Auth Strategies — Basic Authentication

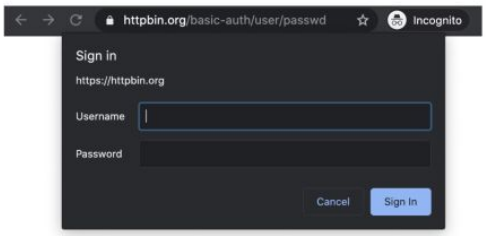
@kamranahmedse

Mechanism to authenticate access to resources over HTTP

Credentials are sent in the request headers

Authorization: Basic aGItOm92ZXJhY2hpZXZlcg==

Base64 encoded username and password



Ever seen this page?
This is basic authentication in play

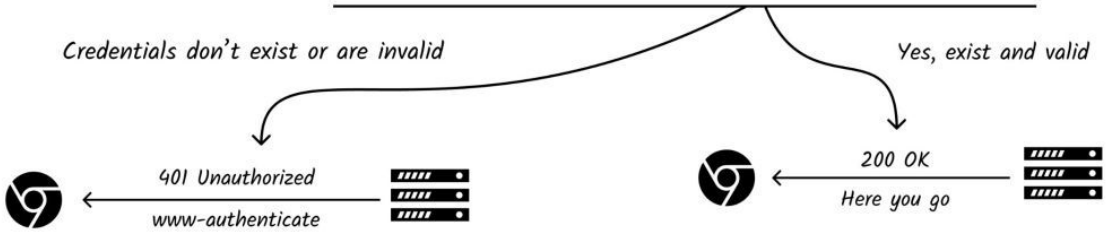
How does it work?

Step 1

Client tries to access some protected URL

Step 2

Server checks if the request has **Authorization** header with valid username and password?




```
HTTP/1.1 401 Unauthorized
Date: Sat, 16 May 2020 16:50:53 GMT
WWW-Authenticate: Basic realm="MyApp"
```

Response to the client

realm or protection space

Group of pages where the same credentials are used. Browser can cache the valid credentials for given realm and use them in future

Freeform text. Can be anything. Server is responsible for defining realms and do the authentication.

Step 3

Browser notices the **www-authenticate** header in response and presents the alert for credentials. (Similar to the one shown on the top right)

Step 4

User submits the credentials. Browser encodes them using base64 and sends in the next request. Credentials are sent in the **Authorization** header in request as follows

Authorization: Basic aGItOm92Y2hpZXZlcg==

base64("username:password")

Step 5

Go to step 2 i.e. server does the authentication and cycle continues.

Basic auth can also be used in APIs but in that case it's just like normal token based authentication

Note

Basic auth is not considered secure unless used with TLS/HTTPS.

Because anyone can eavesdrop and decode the credentials

JWT — JSON Web Tokens

@kamranahmedse

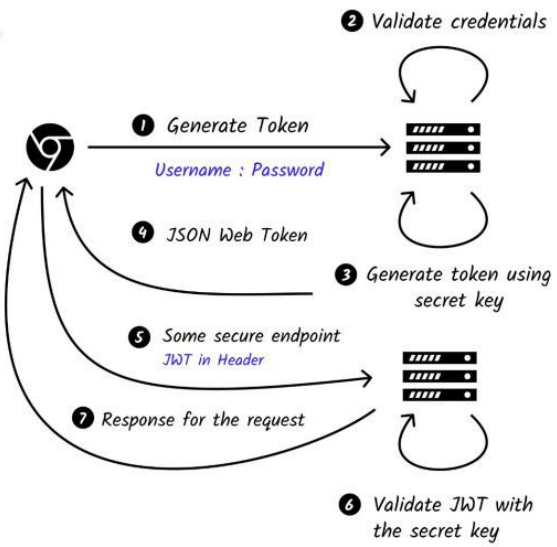
Form of token based authentication. Based on an Open Standard (RFC 7519).

Can be used for authorization as well as secure info exchange.

How does it work?

Just like any other token based auth strategy.

Only differentiator is how the token is generated.



Characteristics of the token

Token is just a normal URL-Safe string and can be passed to server in header, body or URL.

Token is self contained i.e. carries the data.

Anyone can view the content.

Token has three parts separated by a dot

XXXXXXXXXX . YYYYYYYYYY . ZZZZZZZ

header . payload . signature

String generated by hashing the header, payload with a secret i.e.

