Best practices of API authentication and Authorization

Authentication and authorization are practices that are requisite to gain access to an API. The reasons for this are not unethical. Users need to be validated before making requests to APIs for specific reasons; private data has to be secured and protected at all costs to prevent it from getting into the hands of hackers and individuals with malicious intent.

APIs make use of access policies in order to ascertain the identity of the consumer, protect requests from authorized users, and ensure the security of the APIs data.

This article intends to inform you on best practices to manage your API authorization and authentication in order to prevent a breach and protect your data against cyber criminals.

Authentication (AuthN) refers to the process of proving the identity of the user sending the request and verifying their identity and credentials.

Authorization (AuthZ) on the other hand involves verifying whether the authenticated requester has permission to access the data and interact with it.

The following entails the best practices to authorize requests.

**5 Best Practices for API authentication and authorization**

We have enlisted and explained different practices to ensure your private data only gets into the hands of the right users.

* Secure the API with Transport Layer Security (TLS)

In API deployment, transport layer security is used to protect the information of the API. HTTP is the protocol used in the transfer of data between clients and servers. HTTPS is simply an API call for connecting the client and the server which is secured with TLS. TLS secures the data in transit for the API endpoint by encrypting the information. TLS prevents third-party from gaining access to the information by providing encryption to ensure confidentiality and security of the message.

TLS is issued by a certificate authority and the authentication and encryption file is installed on the website to authorize TLS. This file is called an SSL certificate.

A website that uses TLS can be identified with its URL using HTTPS rather than HTTP.

Version 1.3 of TLS is appropriate but version 1.2 could still be used.

* Secure the API by using an OAuth2.0

OAuth2 is an authorization standard that is used to access a resource from a resource server on behalf of the user. It serves as an intermediary between the user, the application, and the resource server (API).

To explain it in detail, OAuth2 process works in this way:

1. The application requests authorization from the user.
2. The user grants the application access to their account. The extent of the application access to the user’s account is limited to what the user authorized.
3. The application presents the proof of authorization to the resource server to get the access token
4. The resource server verifies the user and delivers the token to the application.

Consider this scenario: A user is trying to make a payment to an online business website and the website needs their credit card details. If the website wants to get that information, OAuth will serve as the mediator to enable this transaction without revealing the credit card details to the website. The user will authorize the OAuth application to get the token access from a third party, i.e a payment platform that will grant access to the user’s bank account without revealing the card details.

It is likely that you have come across an OAuth in action in your day-to-day dealings.

OAuth primarily works to authorize and not to authenticate. For instance, in the example explained above.

* Give API keys to the API consumers in your client database

This way you automatically restrict access to your API data by streamlining projects or application requests.

API keys do not directly identify the specific users, they work by identifying the projects or specific requests. That makes it important to build up a list of IP addresses that are allowed.

You can set up a client database of consumers who have access to your API using the [Identity and Service Management (IAM)](https://cloud.google.com/iam/docs#docs) service.

API keys work like passwords for the user making the API request and on the part of the API, they enable the right service access and track how their data are being used.

How the API key is sent is peculiar to the API in question. Some might require that you enter the API key as a query parameter, body parameter, or in the header of the API request.

While it becomes the responsibility of the API consumer to keep the key safe after it has been assigned to them, it is important to enlighten them on the sensitivity of the key and suggest ways to keep it safe.

* Consider implementing session binding

Session binding involves binding the IP addresses of the authenticated user to session tokens and authenticated access tokens. This way if the access token gets stolen or misplaced and gets into the custody of a hacker and the hacker tries to impersonate the authenticated user from a different IP address, they would get restricted. Session binding recognizes devices and IP addresses.

One drawback of session binding is that if an authenticated user attempts to switch between devices they might be required to authenticate anew.

* Enable authorization for access requests

Interestingly, not all requests require authorization. There are basic requests that when provided do not compromise the safety of the API data. These requests do not need authorization. Analyze which requests require authorization and enable authentication on those ones.

There should be a system that distinguishes requests. Access should be given to the particular requests made. The API key should be peculiar to the API request.

Always remember that it’s vital to protect your data. It doesn’t hurt to be a little paranoid about protecting your API. Don’t leave anything to chance or presumption. Secure all ends and treat every request like a new one. It is also necessary to withhold vital details and not share more than required.