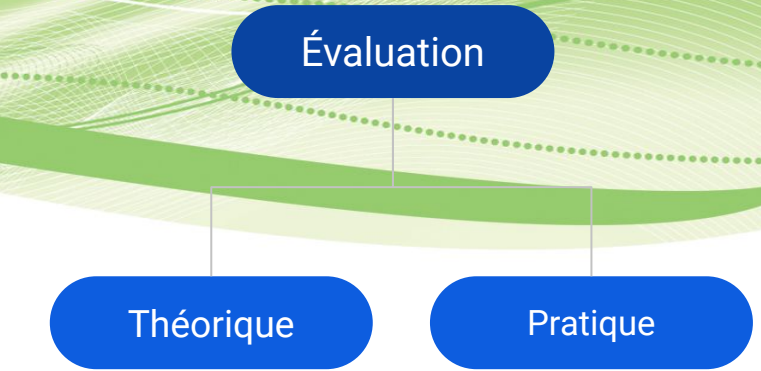
Services Web

S4 - R4.01 - 2022/2023

Plan du cours

- Authentification
- Web sockets et émetteurs d'événements
- Sécurité Web
- Mise en cache (caching)
- Architectures Web (Intro)
- Modèles de conception (Design Patterns)
- Travailler avec une grande quantité de données



BUT_INFO_S4 R4.01 services web

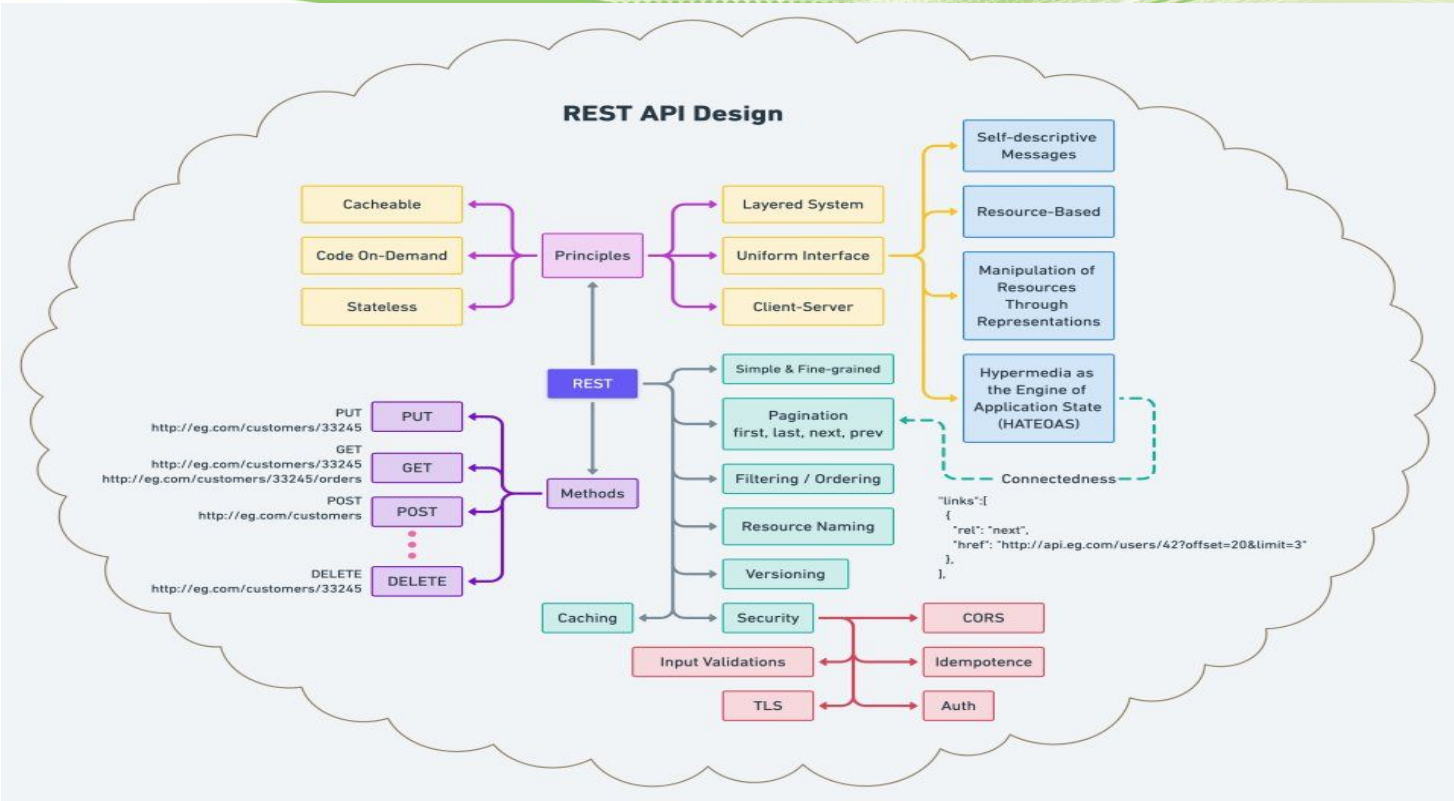
Accueil / Mes cours / IUT Nord Franche-Comté / Informatique / BUT2 / BUT_INFO_S4 R4.01 services web /

pass: ***serviceweb401***

Plan des deux premières séances

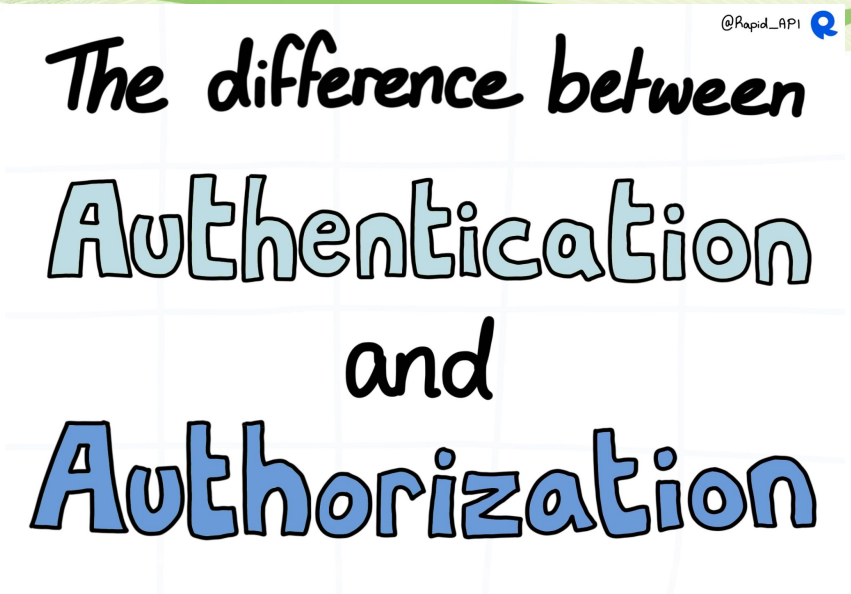
- Introduction à l'authentification
  - Définition de l'authentification
  - Types d'authentification
- Création et gestion des comptes utilisateurs
  - Contrôle d'accès basé sur les rôles
  - Gestion et sécurité des mots de passe
- Techniques d'authentification avancées
  - Connexion OAuth et OpenID
- Comprendre l'authentification multifacteur
- Conclusion et bonnes pratiques

Introduction

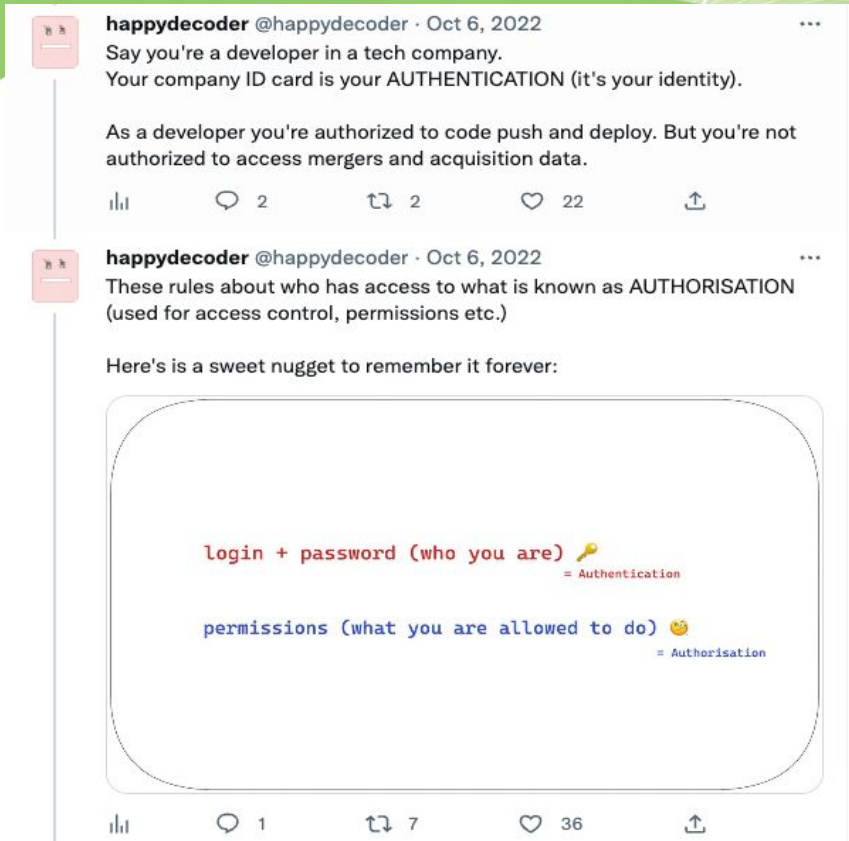


Introduction

L'authentification identifie l'utilisateur et l'autorisation vérifie l'accès de cet utilisateur à une ressource particulière.



