Grenoble INP ENSIMAG

ETUDE DE FAILLE & PSSI

SQL injection vulnerability via the keyword parameter

IF 3A

Ismail LIJAI
Yassine EL OUARDINI
Zakariae MOUTCHOU

16 Décembre 2019

Table des matières

1 Introduction

Une vulnérabilité extrêmement courante consiste en l'utilisation de données en provenance du client dans une requête SQL pour laquelle le serveur n'effectue aucune vérification sur le format de ces données. Un attaquant peut transmettre des données ayant un format tel que la requête générée ait des effets non prévus par le développeur. [4]
Cette faille concerne les machines serveurs puisque c'est la base de donnée qui est menacée. En effet, les machines clientes ne seront pas affectées car la sql statement écrite dans la barre de recherche par l'attaquant cible la base donnée située dans le serveur.

2 Étude de la faille CVE 2018-19331

2.1 Présentation de la faille

Le service compromis par la faille CVE 2018-19331 est l'application S-cms Version 1.5, ce dernier est vulnérable aux attaques de types SQL injection. En effet le code fournit par S-cms pour la création ne gère pas toutes les injections du code sql qui représentent une menace potentielle pour la base de données. [2]

2.2 Le mécanisme permettant l'exploitation de la faille

Pour faire des recherches dans le site S-cms on utilise le champ de recherche de la page d'accueil. Ce champ est associé au fichier S-CMS/search.php. En entrant une recherche contenant la phrase keyword, le code sql suivant est donc exécuté :

```
$sql="select * from SL_text where (T_title like '%".$keyword."%' or
T_content like '%".$keyword."%' ) order by T_id desc";
```

En remplaçant keyword par : %' AND 1=1 AND '%'=' par exemple on obtient [1] :

