ENVRI Community International Winter School on DATA FAIRness

https://www.lifewatch.eu/envri-iws-data-fairness-2020

Some Security Issues Around APIs

AJ Sáenz-Albanés, LifeWatch ERIC

José María García, SCORE Lab, University of Seville, LifeWatch Spain JRU

11-22 January 2021

ENVRI-FAIR has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824068

Outline

- Recommendations to consider before developing a RESTful API
- REST API Authentication Types Overview
- Securing Your REST API: Best Practices

Recommendations to consider before developing a RESTful API

- Data Protection
- TLS
- DOS Attacks
- Anti-Farming

Recommendations to consider before developing a RESTful API

- Data Protection
  - A RESTful API is the way in which a given service can present value to the world.
  - As a result, protection of the data provided via RESTful endpoints should always be a high priority.
  - You have to define clear access rights, especially for methods like DELETE (deletes a resource) and PUT (updates a resource).
  - Those methods must be accessed by authenticated users only, and for each such call, an audit must be saved.

Recommendations to consider before developing a RESTful API

- TLS
  - Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communications security over a computer network.
  - When secured by TLS, connections between a client and a server have one or more of the following properties:
    - The connection is private (or secure) because symmetric cryptography is used to encrypt the data transmitted.
    - The keys for this symmetric encryption are generated uniquely for each connection and are based on a shared secret negotiated at the start of the session.
    - The identity of the communicating parties can be authenticated using public-key cryptography.
    - The connection ensures integrity because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.

Recommendations to consider before developing a RESTful API

- DOS Attacks
  - In a Denial of Service (DOS) attack, the attacker usually sends excessive messages asking the network or server to acknowledge requests that have invalid return addresses.
  - DOS attacks can render a RESTful API into a non-functional state if the right security measures are not taken.
  - A successful DOS attack means that all the connected clients (partners, apps, mobile devices, and more) will not be able to access the API.

Recommendations to consider before developing a RESTful API

- Anti-Farming
  - External aggregated service that takes advantage of provided APIs to obtain the information they want you to utilize.
  - When this happens, the RESTful API is being farmed out for the benefit of another entity.
  - In case your API does not have an Authorization/Authentication mechanism, it might lead to misuse of your API, loading the servers and the API itself, making it less responsive to others.

REST API Authentication Types Overview

- RESTful applications rely on the underlying security of the API ecosystem rather than including security within the REST architecture style.
- In addition to securing RESTful API calls with the HTTPS protocol, session-based authentication should be utilized.
- Currently, most RESTful applications leverage OAuth 2.0 and JWT is the newcomer that is gaining more and more popularity with API developers.
  - Basic Authentication
  - API Keys
  - Security Assessment Markup Language (SAML)
  - OAuth 2
  - JSON Web Token (JWT)

REST API Authentication Types Overview

- Basic Authentication
  - HTTP Basic authentication implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifiers, or login pages; rather, HTTP Basic authentication uses standard fields in the HTTP header, removing the need for handshakes.
  - To receive authorization, the client sends the userid and password, separated by a single colon (":") character, within a Base64 encoded string in the credentials.
    - If the user agent wishes to send the userid "Aladdin" and password "open sesame," it would use the following header field:
    - **Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==**
  - The Basic authentication scheme is not a secure method of user authentication, nor does it in any way protect the entity, which is transmitted in cleartext across the physical network used as the carrier.
  - The most serious flaw in Basic authentication is that it results in the essentially cleartext transmission of the user's password over the physical network.
  - Because Basic authentication involves the cleartext transmission of passwords, it should be used over TLS or SSL protocols (HTTPS) in order to protect sensitive or valuable information.