Introduction



authentication

+

authorisation

=

user session management

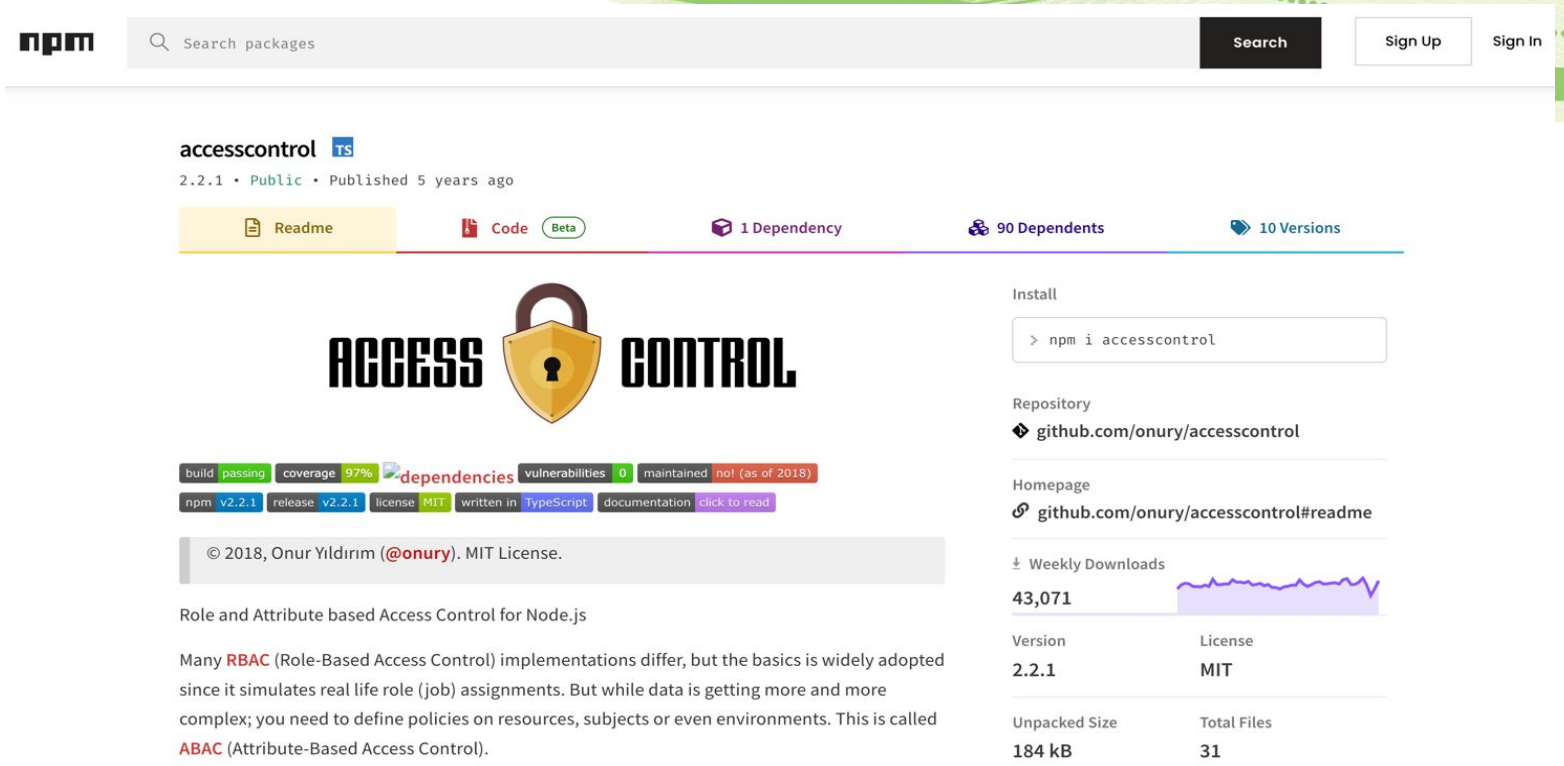
Introduction

- L'authentification est généralement la première étape de tout processus de sécurité Internet et valide qu'un utilisateur est bien celui qu'il prétend être.
- Les méthodes d'authentification standard sont **les mots de passe et les noms d'utilisateur,** la **biométrie** (comme les empreintes digitales) et les **mots de passe à usage unique**.
- En ce qui concerne l'authentification des API, **les API sont incapables d'identifier les utilisateurs individuels**. Par conséquent, la plupart des méthodes **authentifient l'application demandant des données.**
    - Les méthodes d'authentification API standard incluent les **clés API**, **OAuth** et l'**authentification HTTP de base**.

Introduction

- En revanche, l'autorisation contrôle les autorisations d'accès d'un utilisateur à une ressource. **L'autorisation se produit après l'authentification**.
- Une fois qu'un utilisateur est authentifié, l'autorisation détermine les données auxquelles il peut accéder et ce qu'il peut en faire. Cela inclut les limites/restrictions.
- Divers modèles de sécurité implémentent l'autorisation, tels que le contrôle d'accès basé sur les rôles (Role-Based Access Control) et le contrôle d'accès basé sur les attributs (Attribute-Based Access Control).
- Les méthodes d'authentification API standard, telles que **OAuth** et les **clés API**, fournissent également une autorisation API.
    - Par exemple, les clés API délivrées aux clients identifient l'application effectuant les requêtes et déterminent le niveau d'accès de l'utilisateur en fonction du plan d'API choisi. Cela pourrait signifier que leur nombre d'appels d'API est limité.

Introduction



Méthodes d'authentification API

Authentification HTTP

- L'authentification de base HTTP est un mécanisme d'authentification simple intégré au protocole HTTP.
- Il envoie les informations d'identification de l'utilisateur (nom d'utilisateur et mot de passe) en texte clair (encodé en base64) dans les en-têtes de requête.
- Il est généralement utilisé conjointement avec le protocole HTTPS pour fournir une communication sécurisée.

Méthodes d'authentification API

Authentification HTTP

- Avantages :
  - Simple à mettre en œuvre et largement pris en charge par les navigateurs Web et les serveurs
  - Aucune page de connexion ou cookie supplémentaire n'est requis
- Faiblesses:
  - Les mots de passe sont envoyés en texte clair (encodé en base64) et peuvent être facilement interceptés et décodés par un attaquant
  - Il ne fournit aucun mécanisme de déconnexion
  - Il ne fournit aucun mécanisme de gestion de session, de sorte qu'un utilisateur doit saisir ses informations d'identification pour chaque nouvelle session

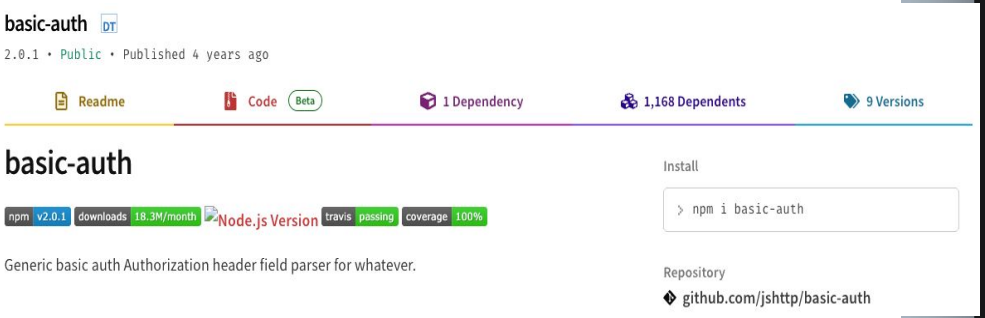
Méthodes d'authentification API

Authentification HTTP

En général, l'authentification HTTP de base ne doit pas être utilisée pour l'authentification sur des connexions non chiffrées ou pour l'authentification qui nécessite un niveau de sécurité élevé. Au lieu de cela, des mécanismes d'authentification plus sécurisés tels que OAuth ou JSON Web Tokens (JWT) doivent être utilisés.

