OAuth 2.1 expliqué simplement (même si tu n'es pas dev) !

Date : 21 avril 2022
Speaker : Julien Topçu (Shodo)
Format : conférence (45mn)

La conférence démarre par des extraits du film The Grand Budapest Hotel.
On y voit 2 convives se taper dessus.
Le film se déroule en 1932, dans un passé lowtech (en référence à la keynote du matin).

Chapitre 1 : les squatteurs

Le gérant Gustave a sécurisé les parties privatives avec une serrure à code posée sur la porte. Le code unique est donné aux convives et aux salariés.
Les anciens convivent arrivent à squatter les chambres en se souvenant de leur code.
On note ici un problème de révocation de code d'accès.

Moyen 1 : générer un nouveau code à chaque départ de convive. Impraticable.
Ce problème de révocation existe sur le web. OAuth 2 y répond.

Julien prend l'exemple d'une ancienne page fictive de Facebook récupérant tous les contacts d'un utilisateur sur AOL, Skype, Yahoo … L'utilisateur devait donner son email et son mot de passe.
Problème : si on donne le mot de passe Gmail à Facebook et que Facebook continue à l'utiliser, pour révoquer ce mot de passe, il faut changer son propre mot de passe.

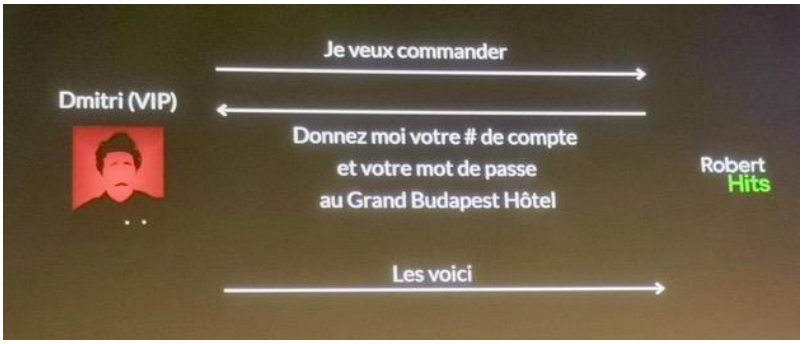
Correspondence film / OAuth 2 :

- Chambres : Resource
- Grand Budapest Hotel : Resource Owner
- Hotel : Resource Server
- Convives : Client(s)

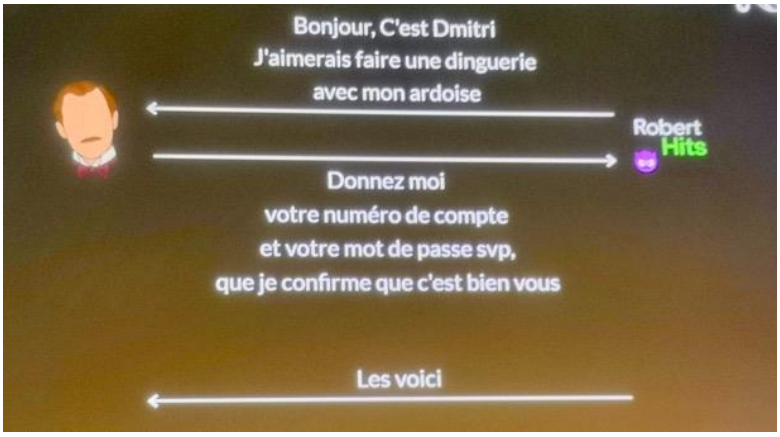
Besoin : distribuer des codes d'accès temporaires et personnalisés.
Le numéro de réservation est unique et peut faire office de Code d'Autorisation (**Authorization Code**). On peut l'utiliser pour ouvrir la serrure à code.
Le problème de révocation est corrigé.

Chapitre 2 : détournement d'ardoise

Les VIP peuvent commander des pâtisseries.
Le prestataire externe (Robert Hits) est choisie pour faire la livraison
Dimitri appelle Robert Hits (VIP). Robert Hits s'assure de l'identité de Dimitri.
Un employé malveillant s'est fait passer par le gérant.

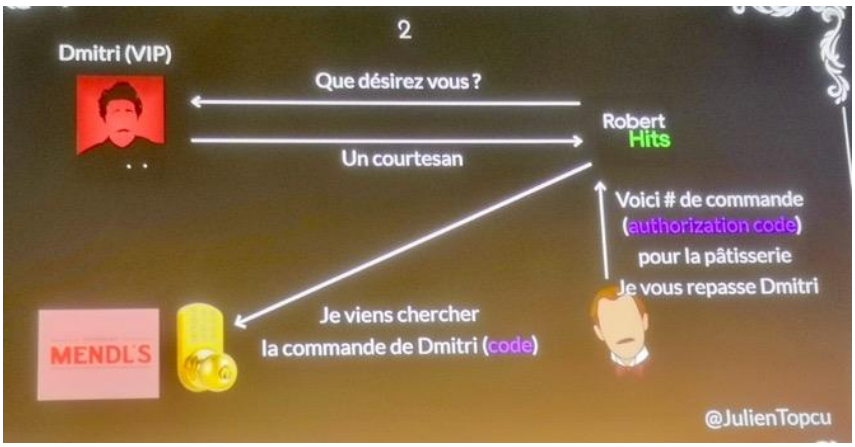


Problème d'imposture.
A chaque fois qu'on donne son mot de passe, on offre un moyen d'usurper son identité.
Cette pratique est à bannir.
OAuth 2 a été inventé pour ne pas avoir à donner son mot de passe.



