Authentification

Les systèmes actuels ne fonctionnent que lorsque l'utilisateur·ice est authentifié·e.

Nous allons parler ici de plusieurs types d'authentification différents et de quelques bonnes pratiques pour chacun d'entre eux.

Identification par mot de passe

Le mot de passe est une méthode courante d'authentification, cependant elle n'est pas forcément très sûre, notamment, car elle dépend beaucoup de l'utilisateur·ice.

Comment un mot de passe peut être découvert

Un mot de passe peut être volé de plusieurs manières, soit en **connaissant l'utilisateur** et en devinant le mot de passe (c'est pourquoi il faut que les mots de passes soient aléatoires), ou de façon **brutale** (par dictionnaire ou énumération).

Il est aussi possible de voler un mot de passe en utilisant un **keylogger**, c'est-à-dire un programme ou appareil qui va enregistrer toutes les entrées clavier de l'utilisateur·ice.

Un mot de passe peut aussi être **sniffé** en écoutant tout ce qui se transmet sur le réseau, si le mot de passe n'est pas transféré de manière chiffrée, alors on peut récupérer le mot de passe.

Un mot de passe peut encore être découvert en analysant des bases de données d'anciens mots de passe découvert, car mal stocké par d'autres entreprises. En effet, comme beaucoup d'utilisateur·ice·s ne changent pas de mot de passe, il y a de grandes chances que ce dernier n'ait pas changé.

Enfin, le mot de passe peut encore être découvert à cause d'autres maladresses de l'utilisateur·ice·s, tel que tomber dans une attaque de **fishing** ou encore l'écrire sur un post-it sur son bureau.

Stockage d'un mot de passe

> Si vous voulez en savoir plus sur les dangers et les différentes manières de stocker des mots de passes, regardez cette vidéo.

Plain text

La pire manière de stocker des mots de passes est de juste les stocker en base de donnée. Car lorsque l'on fait cela, ça signifie que toute personne ayant accès à la base de donnée a accès à tous les utilisateurs et mots de passes.

Pire encore, puis ce que les utilisateur·ice·s réutilisent souvent les mêmes mots de passes, pour beaucoup d'entre eux, cela donne accès à l'entièreté de leur vie en ligne.

Chiffré

Une autre manière est de chiffrer les mots de passes. Cela signifie que si quelqu'un a accès à la base de donnée, il ne pourra rien en faire. Cependant, cela signifie aussi que si quelqu'un a accès à la clé de déchiffrement, cette personne a toujours accès à tous les mots de passes. C'est donc également une chose à éviter.

Un autre problème avec cette méthode est que si certains mots de passes sont identiques, on pourra le reconnaitre, car on verra le même mot de passe chiffré dans la base de donnée plusieurs fois.

Hashing

Cette manière consiste à hasher (faire passer à travers une fonction de hashage) les mots de passes.

Une fonction de hashage est une fonction à sens unique, ainsi, on fait passer le mot de passe à l'intérieur, cela génère un texte, mais il est impossible de retrouver le mot de passe à partir du texte.

Ainsi, il suffit de stocker le hash dans la base de donnée, ainsi lorsque l'utilisateur·ice veut se reconnecter, on hash le mot de psase entré et on compare le hash dans la base de donnée avec celui qui vient d'être généré.

Le problème avec cette méthode est que plusieurs utilisateur·ice·s ayant le même mot de passe vont avoir le même hash. Par conséquent, on le saura directement dans la base de donnée.

Il y a notamment une attaque en particulier qui utilise cette faiblesse, c'est la **rainbow table attack**, à la place d'avoir une liste de mots de passes courants, les hasher et les essayer un par un contre chaque hash (ce qui est une attaque par **dictionaire**), on va avoir une liste de mots de passes pré-hashé et simplement comparer les hash. Ce qui fait que l'attaque est beaucoup plus rapide, surtout que beaucoup d'algorithmes de hashage sont volontairement lents afin de décourager les attaquants.

Hashing avec salt

Cette méthode est l'une des meilleures méthodes aujourd'hui. Elle consiste à utiliser un **salt** avec le hash pour se protéger contre les **rainbow table attacks**.

Un **salt** est une chaine de caractère aléatoire différente pour chaque utilisateur qui va être ajouté à chaque mot de passe. Le salt est ensuite présent juste à côté du hash en clair.

Donc, lorsqu'un·e utilisateur·ice se connecte, on va aller chercher le salt, l'ajouter à son mot de passe et le hasher. Ensuite, on compare ce hash avec celui dans la base de donnée.

Grâce à cette chaine aléatoire (le salt), les rainbow table attacks sont inutiles, car elles ne peuvent plus comparer les hash.

Cette méthode est d'autant plus puissante lorsqu'elle fonctionne avec des algorithmes de hashage lent, parce que l'attaquant n'a pas d'autre choix que de tester les mots de passe un par un et va perdre beaucoup de temps.