Méthodes d'authentification API

Authentification HTTP




```
const express = require('express');
const basicAuth = require('basic-auth');

const app = express();

const auth = (req, res, next) => {
  const user = basicAuth(req);

  if (!user || !user.name || !user.pass) {
    res.set('WWW-Authenticate', 'Basic realm=Authorization Required');
    return res.sendStatus(401);
  }

  if (user.name === 'admin' && user.pass === 'password') {
    return next();
  } else {
    res.set('WWW-Authenticate', 'Basic realm=Authorization Required');
    return res.sendStatus(401);
  }
};

app.get('/', auth, (req, res) => {
  res.send('Welcome to the protected area!');
});

app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

Si le nom d'utilisateur et le mot de passe ne correspondent pas, le serveur envoie une réponse 401 “non autorisé”, ainsi qu'un en-tête WWW-Authenticate, qui invite le navigateur à afficher une boîte de dialogue de connexion

Méthodes d'authentification API

Authentification basée sur les sessions/cookies

- L'authentification basée sur les sessions/cookies est une méthode d'authentification des utilisateurs sur un site Web en créant un identifiant de session unique et en le stockant dans un cookie sur le navigateur de l'utilisateur.
- Lorsque l'utilisateur fait une requête au site Web, l'ID de session est envoyé sous forme de cookie dans les en-têtes de la requête, permettant au serveur d'identifier l'utilisateur et de déterminer s'il est authentifié.

Méthodes d'authentification API

Authentification basée sur les sessions/cookies

Rapid @Rapid_API · Nov 28, 2022

1 Session Cookies

Session Cookies are small data stored both on the server and the client.

The server often keeps track of sessions in a database or memory.

A browser controls cookies on the client side. They're included with every request.

1 15

Rapid @Rapid_API · Nov 28, 2022

Session cookies are stateful.

Each session has a unique ID that the server uses to identify the current user and all the information related to that user.

2 11

Rapid @Rapid_API · Nov 28, 2022

Cookies allow a domain and its subdomains to exchange information.

Sharing cookie information with another domain is not possible.

2 13

Rapid @Rapid_API · Nov 28, 2022

You can use the "HttpOnly" setting to prevent JavaScript tampering on client sites.

Remember that only HTTPS connections are secure for cookies.

The "Secure" flag can be used for this reason.

It ensures that cookies will be sent only if the connection type is HTTPS.

Méthodes d'authentification API

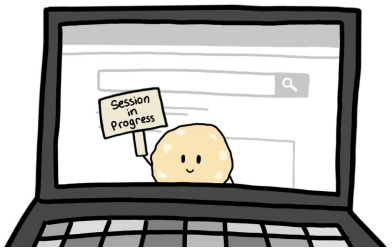
Authentification basée sur les sessions/cookies

@Rapid_API

What is Cookie authentication?

Cookie authentication uses HTTP cookies to authenticate client requests.

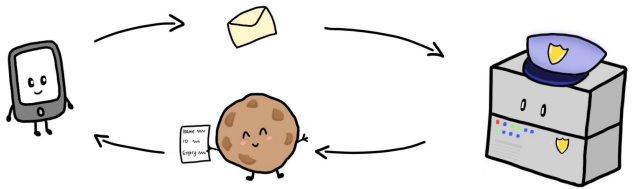
Cookies allow the server to maintain sessions over the stateless HTTP protocol.



@Rapid_API

How it works:

1. The client sends a login request. This contains all the login details the server needs.



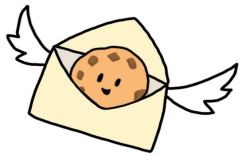
2. The server validates the login if the credentials are correct, and sends back a response with the `Set-Cookie` header. This contains the cookie name, ID, and expiry time, and more.

Méthodes d'authentification API

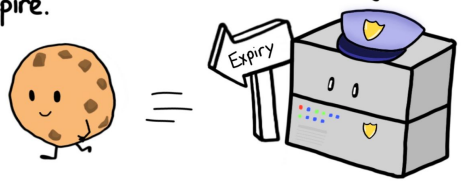
Authentification basée sur les sessions/cookies

@Rapid_API

3 In every following request to the server, the client sends this cookie in the 'Cookie' header.



When the client logs out, the server sends back the 'Set-Cookie' header, causing the cookie to expire. 4



@Rapid_API

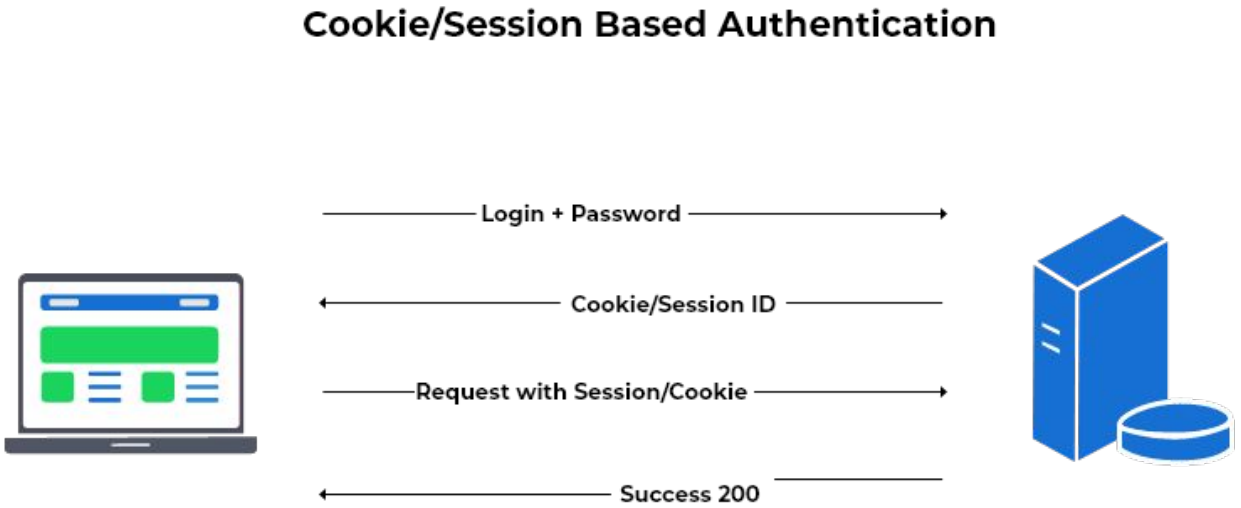
Cookie authentication Limitations

Cookie authentication is vulnerable to Cross-Site Request Forgeries (CSRF), so it's recommended to use them with other security methods, such as CSRF tokens.

Cookies also typically work on one domain only. This can be an issue if, for example, an API service originates from different domains for different platforms.

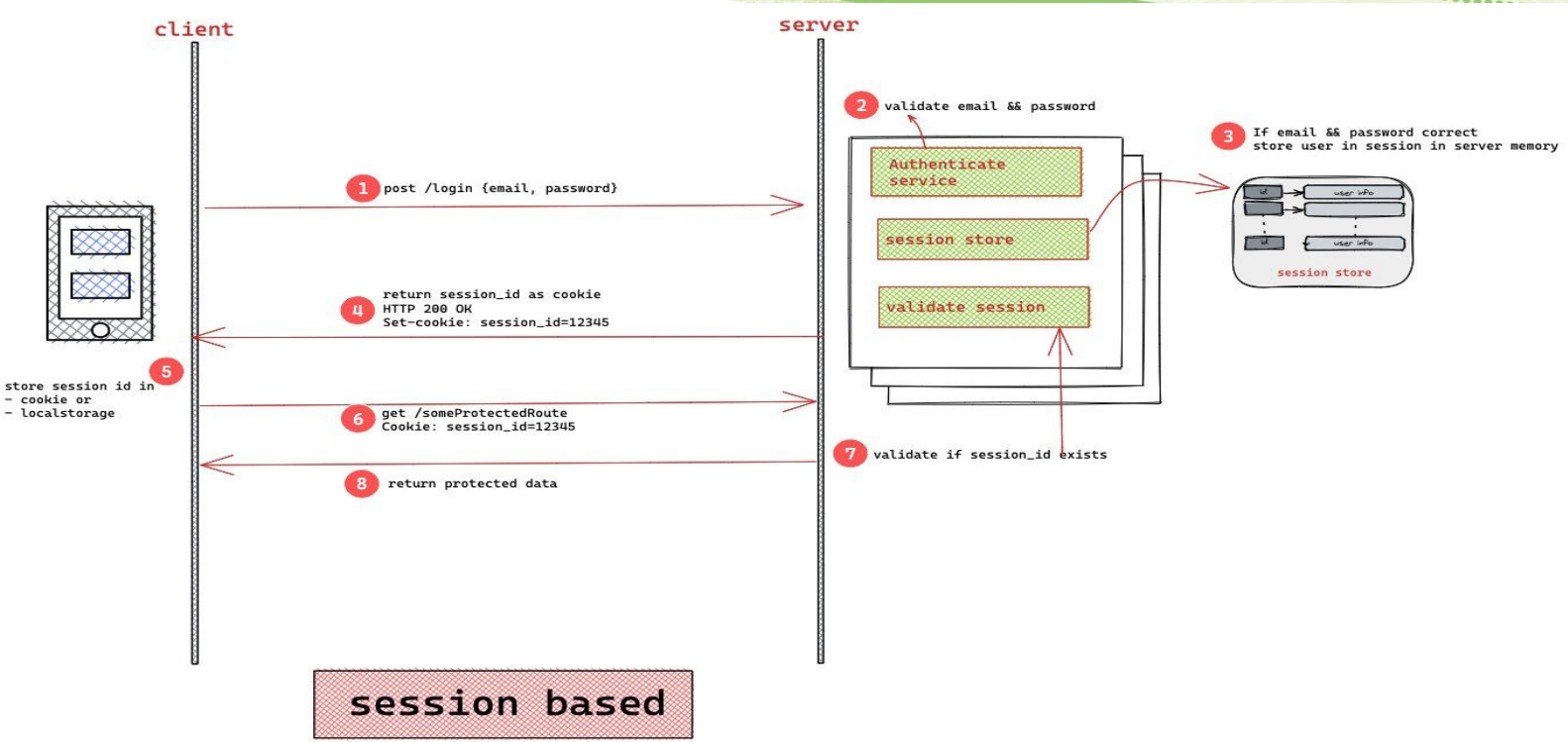
Méthodes d'authentification API

Authentification basée sur les sessions/cookies



Méthodes d'authentification API

Authentification basée sur les sessions/cookies



Méthodes d'authentification API

Authentification basée sur les sessions/cookies

- Avantages :
  - L'ID de session est stocké dans un cookie sur le navigateur de l'utilisateur, de sorte que l'utilisateur n'a pas besoin de saisir ses informations d'identification pour chaque nouvelle session
  - Il permet une gestion de session plus avancée, telle que la déconnexion et la définition de délais d'expiration pour les sessions
  - Il est largement pris en charge par les navigateurs Web et les serveurs
- Faiblesses :
  - Si l'ID de session est volé, un attaquant peut l'utiliser pour accéder au compte de l'utilisateur
  - Les cookies peuvent être supprimés ou bloqués par l'utilisateur, ce qui mettra fin à sa session
  - Les cookies de session sont vulnérables aux attaques de script intersite (XSS), un attaquant pourrait voler le cookie de session et usurper l'identité de l'utilisateur

Méthodes d'authentification API

Authentification basée sur les sessions/cookies

- Il est important de noter que les cookies ne sont pas conçus pour être un moyen sécurisé de stocker des informations sensibles.
  - Pour améliorer la sécurité des cookies de session, il est recommandé de les utiliser conjointement avec **HTTPS**, d'utiliser les flags “secure” et “httpOnly”, et d'utiliser un ID de session aléatoire et unique pour chaque session.
  - Les drapeaux HttpOnly et secure peuvent être utilisés pour rendre les cookies plus sûrs. Lorsqu'un indicateur “secure” est utilisé, le cookie ne sera envoyé que via HTTPS, c'est-à-dire HTTP sur SSL/TLS
- Il convient également de mentionner qu'il existe d'autres façons de gérer les sessions, telles que l'utilisation d'un système d'authentification basé sur des jetons comme JSON Web Tokens (JWT) ou OAuth, qui présentent des avantages et des faiblesses différents selon le cas d'utilisation.

Méthodes d'authentification API

Authentification basée sur les sessions/cookies

