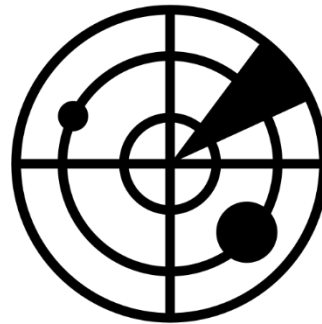


Sécurité :

Authentification Web

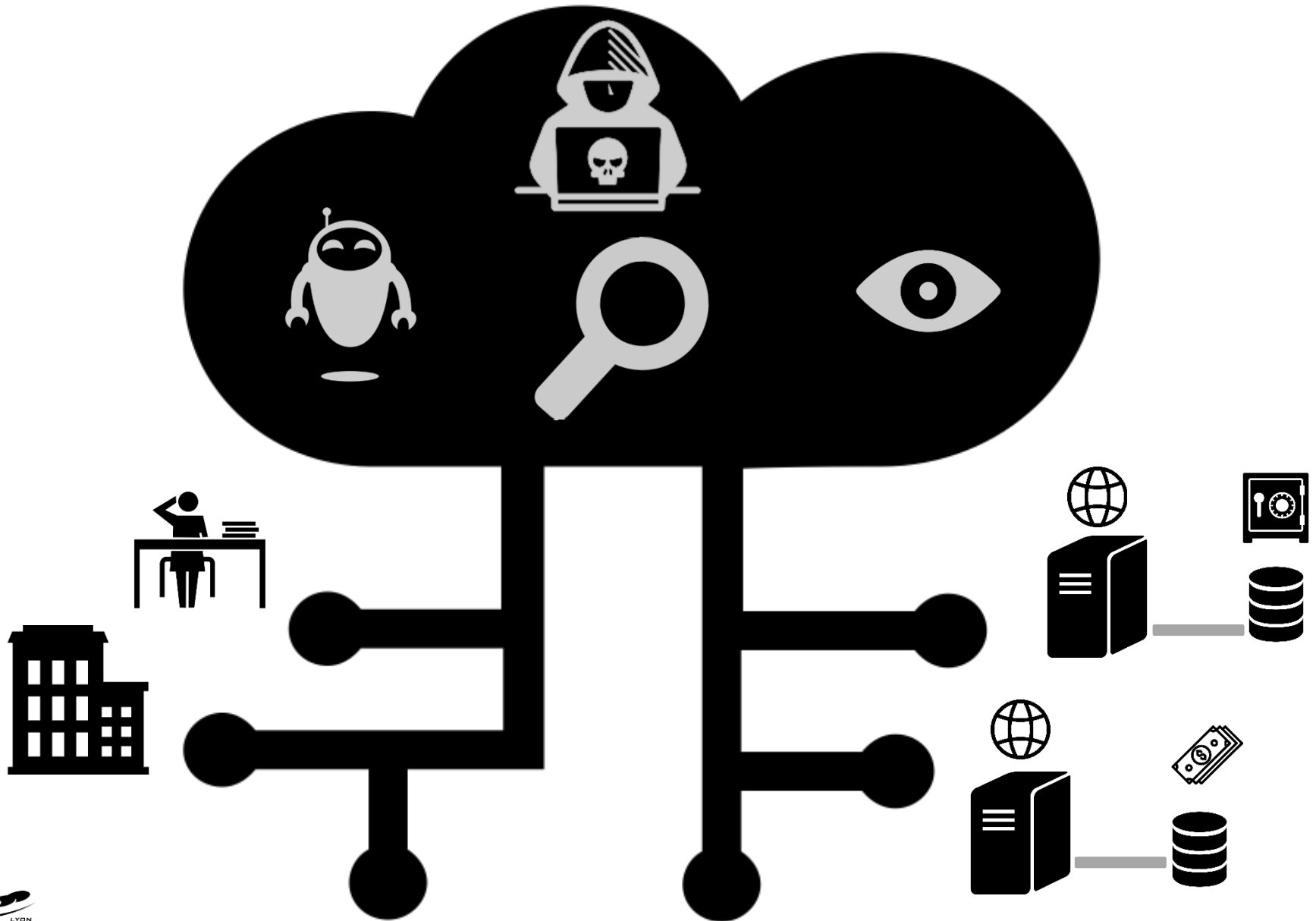
Besoins, Outils, Normes

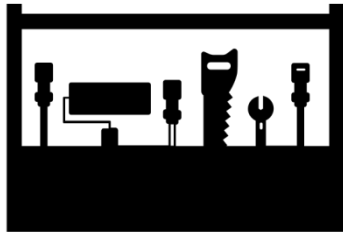
J. Saraydaryan





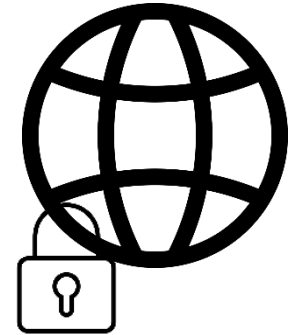
Les Besoins





Les Outils

Les Outils



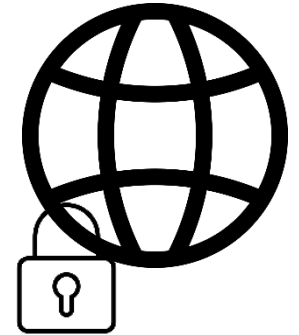
- Authentication Web classique
 - Basique
 - Digest
 - Session Vs Token
 - Cookie or not Cookie

- OAuth

- OpenId

- Autres

Les Outils : Authentification Basique



- RFC 2617: Basic
- Envoyer directement le login/pwd au serveur
- Utiliser le **authorization** des headers HTTP

- Login pwd en une seule chaine **username:password**
- Chaine encodé en Base64
- The mot clé **basic** est ajouté avant la valeur encodé

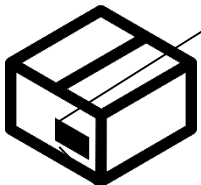
Les Outils : Authentification Basique



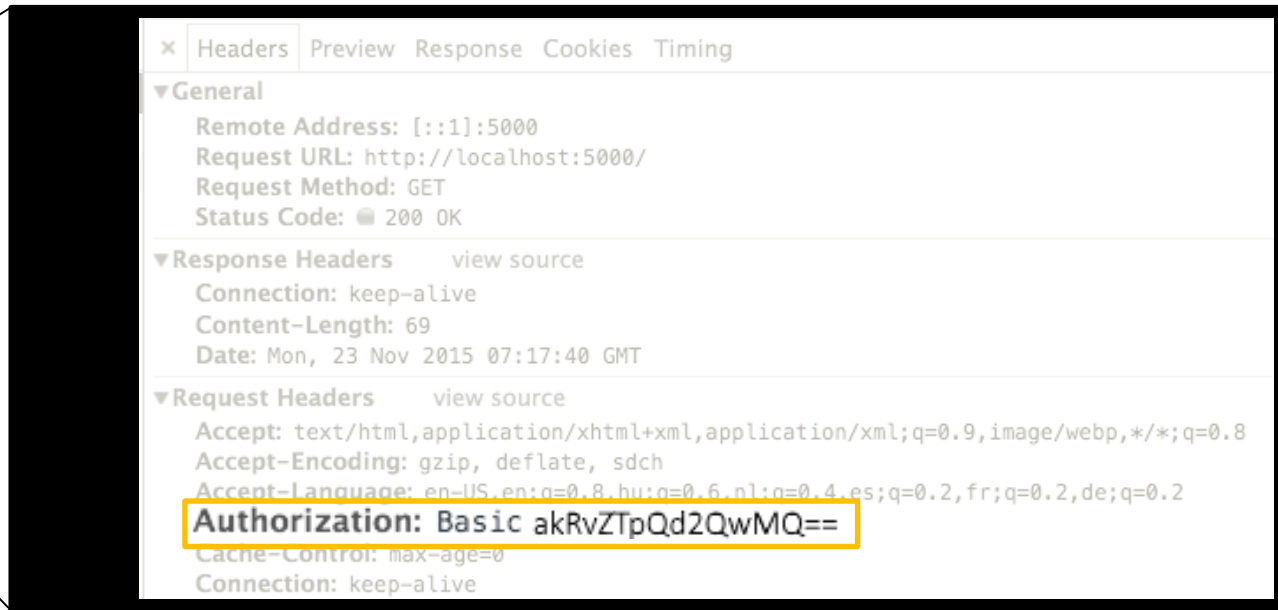
User

Login: jDoe
Password: Pwd01

Base64 (jDoe:Pwd01) =
akRvZTpQd2QwMQ==



HTTP Packet



HTTP Header (Chrome View)

Copyright © Jacques Saraydaryan

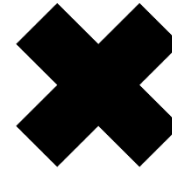
<https://blog.risingstack.com/web-authentication-methods-explained/>

Les Outils : Authentication Basique



Avantages

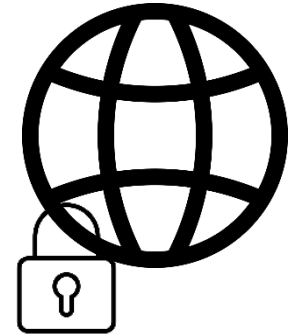
- Authentication envoyée à chaque requête
- Simple à mettre en place (coté serveur et client)