HMACSHA256(header + '.' + payload, 'secret')

held at server and used to generate and verify tokens

String generated using base64(tokenMeta)
e.g. eyJ0eXAiOiJKVIQiLCJhbGciOiJIUzIINiJ9

Token Meta

```
{
  "typ": "jwt",
  "alg": "HS256",
}
```

"typ": "jwt", → Type of token

"alg": "HS256", → hashing algorithm used for the signature part e.g. in this case SHA256

String generated using base64(ourData)
where ourData is the data that we want to embed in the token (aka JWT Claims).
e.g. eyJlc2VySWQiOiJYRjExLTExMjMiLzgiLCJpY

Our Data

```
{
  "userId": "XFII-1123",
  "email": "john@doe.com
  "exp": "1592427938",
  "iat": "1590969600"
}
```

These are called claims

There are three types of claims

Registered Claims

Standard names which are reserved for app usage

iat -> issued at (issuance timestamp)
iss -> issuer (who issued it, app name for example)
sub -> token subject
exp -> expiry time (expiry timestamp)
aud -> token audience (app URL or some string for example)
nbf -> not before (timestamp before which token is not usable)
jti -> unique token identifier (can be used to revoke existing JWT token)

In our sample payload above, we have **exp** and **iat** claims.

Public Claims

Claims which we can define and use for our own data e..g userId, email, above etc.

Private Claims

Names without any meaning to anyone except the consumer and producer of tokens.

OAuth — Open Authorization

@kamranahmedse

Open protocol for authorization that allows users to share their private resources to a third party

- Some examples
  - Authorizing your app's frontend from your API
    - Your custom OAuth Server where users of your website get authorized using OAuth.
  - Allowing some application to log you in using twitter
    - Exposing your twitter info to an external application using twitter's OAuth server.

There are two Versions
- OAuth 1.0 — deprecated
- OAuth 2.0 — active → Not backward compatible
- OAuth 2.1 — draft

How does it work?

There are 4 types of authorization flows for generating a token. → Also called **grant types**

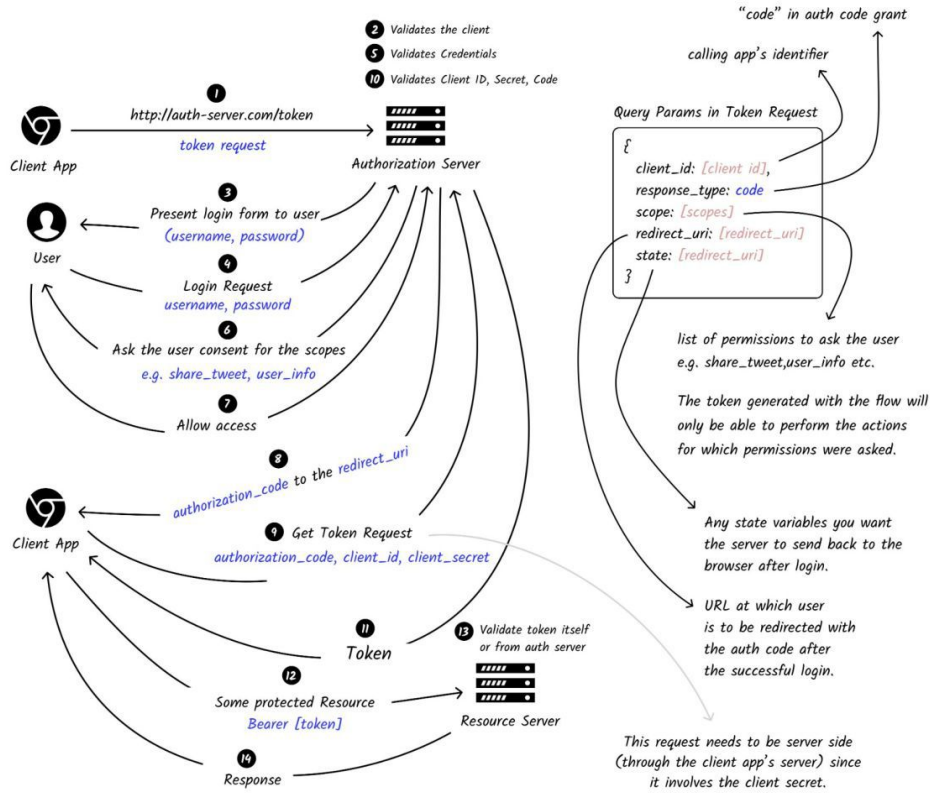
All flows lead to a token which client uses to access protected resource.

