RestCase
REST API Development Cycle Management

MENU

4 Most Used REST API Authentication Methods

26 JULY 2019 on RestCase, REST API Security, REST API, OAS, API Driven Development

While there are as many proprietary authentication methods as there are systems which utilize them, they are largely variations of

a few major approaches. In this post, I will go over the 4 most used in the REST APIs and microservices world.

Authentication vs Authorization

Before I dive into this, let's define what authentication actually is, and more importantly, what it’s not. As much as authentication drives the modern internet, the topic is often conflated with a closely related term: authorization.

The two functions are often tied together in single solutions, but the easiest way to divide authorization and authentication is to ask: **what do they actually state or prove about me**?

Authentication is when an entity proves an identity. In other words, Authentication proves that you are who you say you are. This is like having a driver license which is given by a trusted authority that the requester, such as a police officer, can use as evidence that suggests you are in fact who you say you are.

Authorization is an entirely different concept and in simple terms, Authorization is when an entity proves a right to access. In other words, Authorization proves you have the right to make a request. Consider the following - You have a working key card that allows you to open only some doors in the work area, but not all of them.

In summary:
Authentication: Refers to proving correct identity
Authorization: Refers to allowing a certain action

An API might authenticate you but not authorize you to make a certain request.

Now that we know what authentication is, let's see what are the most used authentication methods in REST APIs.

4 Most Used Authentication Methods

Let's review the 4 most used authentication methods used today.

1. HTTP Authentication Schemes (Basic & Bearer)

The HTTP Protocol also defines HTTP security auth schemes like:

- Basic
- Bearer
- Digest
- OAuth

  and others...

We will go over the two most popular used today when discussing REST API.

Basic Authentication

HTTP Basic Authentication is rarely recommended due to its inherent security vulnerabilities.

This is the most straightforward method and the easiest. With this method, the sender places a username:password into the request header. The username and password are encoded with Base64, which is an encoding technique that converts the username and password into a set of 64 characters to ensure safe transmission.

This method does not require cookies, session IDs, login pages, and other such specialty solutions, and because it uses the HTTP header itself, there's no need to handshakes or other complex response systems.

Here's an example of a Basic Auth in a request header:

```
Authorization: Basic bG9sOnNlY3VyZQ==
```

Bearer Authentication

Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.

The name "Bearer authentication" can be understood as "give access to the bearer of this token." The bearer token allowing access to a certain resource or URL and most likely is a cryptic string, usually generated by the server in response to a login request.

The client must send this token in the Authorization header when making requests to protected resources:

```
Authorization: Bearer <token>
```

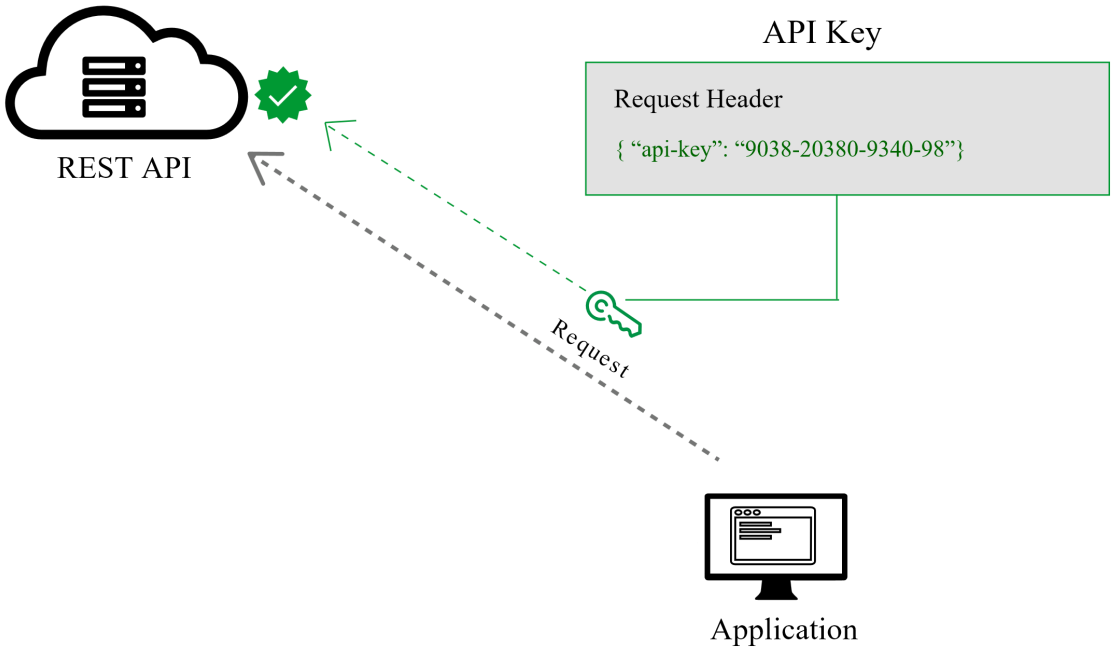
The Bearer authentication scheme was originally created as part of OAuth 2.0 in RFC-6750 but is sometimes also used on its own.