Inconvénients

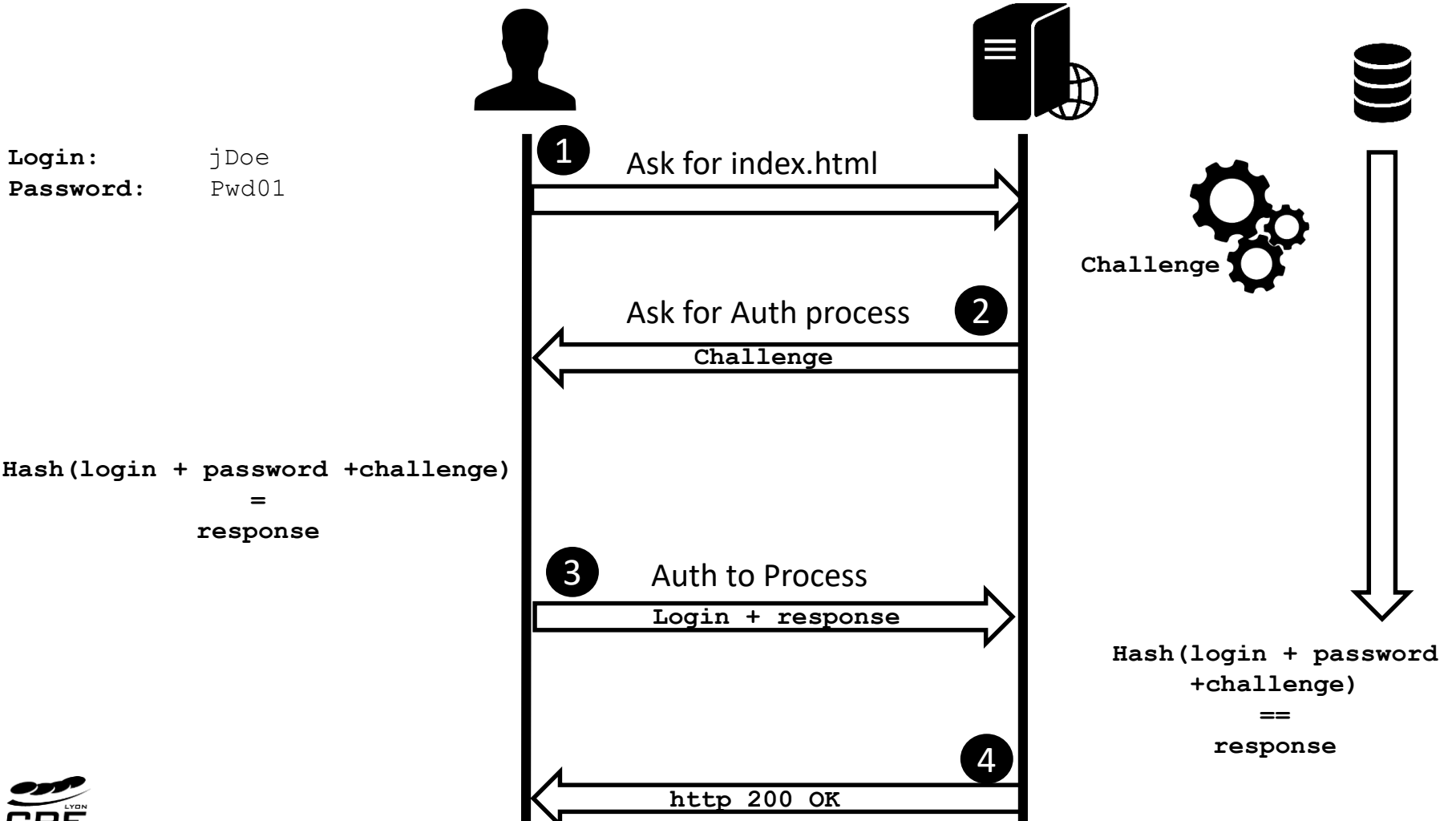
- Authentication envoyée à chaque requête → risque d'interception élevé
- Pas de chiffrement des données
- Pas de possibilité de log out
- Expiration d'authentification complexe (demande de changement de pwd ?)

Les Outils : Authentication Digest



- RFC 2617: Digest
- Authentication via demande réponse
- Négociation d'un algo de « chiffrement »
- Le client répond avec le login, le password et les infos de négociation (appelées realm) « chiffrés ».

Les Outils : Authentication Digest



Les Outils : Authentication Digest

Login: jDoe
Password: Pwd01
HTTP METHOD: Get
URI: /dir/index.html

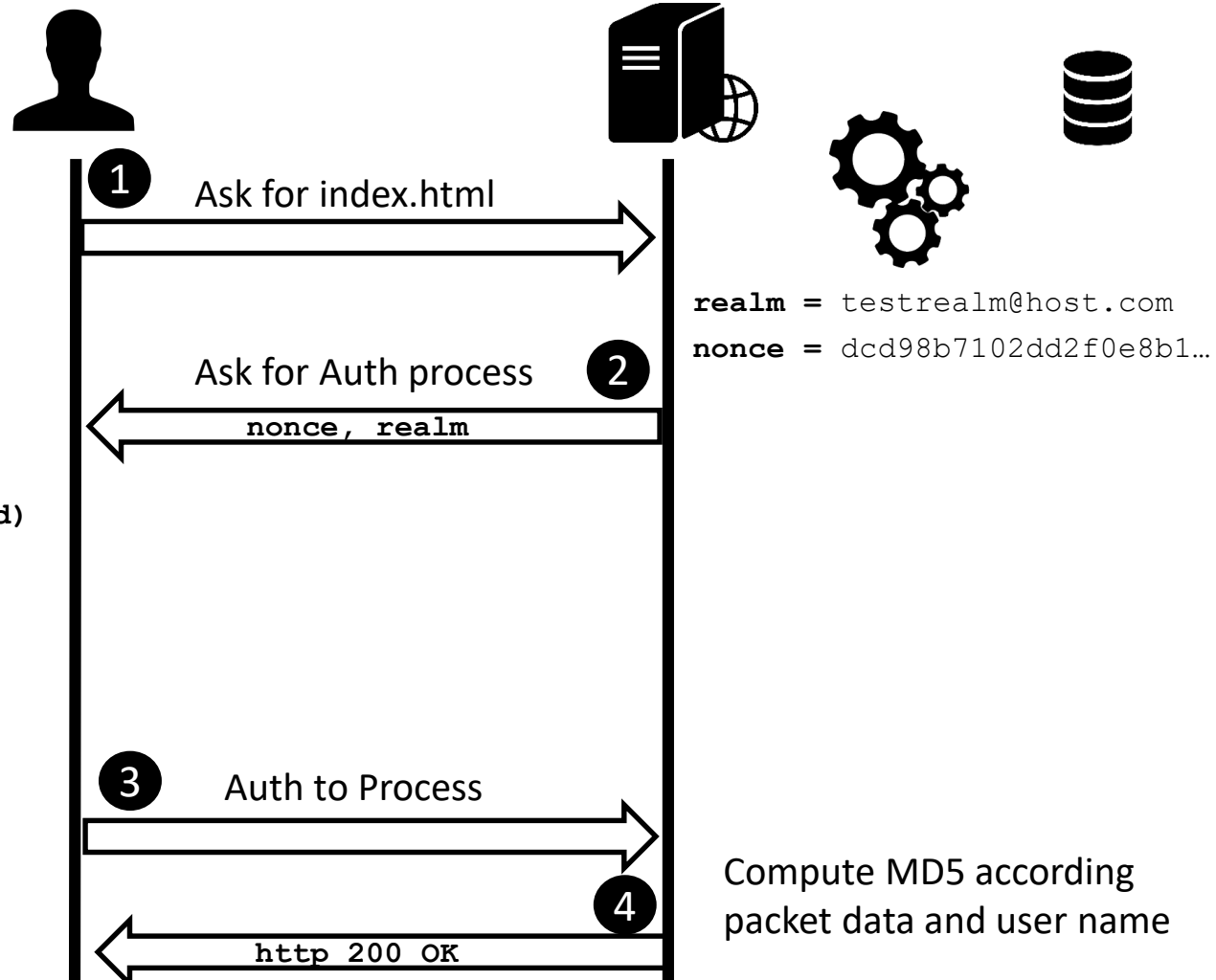
V1=MD5 (username:realm:password)

V1=MD5 (jDoe:testrea...:Pwd01)