Robert Hits s'adresse à Gustave (ici, le tiers de confiance) pour qu'il vérifie que c'est bien Dmitri le client de l'hôtel. Gustave et Dimitri discutent au téléphone. Dmitri a le droit de demander son numéro de client.
Gustave fait office d'**Authorization Server** : il authentifie et protège l'**End User** (Dmitri). Confirme l'accès du End User auprès de Robert Hits.

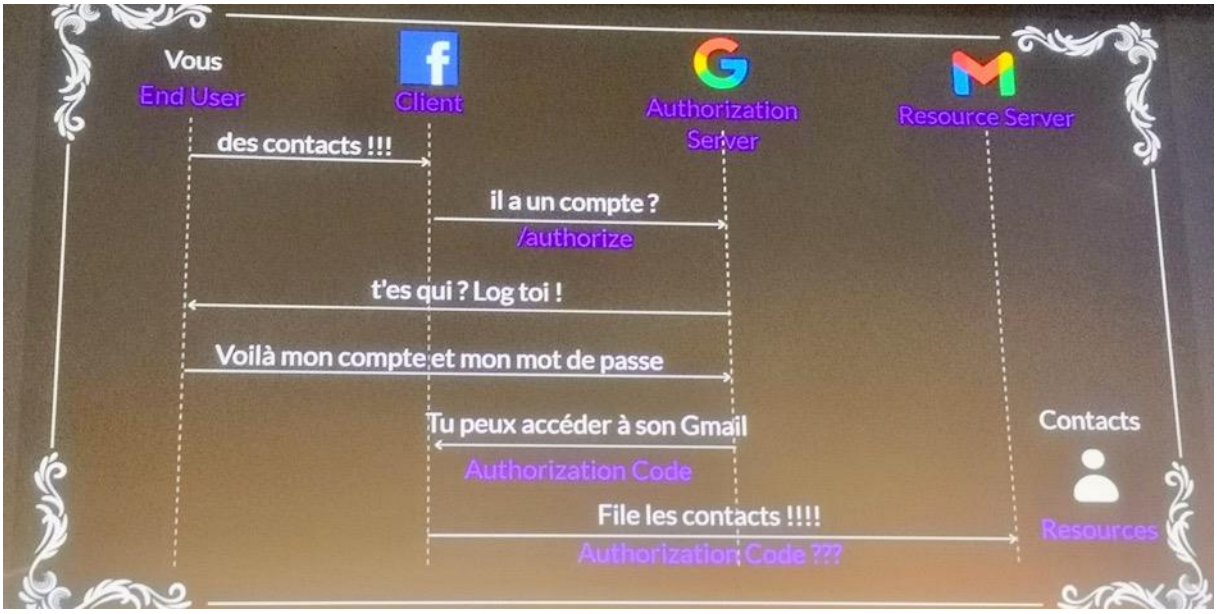
Gustave envoie à Robert Hits un numéro de commande (un équivalent **d'authorization code**) pour qu'il puisse aller chercher la pâtisserie (dans une pièce de l'hôtel)



- Ardoise, Pâtisserie : Resource
- Dmitri : Resource Owner & End User
- Hôtel : Resource Owner
- Numéro de commande : Authorization Code
- Gustave : Authorization Server
- Robert Hits : Client

Dmitri, le End User, délègue la manipulation de ses ressources (les pâtisseries) à un tiers. Robert Hits est le **Client** de **Resources** auprès du Resource Server à la demande d'un **End User** pour lui offrir un service.

Julien reprend l'exemple de l'import de contact dans Facebook
Un Client est toujours un logiciel dans OAuth 2.

