SLO

Laboratoire 5

Analyse des menaces

**Equipe**

Thibaut Loiseau

Romain Mercier

Ioannis Noukakis

Michela Zucca

Documents annexes :

* Jar client
* Jar serveur

Lien Github : <https://github.com/ioannisNoukakis/GEN-projet>

Version du : 21.06.2016

HEIG-VD 2015/2016, SLO.

Table des matières

[1. Introduction 3](#_Toc454297347)

[1.1 Descriptif de l’application 3](#_Toc454297348)

[2. Analyse des menaces 3](#_Toc454297349)

[3. Communication sécurisée Client – Serveur 4](#_Toc454297350)

[3.1 Capture wireshark 4](#_Toc454297351)

[3.2 Chiffrement 4](#_Toc454297352)

[3.3 Implémentation 5](#_Toc454297353)

[4. Sécurité de l’interface administrateur - PHP 6](#_Toc454297354)

[4.1 Déplacement des fonctions PHP dans le même fichier 6](#_Toc454297355)

[4.2 Hachage/salage du mot de passe 6](#_Toc454297356)

[4.3 Utilisation de « prepared statements » PDO pour toutes les requêtes 6](#_Toc454297357)

[4.4 Vérification des entrées dans la base de données 6](#_Toc454297358)

[4.5 Encodage des valeurs 7](#_Toc454297359)

[4.6 Page de connexion séparée 7](#_Toc454297360)

[4.7 Implémentation 8](#_Toc454297361)

[4.7.1 Connection à l’interface administrateur 8](#_Toc454297362)

[4.7.2 Vérification des données avant l’insertion dans la BDD 8](#_Toc454297363)

[4.7.3 Message d’erreur en cas de problème lors de l’insertion 8](#_Toc454297364)

[5. Conclusion 9](#_Toc454297365)

# 1. Introduction

Pour ce laboratoire, nous avons décidé de sécuriser notre application réalisée dans le cadre du cours de Génie Logiciel.

Notre projet est une application client – serveur faisant appel à une base de données.

## 1.1 Descriptif de l’application

Notre application est un jeu en tour par tour, permettant à deux joueurs de s’affronter. Chaque joueur a accès à une liste de personnage prédéfinis par nos soins et enregistrés dans une base de données.

L’accès à la base donnée ne s’effectue que du côté serveur. Un compte administrateur peut se connecter à la base de données et créer de nouveaux personnages ou sortir des statistiques des combats. L’accès administrateur se fait au travers d’une interface web par le biais d’un serveur apache.

Lorsqu’un client veut combattre, il doit s’identifier auprès du serveur pour accéder à son compte, ou cas échéant le créer. Une fois connecté, il peut choisir un personnage et lancer un combat.

# 2. Analyse des menaces

Afin de mieux cerner les différentes menaces de notre programme, nous allons d’abord énumérer les différentes actions recherchées par un potentiel attaquant, puis nous mettre dans la peau de l’attaquant et tenter d’anticiper ses attaques en protégeant notre système.

Que chercherait un attaquant :

* Gagner à coup sûr en prenant le contrôle de son adversaire.
* Bannir de joueurs
* Mettre hors service le serveur (DDOS)
* (À compléter ou à supprimer…)

Comment pourrait-il s’y prendre :

* Brute force sur les mots de passe
* Attaque du type « Man in the middle »
* Surcharge du serveur à l’aide de multiples clients
* (À compléter ou à supprimer…)

Nous avons décidé de nous concentrer sur un chiffrement des communications de notre programme.

Nous traiterons ainsi les 2 cas suivants :

* Communication Client - Serveur
  + Echange chiffré entre le serveur et le client
* Sécurité de l’interface administrateur
  + Echange chiffré entre le serveur Apache et l’administrateur

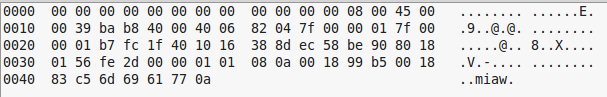
Nous avions également pensé à obfusquer notre code source. Comme nous avions vu pendant le cours SLO, l’obfuscation peut décourager l’attaquant et de ce simple fait rajouter une sécurité à nos produits.

Après quelques tentatives difficiles, et une discussion avec vous, nous sommes arrivés à la conclusion que cette démarche était inutile en Java et dès lors nous avons renoncé à nous attarder là-dessus.

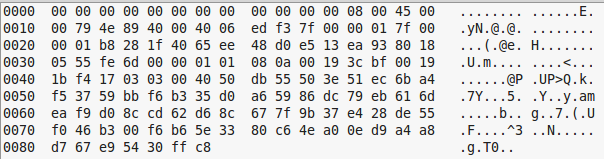
# 3. Communication sécurisée Client – Serveur

## 3.1 Capture wireshark

Avant de sécuriser la communication Client – Serveur, il était possible d’intercepter les paquets et d’accéder au contenu du message (ici avec Wireshark).