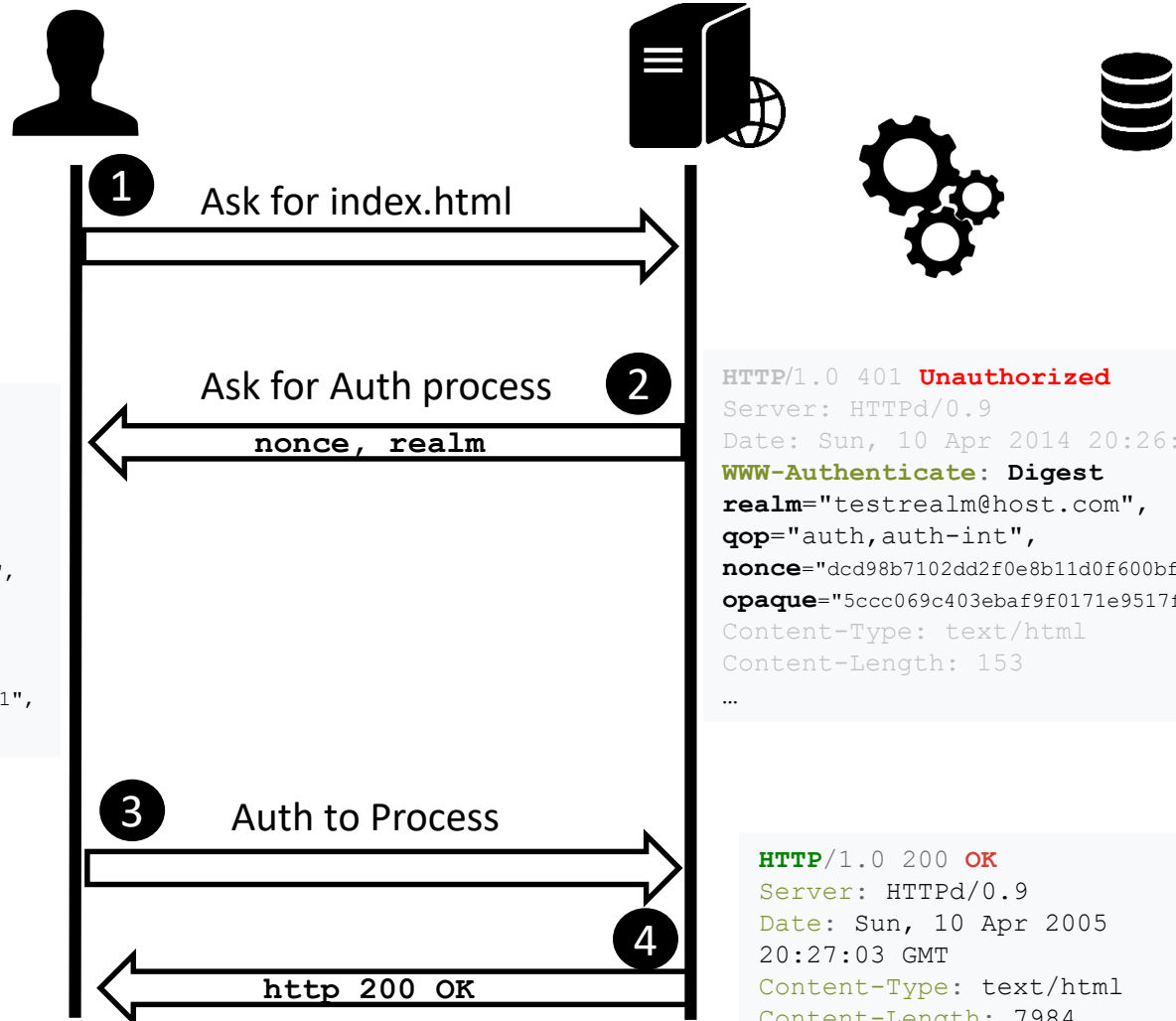
V2=MD5 (method:digestURI)

V2=MD5 (Get:/dir/index.html)

response=MD5 (V1 : nonce : V2)



Les Outils : Authentication Digest



```
GET /dir/index.html
HTTP/1.0 Host: localhost
```

```
GET /dir/index.html HTTP/1.0
Host: localhost
Authorization: Digest
username="Mufasa",
realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="/dir/index.html", qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

```
HTTP/1.0 401 Unauthorized
Server: HTTPd/0.9
Date: Sun, 10 Apr 2014 20:26:47 GMT
WWW-Authenticate: Digest
realm="testrealm@host.com",
qop="auth,auth-int",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: text/html
Content-Length: 153
...
```