```
const express = require('express');
const session = require('express-session');
const cookieParser = require('cookie-parser');

const app = express();

app.use(cookieParser());
app.use(session({
  secret: 'mysecretkey',
  resave: false,
  saveUninitialized: true
}));

app.get('/login', (req, res) => {
  // authenticate the user
  // ...

  req.session.user = { name: 'John Doe' };
  res.redirect('/');
});
```

```
app.get('/', (req, res) => {
  if (req.session.user) {
    res.send(`Welcome ${req.session.user.name}!`);
  } else {
    res.redirect('/login');
  }
});

app.get('/logout', (req, res) => {
  req.session.destroy();
  res.redirect('/');
});

app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

Méthodes d'authentification API

Bearer HTTP Auth

- L'authentification du porteur (Bearer), également connue sous le nom d'authentification basée sur les jetons, est une méthode d'authentification des utilisateurs en passant un jeton d'accès (également appelé « jeton du porteur ») dans les en-têtes de requête.
- Le jeton est généralement généré par le serveur et envoyé au client une fois l'authentification réussie.
- Le client inclut ensuite le jeton dans les requêtes ultérieures adressées au serveur pour prouver son identité.
- Voici un exemple de jeton Bearer dans les en-têtes d'une requête :

```
Authorization: Bearer YOUR_TOKEN_HERE
```

Méthodes d'authentification API

Bearer HTTP Auth

Rapid @Rapid_API · Nov 28, 2022

2 Tokens

In essence, tokens are a collection of letters and numbers.

Tokens have no state.

It implies that no information about the token needs to be stored on the server.

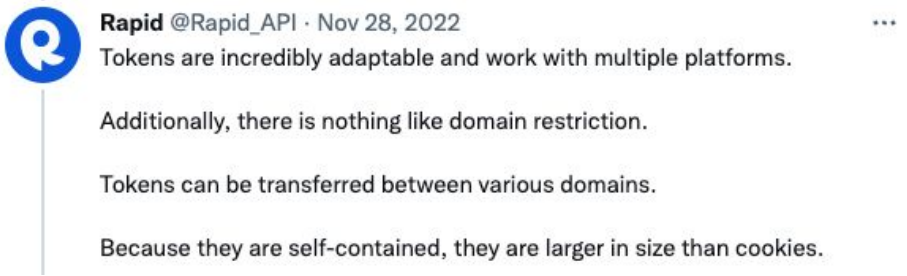
1 16

Rapid @Rapid_API · Nov 28, 2022

The tokens stand alone.

This signifies that the token has all the data needed for server-side verification.

Since no database searches are necessary, they are appropriate for API authentication.



Rapid @Rapid_API · Nov 28, 2022

Tokens are incredibly adaptable and work with multiple platforms.

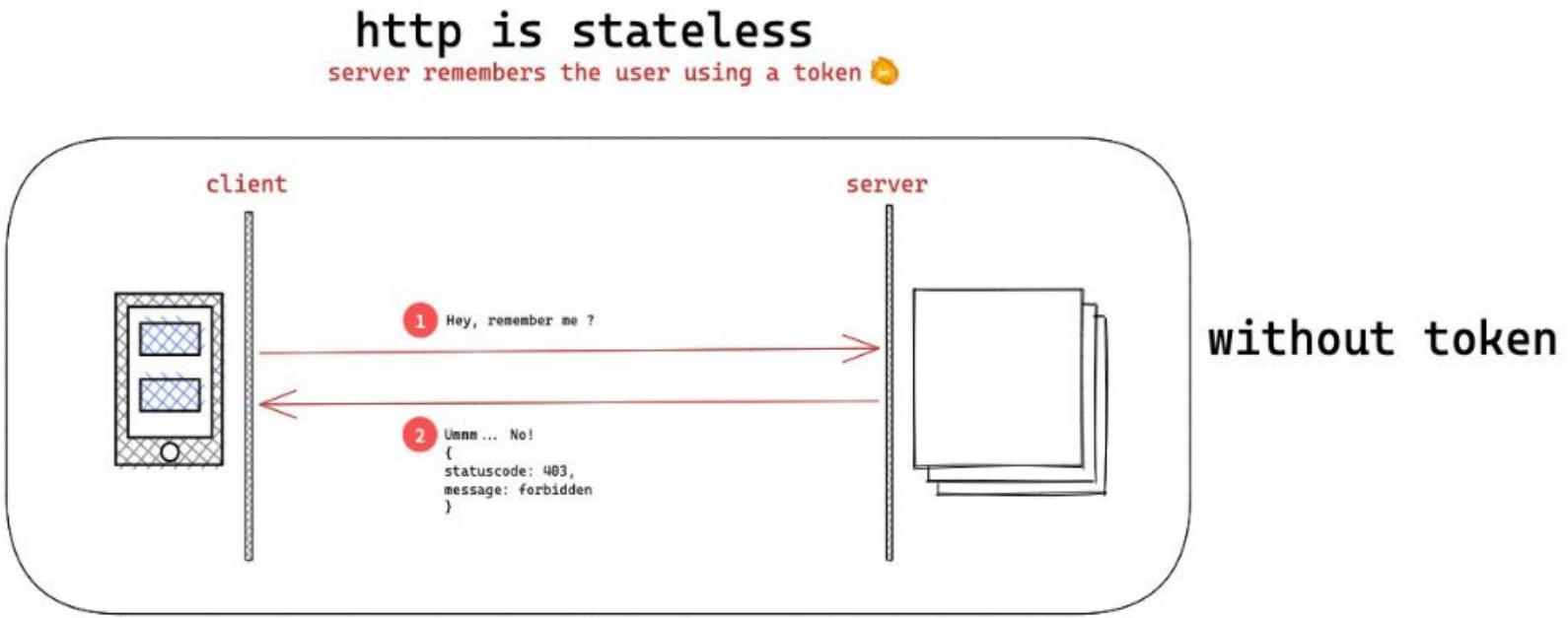
Additionally, there is nothing like domain restriction.

Tokens can be transferred between various domains.

Because they are self-contained, they are larger in size than cookies.

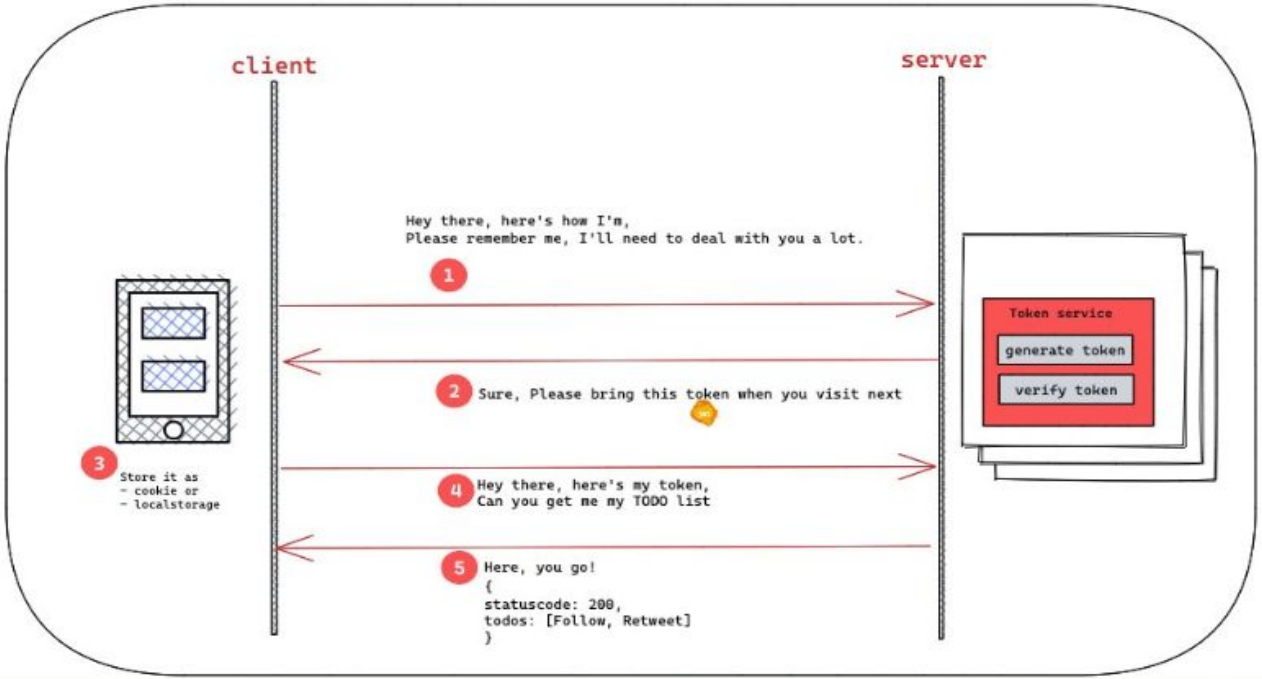
Méthodes d'authentification API

Bearer HTTP Auth



Méthodes d'authentification API

Bearer HTTP Auth



with token

Méthodes d'authentification API

Bearer HTTP Auth

- Avantages :
  - Le jeton est sans état (stateless), ce qui signifie que le serveur n'a pas besoin de maintenir une session ou de stocker des informations sur l'utilisateur
  - Les jetons peuvent être facilement révoqués ou expirés par le serveur
  - Les jetons peuvent être signés cryptographiquement pour empêcher la falsification
- Faiblesses :
  - Les jetons peuvent être volés, il est donc important d'utiliser HTTPS pour chiffrer la communication
  - Les jetons doivent être stockés en toute sécurité côté client, car s'ils sont compromis, ils donneront à l'attaquant l'accès aux ressources de l'utilisateur
  - Les jetons doivent avoir une durée de vie limitée, sinon un attaquant pourrait utiliser un jeton volé indéfiniment

Méthodes d'authentification API

Bearer HTTP Auth

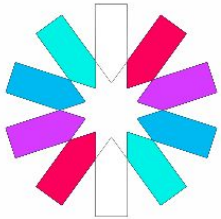
- Les jetons porteurs (Bearer Tokens) sont couramment utilisés dans les mécanismes d'authentification basés sur OAuth et JSON Web Token (JWT).
- Ces mécanismes fournissent un moyen sécurisé de transmettre le jeton du client au serveur et de serveur à serveur
- Il est important de noter que les jetons Bearer peuvent être envoyés dans les en-têtes, le corps ou le paramètre de requête, mais il est recommandé d'utiliser les en-têtes pour des raisons de sécurité

Méthodes d'authentification API

JSON WEB TOKEN (JWT)

JSON Web Token (JWT) est une norme ouverte (RFC 7519) qui définit un moyen compact et autonome pour transmettre en toute sécurité des informations entre les parties en tant qu'objet JSON.

