Authentification par mot de passe

Un FA toujours bien portant

- La majorité des services web utilisent toujours les mots de passe
  - en général comme premier (et souvent unique) facteur d’authentification
- Ce ne serait pas une mauvaise chose en soi si :
  - les mots de passe étaient bien choisis
  - les mots de passe étaient bien protégés côté propriétaire
  - les mots de passe étaient bien protégés côté serveur
- En pratique, c’est trop rarement le cas

Comment ne pas stocker les mots de passe

CNPSLMDP 1: en clair

- Stockage des mots de passe *tel quel* en BDD
- Compromission de la BDD ⇒ compromission immédiate de tous les comptes
- Évident ?
  - pourtant, certaines organisations utilisent cette politique

CNPSLMDP 2 : chiffrés

Stockage des mots de passe *chiffrés* en BDD

- en général par cryptographie symétrique (AES…)

Le problème ? C'est celui du chiffrement symétrique en général :

- **la protection de la clé**
- compromission de la clé ⇒ compromission immédiate de tous les comptes

Autre problème : deux MDP chiffrés identiques ⇒ les deux MDP en clair également identiques (*divulgation d'indices*)

CNPSLMDP 3 : hachés simplement

Stockage des mots de passe *hachés* en BDD avec un algorithme usuel reconnu, comme SHA-2

Le hachage étant un procédé *one-way*, on ne peut pas retrouver le MDP uniquement à partir du condensat (chiffrement : *2-way*)

Ainsi, plus de problème de clé, **mais** :

- toujours le problème : condensats identiques ⇒ MDP identiques
- peu efficace contre la force brute
- inefficace contre *rainbow tables*

Attaques de mots de passe hachés

Force brute pure

Attaqu e par force brute

Principe : tentative systématique de toutes les possibilités

- trouvera tous les mots de passe courts car teste toutes les combinaisons

Contre-mesures basiques :

- pour les portails, accès bloqué après X tentatives infructueuses
- temps de cassage augmente exponentiellement avec la taille du mot de passe

Attaques de mots de passe hachés

Dictionnaires

Attaque par dictionnaire

Principe : on dispose d'un grand ensemble de mots qui ont des chances d'être utilisés comme MDP : c'est le **dictionnaire**

- on teste tous les mots du dictionnaire
- reste de la force brute, mais plus « intelligente »

Très efficace :

- mots que les gens utilisent car faciles à retenir
- souvent issus de bases de MDP réels qui ont fuités
- rend les MDP longs non aléatoires plus fragiles

Attaques de mots de passe hachés

Dictionnaire + force brute pure

Attaques hybrides

- On va combiner dictionnaire et techniques de force brute pure
- À partir d’un mot du dictionnaire, on construit des variations
- ex. : dictionnaire contient *jeanmichel* - on tentera :
  - *Jeanmichel*
  - *JEANMICHEL*
  - *jeanmichel123*
  - *Jeanmichel!*
  - *J34NM1CH31* (*leetspeak*)

Attaques de mots de passe hachés

Rainbow Tables

Tables Arc-en-ciel

Liste précalculée de mots de passe hachés

- réduit **grandement** le temps de cassage (on doit juste comparer les condensats)
- mais augmente **drastiquement** la taille de stockage nécessaire (Tb)
- d’autant qu’il faut une *rainbow table* pour chaque type de hachage

Résoudre le problème de la force brute

Hachage et rapidité

Performance des fonctions de hachage

En général, on veut que le hachage soit rapide (ex. : SHA-2)

- c’est un des critères pour construire un bon algorithme de hachage
- on veut un minimum d’impact sur les performances de l’application ou du serveur

Paradoxalement, **pour les mots de passe, on veut que le hachage soit lent**

- objectif : ralentir les attaques par force brute

Hachage lent - 2 solutions

Utiliser un algorithme de hachage classique rapide, mais avec un **nombre d'itérations de hachage important**

Utiliser un **algorithme de hachage lent** dédié

- KDF : fonction de dérivation de clé (*Key Derivation Function*)
- ex. : scrypt < bcrypt < Argon2
- en plus du nombre d'itérations, permettent de jouer sur la mémoire nécessaire et la parallélisation