Similarly to Basic authentication, Bearer authentication should only be used over HTTPS (SSL).

2. API Keys

In REST API Security - API keys are widely used in the industry and became some sort of standard, however, this method should not be considered a good security measure.

API Keys were created as somewhat of a fix to the early authentication issues of HTTP Basic Authentication and other such systems. In this method, a unique generated value is assigned to each first time user, signifying that the user is known. When the user attempts to re-enter the system, their unique key (sometimes generated from their hardware combination and IP data, and other times randomly generated by the server which knows them) is used to prove that they're the same user as before.



Many API keys are sent in the query string as part of the URL, which makes it easier to discover for someone who should not have access to it. Please do not put any API keys or sensitive information in query string parameters! A better option is to put the API key in the Authorization header. In fact, that’s the proposed standard: `Authorization: Apikey 1234567890abcdef`.

Yet, in practice API keys show up in all sorts of places:

- Authorization Header
- Basic Auth
- Body Data
- Custom Header
- Query String

There are definitely some valid reasons for using API Keys. First and foremost, **API Keys are simple**. The use of a single identifier is simple, and for some use cases, the best solution. For instance, if an API is limited specifically in functionality where "read" is the only possible command, an API Key can be an adequate solution. Without the need to edit, modify, or delete, security is a lower concern.

The problem, however, is that anyone who makes a request to a service, transmits their key and in theory, this key can be picked up just as easy as any network transmission, and if any point in the entire network is insecure, the entire network is exposed.

If you are dealing with Authentication in REST APIs, please consider doing Security Testing, in order to check the common vulnerabilities.

3. OAuth (2.0)

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. OAuth2 combines Authentication and Authorization to allow more sophisticated scope and validity control.

OAuth 2.0 is the best choice for identifying personal user accounts and granting proper permissions. In this method, the user logs into a system. That system will then request authentication, usually in the form of a token. The user will then forward this request to an authentication server, which will either reject or allow this authentication. From here, the token is provided to the user, and then to the requester. Such a token can then be checked at any time independently of the user by the requester for validation and can be used over time with strictly limited scope and age of validity.

This is fundamentally a much more secure and powerful system than the other approaches, mainly because it allows for the establishment of scopes which can provide access to different

parts of the API service and since the token is revoked after a certain time - makes it much harder to re-use by attackers.

OAuth 2.0 Popular Flows

The flows (also called grant types) are scenarios an API client performs to get an access token from the authorization server.

OAuth 2.0 provides several popular flows suitable for different types of API clients:

- **Authorization code** – The most common flow, mostly used for server-side and mobile web applications. This flow is similar to how users sign up into a web application using their Facebook or Google account.
- **Implicit** – This flow requires the client to retrieve an access token directly. It is useful in cases when the user's credentials cannot be stored in the client code because they can be easily accessed by the third party. It is suitable for web, desktop, and mobile applications that do not include any server component.
- **Resource owner password** – Requires logging in with a username and password. Since in that case, the credentials will be a part of the request, this flow is suitable only for trusted clients (for example, official applications released by the API provider).
- **Client Credentials** – Intended for the server-to-server authentication, this flow describes an approach when the client application acts on its own behalf rather than on behalf of any individual user. In most scenarios, this flow provides the means to allow users to specify their credentials in the client

application, so it can access the resources under the client's control.