```
HTTP/1.0 200 OK
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005
20:27:03 GMT
Content-Type: text/html
Content-Length: 7984
```

Les Outils : Authentication Digest

□ Les options

- **domain**: liste d'URI définissant le domaine de protection
- **opaque**: chaîne générée par le serveur que retourne le client sans modification (anti-replay)
- **stale**: retour du serveur indiquant si échec provient de l'usage d'une ancienne version de nonce
- **algorithm**: algos pour les fonctions de Hash (seulement MD5 MD5-sess)
- **qop**: Quality of protection niveau de sécurité auth ou auth-int (indique le type de hash à réaliser)

□ Plus de détails :

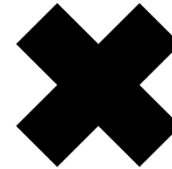
- <https://tools.ietf.org/html/rfc2617>
- https://en.wikipedia.org/wiki/Digest_access_authentication
- https://fr.wikipedia.org/wiki/Authentication_HTTP#M.C3.A9thode_Digest

Les Outils : Authentication Digest



Avantages

- Challenge (le password ne passe jamais seul en clair)
- Password Hashé et Salé
- Evite les attaques par replay



Inconvénients

- Beaucoup d'éléments optionnels
- Pas de chiffrement des données
- Man in the middle possible pour le Digest access
- Pas de possibilité de stocker les passwords coté serveur Hashé et salé (besoin de retrouver login et pwd)

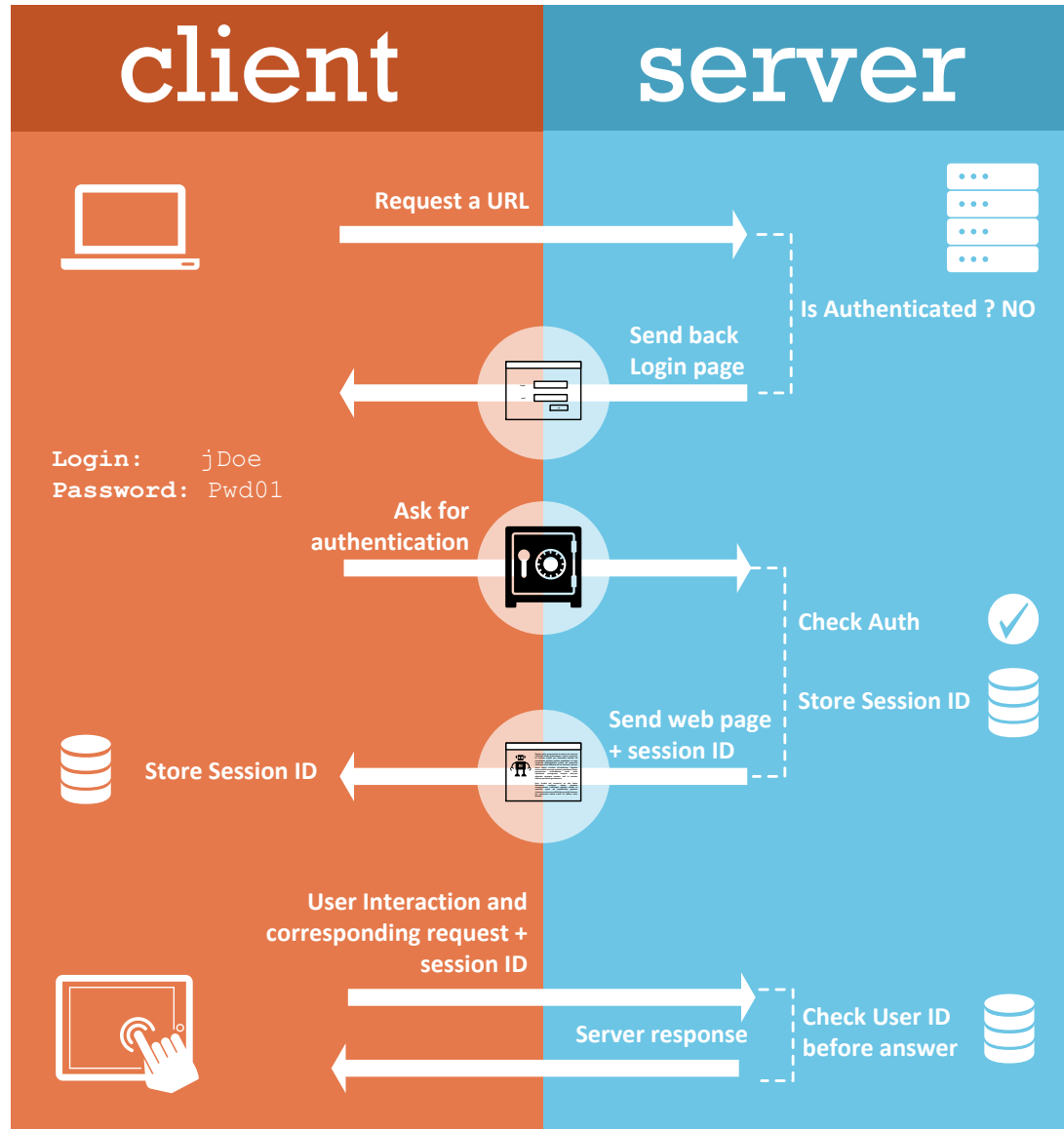


Les Outils : Session VS Token

*Since the **HTTP protocol is stateless**, this means that if we authenticate a user with a username and password, then **on the next request, our application won't know who we are**. We would have to **authenticate again**.*

<https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication>

Les Outils : Session

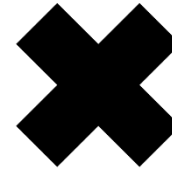


Les Outils : Utilisation de session



Avantages

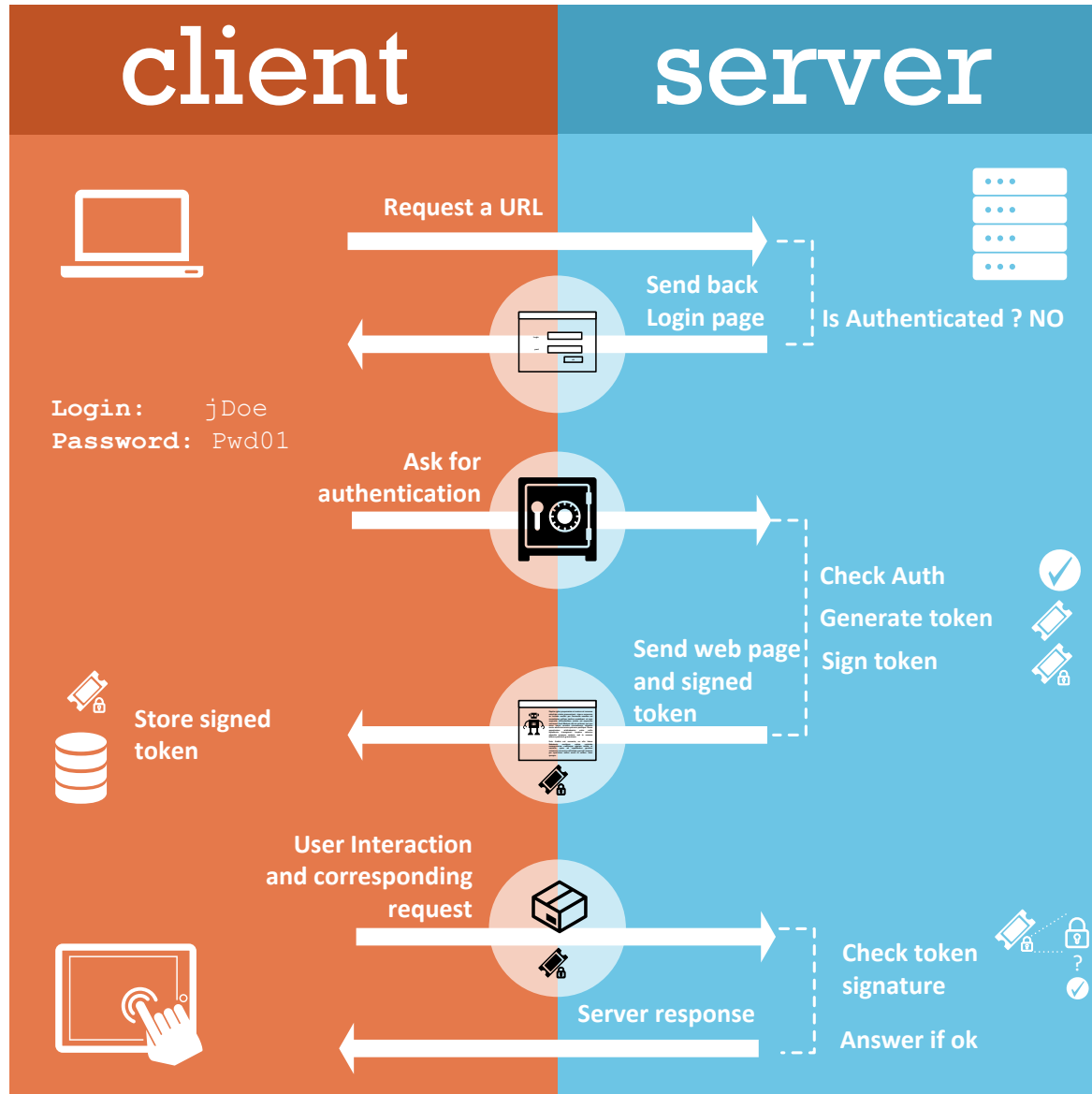
- Conservation d'une trace d'activité de l'utilisateur
- Login/logout plus facile



Inconvénients

- Incompatible Full REST (stateless)
- Stockage d'info coté serveur, difficulté de passage à l'échelle
- Usage d'application Cloud, les coûts associés peuvent être importants (mobilisation de plus de serveurs)

Les Outils : Token

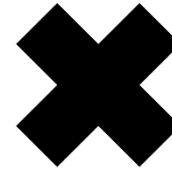


Les Outils : Utilisation de token



Avantages

- Stateless compatible avec les API FullRest
- Vérifier uniquement la signature des tokens, évite de stocker des infos dans le serveur
- Possibilité de passer le token à d'autres applications
- Association d'un TTL au token possible

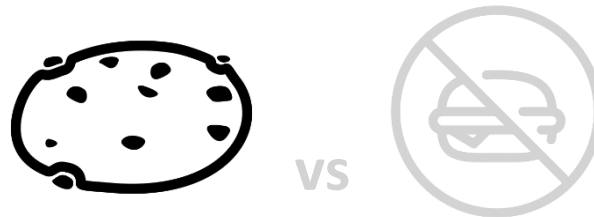


Inconvénients

- Pas de maintien de session utilisateur
- Difficulté des logout (plutôt usage de TTL sur le token)

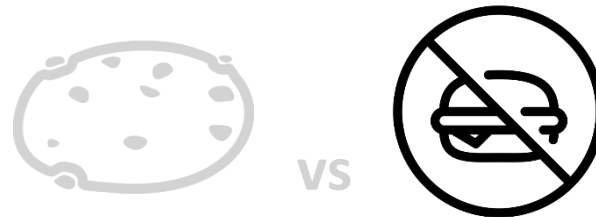
Les Outils : Cookie or not cookie

- Obligatoire en mode session
- Possibilité de se reconnecter directement
- Possibilité d'attaques:
 - Récupération d'information
 - Modification d'information (utiliser plutôt des cookies signés)
 - XSS (utilisé plutôt de préférence http-only)



Les Outils : Cookie or not cookie

- Evite la récupération d'un token (spoofing)
- Adapté pour les single page applications (stockage en mémoire)
- Possibilités d'attaques réduites:
 - Récupération d'information plus complexe
 - Modification d'information très difficile (token signés)



JSON Web Token (JWT):

*JSON Web Token (JWT) is a compact, **URL-safe** means of representing claims to be transferred between two parties. The claims in a **JWT are encoded as a JSON object** that is used as the payload of a **JSON Web Signature (JWS) structure** or as the plaintext of a **JSON Web Encryption (JWE) structure**, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted*

<https://tools.ietf.org/html/rfc7519>

JSON Web Token (JWT):

- Header :
 - Type de token utilisé
 - Type d'algo utilisé pour la signature

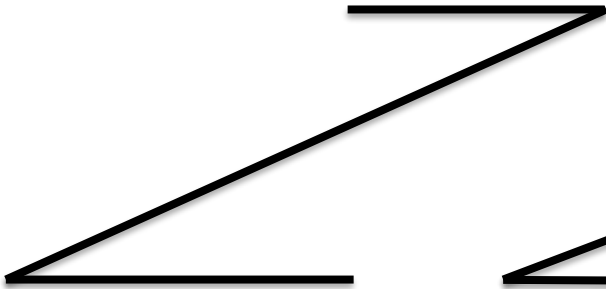
- Payload
 - User defined attributes (public claims)
 - Some are standard (called reserved claims)

- JWT Signature (HMAC or RSA)
 - Header
 - Payload
 - Secret (hmac)

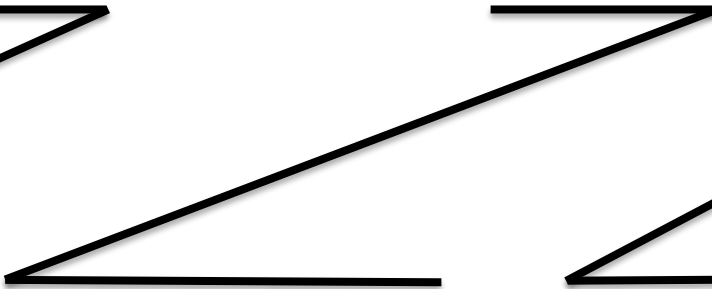
<https://jwt.io/introduction/>

JSON Web Token (JWT):

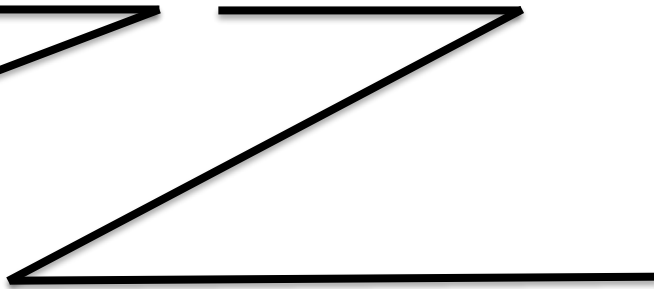
BASE64Url(HEADER).BASE64Url(Payload).Signature



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



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



```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
)
```


JSON Web Token (JWT):

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYWR9LjJVA95OrM7E2cBab3
0RMHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

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

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  
)  secret base64 encoded
```

OAUTH 2.0:

*L'autorisation **OAuth 2.0** est un framework permettant à une **application tierce** d'**obtenir des accès limités** à un **service HTTP**, soit pour le compte du propriétaire de la ressource soit en autorisant l'application tierce d'obtenir l'accès pour son propre compte.*

OAUTH 2.0:

Séparation des rôles :

- Ressource
- Propriétaire de la ressource
- Client souhaitant utiliser la ressource
- Serveur d'autorisation
- Serveur stockant la ressource (ou service)



OAuth 1.0 très différent de OAuth 2.0

OAUTH 2.0:

□ 4 types d'autorisation

- Authorization code
- Implicit
- Resource owner credential
- Client credential



Web Server
authorization



Front end
authorization



Device operating
system

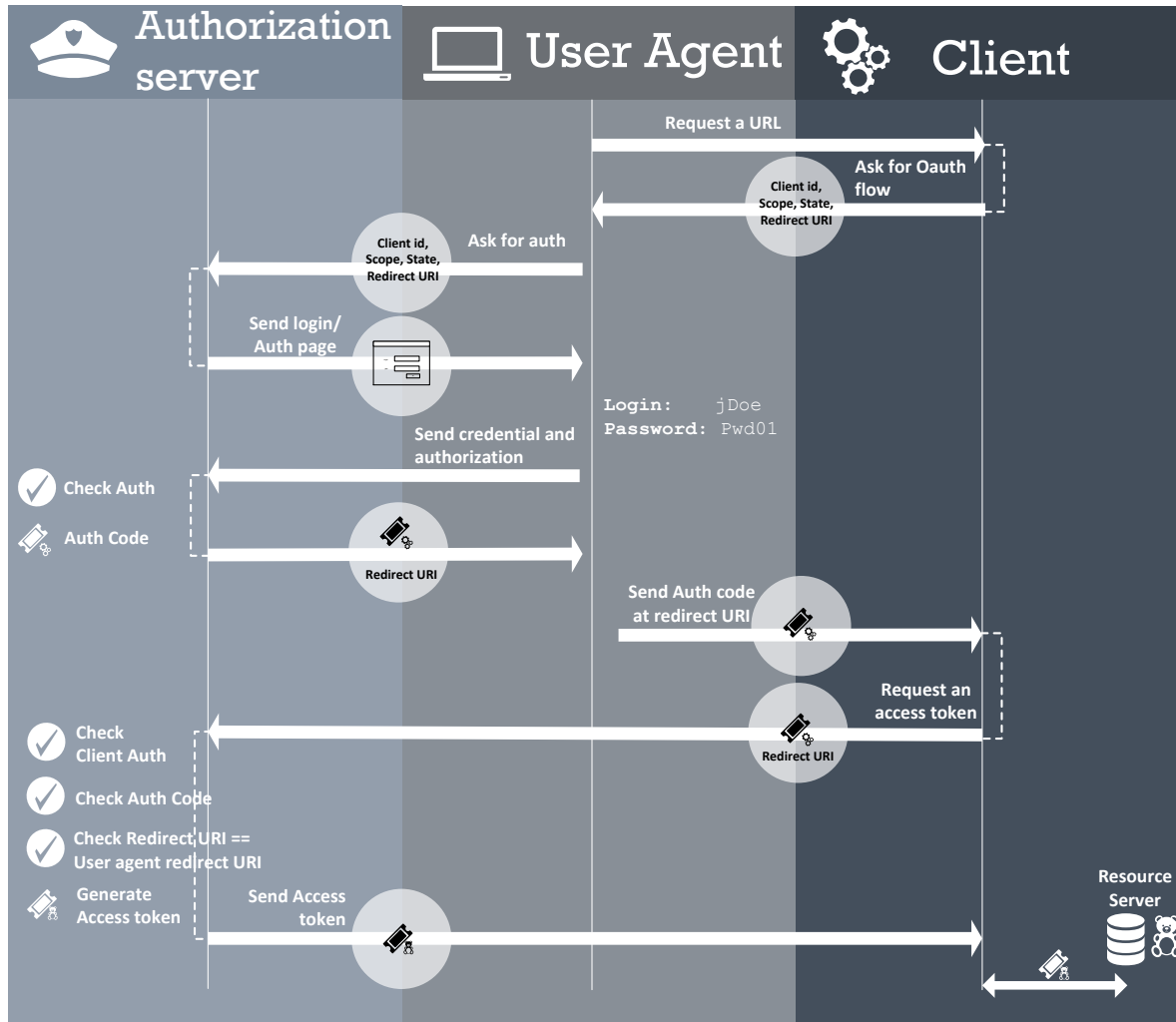


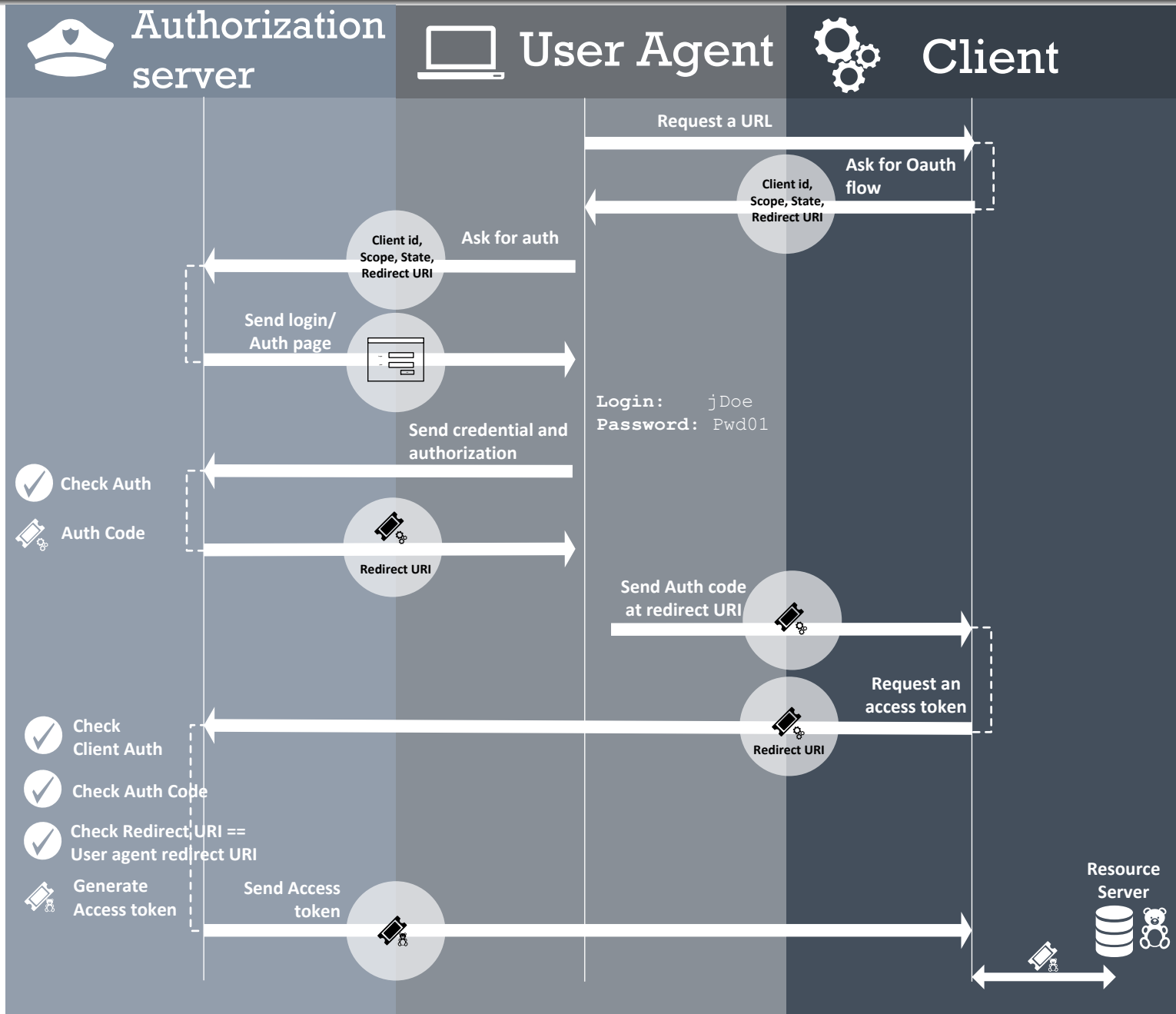
Client by itself

OAUTH 2.0: Authorization Code



Web Server authorization

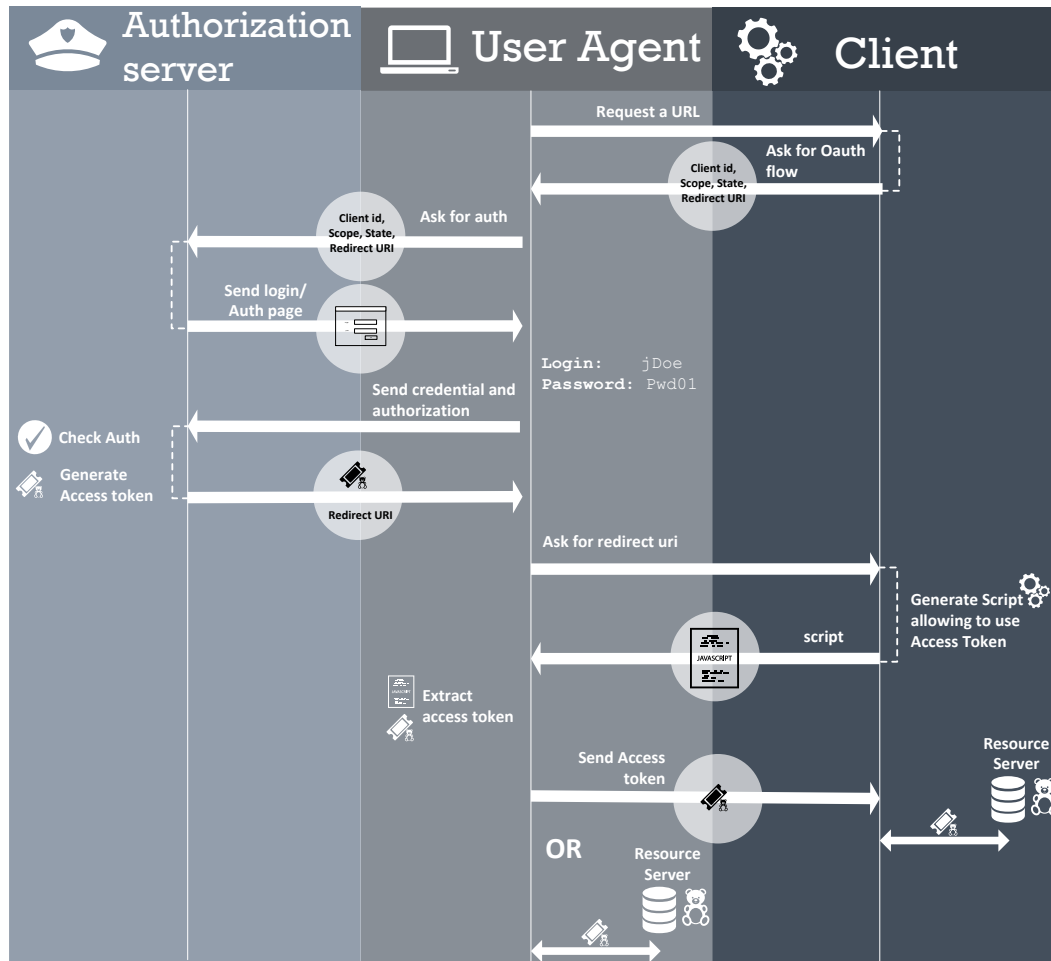


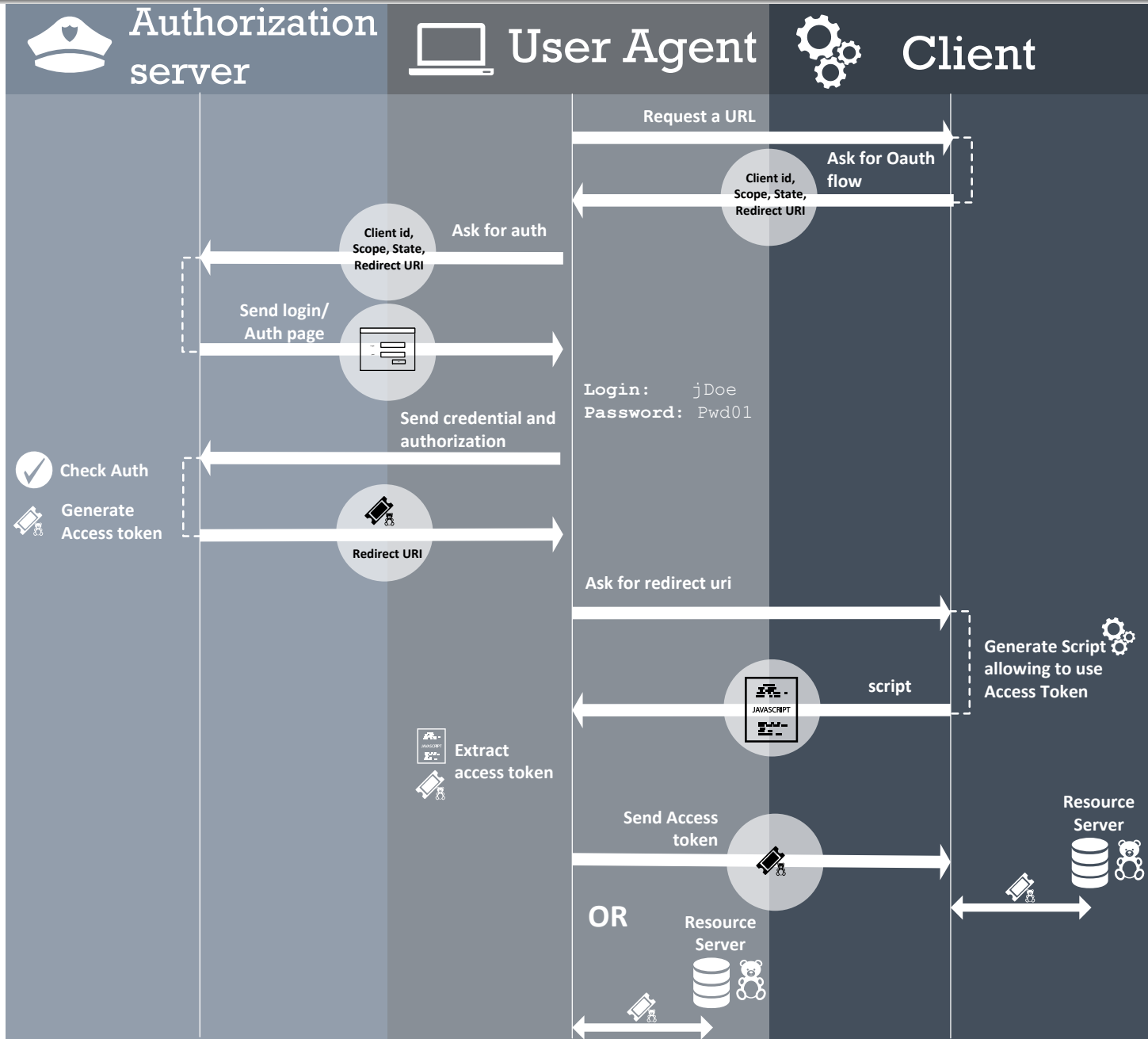


OAUTH 2.0: Implicit grant

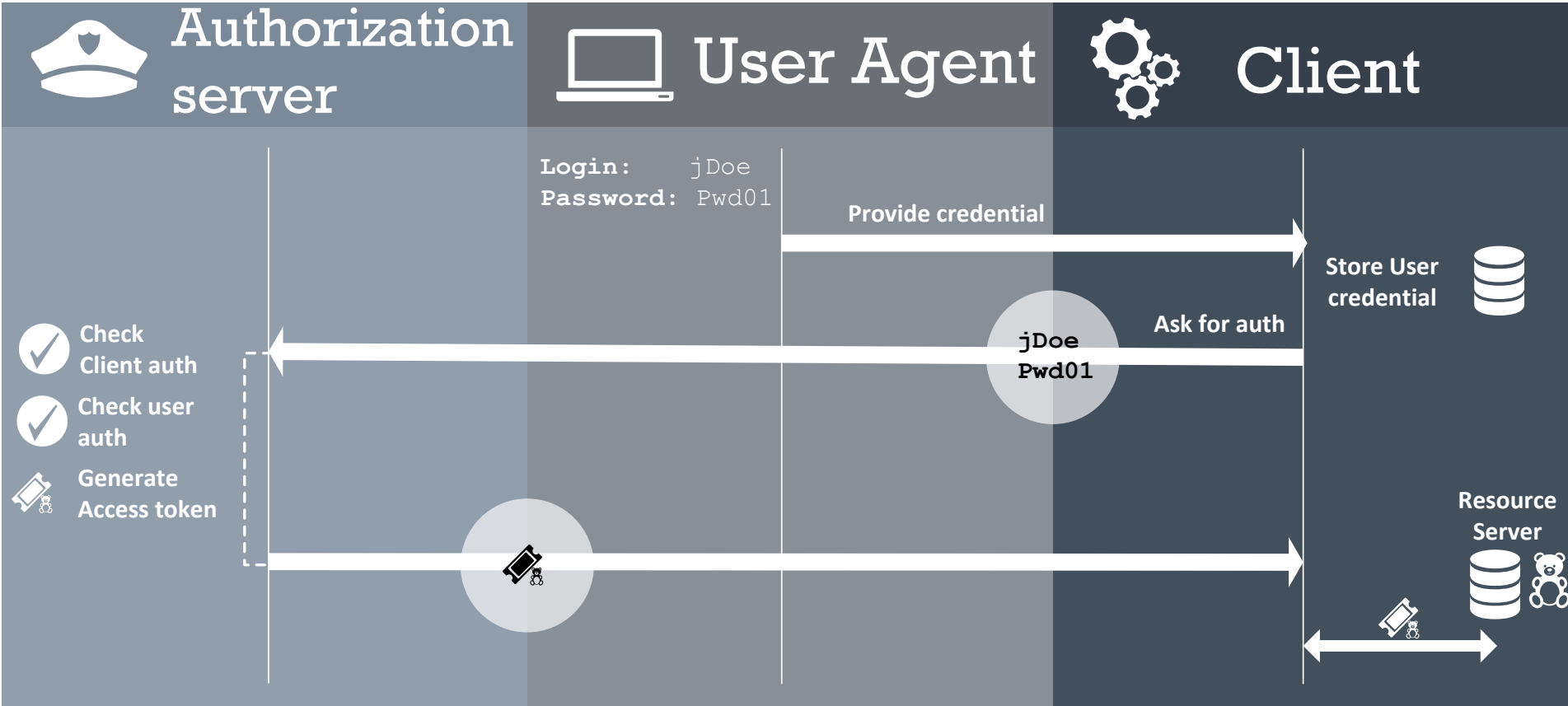


Front end authorization