Résoudre les problèmes de *hash* identiques et les attaques par *rainbow tables*

Salage de mots de passe

Salage

Le *sel cryptographique* est une chaîne de caractères **aléatoire** ajoutée au MDP avant de le hacher

- stocké en BDD (en général dans la même table)
- unique par utilisateur
- pas spécialement secret, peut être stocké en clair

Salage - Implémentation

On concatène "*SEL+MDP*" avant de hacher le tout

L'enregistrement BDD de chaque compte contient :

- le sel unique pour ce compte
- le condensat de la chaîne concaténée "*SEL+MDP*"

Quand l'utilisateur s'authentifie :

- on reconstitue "*SEL+MDP_proposé*" en prenant le sel stocké en BDD pour cet identifiant et on hache
- on compare avec le "*SEL+MDP*" haché stocké en BDD

Salage - Bénéfices

Deux MDP identiques ne produisent plus le même condensat

- plus d’indices pour l’attaquant dans la table

Rainbow tables rendues inefficaces, sauf à produire une *rainbow table* pour chaque sel possible

- impraticable si la taille du sel est suffisante

Salage - Mauvaises pratiques

Ré-utiliser le même sel pour plusieurs comptes

Taille de sel trop courte : *rainbow tables* redeviennent envisageables

Utiliser un sel qui n’est pas aléatoire

- par exemple, l’identifiant ou partie de l’identifiant
- car alors, l’attaquant peut précalculer les *rainbow tables* pour cet identifiant, qui risque d’être réutilisé sur plusieurs services…

Le salage seul ne protège pas des attaques par force brute, il doit être combiné avec un hachage lent.

La technique suivante offre un niveau de protection supplémentaire.

Poivrage

Le *poivre cryptographique* est une chaîne de caractères aléatoire ajoutée au MDP avant de le hacher… Comme le salage, alors ? Non.

- **Sel** : 1 pour chaque utilisateur, stocké en BDD avec "*SEL+MDP*" haché
- **Poivre** : 1 pour toute la BDD, ***stocké ailleurs*** : contrairement au sel, le poivre doit resté inconnu de l’attaquant

Poivrage - Implémentation

Le mécanisme est exactement le même que pour le salage :

- on concatène "*POIVRE+SEL+MDP_proposé*" avant de hacher le tout
- on compare avec "*POIVRE+SEL+MDP*" haché stocké en BDD

Notez que le poivrage seul ne protège pas de la « divulgation d'indices » à l'attaquant car le poivre est commun à toute la table

Poivrage - Bénéfice

L’idée est que, même si l’attaquant parvient à récupérer la BDD, il ne peut pas craquer les MDP même par force brute

- contrairement au sel, connu éventuellement de l’attaquant (le secret n’est pas requis pour son efficacité), le poivre est inconnu de l’attaquant
- il ne peut donc pas calculer le condensat de "*POIVRE+SEL+MDP_tenté*" car **il lui manque une donnée**

Exemple de *hash* avec brypt

`$2y$15$J4muMQBqxa4c84YqM7/TcOJQqcH9OVM9BB7I1/o40o4uBcN…`

- `2y` : indique que c'est un *hash* de type *bcrypt*
- `15` : indique le *coût* de calcul (plus le coût est élevé, plus le *hash* est lent)
- 22 caractères pour le sel
- 31 caractères pour le condensat lui-même

Conclusion (1)

À ce jour, en combinant les techniques suivantes, on parvient à avoir des mots de passe relativement sécurisée :

- imposer des mots de passe forts (**TAILLE** + complexité, idéalement aléatoire)
- bloquer les tentatives de connexion après un certain nombre d'échecs au niveau du portail (IPs + compte)
- appliquer les **mêmes restrictions sur la récupération de MDP**

Conclusion (2)

Pour les attaques sur tables fuitées :

- hachage lent (contre *force brute*)
- salage (contre *divulgation d'indices et rainbow tables*)
- poivrage (contre *force brute*)

L'authentification par MDP reste à combiner à d'autres FA pour une protection réellement efficace, car la gestion du MDP par l'utilisateur reste un point faible évident, malgré toute la sensibilisation que l'on peut (doit) faire