|  |
| --- |
| Mickaël BRENOIT | Michaël FUCHS  14/09/2017 |

|  |
| --- |
| UNIVERSITE DE HAUTE-ALSACE |
| Sécurisation bases de données |
| Mini-projet MIAGE M2 |



SOMMAIRE

[1. Présentation du mini-projet 3](#_Toc493230572)

[2. Création de la base de données 4](#_Toc493230573)

[2.1. Table « users » 4](#_Toc493230574)

[2.2. Table « accounts » 4](#_Toc493230575)

[2.3. Script SQL 4](#_Toc493230576)

[3. Les différentes failles traitées 5](#_Toc493230577)

[3.1. Les injections SQL 5](#_Toc493230578)

[3.1.1. Définition et comment les utiliser 5](#_Toc493230579)

[3.1.2. Comment se protéger 6](#_Toc493230580)

[3.1.3. Notre solution 6](#_Toc493230581)

[3.2. Cross-site scripting (XSS) 7](#_Toc493230582)

[3.2.1. Définition 7](#_Toc493230583)

[3.2.2. Comment éviter ce genre d’attaque 7](#_Toc493230584)

[3.2.3. Notre solution 8](#_Toc493230585)

[3.3. L’attaque dictionnaire et force brute 9](#_Toc493230586)

[3.3.1. Définition et comment s’en protéger 9](#_Toc493230587)

[3.3.2. Notre solution 9](#_Toc493230588)

[3.4. La faille « include » ou « remote file inclusion » 10](#_Toc493230589)

[3.4.1. Définition 10](#_Toc493230590)

[3.4.2. Comment éviter cette faille 10](#_Toc493230591)

[3.4.3. Notre solution 10](#_Toc493230592)

[3.5. La faille « input file … » 11](#_Toc493230593)

[3.5.1. Définition 11](#_Toc493230594)

[3.5.2. Les différentes protections 11](#_Toc493230595)

[3.5.3. Notre solution 12](#_Toc493230596)

[4. Les autres failles non traitées 13](#_Toc493230597)

[4.1. Authentification et gestion de session 13](#_Toc493230598)

[4.2. Une mauvaise configuration du serveur 14](#_Toc493230599)

[4.3. Hameçonnage ou « Phishing » 15](#_Toc493230600)

[5. Travail d’équipe 16](#_Toc493230601)

# Présentation du mini-projet

Le mini-projet présenté dans ce rapport est le fruit de connaissances récoltées lors de nos recherches internet sur les différentes failles possibles sur un site internet. Il nous a été demandé de :

1. Créer une base de données qui contient deux tables :
   1. Une table « users » représentant les utilisateurs de notre application. Les utilisateurs auront un id, un login et un mot de passe.
   2. Une table « accounts » représentant les comptes bancaires que dispose un utilisateur. Les comptes auront un id, un type, un montant et une clé faisant référence à l’id de l’utilisateur à qui appartiennent les comptes.
2. Réaliser un formulaire en PHP qui se connecte à notre base de données et qui demande le login et mot de passe de l’utilisateur.
3. Rechercher les différentes failles importantes qui existent et les tester sur notre formulaire.
   1. Tester les failles « injections SQL » vues en cours
   2. Rechercher d’autres types de faille sur internet
4. Explorer les solutions sur le web et décrire au moins deux méthodes importantes de protection contre ces failles.
5. Illustrer les scénarios avec failles et sans failles.

# Création de la base de données

## Table « users »

La première table créée est « user ». Elle contient :

* Un champ « id » : identifiant unique d’un utilisateur, incrémenté automatiquement et non nul. C’est la clé primaire de cette table.
* Un champ « login » : pseudonyme de l’utilisateur, il est déclaré unique et non null
* Un champ « password » : mot de passe de l’utilisateur haché, non null.