Ne pas stocker de mot de passe

Stocker des mots de passes est une grande responsabilité, il est possible que ce soit plus sûr de ne tout simplement pas les stocker et d'utiliser des mécanismes tel que OAuth pour à la place demander à des tiers de confiance de le gérer pour nous. Par exemple par Google ou Microsoft.

Ou encore d'utiliser des appareils physiques tels qu'une carte électronique ou un appareil FIDO2 (on va en reparler plus tard) à la place ou en plus du mot de passe.

Identification à plusieurs facteurs

Une première manière de faire de l'authentification à plusieurs facteurs est d'utiliser un système de code à usage unique tel que le TOTP. À savoir que certains de ces mécanismes peuvent aussi être utilisés seul, pas seulement en conjonction avec un mot de passe (par exemple une clé FIDO2 pour authentification SSH), on va en reparler dans l'identification password-less.

L'avantage d'utiliser l'identification à plusieurs facteurs est de palier la faiblesse du système de mot de passe en demandant à un autre système de générer un code ou de résoudre un problème (par exemple, envois de SMS, TOTP, hash chain, clé FIDO2, etc).

Hash Chain

Une autre manière de faire est d'utiliser à répétition la fonction de hashage.

Par exemple, on génère une valeur aléatoire de départ (le seed).

Ensuite, on exécute la fonction de hashage un certain nombre de fois (par exemple 1000 fois) sur ce seed, ce qui donne donc un hash de hash de hash de hash de … du seed. Le serveur fait de même et sauvegarde ce hash.

Lors de la première authentification, l'utilisateur·ice génère un code à usage unique en dérivant une fois de mois qu'avant le seed. Dans cet exemple, l'utilisateur va donc hasher le seed 999 fois. Le serveur peut alors vérifier que lorsqu'il hash le seed, il obtient bien le hash de départ, l'authentification est donc réussie. Le serveur défini alors ce nouveau hash (de 999 fois) à la place de l'ancien (de 1000 fois).

Ainsi le processus se répète jusqu'à arriver à zéro où dans quel cas il faut générer un nouveau seed.

TOTP (Time-based One Time Password)

> Cette section est en bonus et n'est pas présente dans le cours

En somme, l'utilisateur·ice a un appareil (smartphone, digipass ou autre), qui partage un secret avec le serveur (token) ainsi que le temps (le temps doit être le même que sur le serveur).

Ainsi, après que l'utilisateur·ice a entré son mot de passe, on lui demande le code à usage unique. L'utilisateur·ice peut ensuite demander à l'appareil de générer ce code.

Ce code est généré en hashant le secret partagé (token) et le temps actuel. Ainsi, le mot de passe n'est valable que pour une certaine durée de temps.

En somme, en plus d'ajouter son mot de passe, l'utilisateur·ice va également

FIDO2

> Cette section est en bonus et n'est pas présente dans le cours. Vous pouvez en apprendre plus sur FIDO2 en regardant cette vidéo.

Lorsque l'on veut s'enregistrer sur un site internet, la clé FIDO2 va créer une paire de clé (clé privée et clé publique, on va y revenir plus tard).

Une fois la paire de clé générée, la clé publique est envoyée au site avec un "handle", cet handle contient des informations uniques sur le site qui a demandé l'authentification.

Ainsi, lorsque l'on souhaite se connecter, le site va envoyer un message aléatoire à signer à la clé, la clé va ensuite vérifier que le handle correspond bien (que c'est le bon site, donc protection contre le fishing, le bon compte, sur la bonne clé).

Si tout correspond, la clé va signer le challenge envoyé par le site. Le site pourra ensuite vérifier que la signature est correcte avec la clé publique.

Ce mécanisme est donc très simple (uniquement un bouton) et très sécurisé (protection contre le fishing, aucune information personnelle à stocker).

Identification password-less

L'identification password-less repose sur l'idée que les mots de passes sont compliqués à gérer, assez faible et que l'authentification à plusieurs facteurs peut être compliquée.

L'identification password-less repose donc sur d'autres principes tels que la biométrie (empreinte digitale, reconnaissance faciale, etc), sur un code unique (par SMS ou application tierce, voir TOTP plus tôt), par lien magique (envois d'un lien de connexion par mail), par notification push, ou par clé FIDO2 (voir plus tôt) par exemple.

>

> Si vous voulez en savoir plus sur l'authentification password-less, vous pouvez regarder cette vidéo.

Identification biométrique

L'identification biométrique a pour but d'authentifier sur base d'une propriété de l'utilisateur·ice (sa tête, ses doigts, ou autre).

Le scanner converti et numérise la propriété, la donnée numérisée est alors traitée comme un mot de passe et est hashée et salée.

Le problème de cette méthode est que la protection de ces données devient vraiment très critique, contrairement aux mots de passes, on ne peut pas changer sa tête ou ses empreintes digitales.

De plus, ces informations sont aussi des informations très confidentielles qui peuvent aussi avoir un impact politique important.

Revision #1
Created 6 January 2024 11:44:00 by SnowCode
Updated 6 January 2024 18:13:15 by SnowCode