens JWT used for this purposes.

|  |  |  |  |
| --- | --- | --- | --- |
| Language | Platform | Library name | Library Sources |
| **C#** | **ASP.NET** | **jwtBeater,jose-jwt** | **Nuget package manager** |
| **PHP** | **Laravel** | **Firebase / php-jwt** | **GitHub** |
| **JS** | **Node/Express.JS** | **npm i jsonwebtoken** | **NPM** |

**JWT (JSON WEB TOKEN)**

* Compact and self-contained way for securely transmitting information between parties as a JSON object.
* Information can be verified and trusted because it is digitally signed.

**USES**

* **Authorization:** Allowing the user to access routes, services, and resources.
* **Information Exchange**: Way of securely transmitting information between parties.

**JSON WEB TOKEN STRUCTURE**

* **Header**
* **Payload**
* **Signature**

**JSON WEB TOKEN HEADER:**

* **Type of the token**
* **Signing algorithm**

**{**

**“alg”:”HS256”,**

**“typ”:”JWT”**

**}**

**JSON WEB TOKEN PAYLOAD:**

* **Registered claims: iss (issuer), exp (expiration time), sub (subject), aud (audience)**
* **Public claims: These can be defined at will by those using JWTs.**
* **Private claims: These are the custom claims created to share information between parties.**

**{**

**“sub”:”12345678900”,**

**“name”:”Ali Akbor”,**

**“id”:”123456”,**

**“exp”:”09876543”**

**}**

**JSON WEB TOKEN SIGNATURE:**

**To create the signature part -**

* **Take the encoded header**
* **Take the encoded payload, a secret**
* **The algorithm specified in the header**

**HMACSHA256(**

**base64UrlEncode(header) + ”.” +**

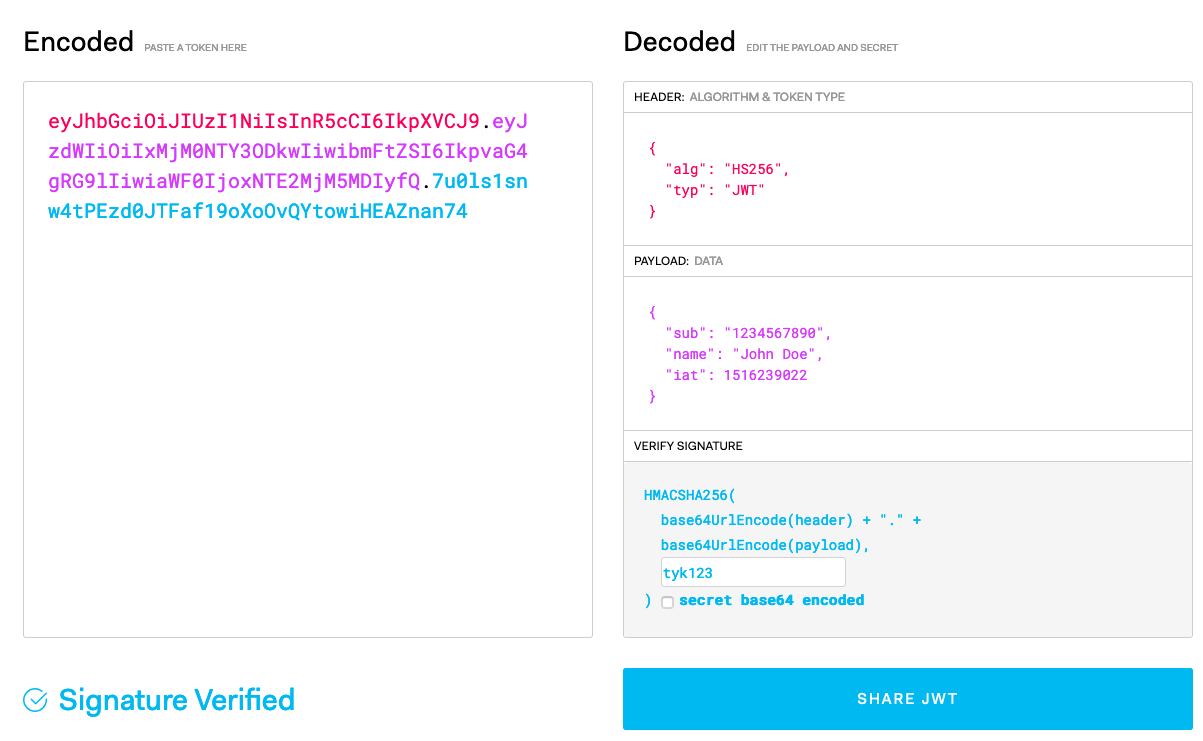
**base64UrlEncode(payload),**

**your-256-bit-secret**

**) secret base64 encoded**

**JSON WEB TOKEN**

**Putting all together**

****

**Essential Packages**

**Express:** The core backbone

**Body-parser:** This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

**Cookie-parser:** Used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.

**Multer:** This is a node.js middleware for handling multipart/form-data.

**Jsonwebtoken:** Securely transmitting information between parties as a JSON object

**MySQL Driver:** To access a MySQL database with Node.js

**Mongo dB Driver:** To access a Mongo database with Node.js

**Dotenv:** Dotenv is a zero-dependency module that loads environment variables

**Cors:** CORS is a node.js package for providing a [Connect](http://www.senchalabs.org/connect/)/[Express](http://expressjs.com/) middleware that can be used to enable [CORS](http://en.wikipedia.org/wiki/Cross-origin_resource_sharing) with various options.

**Express-mongo-sanitize:** Sanitizes user-supplied data to prevent MongoDB Operator Injection.

**Express-rate-limit:** Rate-limiting middleware for Express. Use to limit repeated requests to public APIs and/or endpoints.

**Helmet:** Helmet helps you secure your Express apps by setting various HTTP headers.

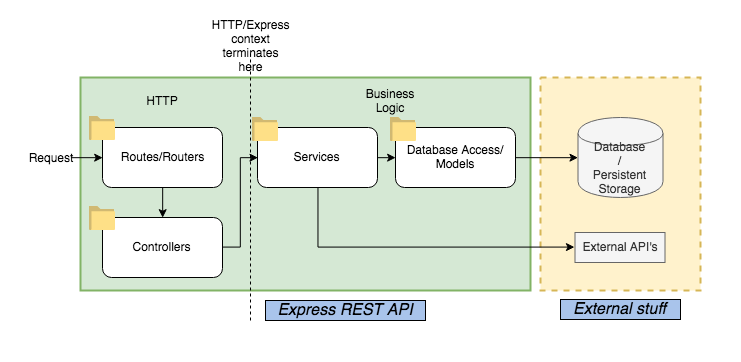
**HPP**: Express middleware to protect against HTTP Parameter Pollution attacks

**Validator:** A library of string validators and sanitizers.

**Xss-clean:** Connect middleware to sanitize user input coming from POST body, GET queries, and url params.

**File Folder Structure**

* **For Monolithic Application MVC (Model, View, Controller)**
* **For Rest API MC (Model, Controller )**
* **Index.js :** Responsible for connecting the MongoDB and starting the server.
* **App.js** : Configure everything that has to do with Express application**.**
* **Config.env:** for Enivrement Variables.
* **Routes -> Routes.js:** The goal of the route is to guide the request to the correct handler function which will be in one of the controllers.
* **Controllers -> Controller.js:** Handle the application request, interact with models and send back the response to the client.
* **Models -> Model.js:** If we use Mongoose this will be schema definition for creating and reading documents from the underlying MongoDB database.

****