REST API Authentication Types Overview

- API Keys
  - Not any formal standard, but something in common practice by API providers, and supported by API management providers.
  - It is the usage of one or two keys what accompany every API call. API keys are really more about identifying the application and user over being anything about security, but is perceived as secure by many.
  - Public REST services without access control run the risk of being farmed, leading to excessive bills for bandwidth or compute cycles. API keys can be used to mitigate this risk.
  - They are also often used by organization to monetize APIs; instead of blocking high-frequency calls, clients are given access in accordance to a purchased access plan.
  - Typically, an API key gives full access to every operation an API can perform, including writing new data or deleting existing data. If you use the same API key in multiple apps, a broken app could destroy your users' data without an easy way to stop just that one app.
  - Some apps let users generate new API keys, or even have multiple API keys with the option to revoke one that may have gone into the wrong hands.
  - The ability to change an API key limits the security downsides.
  - **Beware of GitHub !!!!!**

REST API Authentication Types Overview

- Security Assessment Markup Language (SAML)
  - It is an XML-based framework for authentication and authorization between two entities: A Service Provider and an Identity Provider.
  - The Service Provider agrees to trust the Identity Provider to authenticate users. In return, the Identity Provider generates an authentication assertion, which indicates that a user has been authenticated.
  - SAML is a standard single sign-on (SSO) format. Authentication information is exchanged through digitally signed XML documents.
  - It's a complex single sign-on (SSO) implementation that enables seamless authentication, mostly between businesses and enterprises.

REST API Authentication Types Overview

- OAuth 2
  - Created in 2006, OAuth 2 is an open standard for authentication protocol that provides authorization workflow over HTTP and authorizes devices, servers, applications, and APIs with access tokens instead of credentials. OAuth gained popularity from usage by Facebook, Google, Microsoft, and Twitter, who allow usage of their accounts to be shared with third-party applications or websites.
  - OAuth 2.0 can be used to read data of a user from another application without compromising the user's personal and sensitive data, like user credentials. It also supplies the authorization workflow for web, desktop applications, and mobile devices.
  - The previous versions of this spec, OAuth 1.0 and 1.0a, were much more complicated than OAuth 2.0. The biggest change in the latest version is that it's no longer required to sign each call with a keyed hash. The most common implementations of OAuth use one or both of these tokens instead:
    - **access token:** sent like an API key, it allows the application to access a user's data; optionally, access tokens can expire.
    - **refresh token:** optionally part of an OAuth flow, refresh tokens retrieve a new access token if they have expired.
  - Since an access token is like a special type of API key, the most likely place to put it is the authorization header, like so:
    - **Authorization: Bearer 1234567890abcdef**

REST API Authentication Types Overview

- JSON Web Token (JWT) (1)
  - It is an open standard extension of OAuth 2.0 for creating access tokens that assert some number of claims.
  - Whereas API keys and OAuth tokens are always used to access APIs, JSON Web Tokens (JWT) can be used in many different scenarios. In fact, JWT can store any type of data, which is where it excels in combination with OAuth.
  - Like OAuth access tokens, JWT tokens should be passed in the Authorization header:
    - **Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJob21lcGFnZSI6Imh0dHBzOi8vemFwaWVyLmNvbSIsInRhZ2xpbmUiOiJaYXBpZXIgbWFrZXMgeW91IGhhcHBpZXIiLCJmb3JtIjoiaHR0cHM6Ly96YXBpZXIudHlwZWZvcm0uY29tL3RvL0hkRVk0eiJ9.E3EtYy2y7BRn4eS0RIyDAAh-KAsa6dVV91ULbBJCRJw**

REST API Authentication Types Overview

- JSON Web Token (JWT) (2) What Are JSON Web Tokens?
  - JSON Web Token (JWT), pronounced "jot," is an open standard (RFC 7519 https://tools.ietf.org/html/rfc7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.
  - JWT features:
    - **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.

REST API Authentication Types Overview

- JSON Web Token (JWT) (3) What Does a JWT Look Like?
  - JSON Web Tokens consist of three parts separated by dots (.), which are:
    - Header
    - Payload
    - Signature
  - For example:
    - aaaaaaaaaa.bbbbbbbbbbb.ccccccccccc
  - Since there are three parts separated by a (.), each section is created differently.