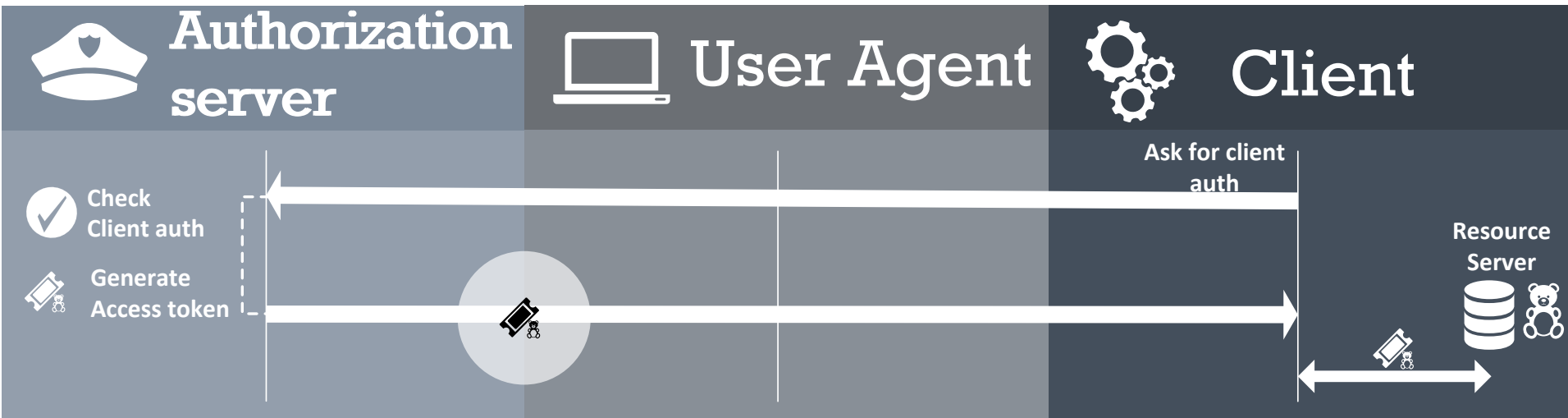




OAUTH 2.0: Resource owner credential

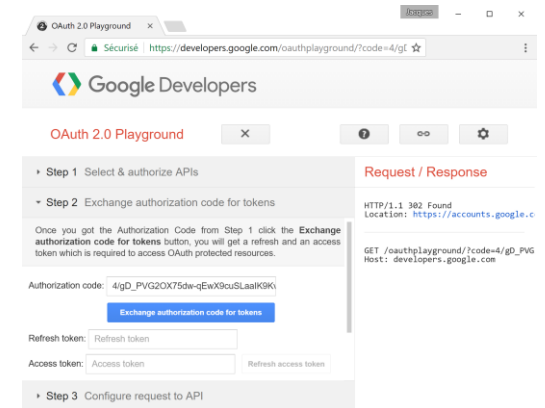
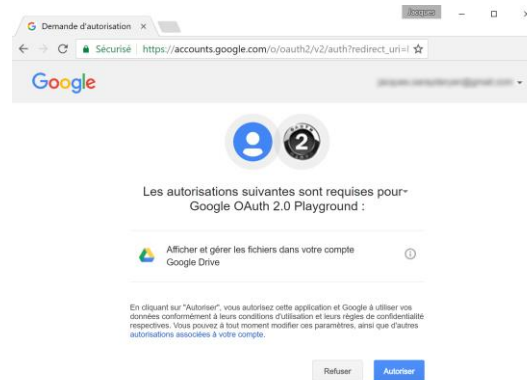
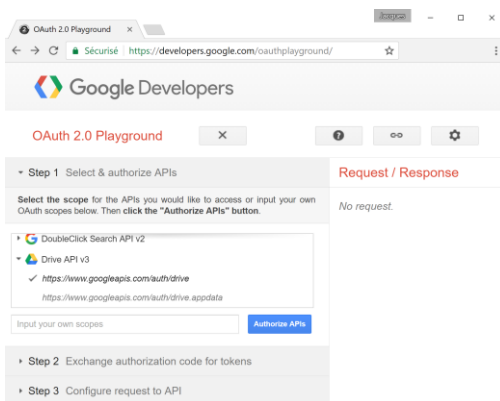


OAUTH 2.0: Client credential



OAUTH 2.0: google exemple

<https://developers.google.com/oauthplayground/>

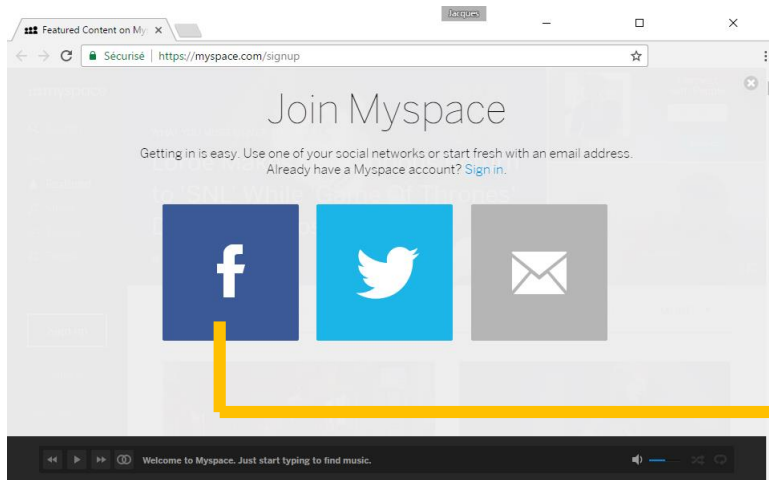


OpenID :

- ❑ Protocol de vérification d'identité
 - Protocol HTTP
 - Possibilité de réaliser une Single Sign-On (SSO)
 - réutiliser une identité provenant d'un fournisseur OpenID