De base, nous aurons deux entrées :

* Id : 1, login : mickael et password : md5(aA4\*7777)
* Id : 2, login : framboise et password : md5(strawberry)

## Table « accounts »

La seconde table créée est « accounts ». Elle contient :

* Un champ « id » : identifiant unique d’un compte, incrémenté automatiquement et non nul. C’est la clé primaire de cette table.
* Un champ « type » : intitulé du compte bancaire, unique et non null
* Un champ « amount » : montant représenté par un entier réel positif, non null
* Un champ « iduser » : identifiant de l’utilisateur qui détient ce compte, non null

De base, nous aurons quatre entrées :

* Id : 1, type : Livre bleu, amount : 1100 et iduser : 1
* Id : 2, type : Compte Courant, amount : 1600 et iduser : 1
* Id : 3, type : Livret Framboise 1, amount : 600 et iduser : 2
* Id : 4, type : Livret Framboise 2, amount : 750 et iduser : 2

## Script SQL

Image

# Les différentes failles traitées

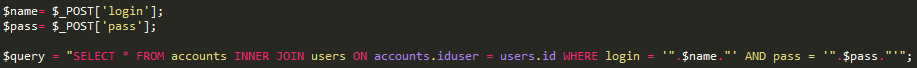
## Les injections SQL

### Définition et comment les utiliser

Les injections SQL (SQLi) se réfère à une attaque d'injection dans laquelle un attaquant peut exécuter des instructions SQL malveillantes qui contrôlent le serveur de base de données d'une application Web (également communément appelé système de gestion de base de données relationnelle - SGBDR). Étant donné qu'une vulnérabilité SQL Injection pourrait affecter n'importe quel site Web ou application Web qui utilise une base de données basée sur SQL, cette vulnérabilité est l'une des vulnérabilités les plus anciennes, les plus répandues et les plus dangereuses.

Pour exécuter des requêtes SQL malveillantes contre un serveur de base de données, un attaquant doit d'abord trouver une entrée dans l'application Web qui permet d’inclure une requête SQL. Un attaquant peut alors insérer une partie de code SQL qui sera incluse dans la requête SQL et s'exécuter en fonction du serveur de la base de données.

Le pseudo-code suivant du côté serveur sert à authentifier les utilisateurs sur l'application Web.



Le script ci-dessus est un exemple simple d'authentification d'un utilisateur avec un nom d'utilisateur et un mot de passe contre une base de données avec une table nommée utilisateurs et une colonne de nom d'utilisateur et de mot de passe. Le script ci-dessus est vulnérable à l'injection SQL, car un attaquant peut envoyer des entrées malveillantes de manière à modifier l'instruction SQL exécutée par le serveur de base de données. Un exemple simple d'une charge utile d'injection SQL pourrait être quelque chose d'aussi simple que le champ de mot de passe en password 'OR 1 = 1**.**

Cela entraînerait que la requête SQL suivante soit exécutée contre le serveur de base de données.

**SELECT id FROM users WHERE username=’username’ AND password=’password’ OR 1=1’**

Dans le cas où le contournement d'authentification est possible, l'application enregistrera probablement l'attaquant avec le premier compte à partir du résultat de la requête - le premier compte dans une base de données est habituellement un utilisateur administratif.

### Comment se protéger

Selon OWASP « Open Web Application Security Project (OWASP) est une communauté en ligne travaillant sur la sécurité des applications Web. Sa philosophie est d'être à la fois libre et ouverte à tous. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web. », la mise en place de trois défenses est obligatoire pour lutter contre les injections SQL.

Voici la liste des trois défenses :

* Utilisation des requêtes préparées
* Utilisation des procédures stockées
* Vérifier toutes les entrées utilisateurs

Les requêtes préparées garantissent qu'un attaquant ne peut pas modifier l'intention d'une requête, même si les commandes SQL sont insérées par un attaquant. Dans l'exemple de sécurité vu précédemment, si un attaquant devait entrer l'ID utilisateur de tom 'ou' 1 '=' 1, la requête paramétrée ne serait pas vulnérable et rechercherait un nom d'utilisateur qui correspondait littéralement à la chaîne entière tom 'ou' 1 '=' 1.

L’utilisation des procédures stockées est très similaire à l’utilisation des requêtes préparées. La seule différence entre les instructions préparées et les procédures stockées est que le code SQL pour une procédure stockée est défini et stocké dans la base de données elle-même, puis appelé à partir de l'application.

