API Authentication Summary Report

Prepared by: Falilat Owolabi
Date: Dec 1st, 2025

As the week progressed, I was able to understand the history of Oauth 1.0 and oauth 2.0, Authentication processed in Api, interaction patterns, jwt scopes and claim within a token, Api gateways and Api Authorization, here is a detailed summary of each Concept :

OAuth 1.0History:
Standardized in 2010 by IETF, Driven by Twitter and Google created to enable third-party integrations without exposing user credentials (avoiding asic auth)Relied heavily on signing (messages/exchanges signed). It not require HTTPS, difficult to implement (signing complexity) and it did not gain much traction

OAuth 2.0 History:
Standardized in late 2012 created to Simplify implementation by removing signing requirements. Easier to build OAuth clients/applications, the key difference with Oath 1.0 is that No signing required, requires HTTPS, much easier to implement (especially clients). It is the current standard for API security and not compatible with OAuth 1.0

OAuth 2.1: A cleanup/consolidation of additions after 2012; backward compatible.
OAuth 2.0 replaced OAuth 1.0 primarily by removing signing, which made implementation much simpler and led to broader adoption.

OAuth lets third-party apps access APIs on behalf of users without sharing the user's password, using tokens that can expire, be scoped to specific audiences, and convey both user and machine identity for authentication and authorization.

Variant of Api Authentication: API authentication identifies who is making the request so the API can determine what they're allowed to do and prevent unauthorized access. An API can be authenticated in different ways which include - Basic Authentication (i.e username and password) - API keys - Certificate base (tls Authentication) - token based authentication - and finally token based Authorization. These variants are either machine level Authentication or user level Authentication and oftentimes both are needed.

OAUTH Actors: there are 4 actors involved in OAuth and they include - Resource Owner(the owner of the data to be accessed by an application) - Client(The Application accessing the owner data/information - Authorization server (This is the server that gives/Authorize the app to access the owner's data from the resource server usually by sending tokens back -Resource server(this is the server that has the data that needs to be accessed by the client and it needs a valid token to determine if the client do has the authorization to access the data indeed).
It should be noted that the Authorization given to the client to access the resources is not an open authorization but a delegation. What this means is that the client being authorized cannot access the data because he has a token from the authorization server until a confirmation is being made with the authorization server by the resource server before access can be granted to the client and it is based on the condition of the Authorization. If anything goes wrong even with a valid token from the Authorization server, the resource server may still deny the client in the end.

Oauth Interaction patterns: there are different kinds of interaction patterns in Oath 2.0 and the kind to use will depend on the kind of application being built, example includes code flow, refresh flow, client credential flow.

JWT Token: A token consist of 3 things which are - The format (how to read the content, how to validate it) - Purpose(who is the token for, what is the intended use, where can I send it) - Type(Overloaded word, How can the token be transmitted).
FORMAT can be by reference or by value, by reference means it is opaque to the holder, it cannot be decrypt or decoded , it will require calls to the Authorization server to read and validate and it is safe to use on the internet and By Value means that all the information is contained the token and can be validated by the receiver without making a call to the Authorization server.
Purpose The purpose of a token is determined on what kind of token it is, there is access token and there is refresh token, access token is meant to be sent to the resource server i.e the final recipient of this token and the one who is intended to be reading the contents of the token. The refresh token is only meant to be sent to the authorization server directly from the client when it needs to do a refresh. There is also ID token, ID token in OpenID Connect is received after the code flow is finished and is for the client. It should not leave the client. It's not meant to be sent to anyone else.

Type The response from the token endpoint can be either of two types, which can be either a bearer token or Proof of possession (POP) token. The bearer token is like cash which can be used by anyone, the sender is not verified and because of these attributes, it needs to be protected and should not be logged or kept in the browser. The proof of possession is like a credit card just like the name sender need to provide proof of ownership which means it cannot be sent by others.There are a couple of common mechanisms, DPOP, demonstration of proof of possession, and mutual TLS. So you can bind. the token to the mutual TLS channel. So only the client with that client certificate can use it.
A JWT is a format, it is not a purpose and it is not a type, which can be used for many purposes. Most id tokens are always jwt s and an access token can never be a jwt. A jwt is most often signed referred to as jws and can be encrypted referred to as jwe.

3 part of a jwt are:
The header: A few details about the token. I.e the key, certificate, signing algorithm
The Payload: The content of the token. The part we can tell things about the client
The Signature: the signature signs the other parts. You cannot manipulate the contents of this token without the signature becoming invalid, and that's why a JSON Web Token is very handy.
JSON Web Tokens can come in different forms. They can be signed, we call that JWS, JSON Web Signatures, or they can be encrypted, JWE, JSON Web Encryption. The signing is the most common version of JWTs out there, proves who issued the token and It uses asymmetric signatures. A valid JSON Web Token signature doesn't mean the token is still valid. It could have expired, it could be issued from the wrong party, or have the wrong audience.
A jwt is not a protocol, even thou they can be used in different ways it should be used depending on the protocol and not used against the protocol intention

Scopes and claim
scopes defines the scope of access for the token, it is a key not a value which is requested by the client and consented by the user, authorized by the authorization server used to express the client privileges and not the user level privileges best used for coarsed grained.

Claims provide users details, it is a key-value item inside the token asserted by the issuer, proven truth about the subject which can be used for fine grained.

APIs and gateways
Gateways are layer firewalls used to protect the Api, it is security focused, give you a visibility of the APIs that you expose to the internet,
Phantom token flows:
All APIs should depend on jwt
Accept no request without a jwt, verify issuers, audience validity scopes and claim
Uniform security model internally
Easy to re-use
Zero trust enabled
Identity data is kept and can be trusted at api level.

Introspection
The 3 ways Api to APi calls can happen
- Exchange: use the token exchange to get another token that will be used in the next Api, this used on demand, powerful option
- Embed: put more token inside the first token, this can be used when it is known to happen to every request
- Share: use the same token, this can happen if the api are in the same security domain.

End of Report!