1 Authorization Code Grant Flow

Terminology

```
client_id            -> Like a username for your app
client_secret        -> Like a password for your app
authorization server -> server that handles authentication and authorization
resource server      -> server that exposes protected resources
```

} -> They could both be a single server also



2 Implicit Grant Flow

Similar to the "Authorization Code Grant Flow" but **in step 8 there is a token instead of authorization code**.
So no need for 9, 10, and 11 steps. Also in the token request the **response_type parameter is set to `token` instead of `code`**

3 Client Credential Grant Flow

Similar to the "Authorization Code Grant Flow" but there is no user interaction. It starts at **step 9 and immediately gets the token.**
So step 9 would be step 1 in this grant flow. Also in the **request at step 9 there is no `authorization_code` and an additional parameter of the `grant_type` set to `token`**

4 Password Grant Flow

Similar to the "Authorization Code Grant Flow" but user enters the credentials owned by the the authorization server instead of the app.
So **step 4 to 8 are replaced by the get token request (i.e. step 9) and the credentials are directly sent to the authorization server along with an additional parameter of the `grant_type` set to `password`.**

Note

Token response in all grant types is normally accompanied by an expiry date for the token and a refresh token which is used to refresh the token when expired.

Auth Strategies — Session Based Authentication

@kamranahmedse

User is assigned some unique identifier and this identifier is stored on the server in memory. Client sends this session id in all the requests and server uses it to identify the user.

A stateful authentication methodology.

How does it work?

Server keeps track of the loggedin users in memory or in storage.

Stateful vs Stateless Methodologies

Stateful — Authentication session can be revoked
Stateless — Authentication session can't be revoked

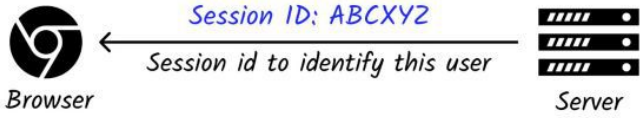
Step 1

Client sends the login request

Step 2

Server validates the credentials, creates a session and stores it in memory assigned to current user and returns back the generated session id.

Some random unique identifier to identify the user



Server stores the generated session id in memory. This could be anywhere e.g. redis, memory, database, filesystem etc.

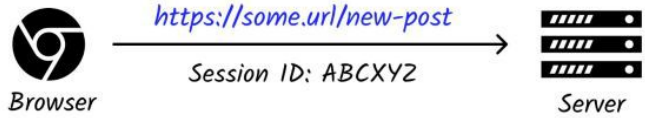
ABCXYZ now refers to user with id **1224**

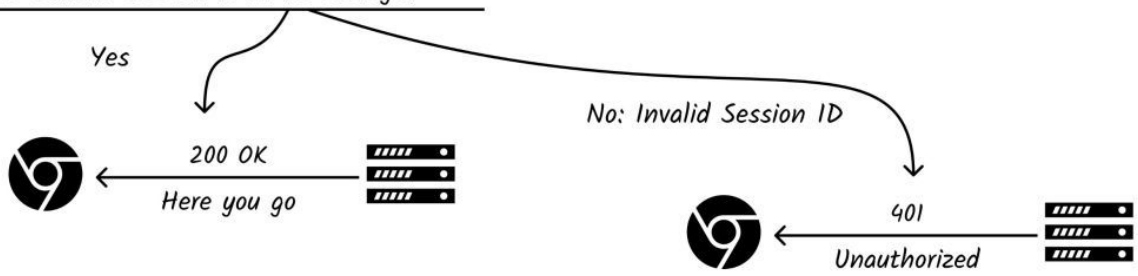
Step 3

Client receives the session id and stores it in a cookie if cookies enabled by client or somewhere else (e.g. in local/session storage) if cookies are not enabled.

Step 4

Client sends next requests with session id. Server checks: if there exists a user with the current session id in its storage?