REST API Authentication Types Overview

- JSON Web Token (JWT) (4) Header
  - The header carries two parts:
  - Declaring the type, which is JWT, and the hashing algorithm to use (HMAC SHA256 in this case).
    - **{**
    - **"typ": "JWT",**
    - **"alg": "HS256"**
    - **}**
  - In this JSON, the value of the “typ” key specifies that the object is a JWT, and the value of the “alg” key specifies which hashing algorithm is being used to create the JWT signature component. In our example, we’re using the HMAC-SHA256 algorithm, a hashing algorithm that uses a secret key, to compute the signature.
  - Now all that is left is to Base64 Encode this JSON and we have the first part of our JSON web token:
    - **eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9**

REST API Authentication Types Overview

- JSON Web Token (JWT) (5) Payload
  - The second part of the token is the payload, which contains the claims.
  - Claims are statements about an entity (typically the user) and additional metadata. Notice that the claim names are only three characters long, as JWT is meant to be compact.
  - There are three types of claims: reserved, public, and private claims.
  - **Reserved claims:** These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. In the preferred JWT standard, some claims are:
    - **exp**: Expiration Value (Type: Number – DateTime). This is commonly used by the issuer. It has a numeric value that denotes a time interval. The token will expire after this time interval elapses from the current time.
    - **iss**: Issuer (Type: String or URI)
    - **sub**: Subject (Type: String)
    - **aud**: Audience (Type: String or URI)
    - **nbf**: Not Before (Type: Number – DateTime). Denotes a numeric value of time, before which the token must not be accepted for processing.
    - **iat**: Issued At (Type: Number – DateTime). Denotes the numeric format of issued date and time.
    - **jti**: JWT ID (Type: String). The unique identifier for the JWT token.

REST API Authentication Types Overview

- JSON Web Token (JWT) (6) Payload
  - **Public claim**s: These can be defined at will using JWTs, but to avoid collisions, they should be defined in the IANA JSON Web Token Registry or defined as a URI that contains a collision-resistant namespace.
  - **Private claims**: These are the custom claims created to share information between parties that agree on using them.
    - **{**
    - **"sub": "AB324901",**
    - **"name": "Michael",**
    - **"admin": true,**
    - **"exp": 2601638760**
    - **}**
  - The Base64 of the above encoded form should look like:
    - **eyJzdWIiOiJBQjMyNDkwMSIsIm5hbWUiOiJNaWNoYWVsIiwiYWRtaW4iOnRydWUsImV4cCI6MjYwMTYzODc2MH0**

REST API Authentication Types Overview

- JSON Web Token (JWT) (7) Signature
  - To create the signature part, you have to take the encoded header, the encoded payload, a secret, and the algorithm specified in the header, and sign it.
  - For example, if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:
    - **data = base64urlEncode(header) + "." + base64urlEncode(payload)**
    - **signature = Hash(data, secret);**
  - The signature is used to verify that the sender of the JWT is who they say they are, and to ensure that the message wasn't changed along the way.

REST API Authentication Types Overview

- JSON Web Token (JWT) (8) Putting It All Together
  - The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML.
  - The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.
    - **eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJBQjMyNDkwMSIsIm5hbWUiOiJNaWNoYWVsIiwiYWRtaW4iOnRydWUsImV4cCI6MjYwMTYzODc2MH0.o79bwAmsmfxmORr6G_TpAawSv5nGu6UoB9QfT6XYVws**
  - You can create your own JWT here - https://jwt.io/