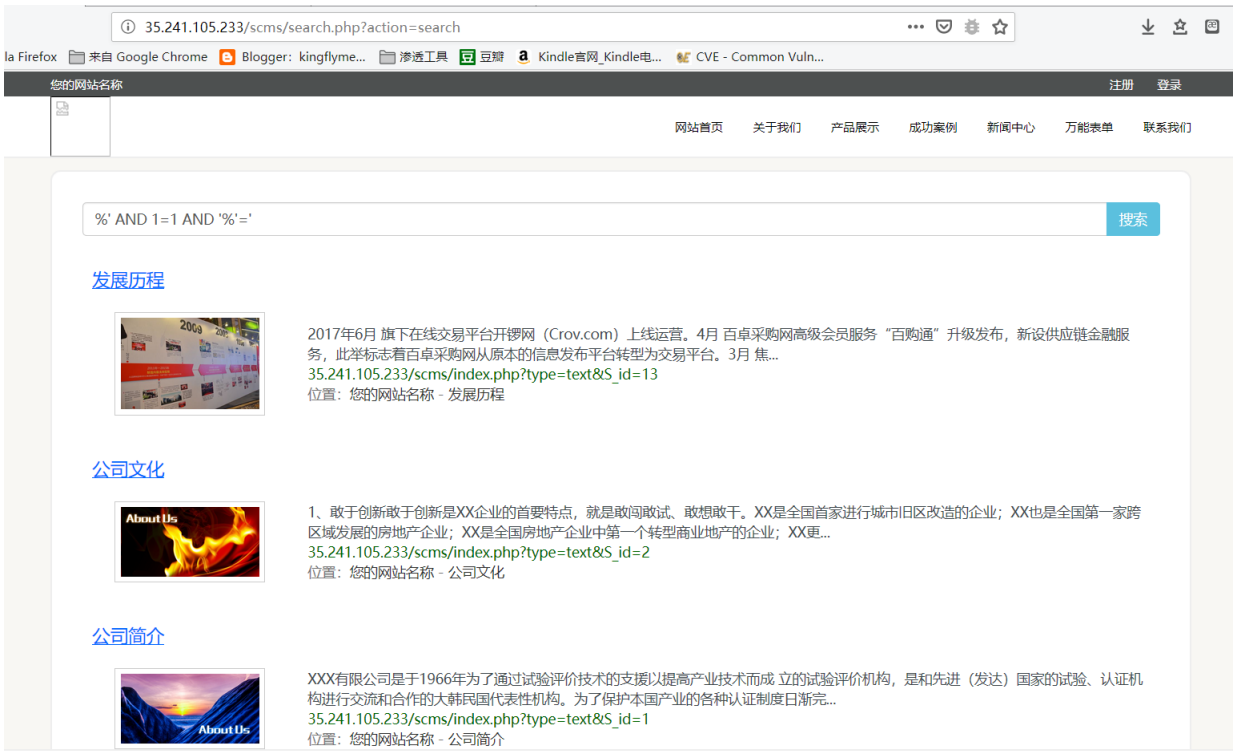

FIGURE 1 – Résultat du premier cas

En remplaçant keyword par : %' AND 1=2 AND '%'='par exemple on obtient [1] :

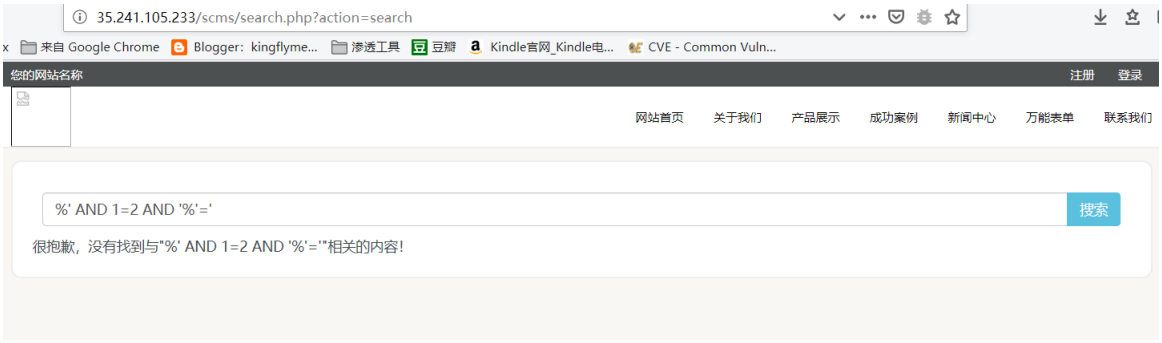

FIGURE 2 – Résultat du deuxième cas

2.3 L'architecture typique du système d'information impliqué

Le système d'information qui pourrait être impliquée est un système de site Web d'entreprise S-CMS. L'ensemble de programmes est un framework PHP + MYSQL.

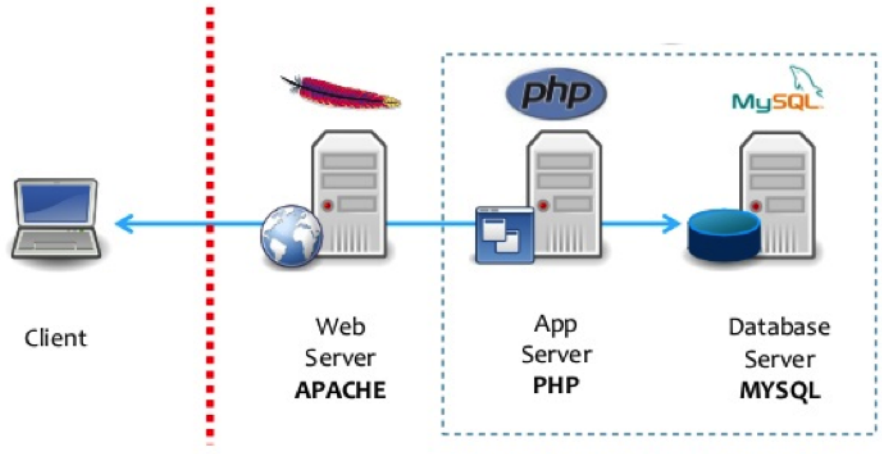

FIGURE 3 – Architecture typique du système d'information impliqué

2.4 Bonnes pratiques et recommandations

Pour limiter l'impact de l'exploitation de ces failles plusieurs précautions peuvent être mises en place :