Ces informations peuvent être vérifiées et approuvées car elles sont signées numériquement. Les JWT peuvent être signés à l'aide d'un secret (avec l'algorithme HMAC) ou d'une paire de clés publique/privée utilisant RSA ou ECDSA.



JWT

Méthodes d'authentification API

JSON WEB TOKEN (JWT)

- Dans sa forme compacte, les jetons Web JSON se composent de trois parties séparées par des points (.), qui sont :
  - Entête
  - Payload
  - Signature
- Par conséquent, un JWT ressemble généralement à ce qui suit :

```
xxxxx.yyyyy.zzzzz
```

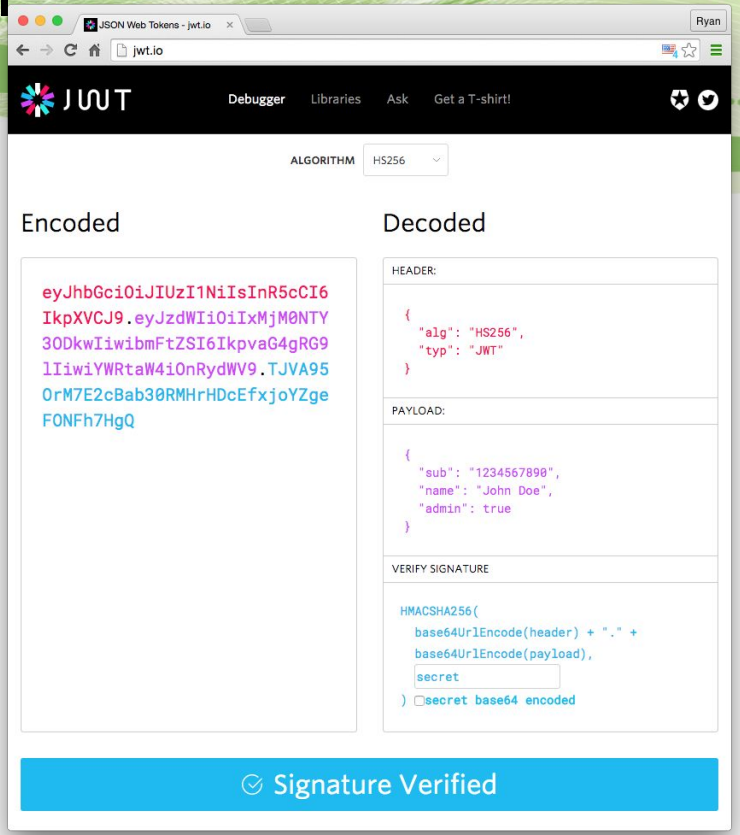
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

Méthodes d'authentification API

JSON WEB TOKEN (JWT)

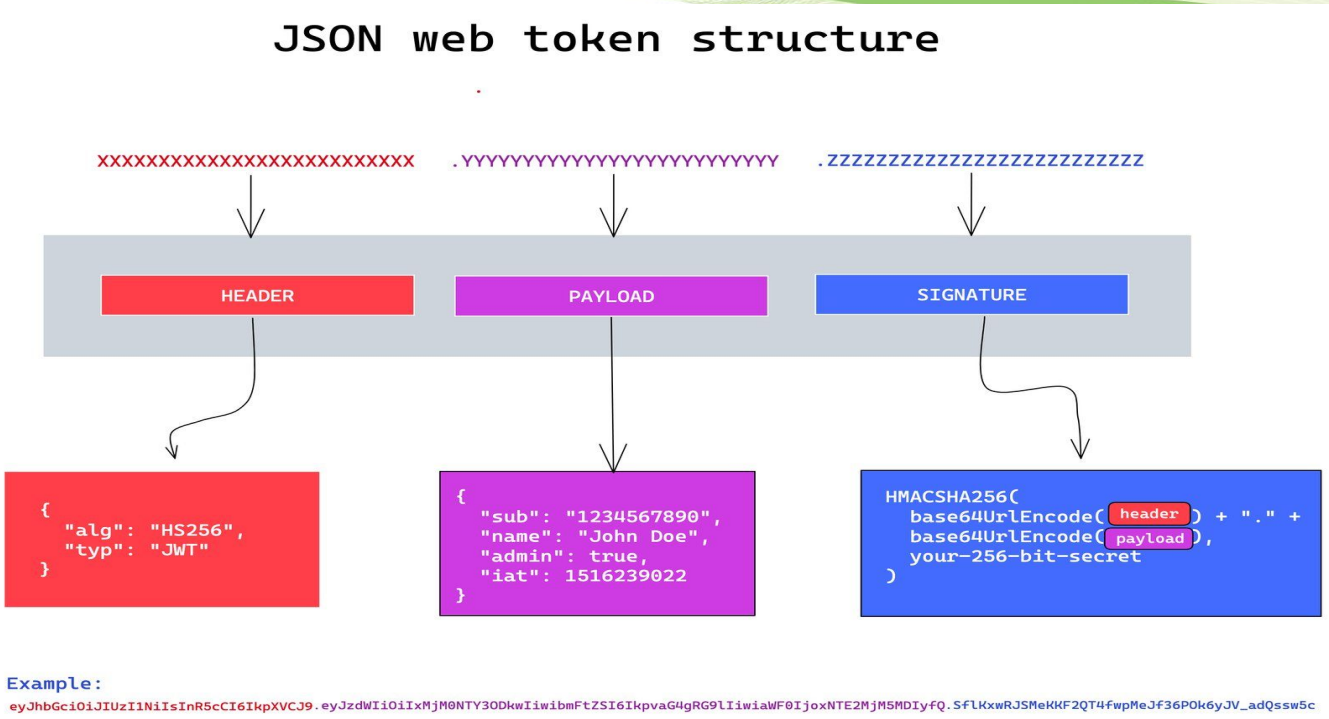
Pour créer la partie signature, vous devez prendre l'en-tête encodé, le payload encodé, un secret, l'algorithme spécifié dans l'en-tête et le signer.

La signature est utilisée pour vérifier que le message n'a pas été modifié en cours de route et, dans le cas de jetons signés avec une clé privée, elle peut également vérifier que l'expéditeur du JWT est celui qu'il prétend être.



Méthodes d'authentification API

JSON WEB TOKEN (JWT)



Méthodes d'authentification API

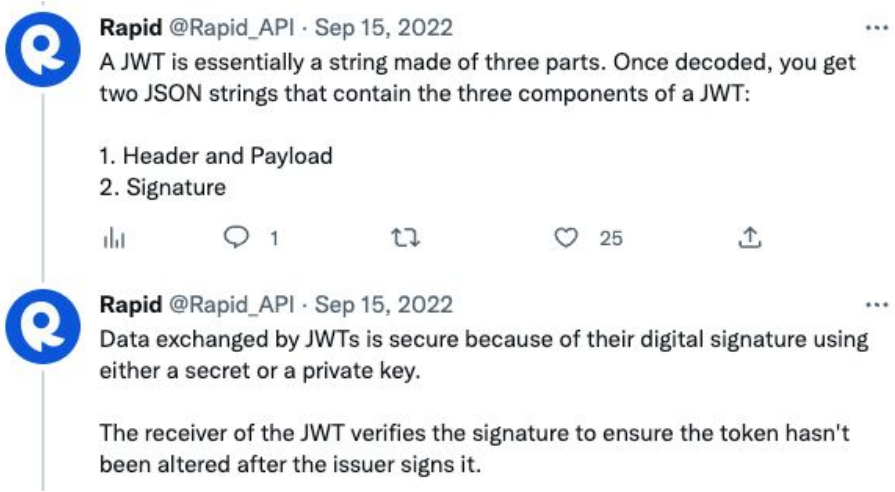
JSON WEB TOKEN (JWT) - Quel problème résout-il ?

Bien que l'objectif principal des JWT soit de transférer des réclamations entre deux parties, l'aspect le plus important est sans doute l'effort de normalisation sous la forme d'un format de conteneur simple, éventuellement validé et/ou crypté. Certaines de ces applications incluent :

- Authentification
- Autorisation
- Identité fédérée (Federated identity)
- Sessions côté client (sessions "sans état")

Méthodes d'authentification API

JSON WEB TOKEN (JWT)



Rapid @Rapid_API · Sep 15, 2022

A JWT is essentially a string made of three parts. Once decoded, you get two JSON strings that contain the three components of a JWT:

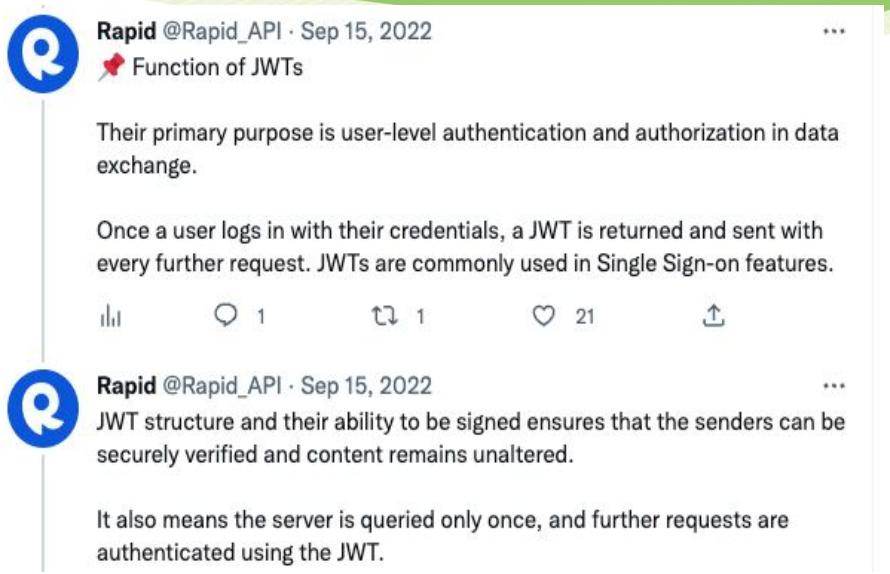
1. Header and Payload
2. Signature

1 25

Rapid @Rapid_API · Sep 15, 2022

Data exchanged by JWTs is secure because of their digital signature using either a secret or a private key.