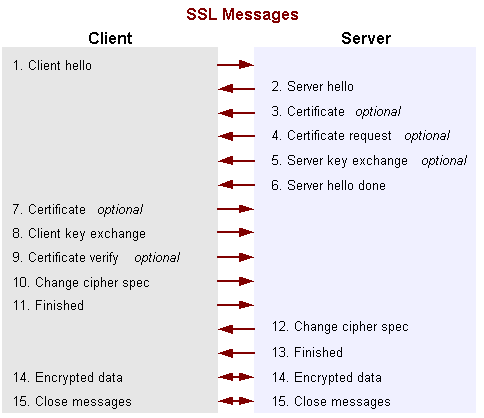


Désormais lors d’une interception du paquet le message est bien chiffré.



## 3.2 Chiffrement

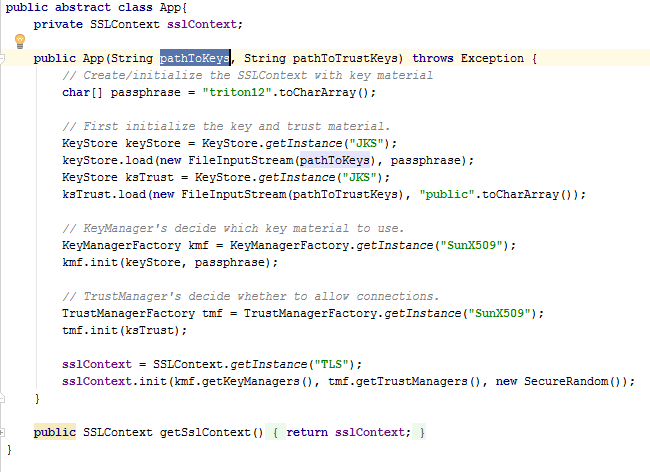
Ce chiffrement s’est effectué à l’aide de la JSSE (Java Secure Socket Extension) qui s’appuie sur la technologie SSL (Secure Socket Layer). Nous avons utilisé le protocole TLS (Transport Layer Security). Pour authentifier les deux parties nous utilisons des certificats auto signés. Ils permettent, par exemple, à un client de s’assurer qu’il se connecte à un serveur légitime. Toutefois, pour un projet réel, remis à un client réel, un certificat doit être signé par une autorité agréée.



La JSSE fonctionne ainsi :

* Le client et le serveur négocient quelle suite cryptographique ils vont utiliser.
* Le serveur et le client échangent un secret partagé pour un futur échange symétrique par un procédé asymétrique.
* Le client et le serveur échangent des données chiffrées par un secret partagé selon un procédé symétrique.

## 3.3 Implémentation



L’illustration ci-dessus démontre la création d’un contexte TLS. Ce dernier sera utilisé afin de créer des SSLServerSocket ou des SSLSocket.

Voici ce qui se passe.

1. Nous lisons notre clé privée qui sera celle de l’application que nous contrôlons.
2. Nous lisons la clé publique de l’application distante.
3. Nous créons un contexte SSL avec ces deux clés.

Une fois ces opérations effectuées, nous pouvons instancier des SSLSocket et des SSLServerSocket issus du contexte SSL crée précédemment.

Ces derniers se chargeront alors, lors du premier envoi de données, de négocier un secret partagé pour par la suite effectuer des échanges de données chiffré.

# 4. Sécurité de l’interface administrateur - PHP

L'interface administrateur accédant elle aussi directement à la base de données et possédant un accès authentifié, il est donc normal qu'elle soit elle aussi sécurisée.

Voici les mesures sécuritaires qui ont été prises :

## 4.1 Déplacement des fonctions PHP dans le même fichier

Ce n'est pas une mesure sécuritaire à proprement parler, mais cela permet de simplifier l'affichage de messages d'erreur pour l'utilisateur, car une fois le formulaire traité, le code a directement accès au code de la page et peut dont directement afficher le bon message.

## 4.2 Hachage/salage du mot de passe

Sans doute l'une des plus grosses failles de l’interface administrateur : le mot de passe était traité et stocké en clair, et n'était pas salé.

Pour remédier à cela, un hash a été appliqué (SHA256) sur le mot de passe salé au préalable selon les recommandations trouvées sur le tutoriel suivant :

<https://crackstation.net/hashing-security.htm#properhashing>

* Le sel utilisé est stocké dans la base de données
* Lors de la connexion, on obtient le sel correspondant à l'utilisateur choisi (si existant), puis on l'ajoute au mot de passe entré par l’utilisateur
* Puis on hash le tout, et on le compare avec le mot de passe existant dans la base

## 4.3 Utilisation de « prepared statements » PDO pour toutes les requêtes

Cette précaution avait déjà été effectuée auparavant, mais il nous a semblé nécessaire de vérifier que toutes les requêtes étaient bien implémentées de cette manière.