REST API Authentication Types Overview

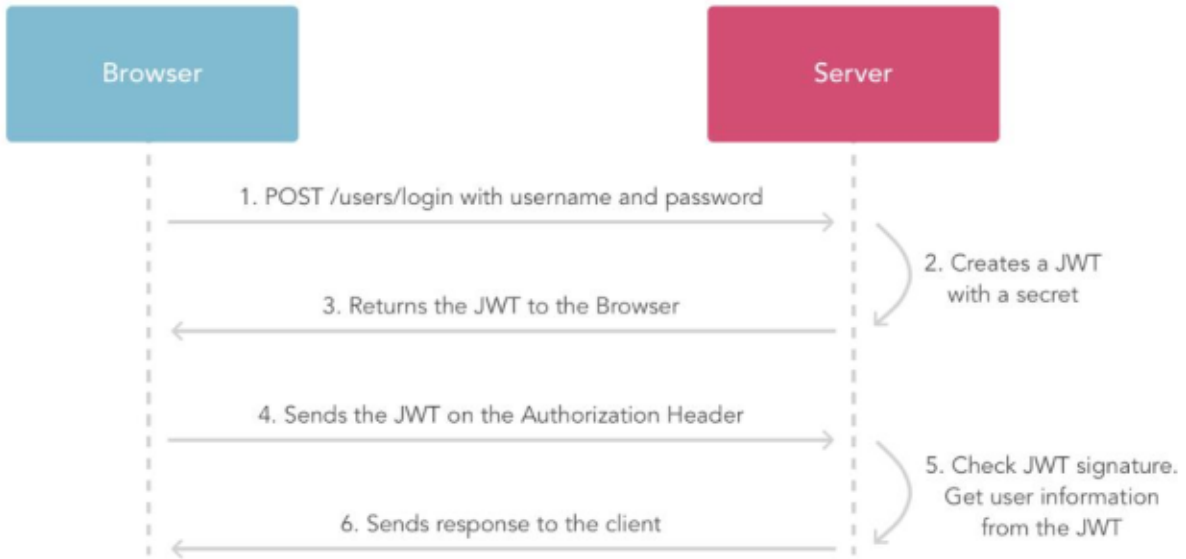
- JSON Web Token (JWT) (9) When Should You Use JSON Web Tokens?
  - **Authentication**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single sign-on is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
  - **Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

REST API Authentication Types Overview

- JSON Web Token (JWT) (10) How Do JSON Web Tokens Work?
  - In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned and must be saved locally (typically in local storage, but cookies can be also used), instead of the traditional approach of creating a session in the server and returning a cookie.
  - Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:
    - **Authorization: Bearer <token>**
  - This is a stateless authentication mechanism as the user state is never saved in the server's memory. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. As JWTs are self-contained, all the necessary information is there, reducing the need to query the database multiple times.
  - This allows you to fully rely on data APIs that are stateless, and even make requests to downstream services. It doesn't matter which domains are serving your APIs, so Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.

REST API Authentication Types Overview

- JSON Web Token (JWT) (11) How Do JSON Web Tokens Work? (Diagram)



Securing Your REST API: Best Practices

- Rate Limiting
- Authorization
- Input Validation
- Output Encoding
- HTTP Status Codes

Securing Your REST API: Best Practices

- Rate Limiting
  - An API key is a valuable strategy to provide a level of identity to consumers of a RESTful API.
  - In addition to providing tiered services, another benefit of using API keys is the ability to throttle usage of the API, and the logging of all API calls made allows API providers to limit the rate of consumption for all API users.
  - Putting caps on the number of API calls that can be made for any single API resource, dictating consumption by the second, minute, day, other relevant constraint.

Securing Your REST API: Best Practices

- Authorization (1)
  - Protect HTTP Methods
    - RESTful APIs often use GET (read), POST (create), PUT (replace/update) and DELETE (to delete a record).
    - Not all of these are valid choices for every single resource collection, user, or action. Make sure the incoming HTTP method is valid for the session token/API key and associated resource collection, action, and record.
  - Whitelist Allowable Methods
    - It is common with RESTful services to allow multiple methods for a given URL for different operations on that entity.
    - For example, a GET request might read the entity, while PUT would update an existing entity, POST would create a new entity, and DELETE would delete an existing entity.
    - It is important for the service to properly restrict the allowable verbs such that only the allowed verbs would work, while all others would return a proper response code (for example, a 403 Forbidden).