The receiver of the JWT verifies the signature to ensure the token hasn't been altered after the issuer signs it.



Rapid @Rapid_API · Sep 15, 2022

📌 Function of JWTs

Their primary purpose is user-level authentication and authorization in data exchange.

Once a user logs in with their credentials, a JWT is returned and sent with every further request. JWTs are commonly used in Single Sign-on features.

1 1 21

Rapid @Rapid_API · Sep 15, 2022

JWT structure and their ability to be signed ensures that the senders can be securely verified and content remains unaltered.

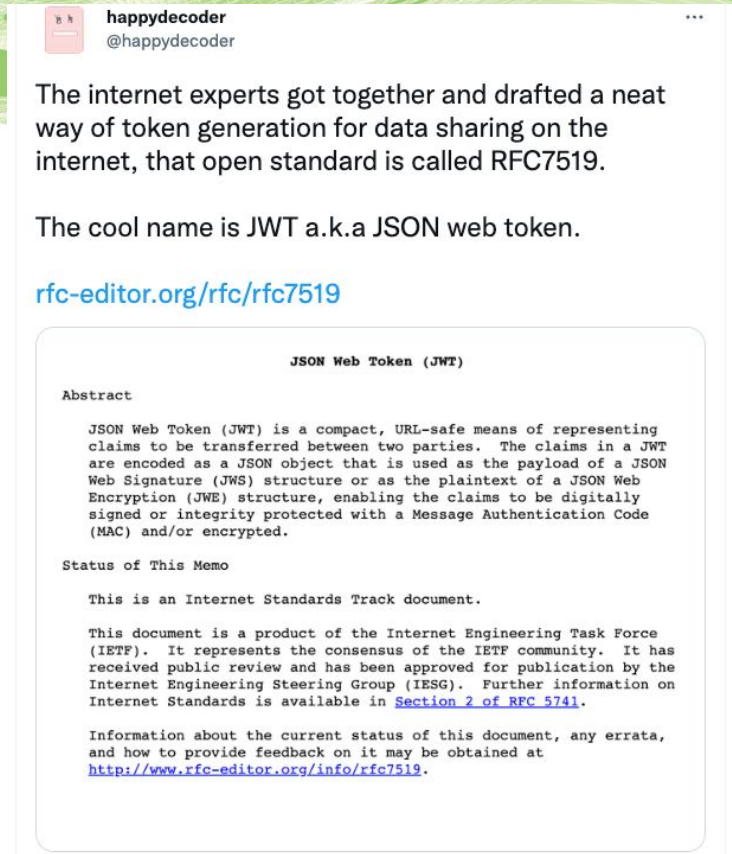
It also means the server is queried only once, and further requests are authenticated using the JWT.

Méthodes d'authentification API

JSON WEB TOKEN (JWT)

Bien que JWT soit couramment utilisé pour gérer les autorisations, l'idée derrière JWT est de définir un moyen standard entre deux parties pour communiquer des informations en toute sécurité.

La norme RFC7519 dicte simplement
- comment les données JSON doivent être structurées
- les moyens de le crypter
- façons de le signer



Méthodes d'authentification API

JSON WEB TOKEN (JWT)

- La phrase "**Cela signifie également que le serveur n'est interrogé qu'une seule fois**" fait référence au fait que lors de l'utilisation de JWT (JSON Web Tokens), **le serveur n'a pas besoin de maintenir une session ou de suivre l'état de connexion de l'utilisateur** à chaque demande ultérieure.
- Normalement, lorsqu'un utilisateur se connecte, le serveur vérifie ses informations d'identification, crée une session et stocke les informations de session côté serveur. Pour chaque requête ultérieure, le serveur devra rechercher les informations de session pour vérifier si l'utilisateur est toujours connecté. Cela nécessite que le serveur interroge le jeu de données de session à chaque requête.

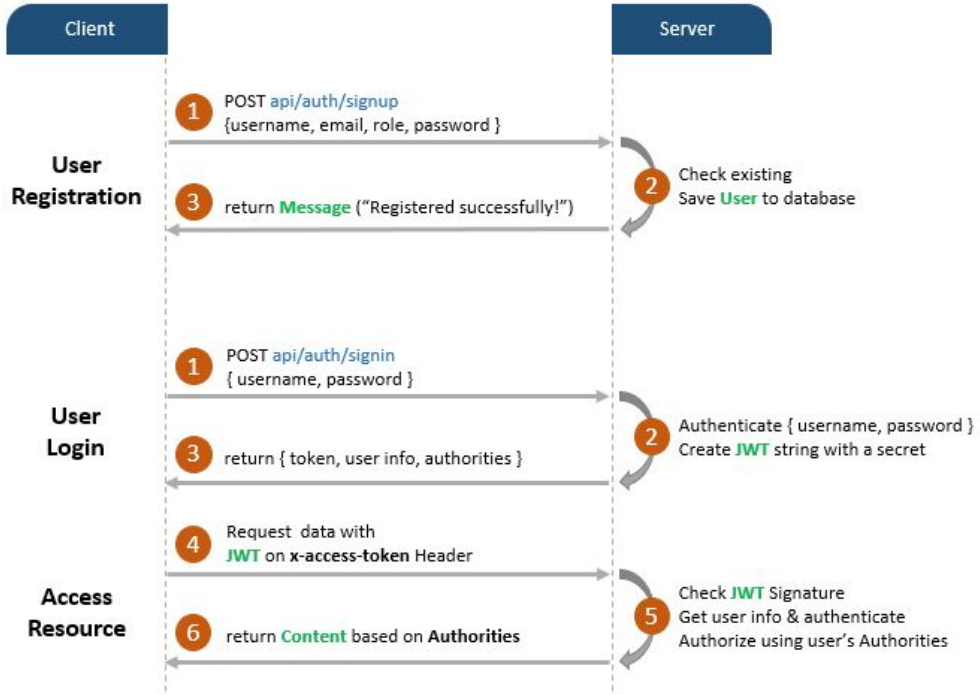
Méthodes d'authentification API

JSON WEB TOKEN (JWT)

- Avec JWT, cependant, **le serveur émettra un jeton au client une fois l'authentification réussie**. Ce jeton contient les informations de l'utilisateur et est signé cryptographiquement par le serveur. Le client peut ensuite inclure ce jeton dans chaque requête ultérieure, et le serveur peut utiliser le jeton pour vérifier l'identité de l'utilisateur **sans avoir à interroger le jeu de données de session** côté serveur. Comme le jeton est autonome, il contient toutes les informations nécessaires pour vérifier l'identité de l'utilisateur, c'est pourquoi le serveur n'est interrogé qu'une seule fois.
- Il s'agit d'un aspect important de JWT, car il permet une authentification sans état, qui peut être plus efficace et évolutive que de maintenir une session côté serveur pour chaque utilisateur.

Méthodes d'authentification API

JSON WEB TOKEN (JWT)



Méthodes d'authentification API

JSON WEB TOKEN (JWT)

- Avantages :
  - Il est simple à mettre en œuvre, car il est basé sur JSON, un format d'échange de données largement utilisé
  - Il est sans état, ce qui signifie que le serveur n'a pas besoin de maintenir une session pour le client, ce qui peut être utile pour l'évolutivité
  - Il est autonome, ce qui signifie que le jeton contient toutes les informations nécessaires pour authentifier l'utilisateur, éliminant ainsi le besoin d'interroger la base de données plusieurs fois
  - Il est largement pris en charge dans différentes plates-formes
- Faiblesses :
  - La principale faiblesse de l'authentification JWT est qu'elle est vulnérable aux attaques par relecture (replay attacks) si le jeton est intercepté par un attaquant
  - Pour atténuer cela, les jetons doivent avoir un délai d'expiration court et être transmis via une connexion sécurisée (HTTPS)
  - De plus, les jetons doivent être stockés en toute sécurité côté client et non exposés dans la console développeur du navigateur

Méthodes d'authentification API

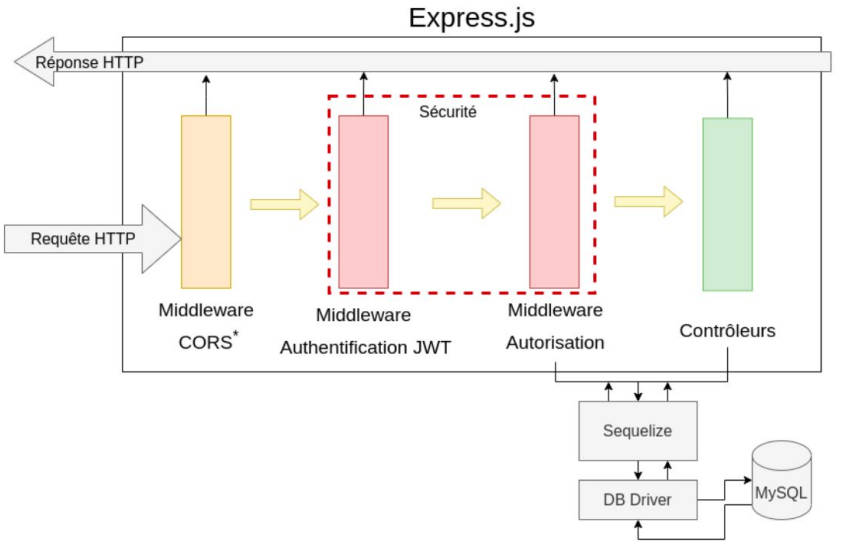
JSON WEB TOKEN (JWT)

```
const express = require('express');
const jwt = require('jsonwebtoken');

const app = express();

// Secret key for signing the JWT
const SECRET_KEY = 'mysecretkey';

// Middleware function to check for a valid JWT
function checkJWT(req, res, next) {
  // Get the JWT from the request headers
  const token = req.headers.authorization;

  // Try to verify the JWT and get the user's data
  try {
    const userData = jwt.verify(token, SECRET_KEY);

    // Attach the user data to the request object
    req.user = userData;

    // Call the next middleware function
    next();
  } catch (err) {
    // If the JWT is invalid, return an error
    res.status(401).json({ message: 'Invalid JWT' });
  }
}
```