Ce mécanisme a pour but d’éviter les injections sql en interprétant les entrées utilisateurs comme de simple chaine de caractères. Ainsi quel que soient les commandes entrées elles ne pourront jamais être exécutées côté serveur.

## 4.4 Vérification des entrées dans la base de données

Toutes les valeurs reçues via le formulaire sont vérifiées avant d'être insérées dans la base de données. Bien que certains navigateurs effectuent déjà certaines vérifications, typiquement les champs "required" sont vérifiés automatiquement par Google Chrome, tout comme la longueur maximum du texte inséré dans un champ texte HTML.

Ces précautions peuvent être contournées facilement en éditant le code HTML, via Firebug ou autre, ou en utilisant un navigateur n’offrant pas la vérification automatique, tel que K-Meleon.

Pour se protéger de ce type d’attaque, il est nécessaire de vérifier toutes les valeurs des formulaires avant son insertion dans le code PHP de la page.

* Vérification de l’authenticité du formulaire (variables)
* La taille de la saisie (0 < nb caractères < max caractères)
* Le format de la saisie

## 4.5 Encodage des valeurs

Lors d’une saisie d’utilisateurs on ne peut pas se fier à son contenu, un utilisateur n’est pas une personne avec de bonnes intentions. Avec « prepared statements », cité précédemment, nous nous sommes protégés contre les injections sql. Toutefois, il est nécessaire de se protéger contre les injections de script, tel qu’un script JavaScript.

Pour remédier à ce problème, toutes les valeurs textuelles insérées dans la bases de données sont encodées à l’aide de la fonction *htmlentities()*. Celle-ci, encode les caractères spéciaux au format HTML (*&amp;*, *&eacute;*...), dans un premier temps créé pour éviter des erreurs d’encodage, par exemple compatibilité ISO et UTF-8. Elle permet aussi d’éviter l’insertion d’un script. Les caractères « < » et « > » sont transformés en *&lt;* et *&gt;*, les balises <script> sont donc affichables, mais pas interprétables.

## 4.6 Page de connexion séparée

Pour simplifier la connexion, lorsque l’administrateur se connecte, il est redirigé immédiatement vers la page d’administration.

Les accès sont contrôlés par une variable de session « username ».

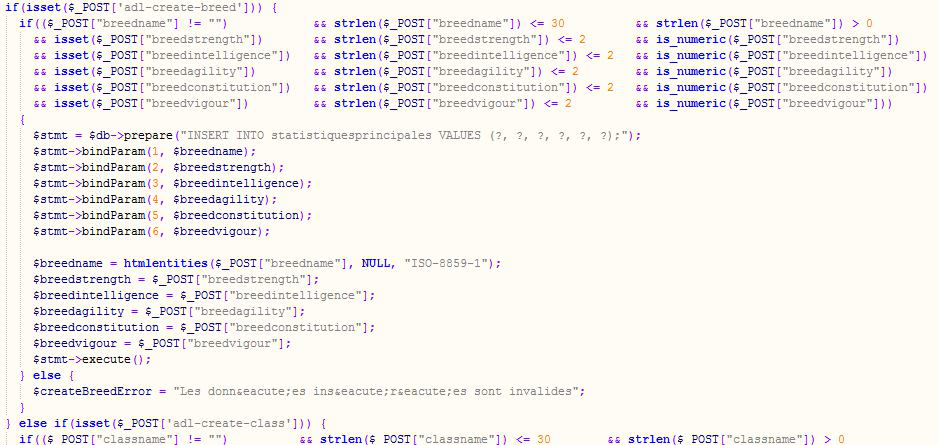
* L’accès à la page de connexion n’est pas possible si la variable n’existe pas
* L’accès à la page d’administration est possible que si la variable existe et non vide.

## 4.7 Implémentation

### 4.7.1 Connection à l’interface administrateur



### 4.7.2 Vérification des données avant l’insertion dans la BDD



### 4.7.3 Message d’erreur en cas de problème lors de l’insertion



# 5. Conclusion

Notre application bénéficie désormais d’une meilleure sécurité au niveau des communications entre le serveur et le client, l’administrateur et la base de données. Nous aurions souhaité pouvoir protéger notre application contre les attaques de types déni de services. Celle-ci restant très probable contre notre produit.

Nous fournissons un service de jeu à notre clientèle. Si nous nous mettons dans le rôle et responsabilités d’une entreprise, ce genre d’attaque serait fortement préjudiciable. Nous pourrions perdre notre clientèle, ce qui pourrait entrainer la faillite de ce produit.

Ce dernier ne pourrait pas s’épanouir et offrir d’autre service, tel que des achats de personnages, d’extension ou autres. Le travail fournit sur ce logiciel serait ainsi perdu. Le prix consacré une perte financière pour l’entreprise.

D’autres mécanismes pourraient être envisagés comme par exemple une identification à l’aide d’un ID généré aléatoirement (système de PostFinance). Notre application restant très basic sur les services offerts, il serait inutile de la protéger avec de tels moyens.