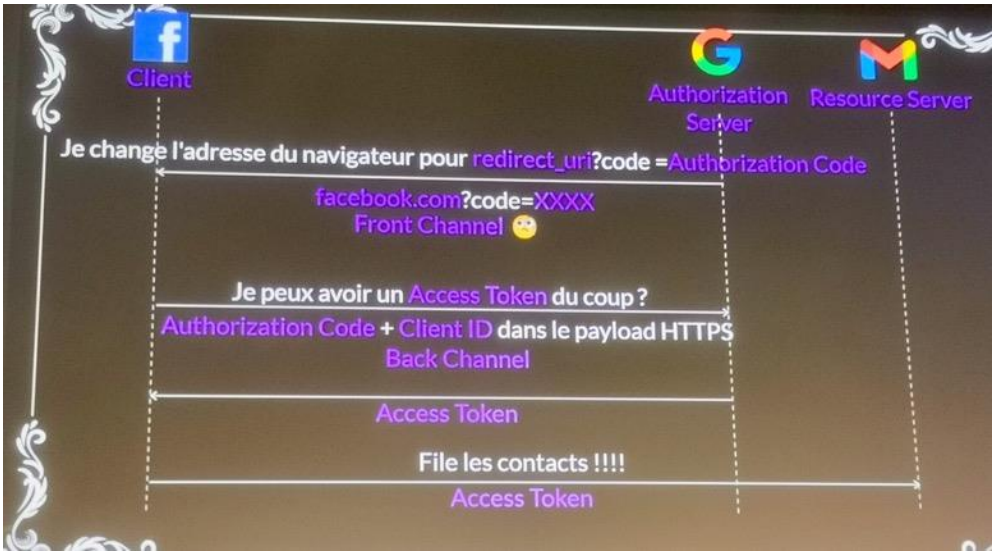


A aucun moment Facebook n'a désormais accès au mot de passe.

Chapitre 3 : Les margoulins de la concurrence

Des concurrents de Just Munch ont réussi à livrer des pâtisseries en se faisant passer par Robert Hits. Just Munch appelle des VIP en le hameçonnant.
Pour se prémunir, Gustave liste tous les numéros de téléphone de Robert Hits afin de vérifier qui l'appelle.

Facebook ne dialogue pas directement avec Google. La direction contient le client_id (de Facebook) et l'adresse de redirection.
Un hacker pourrait modifier l'URL de redirection. Exact match depuis OAuth 2.1



Chapitre 4 : les regards indiscrets

2 convives ont le même numéro de réservation.
Le problème d'un courrier est qu'on ne sait pas si le destinataire l'a reçu. Dans notre cas, un voisin a regardé le courrier et l'a précédé à l'hôtel.
Envoi d'un courrier : **Front Channel** => pas sécurisé
La remise en main propre des accès est à privilégier.
On remplace le code de la porte par une badgeuse avec carte. La carte est remise en main propre. Le numéro de réservation est échangé contre une carte d'accès. Une fois échangé, l'**authorization code** n'est plus valide. La carte d'accès est un **access code**.

La redirection d'URL avec l'Authorization Code est un Front Channel.
Facebook demande un Access Token à Google via un Back Chanel. Google renvoie l'Access Token à Facebook.

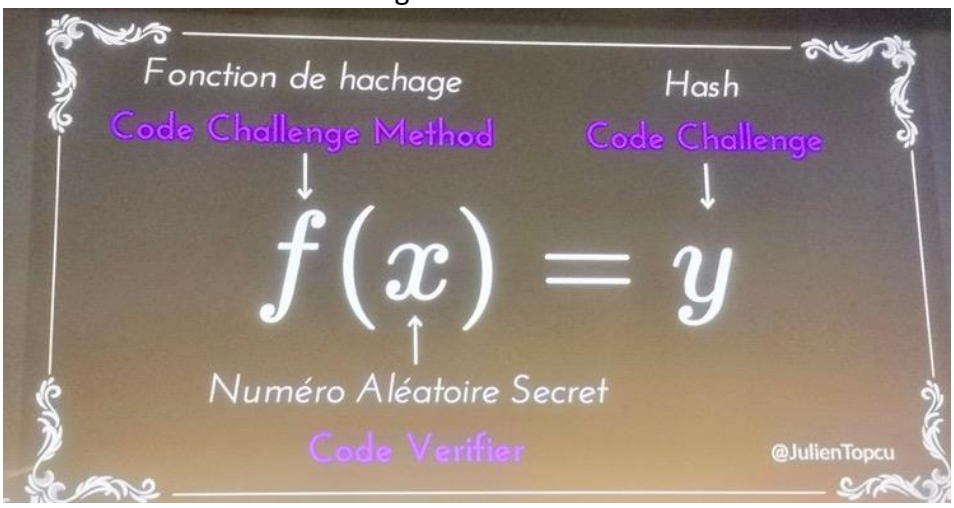
On vient de voir l'**Implicit Flow**, la cinématique la plus utilisée pour les webapps.
Ce flow est aujourd'hui, en 2022, à éviter car hackable.

Chapitre 5 : les infiltrés

Un infiltré peut récupérer l'**authorization code** et l'utiliser avant la vrai VIP.
Avant de passer la commande, Valentin (le gentil) choisit un algo de hash (ex : SHA-256)
Valentin encode un numéro aléatoire secret qu'il a choisi avec la fonction de hash
Ce numéro (non hashé) n'est communiqué que sur un Back Channel

Ce **flow Auhtorization Code flow with PKCE** s'appuie sur les fonctions de hachage f(x)=y
Ce flow vient du mobile. C'est la best practice de OAuth2.1.

Julien invite tous les dévs à migrer tous les fronts



Les scopes

La Carte d'Accès est Scopée au consentement de Dmitri.
Les scopes permettent de limiter l'accès à certaines ressources.

Les **Scopes** ne permettent que d'octroyer des accès à un **Client** (une application) à des **Resources**. Ils ne permettent PAS d'assigner des accès à un **End User** sur des **Resources**.