Vérifier toutes les entrées utilisateurs consiste à échapper les saisies des utilisateurs avant de les mettre dans une requête SQL. Pour cela on utilise des fonctions spéciales qui vont filtrer les données en prenant soin de virer les balises <elem></elem> ainsi que les apostrophes.

### Notre solution

Blabla

## Cross-site scripting (XSS)

### Définition

Cross-site Scripting, également appelé XSS, est un moyen de contourner le concept SOP. En anglais SOP signifie « standard operating procedure (SOP). D’après Wikipédia, « *SOP est une procédure de sécurité qui décrit comment affronter une menace et comment agir pour en diminuer le risque. Elle décrit les étapes à suivre pour réduire la possibilité qu'un incident se produise et s'il se produit ce qu'il faut faire pour en limiter les conséquences.* »

Chaque fois que le code HTML est généré dynamiquement et que l'entrée de l'utilisateur n'est pas « désinfectée » et qu'elle se reflète sur la page, un attaquant peut insérer son propre code HTML/JavaScript. Dans ce cas, un attaquant peut facilement insérer un code JavaScript. Ce faisant, l'attaquant peut accéder à d'autres pages sur le même domaine et peut lire des données comme les « CSRF-Tokens » ou les « cookies » définis.

Si les « cookies », qui contiennent généralement des informations d'identifiant de session, peuvent être lus par le JavaScript, l'attaquant peut les utiliser sur son propre navigateur et se connecter à l'application Web en tant que victime. Si cela ne fonctionne pas, l'attaquant peut encore lire des informations privées à partir des pages, telles que les jetons « CSRF » et faire des demandes au nom de l'utilisateur. En somme, usurper l’identité d’une personne.

### Comment éviter ce genre d’attaque

Dans certains cas, il pourrait suffire de coder les caractères spéciaux HTML, tels que l'ouverture et la fermeture des balises. Dans d'autres cas, un codage d'URL correctement appliqué est nécessaire. Les liens devraient généralement être refusés s'ils ne commencent pas par un protocole appartenant à une « whitelist » tel que « http: // » ou « https: // », empêchant ainsi l'utilisation de schémas URI tels que « javascript: // ».

Bien que la plupart des navigateurs Web modernes possèdent un filtre XSS intégré, ils ne doivent pas être considérés comme une alternative à la désinfection. Ils ne peuvent pas attraper toutes sortes d'attaques de scripts entre sites et ne sont pas stricts, sinon cela empêcherait certaines pages de se charger correctement. Le filtre XSS d'un navigateur Web ne doit être qu'une «seconde ligne de défense», la première étant la barrière qu’impose le développeur grâce aux filtrages des entrées utilisateurs.

Pour contrer ce genre d’attaque, le développeur peut utiliser par exemple la fonction « filter\_var() » qui permet de filtrer les données selon un filtrage spécifique. Heureusement, PHP met à disposition des filtres prêts à l’emploi qui sont efficaces et optimisés que nous allons voir dans la partie ci-dessous

### Notre solution

Blabla

## L’attaque dictionnaire et force brute

### Définition et comment s’en protéger

Ce genre d’attaque consiste simplement à tester chaque mot du dictionnaire pour un « login » donné. Les attaques de dictionnaires fonctionnent parce que de nombreux utilisateurs particulier ou en entreprise insistent sur l'utilisation de mots ordinaires comme mots de passe. Les attaques de dictionnaires sont rarement couronnées de succès contre les systèmes qui utilisent plusieurs mots et échouent contre les systèmes qui utilisent des combinaisons aléatoires de majuscules et minuscules mélangées avec des chiffres. Contre ces systèmes, la méthode d'attaque de la force brute (dans laquelle chaque combinaison possible de caractères et d'espaces est essayée jusqu'à une certaine longueur maximale) peut parfois être efficace, même si cette approche peut prendre beaucoup de temps pour produire des résultats.

