Comment (ne pas) stocker les mots de passe de vos utilisateurs

Mathieu Pillard (@dioxmat) – Sudweb 2011

Enjeux

- Hypothèses :
  - L'attaquant a eu accès à la base de données, voire à tout le système
  - Nos utilisateurs ré-utilisent les mêmes mots de passe partout
  - On souhaite les protéger malgré tout
- Conclusion :
  - Il faut ralentir l'attaquant un maximum, l'empêcher de déchiffrer nos mots de passe

Ce qu'il ne faut pas faire (1) : stockage en clair

- Méthode la plus simple et la moins sécurisée :
  - Quiquonque avec un accès à la base de données peut voir les mots de passe...
  - ...Y compris vos développeurs et administrateurs !

Ce qu'il ne faut pas faire (2) : stockage chiffré

- Sécurité toute relative, puisqu'il est possible de retrouver la version en clair:
  - Vos administrateurs ou développeurs peuvent toujours déchiffrer les mots de passe...
  - ... et un attaquant qui a eu accès au système et/ou à la base peut trouver la clé facilement aussi !

Ce qu'il ne faut pas faire (3) : stockage d'un hash simple

- Illusion de sécurité – les algos (MD5, SHA1, etc) sont faits pour calculer des empreintes très rapidement, pas pour stocker des mots de passe!
- Vulnérable aux attaques *brute-force* : Un PC de base calcule 100.000+ hashes SHA-1 par seconde et par coeur!
- Vulnérable aux attaques par *rainbow tables* : gigantesques tables de correspondance mot de passe – hash
- Le matériel a disposition de tout le monde et donc de vos attaquants est de plus en plus rapide

Ce qu'il ne faut pas faire (4) : stockage d'un hash avec salt

- Principe : un ~~petit~~ gros bout aléatoire mais connu (car stocké) qu'on rajoute au mot de passe lors de la génération du hash :
  - hashed_password = hash(salt + password)
- C'est déjà (beaucoup) mieux...
- ... Mais:
  - Vous êtes en train de ré-inventer la roue
  - Ça reste vulnérable aux attaques de brute-force !
- On peut faire mieux...

Ce qu'il faut faire

- Garder le principe du salt aléatoire (un par mot de passe, stocké en base avec le password hashé)
- Utiliser des implémentations spécifiques dont le but est de *ralentir* la génération du hash: bcrypt, PBKDF2, SHA-256/512 Crypt.
  - Volontairement lents: 500 ms ou carrément 1 seconde c'est négligeable pour vous, c'est horrible pour l'attaquant!
  - Ajustables pour pallier à la montée en puissance du matériel (iteration count)
  - Utilisés actuellement par différents systèmes d'exploitation
  - Pas mal d'implémentations existantes dans tous les langages de programmation
- Le raffinement ultime: scrypt
  - Lent **et** couteux en mémoire
  - Peu d'implémentations / tests pour le moment (date de 2009)

Bonus

- N'oubliez pas d'éduquer vos utilisateurs
- Incitez les à utiliser des mots de passe longs, avec caractères “spéciaux”, majuscules, chiffres etc.
- Ne cédez pas aux sirènes du “mais c'est plus facile pour les utilisateurs si c'est en clair”, ca finira par se retourner contre vous!
- Faites du HTTPS!

Liens

- How To Safely Store A Password
  http://codahale.com/how-to-safely-store-a-password/
- SHA-512 w/ per Users Salts (in)security
  http://blog.mozilla.com/webappsec/category/passwords-2/
- Advanced Password Recovery
  http://hashcat.net/oclhashcat-plus/
- PasswordHash2 PHP Password Hashing class
  http://t.co/T8sfsy5
- Python bcrypt
  http://pypi.python.org/pypi/py-bcrypt/
- Ruby bcrypt
  https://github.com/codahale/bcrypt-ruby
- Python Password Hashing Schemes (n'est pas l'implémentation de référence, surtout utile pour le coté documentation & historique des techniques de stockages de password)
  http://packages.python.org/passlib/lib/passlib.hash.html