PGON.CA

Connexion sécuritaire sans HTTPS

Simon Levesque

2011-07-16

Contenu

Introduction

Lorsque nous désirons nous connecter sur un site avec notre nom d'utilisateur et mot de passe, régulièrement, nous serons redirigés vers le site en HTTPS pour le temps d'entrer nos données sensibles. Ensuite nous recevrons un cookie de session qui sera valide en HTTP. Si l'hébergeur web ne possède pas d'HTTPS et qu'il faut se connecter, les données des utilisateurs sont envoyées en clair sur le réseau et une personne malveillante pourrait s'en emparer. C'est ce qui se passe avec Joomla!, WordPress et autres qui sont installés sur du HTTP.

Avant de continuer, cet article est pour les programmeurs et non pour les utilisateurs. Si votre programme comme WordPress est sur du HTTP, vous ne pouvez pas vous connecter sécuritairement. Si par contre vous développez une application web, vous pouvez améliorer votre méthode de connexion avec cette technique.

Pour résumer la technique, il suffit de faire un hash du mot de passe salé avant de l'envoyer sur le réseau.

L'authentication

Le principe

Premièrement, ce que vous voulez cacher n'est pas le nom d'utilisateur, mais le mot de passe. Une bonne façon de le faire, c'est en **utilisant un hash cryptographique** tel SHA-512[1]. Un hash cryptographique permet de créer une série de caractères unique, mais impossible de retrouver le mot initial. C'est justement ce qui est utilisé pour enregistrer les mots de passes dans une base de données pour ne pas que les administrateurs puissent connaitre votre mot de passe. Je vais utiliser la notation *H(motdepasse)* pour parler du mot de passe hashé, de *c.motdepasse* pour le mot de passe du client et *s.motdepasse* pour le serveur. Puisque le serveur a déjà un hash du mot de passe, tout ce que le serveur connait c'est *s.motdepasse = H(c.motdepasse).*

Deuxièmement, si on envoi sur le réseau toujours *H(c.motdepasse)*, alors il est très simple pour un serveur de regarder si c'est égal à sa valeur de mot de passe. Par contre, cette valeur sera toujours la même que ce soit cette connexion ou la 5ième. Pour quelqu'un qui espionne le réseau, que ce soit le mot de passe en clair ou le hash de celui-ci en clair, il va simplement le transférer au serveur et obtenir l'autorisation. Il faut donc le **saler avant de l'envoyer**. Saler un hash, c'est dire *H(c.motdepasse+sel)*. Ici, puisque le serveur ne connait pas le mot de passe, mais uniquement le hash de celui-ci, il faut hasher 2 fois pour ajouter le sel: *H(sel+H(c.motdepasse))* pour le client et *H(sel+s.motdepasse)* pour le serveur. La valeur du sel est choisie par le serveur, comme cela un attaquant n'aurait pas le choix de cette valeur et il ne pourrait pas créer la valeur finale sans connaitre le mot de passe. Le sel étant changeant à chaque demande de connexion, la valeur de mot de passe sur le réseau sera toujours changeante.

[1] http://fr.wikipedia.org/wiki/Sha-512

La procédure complète

Voici comment implémenter cette protection :

- Le serveur crée un *sel* qu'il va garder en mémoire pour cette session
- Le serveur envoie le *sel* au client en même temps que son formulaire de connexion
- Le client envoie au serveur le *nom d'utilisateur* et *H(sel+H(c.motdepasse))*
- Le serveur vérifie les données
    - Il obtient de sa base de données le *s.motdepasse* qui est *H(c.motdepasse)*
    - Il vérifie que ce que le client vient de lui envoyer est égal à *H(sel+s.motdepasse)*
    - Si c'est le cas, il donne l'autorisation à cet utilisateur pour sa session

La faille

Pour l'autorisation, cette technique est parfaite, mais le problème est lors de la création du compte utilisateur. Puisqu'il n'est pas possible de décoder un hash, il n'est pas possible de cacher un mot de passe à utiliser. **Cette technique fonctionne uniquement lorsque le client et le serveur connaissent une phrase secrète au préalable**. Pour la création du compte, il faudrait soit utiliser HTTPS ou faire confiance en son réseau. Lorsque nous sommes chez soi, le réseau est normalement de confiance, mais lorsque nous sommes dans un endroit public, mieux vaut ne pas créer de comptes sur HTTP. Par contre, dans un endroit public, mieux vaut avoir cette technique de connexion que d'envoyer les mots de passes en clair sur ce réseau!

Les outils pour faire du SHA-512

Tout dépendant du langage de programmation utilisé, des fonctionnalités internes ou des librairies peuvent créer le hash. Voici quelques-unes :

- Javascript: jsSHA[2]
- PHP: $out = hash('sha512', $in);[3]
- Java: MessageDigest.getInstance("SHA-512");[4]
- DotNet: SHA512Managed[5]

[2] http://jssha.sourceforge.net
[3] http://fr.php.net/manual/fr/function.hash.php
[4] http://download.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html
[5] http://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512managed.aspx

La session sécurisée

Le problème

Avec l'id de la session qui est envoyé en clair, une personne tierce pourrait utiliser cet id et se faire passer pour vous sans même avoir à s'authentifier. C'est ce qui se passe quand les gens volent des cookies avec l'id d'une session d'un site web. Avec cette information, une personne pourrait aisément donner des ordres à un serveur de la part d'une autre personne et ce n'est pas souhaitable.

Les éléments connus

À cette étape-ci, voici ce que toutes les parties connaissaient:

- Client
  - Le *mot de passe* (et par conséquent *H(c.motdepasse)*)
  - L'id de la *session*
- Serveur
  - Le *s.motdepasse* qui est égal à *H(c.motdepasse)*
  - L'id de la *session*
- Tierce personne
  - L'id de la *session*

Les principes sortis de l'authentication

1. Le client et le serveur doivent partager un secret connu seulement entre-eux (le hash du mot de passe)
2. Toutes les requêtes doivent avoir une partie dynamique pour éviter que la tierce personne puisse rejouer une requête

La solution

Il faut encore utiliser un sel différent pour chaque requête, mais cette fois-ci, au lieu d'avoir le serveur qui en dit un, ce sera le client. Le sel sera tout simplement un numéro de séquence et le serveur doit garder en mémoire le dernier numéro pour cette session et ne permettre que des numéros supérieurs. Ainsi, il n'est pas possible pour la tierce personne de créer une requête ou de répéter une requête.

Si la personne tierce peut modifier une requête parce qu'elle se trouve entre les deux parties, le hash peut aussi inclure les champs importants pour ainsi éviter leur altération.

La procédure complète

- Le client choisi un numéro de *séquence* supérieur à l'ancien utilisé
- Le client envoie au serveur les *champs* de sa requête, l'id de la *session*, le numéro de *séquence* utilisé et *H(H(c.motdepasse) + sequence + session + champ1 + champ2 + ...)*
- Le serveur vérifie les données
    - Il obtient de sa base de données le *s.motdepasse* qui est *H(c.motdepasse)* associé à la *session*
    - Il vérifie que ce que le client vient de lui envoyer est égal à *H(s.motdepasse + sequence + session + champ1 + champ2 + ...)*

Limitation

Comme mentionné, toutes les informations restent transmises en clair sur le réseau. **S'il y a des informations confidentielles, elles ne le seront plus**. Cette technique est donc bonne pour donner des ordres qui peuvent être vus publiquement. Par exemple, pour un système de blogue, les articles vont finir par être connus du public, mais ce n'est pas tout le monde qui espionne qui pourra créer des articles.