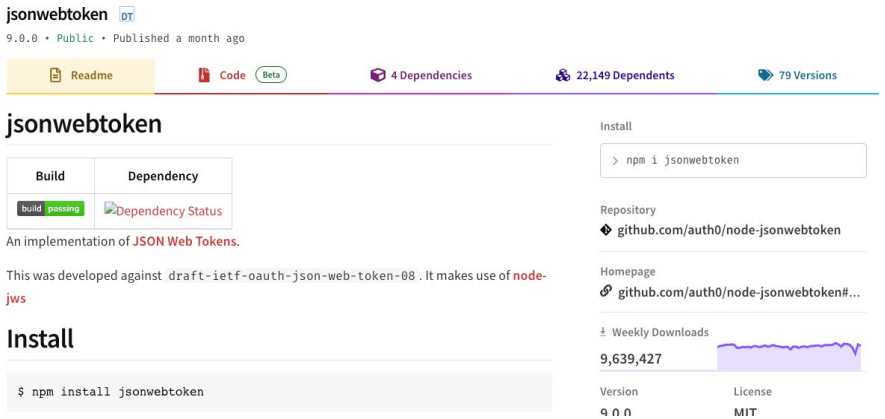
```
// Route to login and get a JWT
app.post('/login', (req, res) => {
  // Get the user's data from the request body
  const { username, password } = req.body;

  // Check if the user's data is valid
  if (username === 'user' && password === 'password') {
    // Create a JWT with the user's data
    const token = jwt.sign({ username }, SECRET_KEY);

    // Send the JWT to the client
    res.json({ token });
  } else {
    // If the user's data is invalid, return an error
    res.status(401).json({ message: 'Invalid username or password' });

});
// Roue that requires a valid JWT to access
app.get('/secret', checkJWT, (req, res) => {
  // Send a message to the client
  res.json({ message: 'You have access to the secret!' });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Méthodes d'authentification API

JSON WEB TOKEN (JWT)



*Cross-origin resource sharing (CORS) est un mécanisme qui permet d'accéder à des ressources restreintes sur une page Web à partir d'un autre domaine en dehors du domaine à partir duquel la première ressource a été servie



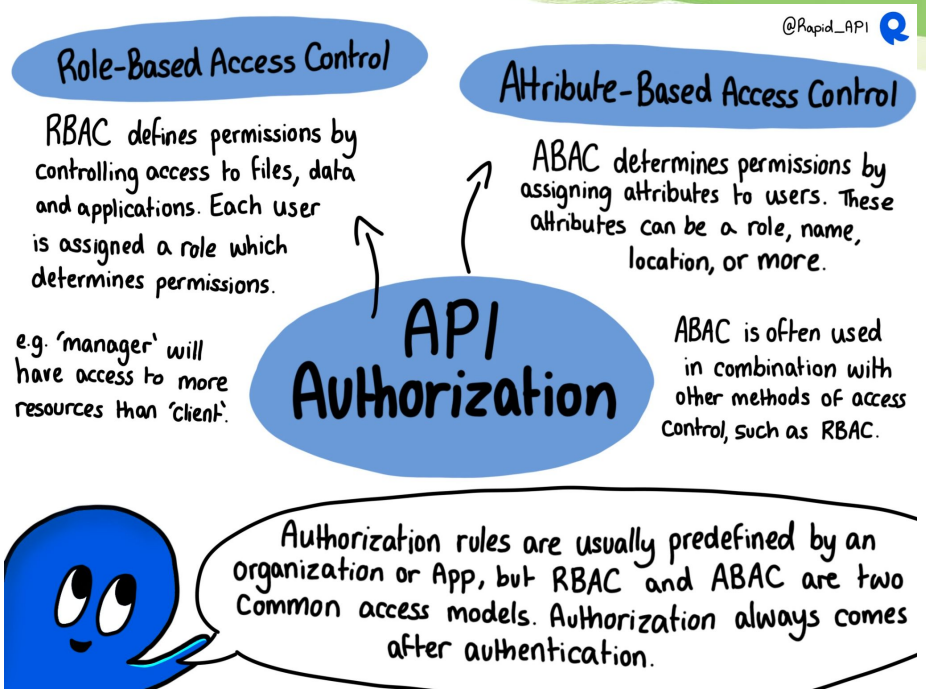
Méthodes d'autorisation

Comment gérer les rôles et les autorisations dans une application NodeJS

- Utilisez une bibliothèque comme **Passport.js** qui fournit un moyen simple de gérer l'authentification et l'autorisation en utilisant différentes stratégies. Cette bibliothèque propose également un middleware qui vérifie les rôles et les autorisations des utilisateurs en fonction de la stratégie utilisée.
- Créez une fonction middleware personnalisée qui vérifie le rôle et l'autorisation de l'utilisateur avant d'autoriser l'accès à des itinéraires ou à des ressources spécifiques. Les informations de rôle et d'autorisation peuvent être stockées dans la session de l'utilisateur ou dans un jeton Web JSON.
- Utilisez une bibliothèque RBAC (Role-Based Access Control) telle que **node-rbac** ou **express-rbac**, qui fournit un ensemble de rôles et d'autorisations prédéfinis pouvant être attribués aux utilisateurs.
- Utilisez un framework comme **Nest.js** qui fournit un mécanisme intégré pour gérer les rôles et les autorisations. Il utilise des décorateurs pour définir les rôles et les autorisations et il est facile à utiliser.
- Utilisez un service tiers comme **Auth0** ou **Firebase Authentication** qui fournissent la fonctionnalité de gestion des utilisateurs, des rôles et des autorisations prêtes à l'emploi.

Méthodes d'autorisation

Comment gérer les rôles et les autorisations dans une application NodeJS



Méthodes d'autorisation

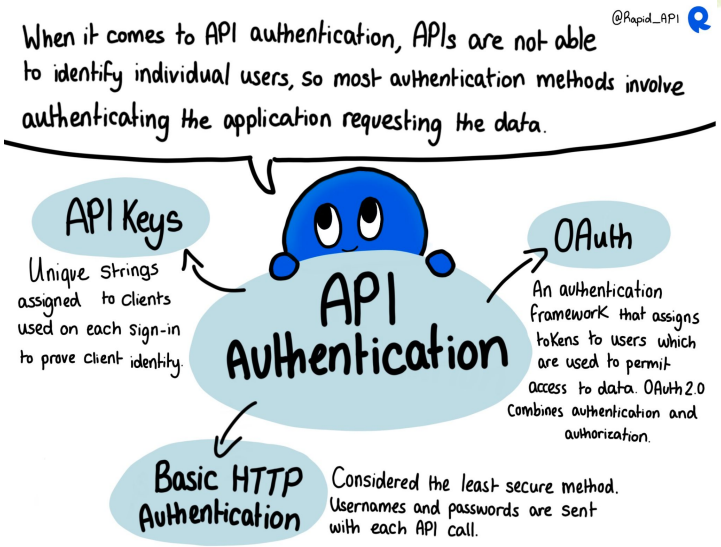
Comment gérer les rôles et les autorisations dans une application NodeJS

Quelle que soit la méthode que vous choisissez, il est important de vous assurer que les rôles et les informations d'autorisation sont stockés en toute sécurité et qu'ils sont correctement vérifiés et appliqués aux points appropriés de l'application.

De plus, il est important d'avoir un processus d'ajout, de mise à jour et de suppression des rôles et des autorisations des utilisateurs.

Authentification basée sur les clés API (API Keys)

- L'authentification par clés API est une méthode d'authentification dans laquelle une clé unique est générée pour chaque utilisateur ou développeur qui souhaite accéder à l'API. La clé est ensuite transmise à chaque requête à l'API afin d'identifier l'utilisateur ou le développeur à l'origine de la requête.
- Les avantages de l'authentification par clés API incluent :
  - Elle est facile à mettre en œuvre et ne nécessite pas l'utilisation d'algorithmes cryptographiques complexes
  - Elle est sans état, ce qui signifie que le serveur n'a pas besoin de maintenir une session pour le client, ce qui peut être utile pour l'évolutivité
  - Elle peut être facilement intégrée dans les systèmes existants et ne nécessite pas l'utilisation de cookies ou de sessions
  - C'est un moyen simple de limiter les requêtes à une API



Authentification basée sur les clés API (API Keys)

- Cet exemple utilise Express.js pour créer un serveur simple qui écoute les requêtes et utilise une fonction middleware appelée **checkAPIKey** pour vérifier si une clé API valide est présente dans les en-têtes de requête. Si la clé API est valide, la fonction middleware joint les données de l'utilisateur à l'objet de requête et appelle la fonction middleware suivante. Si la clé API n'est pas valide, la fonction middleware renvoie une erreur au client.
- Dans cet exemple, les clés API sont stockées dans un objet Map, mais dans une application réelle, elles doivent être stockées dans un stockage sécurisé et fiable, comme une base de données.
- Il est important de noter qu'il s'agit d'un exemple simple et qu'il y a d'autres éléments à prendre en compte lors de la mise en œuvre de l'authentification par clés d'API dans une application réelle, telles que l'expiration des clés, la rotation des clés et la surveillance de l'utilisation des clés.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

// Database of API keys
const apiKeys = new Map();
apiKeys.set('key1', { userId: 'user1', role: 'admin' });
apiKeys.set('key2', { userId: 'user2', role: 'user' });

// Middleware function to check for a valid API key
function checkAPIKey(req, res, next) {
  // Get the API key from the request headers
  const apiKey = req.headers['x-api-key'];

  // Check if the API key is valid
  if (apiKeys.has(apiKey)) {
    // Attach the user data to the request object
    req.user = apiKeys.get(apiKey);

    // Call the next middleware function
    next();
  } else {
    // If the API key is invalid, return an error
    res.status(401).json({ message: 'Invalid API key' });
  }
}

app.use(bodyParser.json());

// Route that requires a valid API key to access
app.get('/secret', checkAPIKey, (req, res) => {