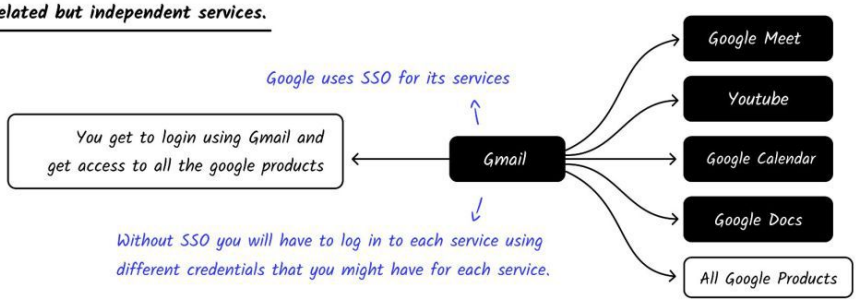
Step 5

When the user logs out, the session is destroyed (cookie removed + session removed from the server) and same Session ID cannot be reused.

Single Sign-On — SSO

@kamranahmedse

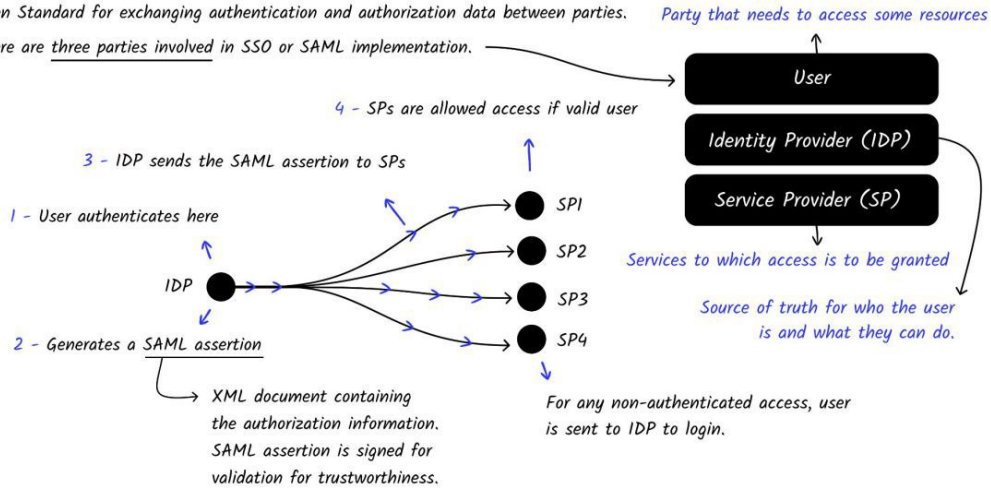
Authentication strategy that allows a user to **login with single username and password to several related but independent services.**



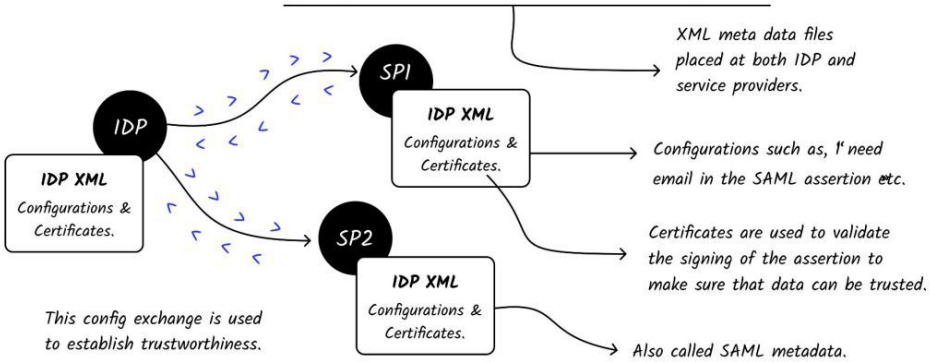
SAML — Security Assertion Markup Language

Open Standard for exchanging authentication and authorization data between parties.

There are three parties involved in SSO or SAML implementation.



SAML assertion format and content is agreed upon between the Service Provider and Identity Provider.