La réussite d’un assaut de type dictionnaire peuvent être proche de zéro en limitant le nombre de tentatives autorisées dans une période de temps donnée et en choisissant judicieusement le mot de passe. Par exemple, si seulement trois tentatives sont autorisées, une période de 15 minutes doit s'écouler avant que les trois tentatives suivantes ne soient autorisées, et si le mot de passe est un énorme mélange de lettres et de chiffres sans importance, un système peut être rendu immunisé contre les attaques de dictionnaires et pratiquement immunisées contre les attaques de force brute.

### Notre solution

Blabla

## La faille « include » ou « remote file inclusion »

### Définition

L'inclusion de fichiers à distance (RFI) fait référence à une attaque d'inclusion dans laquelle un attaquant peut amener l'application Web à inclure un fichier distant (celui de l’attaquant). Les conséquences d'une attaque RFI réussie incluent la divulgation des informations du serveur, de la base de données, des personnes…

L'inclusion de fichiers à distance (RFI) se produit habituellement, lorsqu'une application reçoit le chemin d'accès d’un fichier dynamiquement par la variable « $\_GET[‘nom\_du\_fichier’] ». Cela permettrait d'envoyer une URL externe à l'instruction « include ».

### Comment éviter cette faille

La meilleure façon d'éliminer les vulnérabilités à l'inclusion de fichiers distants (RFI) est d'éviter d'inclure dynamiquement des fichiers en fonction de l'entrée de l'utilisateur. Si cela n'est pas possible, l'application devrait conserver une « whitelist » des fichiers pouvant être inclus afin de limiter le contrôle de l'attaquant sur ce qui est inclus.

En outre, dans le cas de PHP, la plupart des configurations PHP modernes sont configurées avec « allow\_url\_include » désactivé, ce qui n'autoriserait pas les utilisateurs malveillants à inclure des fichiers distants. Cela étant dit, l'inclusion de fichiers locaux (LFI) serait toujours possible.

### Notre solution

Blabla

## La faille « input file … »

### Définition

La vulnérabilité d’ « upload » de fichiers est un problème majeur avec les applications basées sur le Web. Sur de nombreux serveurs Web, cette vulnérabilité permet à un attaquant d’ « uploader » un fichier avec un code malveillant qui peut être exécuté sur le serveur. Un attaquant pourrait être en mesure de mettre une page d'hameçonnage dans le site Web.

Dans la plupart du temps, l’attaquant va utiliser cette faille en « uploadant » un fichier avec double extension comme « mon\_image.php.jpeg ». On devine ainsi que le fichier est bien un script PHP et non une image JPEG mais lors de l’ « upload », sans contre-mesure, le fichier est considéré comme une image.

L'attaquant peut ainsi révéler des informations internes du serveur Web à d'autres et, dans certaines occasions, des données sensibles peuvent être acquises comme celles existantes au sein d’une base de données, par des personnes non autorisées.

### Les différentes protections

Voici une liste des meilleures pratiques qui devraient être appliquées lorsque les « upload » de fichiers sont autorisés sur les sites Web et les applications Web. Ces pratiques aideront à sécuriser les fichiers « uploader » utilisés dans les applications Web :

* Définir un fichier « .htaccess » qui autorise uniquement l'accès aux fichiers avec des extensions autorisées.
* Ne placez pas le fichier « .htaccess » dans le même répertoire où les fichiers « uploader » seront stockés, placez-le dans le répertoire parent. De cette façon, le fichier « .htaccess » ne peut jamais être écrasé par un attaquant.
* Généralement, le fichier « .htaccess » devrait autoriser seulement les fichiers qui ont l’extension gif, jpg, jpeg et png. Le code qui va suivre permet d’éviter l’ « upload » de fichier à double extension :

**deny from all**

**<Files ~ “^w+.(gif|jpe?g|png)$”>**

**order deny,allow**

**allow from all**

**</Files>**

* Si possible, « uploader » les fichiers dans un répertoire en dehors de la racine du serveur.
* Ne comptez pas uniquement sur la validation côté client, car ce n'est pas suffisant. Idéalement, la validation côté serveur et côté client devrait être implémentée.