4. OpenID Connect

OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol, which allows computing clients to verify the identity of an end-user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner.

In technical terms, OpenID Connect specifies a RESTful HTTP API, using JSON as a data format.

OpenID Connect allows a range of clients, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. The specification suite is extensible, supporting optional features such as encryption of identity data, the discovery of OpenID Providers, and session management.

OpenID Connect defines a sign-in flow that enables a client application to authenticate a user, and to obtain information (or "claims") about that user, such as the user name, email, and so

on. User identity information is encoded in a secure JSON Web Token (JWT), called ID token.

- JWT

JSON Web Tokens are an open, industry-standard RFC 7519 method
for representing claims securely between two parties. JWT allows you to decode, verify and generate JWT. While JWT is a standard it was developed by Auth0, an API driven identity, and authentication
management company.

OpenID Connect defines a discovery mechanism, called OpenID Connect Discovery, where an OpenID server publishes its metadata at a well-known URL, typically `https://server.com/openid-configuration`.

This URL returns a JSON listing of the OpenID/OAuth endpoints, supported scopes and claims, public keys used to sign the tokens, and other details. The clients can use this information to construct a request to the OpenID server. The field names and values are defined in the OpenID Connect Discovery Specification.

OpenAPI Security Schemes

In OpenAPI specification, in order to define what kind of a security mechanism is used across the API - API security schemes are used to define what API resources are secured and what means.

In OpenAPI specification there are a number of standard authentication protocols you can pick from, each with their own strengths and weaknesses.

Basic API Authentication

- Easy to implement, supported by nearly all web servers
- Entails sending base-64 encoded username and passwords
- Should not be used without SSL
- Can easily be combined with other security methods

***Note**: basic authentication is very vulnerable to hijacks and man-in-the-middle attacks when no encryption is in use. Due to this limitation, this method of authentication is only recommended when paired with SSL.*

OAuth1.0 (Digest Scheme)

- Popular, tested, secure, signature driven, well-defined protocol
- Uses cryptographic signature, which is a mix of a token secret, nonce, and other request based information
- Can be used with or without SSL

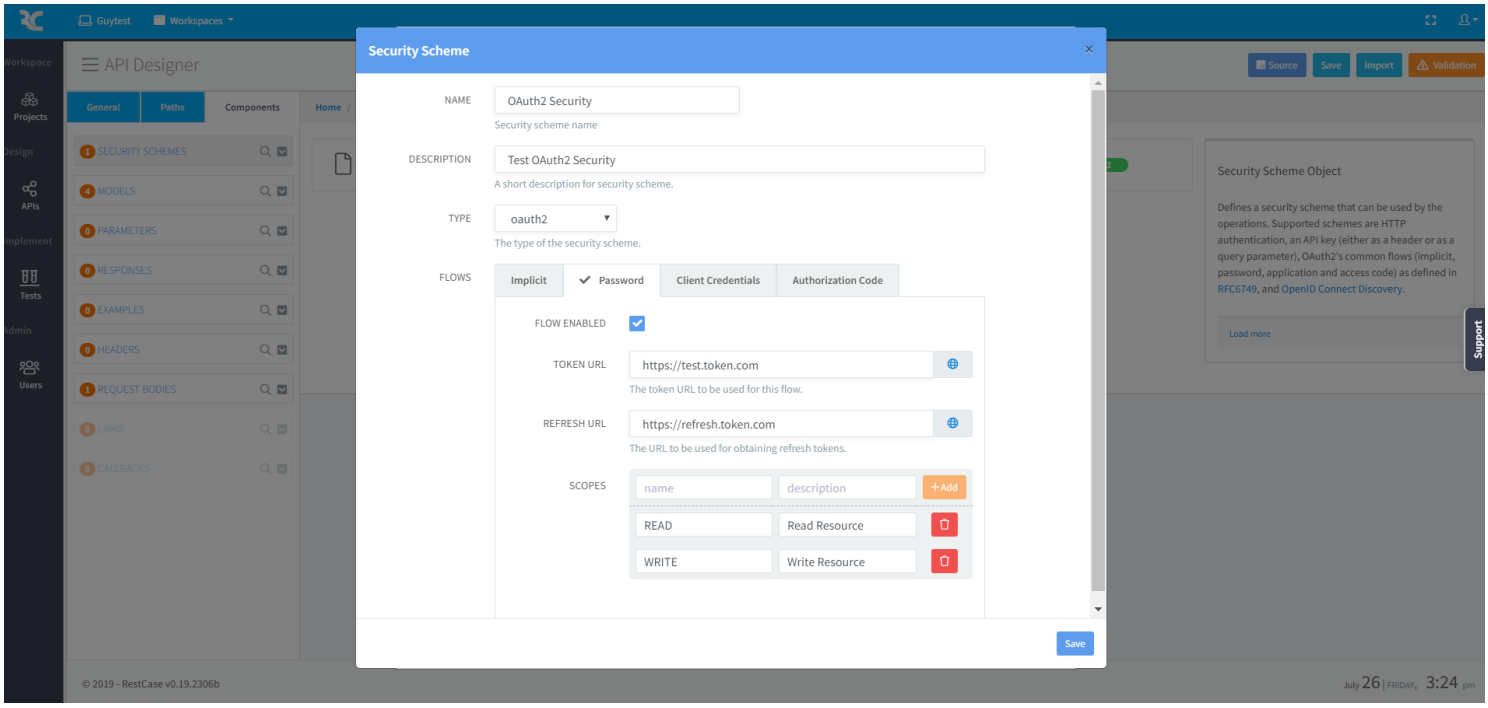
OAuth2 (Bearer Token Scheme)