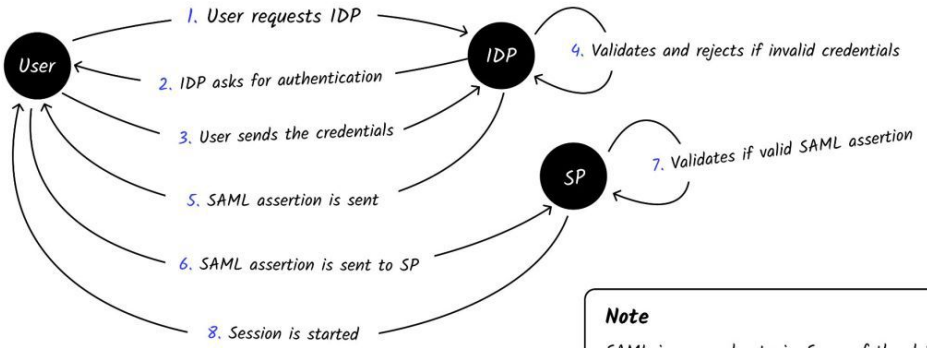
How does it work?

There are two main ways of initiating the authentication flow.

- Identity Provider Initiated Flow
- Service Provider Initiated Flow

Identity Provider Initiated Flow

Client requests the IDP to start the authentication flow



Note

SAML is a complex topic. Some of the details has been intentionally left or simplified to get the point across.

Service Provider Initiated Flow

Client requests the service provider to start the authentication flow.

All the steps are same, except the first request is sent to service provider which directs the user to IDP for authentication.

Token Based Authentication

@kamranahmedse

Unlike the basic auth where username and passwords are sent in each request, in token based authentication a token is sent from client to server in each request.

Some string generated by the server for client to send in each request

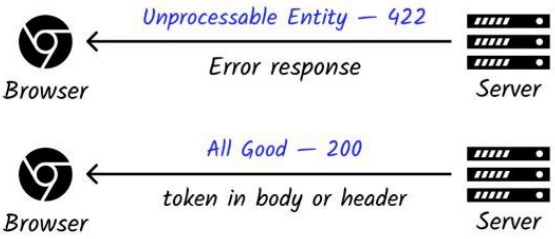
How does it Work?

Client sends the credentials to generate a token

Server validates the credentials

Credentials were invalid

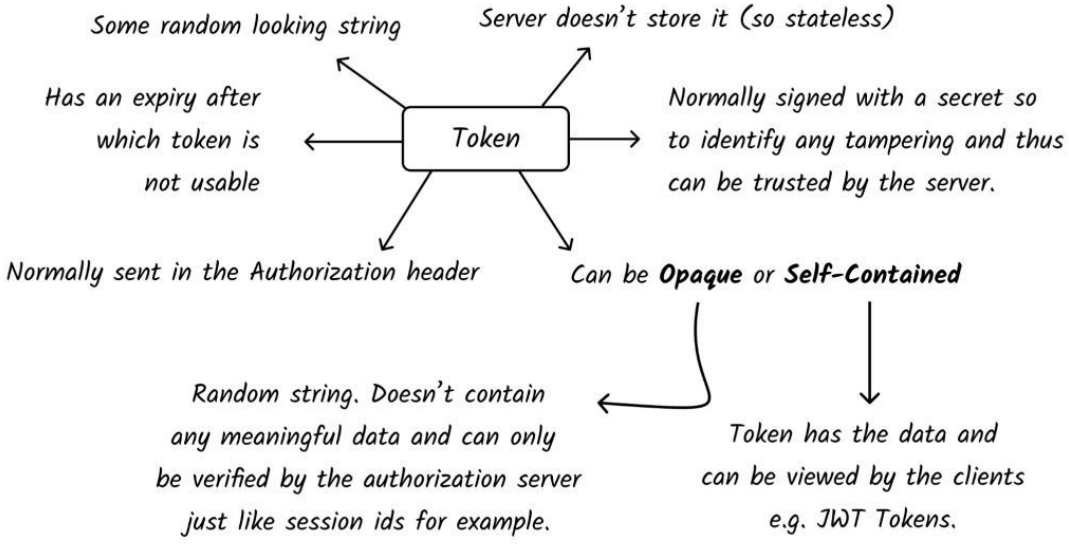
Credentials were valid



Client stores this token in local storage or a cookie and sends in subsequent requests.

Server validates this token and returns response if token is valid. Returns 401 Unauthorized otherwise.

Characteristics of token



Examples of Token Based Authentication Strategies

- SWT (Simple Web Tokens)
- JWT (JSON Web Tokens)
- OAuth (Open Authorization)
- SAML (Security Assertions Markup Language)
- OpenID