### Notre solution

Blabla

# Les autres failles non traitées

Bien entendu, lors de cette partie nous verrons différentes failles que nous n’avons pas traitées. Cependant, nous ne pouvons pas toutes les traiter car ils en existent une multitude. Ainsi, par avance nous vous prévenons qu’il s’agit d’une liste non exhaustive des failles potentielles pouvant se trouver sur des sites internet et applications WEB.

Cette partie ne sert qu’à montrer les recherches menées par l’ensemble du groupe dans le but de prendre conscience à quel point la sécurisation des applications est une étape importante lors du développement de ces dernières.

## Authentification et gestion de session

Cette faille est l’une des plus courantes et l’une des plus appréciées par les attaquants de nos jours. La finalité de cette attaque est d’usurper l’identité d’une personne en s’attribuant ces identifiants de session.

Sur certains sites internet, cette faille est visible dans l’url lorsque nous sommes connectés. Par exemple, imaginons le cas suivant où nous nous connectons sur un forum, nous pourrions obtenir une URL du genre :

« http://www.forum.com/moncompte/26/monprofil »

Ici, le nombre 26 indique de manière non sécurisée un identifiant de session. Pour l’attaquant il ne reste alors qu’à se créer un compte sur ce forum, il aura par exemple pour url :

« http://www.forum.com/moncompte/57/monprofil »

Il lui suffira alors de remplacer dans son url le nombre « 57 » par « 26 » pour usurper l’identité d’une personne.

Pour contrer ce genre d’attaque, il faut respecter les défenses indiquées précédemment dans les parties précédentes comme un mot de passe compliqué, ne pas faire apparaitre des données sensible au sein de l’url, contrôler les entrées utilisateurs sur le site… De plus, pour les sessions, bien souvent nous générons un « ticket » unique pour un utilisateur avec des informations totalement aléatoire pour s’assurer qu’un attaquant puisse à aucun moment deviner un identifiant de session.

## Une mauvaise configuration du serveur

Aujourd’hui, la mauvaise configuration d’un serveur fait partie des failles les plus fréquentes que nous pouvons trouver.

En fait, lorsque nous achetons (plutôt louons) un hébergement, nous obtenons alors accès à un serveur. Ce dernier peut être configuré ou non selon le contrat de vente propre à chaque hébergeur.

Sur ces serveurs, nous avons la possibilité des changer des paramètres qui autorisent plus ou moins certaines choses, dont une où nous avons utilisé pour exploiter la faille de « d’inclusion de fichier distants » ou « Remote file inclusion » en anglais. Tous ces paramètres sensibles se trouvent dans un fichier appelé « php.ini » qui regroupe toutes les variables de configuration du serveur.

Par exemple, il existe en PHP une fonction « allow\_url\_include() » qui permet de se protéger contre la faille précédemment citée.

Pour plus d’exemple de fonctions de configuration, allez sur le site :

<http://php.net/manual/en/filesystem.configuration.php>

## Hameçonnage ou « Phishing »

Le « phishing » est la tentative d'obtenir des informations sensibles telles que les noms d'utilisateur, les mots de passe et les détails de la carte de crédit (et, indirectement, l'argent), souvent pour des raisons malveillantes, en se déguisant en tant qu'entité digne de confiance sur un site ou application WEB. Le mot est un néologisme créé en tant que homophone de la pêche en raison de la similitude de l'utilisation d'un appât dans le but d'attraper une victime.

L'hameçonnage est généralement effectué par courrier électronique ou par messagerie instantanée, et il ordonne souvent aux utilisateurs d'entrer des informations personnelles sur un faux site Web dont l'apparence est identique à l’original et la seule différence est l'URL du site concerné. Cela concerne souvent des sites Web sociaux, des sites d'enchères, des banques, des sites de paiement en ligne qui sont souvent utilisées pour attirer les victimes. Les e-mails de phishing peuvent contenir des liens vers des sites Web infectés par des logiciels malveillants.

Pour contrer ce genre « d’attaque », il suffit à l’utilisateur de vérifier les URLs envoyés par mail. Malheureusement, beaucoup de personnes non-initiées à ces dangers se font encore avoir aujourd’hui.

# Travail d’équipe