- The current OAuth2 specification eliminates the need for cryptographic signatures, passwords, and usernames
- OAuth2 works with authentication scenarios called flows, these flows include:
  - Authorization Code flow
  - Implicit flow
  - Resource Owner Password flow
  - Client Credentials flow

OpenID Connect Discovery

- Based on the OAuth 2.0 protocol
- Uses a sign-in flow that permits user authentication and information access by a client app
- The user information is encoded via a secure JSON Web Token (JWT)

RestCase development platform, allows you to define these Security schemes visually, allowing to build and define the entire API without any coding knowledge.



Please feel free to join our Beta, just sign-up and start building APIs - It's free!

Summary

For now, the clear winner of the four methods is OAuth 2.0, there are some use cases in which API keys or HTTP Authentication methods might be appropriate and the new OpenID connect is getting more and more popular, mainly because it is based on an already popular OAuth 2.0.

OAuth 2.0 delivers a ton of benefits, from ease of use to a federated system module, and most importantly offers scalability of security – providers may only be seeking authentication at this time, but having a system that natively supports strong authorization in addition to the baked-in authentication methods is very valuable, and decreases cost of implementation over the long run.

Guy Levin

Read more posts by this author.

http://www.restcase.com

Share this post

ALSO ON **RESTCASE BLOG**

7 Rules for REST API URI Design

4 years ago • 19 comments

By following the 7 rules for REST API URI design in the post, you will create a ...

4 Maturity Levels of REST API Design

2 years ago • 7 comments

4 maturity levels of REST API with a more vide view along with REST API ...

5 Basic REST API Design Guidelines

4 years ago • 7 comments

As soon as we start workin on an API, design issues arise. A robust and strong .

What do you think?

116 Responses

Upvote Funny Love Surprised Angry

Sad

1 Comment RestCase Blog Disqus' Privacy Policy Login

Subscribe to REST API and Beyond

Get the latest posts delivered right to your inbox.

Your email address SUBSCRIBE

or subscribe via RSS with Feedly!

READ THIS NEXT

REST APIs - How To Handle "Man In The Middle" Security Threat

An API, or Application Programming Interface, is how software talks to other software. Every day, the variety of APIs...

YOU MIGHT ENJOY

OpenAPI Spec: Documentation and Beyond

OpenAPI has become the industry standard for defining an API, yet it is often treated as a documentation tool...