Securing Your REST API: Best Practices

- Authorization (2)
  - Protect Privileged Actions and Sensitive Resource Collections
    - Not every user has a right to every web service. This is vital, as you don't want administrative web services to be misused: https://example.com/admin/exportAllData
    - The session token or API key should be sent along as a cookie or body parameter to ensure that privileged collections or actions are properly protected from unauthorized use.
  - Protect Against Cross-Site Request Forgery
    - For resources exposed by RESTful web services, it's important to make sure any PUT, POST, and DELETE request is protected from Cross-Site Request Forgery. Typically, one would use a token-based approach.
    - CSRF is easily achieved — even using random tokens — if any XSS exists within your application, so please make sure you understand how to prevent XSS.

Securing Your REST API: Best Practices

- Input Validation
  - Assist the user > Reject input > Sanitize (filtering) > No input validation
    - Assisting the user makes the most sense, as the most common scenario is "problem exists between keyboard and chair" (PEBKAC).
  - URL Validations
    - Web applications/web services use input from HTTP requests (and occasionally files) to determine how to respond.
    - Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms.
    - Common names for common input tampering attacks include: forced browsing, command insertion, cross site scripting, buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation.
  - XML Input Validation
    - XML-based services must ensure that they are protected against common XML-based attacks by using secure XML-parsing.
    - This typically means protecting against XML External Entity attacks, XML-signature wrapping, etc.
    - See http://ws-attacks.org for examples of such attacks.

Securing Your REST API: Best Practices

- Output Encoding
  - Security Headers
    - To make sure the content of a given resource is interpreted correctly by the browser, the server should always send the Content-Type header with the correct Content-Type, and the Content-Type header should preferably include a charset.
    - The server should also send an X-Content-Type-Options: nosniff to make sure the browser does not try to detect a different Content-Type than what is actually sent (as this can lead to XSS).
    - Additionally, the client should send an X-Frame-Options: deny to protect against drag-and-drop clickjacking attacks in older browsers.
  - JSON Encoding
    - A key concern with JSON encoders is preventing arbitrary JavaScript remote code execution within the browser... or, if you're using Node.js, on the server. It's vital that you use a proper JSON serializer to encode user-supplied data properly to prevent the execution of user-supplied input on the browser.
  - XML Encoding
    - XML should never be built by string concatenation. It should always be constructed using an XML serializer. This ensures that the XML content sent to the browser is parseable and does not contain XML injection.

Securing Your REST API: Best Practices

- HTTP Status Codes (not just 200 / 404)

Return Code	Message	Description
200	OK	Response to a successful REST API action. The HTTP method can be GET, POST, PUT, PATCH or DELETE.
201	Created	The request has been fulfilled and the resource created. A URI for the created resource is returned in the Location header.
400	Bad Request	The request is malformed, such as a message body format error, missing headers, etc.
401	Unauthorized	Wrong or no authentication ID/password provided.
403	Forbidden	Used when the authentication succeeded but the authenticated user doesn't have permission to the requested resource.
404	Not Found	When a non-existent resource is requested.
406	Unacceptable	The client presented a content type in the Accept header which is not supported by the server API.
405	Method Not Allowed	The error for an unexpected HTTP method. For example, the REST API is expecting HTTP GET, but HTTP PUT is used.
413	Payload Too Large	Used to signal that the request size exceeded the given limit, e.g. regarding file uploads and to ensure that the requests have reasonable sizes.
415	Unsupported Media Type	The requested content type is not supported by the REST service. This is especially effective when you are working primary with JSON or XML media types.
429	Too Many Requests	The error is used when there may be a DOS attack detected or the request is rejected due to rate limiting.

ENVRI
FAIR

envri.eu/envri-fair

@envri_fair

company/envri-fair

facebook.com/ENVRIcomm