

**05 juillet 2020**



**Audit de sécurité**

**Projet complet**

**Joëlle CASTELLI – Antoine FÈVRE – Noé LARRIEU-LACOSTE**

*ESGI 2A2 – Groupe 6*

1. **Risques Applications C**
2. **Informations sensibles**

**Détails :**

* Lecture des informations de connexions (à la BDD par exemple) stockés en dur sur l’application client par un utilisateur mal intentionné qui voudrait décompiler le programme.

**Comment y remédier :** Stocker les informations de connexion dans un fichier chiffré, avec une clé sous la forme d’une matrice d’encodage CODEC et déchiffrer ce fichier au lancement du programme.

1. **Informations entrés incorrects**

**Détails :**

* Les informations rentrées par l’utilisateur sont incorrectes sur l’application coté client
* Les informations rentrées par l’utilisateur peuvent représenter une faille SQL au moment de l’insertion sur l’application coté serveur.

**Comment y remédier :** Vérifier les informations saisies par l’utilisateur aussi bien sur l’application client que serveur.

1. **Risques Application Java**
2. **Informations sensibles**

**Détails :**

* Lecture des informations de connexions (à la BDD par exemple) stockés en dur sur l’application client par un utilisateur mal intentionné qui voudrait décompiler le programme.

**Comment y remédier :** Stocker les informations de connexion dans un fichier chiffré, avec une clé sous la forme d’une matrice d’encodage CODEC et déchiffrer ce fichier au lancement du programme.

1. **Reverse**

Les applications Java sont très facilement décompilable, et donc lisible. Il faut donc faire attention à ne laisser aucune information sensible en dur et laisser le moins d’indice possibles sur le fonctionnement global de la base de données par exemple.

1. **Risques Application Web**
2. **Faille XSS**

**Détails :**

* Injection de code HTML ou JavaScript dans des variables
* Deux types :
  + XSS réfléchi : temporaire
  + XSS stocké en base de données : permanent

**Comment y remédier :** utiliser la fonction PHP htmlspecialchars() pour filtrer et traduire les caractères spéciaux comme en entités HTML. Dans notre cas, le framework Laravel gère ça pour nous.

1. **Faille include**

**Détails :**

* Exploitation de la fonction PHP include()
* Deux types :
  + A distance : inclusion de fichiers contenus sur un autre serveur que celui du site web
  + En local : inclusion et exploitation de fichiers situés sur le serveur du site, possibilité de naviguer dans les répertoires du site voire ceux du serveur

**Comment y remédier :**

* Tester de la présence de la faille sur le site en essayant d’inclure une page qui n'existe pas
* Développer d’un code d’inclusion sécurisé :
  + Limiter l'inclusion aux fichiers PHP en ajoutant dynamiquement l'extension
  + Remplacer les caractères permettant de naviguer dans les répertoires
  + Protéger les dossiers et fichiers importants par htaccess et interdiction de les inclure
  + Vérifier que la page à inclure existe bien sur le serveur
* Utiliser le système de templating de Laravel pour limiter le plus possible les inclusions

1. **Injection SQL**

**Détails :**

* Modification d’une requête SQL en injectant des morceaux de code non filtrés
* Généralement exploitée sur les formulaires

**Comment y remédier :** Utiliser l’ORM Eloquent permettant de construire des requêtes préparés et sécurisés.

1. **CSRF (pour *Cross Site Request Forgery*)**

**Détails :** exécution d’actions sur l’ordinateur de l’utilisateur à son insu

**Comment y remédier :**

* Authentifier l’utilisateur par un token
* Demander la confirmation de l’action par un clic sur un bouton, un mot de passe ou un captcha
* Utiliser le token CSRF généré par Laravel dans les formulaires POST

1. **Attaque par brute force**

**Détails :**

* Tentative de crack d’un mot de passe en utilisant toutes les combinaisons possibles
* Attaque par dictionnaire : utilisation des mots de passe régulièrement utilisés

**Comment y remédier :**

* Vérifier les formulaires par captcha
* Exiger des mots de passe forts
* Ralentir l’exécution du formulaire de connexion en ajoutant une pause d’une seconde (indolore pour l’utilisateur normal mais gros ralentissement pour le pirate)
* Limiter le nombre de tentatives par personne et par jour

1. **Vol de session**

**Détails :** récupération de l’ID unique de la session d’un utilisateur (PHPSESSID) pour se faire passer pour lui sur le site

**Comment y remédier :**

* Identifier l’utilisateur de la façon la plus sûre possible en créant un système de communication client/serveur par « tickets » : le serveur génère et garde en mémoire un ticket stocké dans les cookies de l’utilisateur. A chaque demande de page, le serveur vérifie que le client a bien le même ticket que lui et en génère un nouveau.
* Système faillible : ajouter un système de déconnexion automatique après un certain temps d’inactivité.

1. **Risques serveur VPS**
2. **Sécurisation du site**

* Changer les ports par défaut et sécuriser les mots de passe afin d’éviter les connexions sur la base de données
* Passer les inclusions de fichiers critiques en require (et non include) afin d’obtenir une erreur fatale en cas d’échec de chargement du fichier
* Ne divulguer que le minimum d’informations dans les messages d’erreur afin de ne pas faciliter les injections SQL
* Protéger les répertoires sensibles par .htaccess
* Forcer le https et avoir un certificat SSL (let’s encrypt le fait très bien)
* Ne pas se contenter de la clé générée par let’encrypt, générer une clé RSA plus forte (4096 bits) pour renforcer la sécurité du site.
* Passer en support TLS 1.3 et retirer le support des anciennes versions, potentiellement vulnérable
* Enregistrer son domaine auprès de DNS Certification Authority Authorization (CAA) pour assurer la signature de notre domaine.
* Passer en http/2
* Utiliser le filtrage cipher pour empêcher les navigateurs trop anciens et potentiellement vulnérable d’accéder au site.
* Viser la note A+ sur SSL Labs.

1. **Sécurisation des connexions sur le VPS**

* Restreindre l’accès au VPS le plus possible
* Mot de passe fort pour la BDD et si possible, restreindre son accès en local
* Mot de passe fort pour la connexion SSH et si possible, authentification par clé uniquement.

1. **Intégrité du serveur et de ses données**

* Programmer un « batch update » quotidien via l’API afin de garder à jour toutes les informations relatives aux séries
* Si ces informations sont erronées ou que l’API est inutilisable, permettre l’ajout ou la modification d’informations manuellement
* Prévoir des sauvegardes régulières de la base de données, des informations du serveur et du code.