  // Send a message to the client
  res.json({ message: 'You have access to the secret!' });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Authentification basée sur les clés API (API Keys)

- La principale faiblesse de l'authentification par clés API est qu'elle n'est pas aussi sécurisée que d'autres méthodes telles que **OAuth2** ou **JWT**. En effet, la clé peut être facilement interceptée si elle est envoyée en clair ou si elle est stockée de manière non sécurisée côté client. De plus, si la clé est compromise, un attaquant pourrait l'utiliser pour accéder à l'API au nom de l'utilisateur ou du développeur.
- Il est important de mentionner que les clés API sont généralement utilisées en combinaison avec d'autres méthodes d'authentification et d'autorisation, telles que **JWT**, **OAuth2**, etc. Il s'agit de fournir une couche de sécurité supplémentaire, d'empêcher les abus, d'identifier et de suivre l'utilisation de l'API.
- Pour atténuer les faiblesses de l'authentification des clés API, il est recommandé d'utiliser Secure HTTP (HTTPS) pour chiffrer la clé en transit, et d'utiliser des clés de courte durée qui expirent ou peuvent être facilement révoquées. Il est également important de mettre en œuvre un processus de surveillance, de rotation et de révocation des clés, et d'utiliser différentes clés pour différents environnements (par exemple, développement, production, etc).

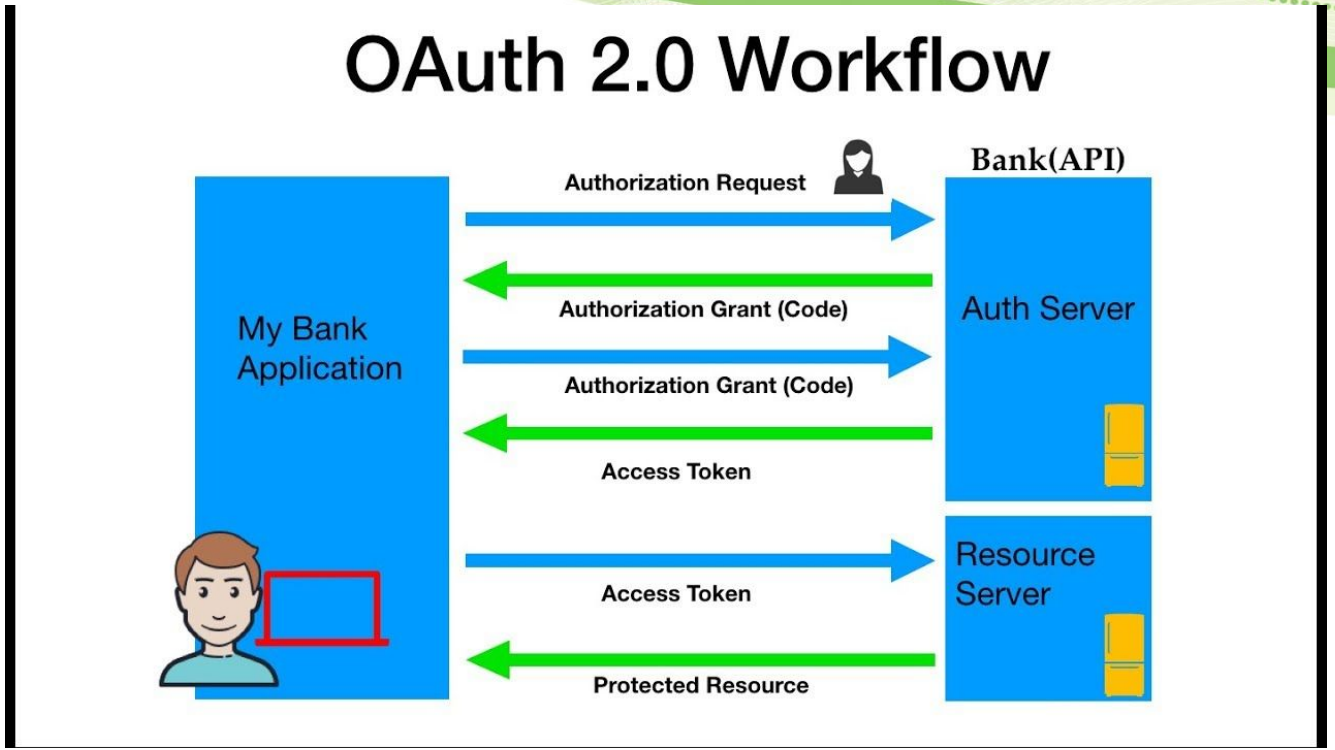
OAuth (Open Authorization)

- OAuth (Open Authorization) est une norme ouverte d'autorisation qui permet aux utilisateurs de partager leurs ressources privées (par exemple, des photos, des vidéos, des listes de contacts) stockées sur un site avec un autre site sans avoir à donner leurs informations d'identification, généralement un nom d'utilisateur et un mot de passe.
- **OAuth1** et **OAuth2** sont deux versions du protocole **OAuth**.

OAuth (Open Authorization)

- **OAuth1** était la première version **d'OAuth**, elle a été créée en 2007 et elle est principalement utilisée pour les API des systèmes hérités (legacy systems). Elle nécessite l'utilisation d'un algorithme de signature numérique (DSA) et utilise un processus complexe de génération et de vérification de signature. Elle nécessite également l'utilisation d'un jeton de demande et d'un jeton d'accès, qui sont échangés lors du processus d'autorisation.
- **OAuth2**, sorti en 2012, est la version actuelle du protocole **OAuth**. Elle est plus simple que **OAuth1** et convient mieux aux applications Web et mobiles. **OAuth2** a introduit l'utilisation de l'authentification basée sur les jetons, qui permet l'utilisation de jetons d'accès au lieu de signatures numériques. Elle a également introduit un flux d'autorisation simplifié, plus facile à mettre en œuvre et à comprendre. Elle prend également en charge plusieurs types de subventions (**GRANT**) différents, tels que :
  - Authorization Code Grant
  - Implicit Grant
  - Resource Owner Password Credentials Grant
  - Client Credentials Grant

OAuth (Open Authorization)

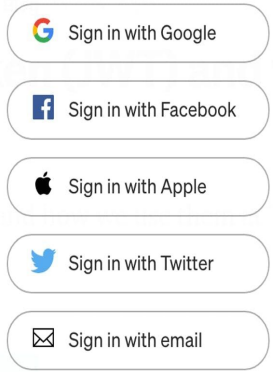


OAuth (Open Authorization)

- **OAuth0** n'est pas une version du protocole **OAuth**, c'est une **implémentation du protocole OAuth2**, qui est fourni par **Auth0** (https://auth0.com/) en tant que service, qui permet d'authentifier et d'autoriser les utilisateurs dans une application Web ou mobile. **OAuth0** est conçu pour gérer les complexités de l'authentification et de l'autorisation et fournit des fonctionnalités supplémentaires telles que la gestion des utilisateurs, l'authentification multifacteur, etc.
- OAuth0 est un service basé sur le cloud, qui permet d'authentifier et d'autoriser les utilisateurs dans une application Web ou mobile. Il peut également fonctionner avec des fournisseurs d'identité externes tels que Google, Facebook, etc.
- En résumé, **OAuth** est un protocole qui permet aux utilisateurs de partager leurs ressources privées avec un autre site sans donner leurs informations d'identification. **OAuth1** est la première version **d'OAuth**, elle est complexe et principalement utilisée pour les systèmes hérités. **OAuth2** est la version actuelle du protocole OAuth, il est plus simple et mieux adapté aux applications Web et mobiles.

"Social Sign-in" authentication

- Lorsque vous vous connectez à un site Web à l'aide de votre compte Google ou Facebook, cela s'appelle l'authentification "**Social Login**" ou "**Social Sign-in**". Il s'agit d'une forme d'authentification qui permet aux utilisateurs de se connecter à un site Web ou à une application à l'aide de leur compte de réseau social existant plutôt que de créer un nouveau compte spécifiquement pour ce site Web.
- Ce type d'authentification est basé sur le protocole **OAuth2**, qui est un standard ouvert d'autorisation.



"Social Sign-in" authentication

- Lorsque vous cliquez sur le bouton **"Connexion avec Google"** ou **"Connexion avec Facebook"**, le site Web vous redirige vers le site de réseau social correspondant, où vous êtes invité à saisir vos identifiants de connexion. Une fois connecté, le site de réseau social vous redirigera vers le site Web, avec un jeton d'accès. Le site Web peut ensuite utiliser ce jeton pour authentifier l'utilisateur et accéder à ses informations de profil de base, telles que son nom, son adresse e-mail et sa photo de profil.
- L'avantage de la connexion sociale est qu'elle rend le processus d'inscription plus rapide et plus facile pour l'utilisateur, car il n'a pas besoin de créer un nouveau compte ou de se souvenir d'un autre ensemble d'identifiants de connexion.
- De plus, cela permet également au site Web d'accéder aux données des médias sociaux de l'utilisateur, qui peuvent être utilisées pour personnaliser l'expérience de l'utilisateur et fournir des fonctionnalités sociales.