 - Uniquement le fournisseur OpenID gère les mots de passe des utilisateurs

- ❑ Beaucoup utilisé:
 - Google, Facebook, Stack Exchange, Yahoo!

OpenID :



Myspace recevra :
vos profil public, adresse e-mail, anniversaire, ville actuelle, description personnelle, mentions J'aime, activité musicale et activité vidéo. ⓘ

[Modifier ça](#)

Continuer en tant que

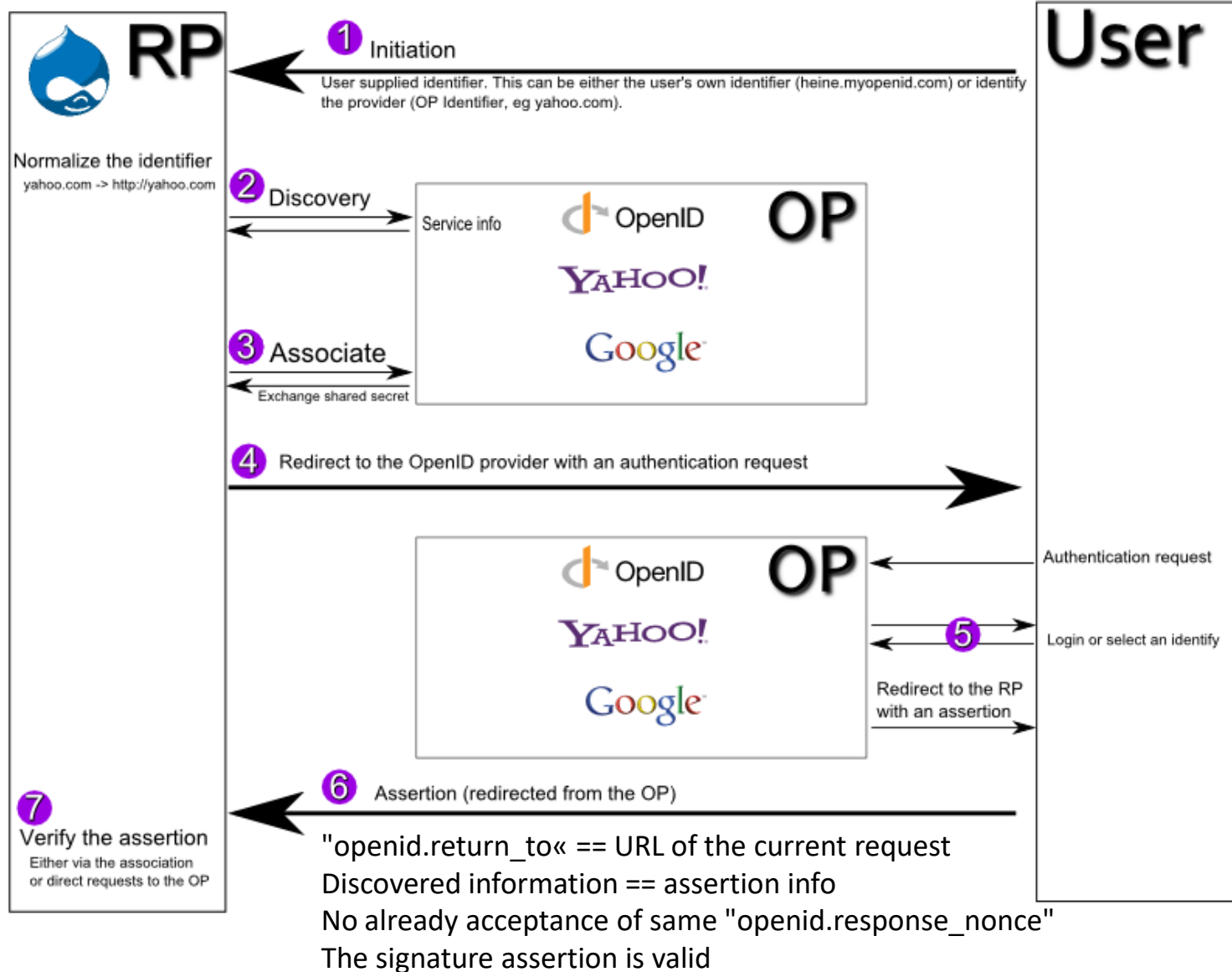
Annuler

🔒 Cela ne permet pas à l'application de publier sur Facebook

[Conditions d'utilisation de l'application](#) · [Politique de confidentialité](#)

OpenID 2.0:

User: Web browser
OP: OpenID provider
RP: Relying party



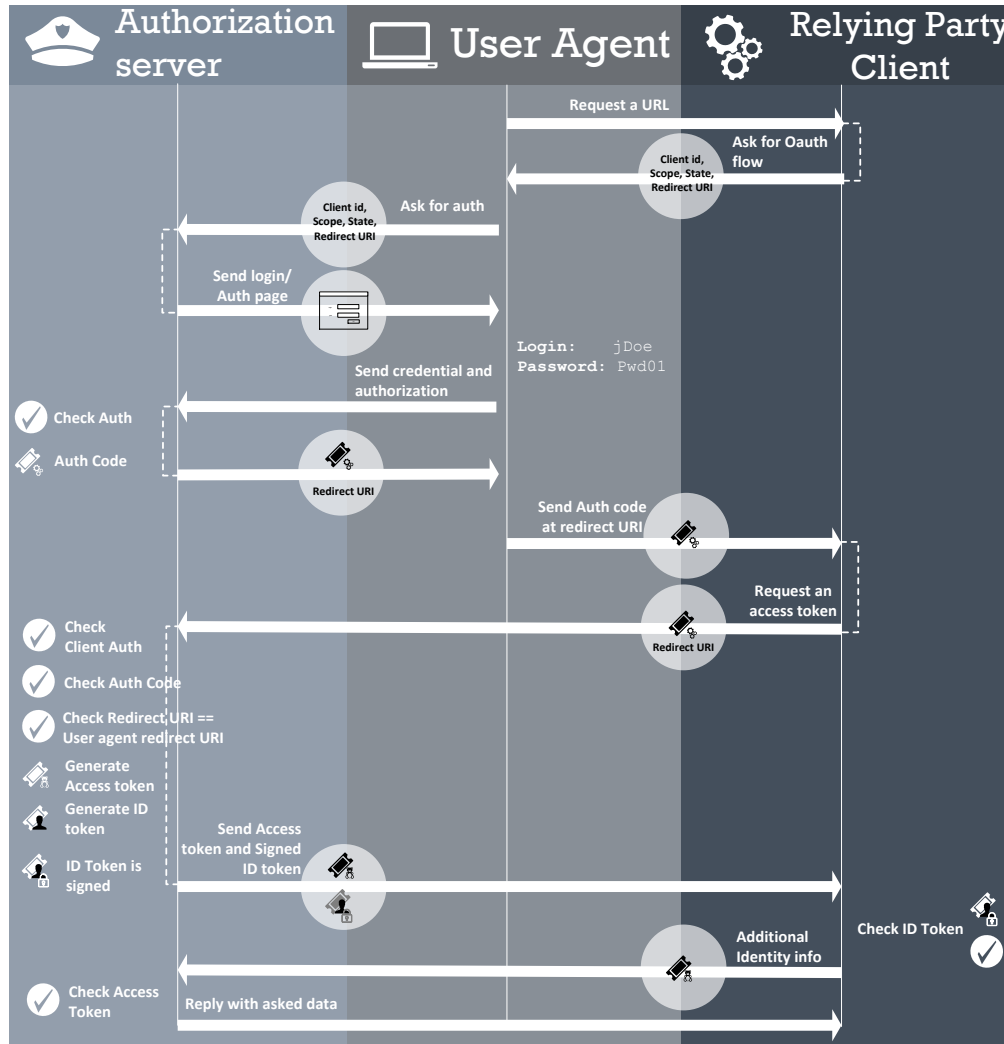
OpenID Connect:

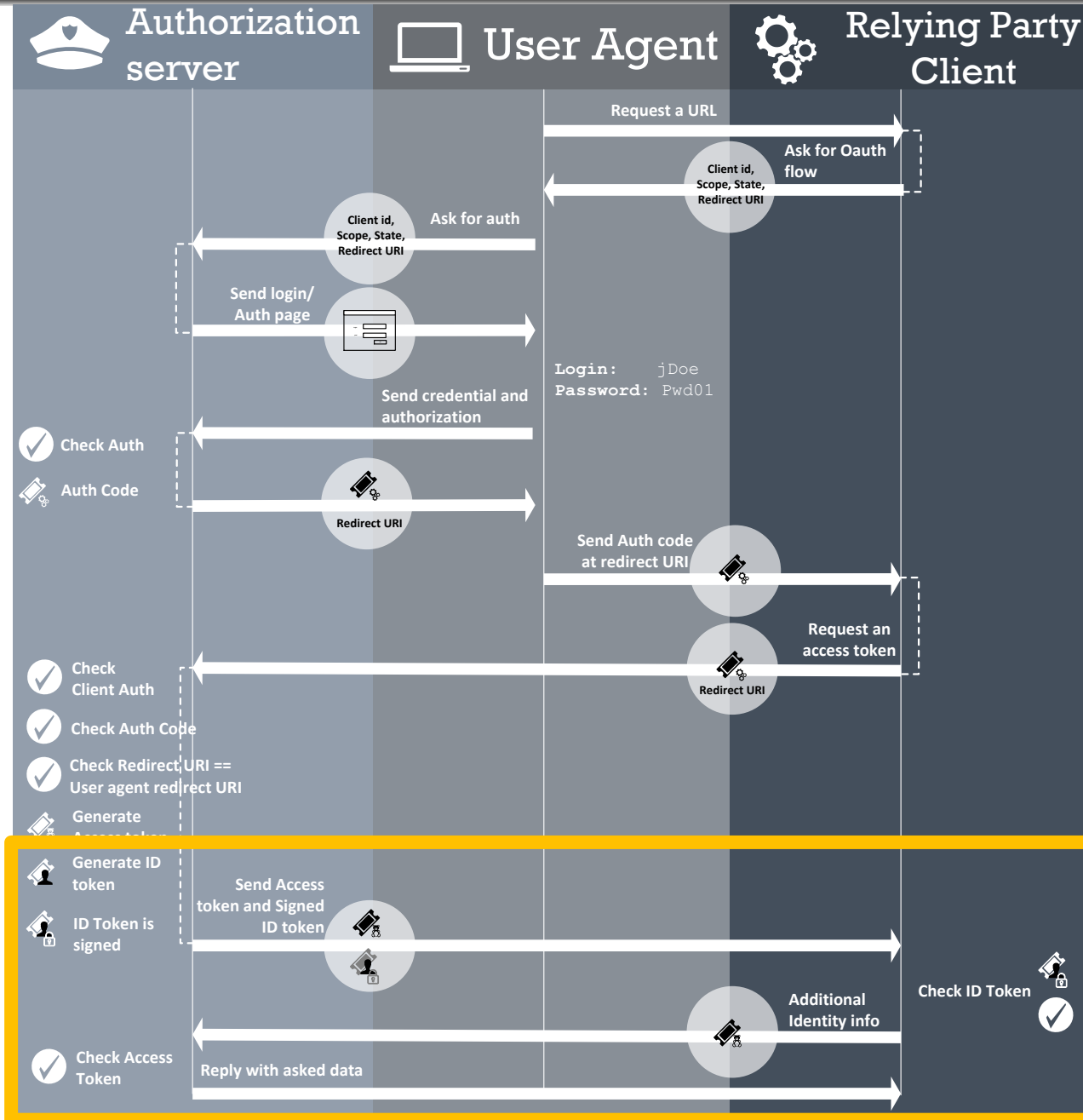
- Protocol de vérification d'identité

- Construit sur OAUTH 2.0

- OpenId Connect
 - Mêmes fonctionnalités assurées que OpenID 2.0
 - plus orienté API-friendly (meilleure interopérabilité, « REST-Like ») que OpenID 2.0
 - Utilisable par des applications natives et mobiles

OpenID Connect:





Specific OpenID Connect



Questions ?



References



References

- Web authentication RFC 2617
- JWT RFC 7519

- Web links
 - <https://blog.risingstack.com/web-authentication-methods-explained/>
 - https://en.wikipedia.org/wiki/Digest_access_authentication
 - https://www.owasp.org/index.php/Authentication_Cheat_Sheet
 - https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
 - https://www.owasp.org/index.php/Web_Service_Security_Cheat_Sheet
 - <https://www.w3.org/TR/webauthn/>
 - <https://blog.netapsys.fr/oauth-comment-ca-marche/>
 - <https://api.slack.com/docs/oauth>
 - <https://developers.google.com/identity/protocols/OAuth2>
 - <https://oauth.net/getting-started/>
 - <https://heine.familiedeelstra.com/openid-compliance-crusade-part-1>
 - http://openid.net/specs/openid-authentication-2_0.html
 - <https://auth0.com/blog/cookies-vs-tokens-definitive-guide/>
 - <https://jwt.io/introduction/>



© creative commons



Created by Unrough™
from the Noun Project



Created by jayjayr
from the Noun Project



Globe by Richard Schumann
from The Noun Project



Created by JAMES ADAM TITTEL
from the Noun Project



Created by Lisa English
from the Noun Project



Created by Gerdit Albrecht
from the Noun Project



Created by Gregory Rabinovich
from the Noun Project



Created by Peter van Diehl
from the Noun Project



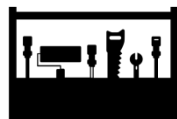
Created by Edoardo Esposito
from the Noun Project



Created by Rutmer Zijlstra
from the Noun Project



Created by Christiane Beal
from the Noun Project



Created by Björn Andersson
from the Noun Project



Created by Gerdit Albrecht
from the Noun Project



Created by Edoardo Esposito
from the Noun Project



Created by Edoardo Esposito
from the Noun Project



Created by Jason Aherne
from the Noun Project



Created by Stefan Hering
from the Noun Project



Created by Gerdit Albrecht
from the Noun Project



Created by A-Hojo Pichler
from the Noun Project



Created by Aaron K. Kern
from the Noun Project



Chat by Luiz Henrique Bello Cera
from The Noun Project



Quote by Irene Hoffman
from The Noun Project



Jacques Saraydaryan

Jacques.saraydaryan@cpe.fr