- Mise en place des pare-feux de base de données pour détecter les tentatives d'une injection SQL quand le nombre de requêtes invalides dépasse un seuil prédéfini. [3]
- Limiter l'accès aux bases de données à un accès en lecture seule pour les données sensibles. [3]
- Partager les données sur plusieurs comptes SQL afin de limiter les dégâts si un des comptes est détruit par une injection. [3]

D'une autre part, afin d'empêcher l'exploitation de ces failles, des mesures de sécurité doivent être respectées lors de l'écriture du code sql telles que :

- L'échappement des caractères spéciaux entrés par l'utilisateur, cependant, il n'est pas recommandé, notamment parce qu'il induit un couplage fort entre la sécurité du code et la technologie de bases de données et, d'autre part, parce qu'il tend à faire négliger certains cas particuliers, comme les injections SQL sur les variables censées être de type numérique. [3]

— L'utilisation des requêtes SQL paramétrées : au lieu d'injecter directement la phrase entrée par l'utilisateur dans la requête SQL, cette phrase va être considérée comme un paramètre et va être traité et vérifiée avant son injection dans le code SQL. Ainsi une tentative d'injection SQL va être simplement considérée comme un paramètre invalide et va donc être rejetée avant d'être traitée en tant que requête SQL.[3]

2.5 Extrait du rapport PSSI

Les failles relatives aux injections SQL sont d'une criticité très haute puisqu'elles peuvent entraîner la perte totale des données sauvegardées dans une base de données. Afin d'éviter ce type de dangers, le système informatique (le code du site et la base de données qui lui est attachée dans notre cas) doit respecter parfaitement les consignes de codage mentionnées ci-haut. Les collaborateurs responsables du développement du site doivent être formés sur les bonnes pratiques du développement informatique, et en particulier le bon usage des requêtes SQL.

3 Mise en place de l'expérimentation

3.1 Architecture

Afin d'exhiber la faille, il faut pouvoir ouvrir le site et répliquer les manoeuvres présentées en 2.2. Pour cela, il faut compiler le code `PHP` du site, faire la liaison avec la base de données, et tout ouvrir dans une image du container `Docker`. On s'est inspiré principalement du $TP1$(Vagrant et Docker) pour réaliser notre architecture qui consistait en deux parties.

La partie **serveur web** : C'est le dossier *http*. Elle contient un `Dockerfile` spécifiant les packages nécessaires à installer sur le container, les commandes à lancer automatiquement et le port sur lequel le site sera visible et accessible, ainsi que d'autres informations. En plus, elle contient tous les répertoires et les fichiers `PHP` nécessaires à la compilation.

La partie **Base de données** : C'est le dossier *bd*. De même, elle contient un `Dockerfile` et un fichier `sql` permettant de créer et de peupler la base de données.

3.2 Expérimentation

Pour pouvoir lancer l'expérimentation, veuillez récupérer les sources à partir du ce lien : `https://github.com/lijaii15/Faille`

On se place dans la racine du projet (le répertoire contenant le fichier `docker-compose.yml`. On exécute les commandes suivantes sur un terminal :

— **docker-compose build**
— **docker-compose up**

Ensuite, on ouvre un navigateur web tel que *Chrome* ou $Firefox$. L'adresse locale du site est : **http ://localhost :8080**. Après s'être rendu sur cette adresse, une première page s'affiche (Figure 4), contenant l'accord utilisateur à accepter. Notons que la page s'affiche initialement en chinois. On peut se servir du traducteur automatique de *Chrome* pour la traduire sur place.

FIGURE 4 – Image de l'accord utilisateur qui s'affiche en ouvrant le site

Une deuxième page similaire s'affiche contenant des informations à propos du site à valider. En validant, une dernière page s'affiche, comme montre la figure 5. Il faut y remplir les informations pour initialiser le site et accéder à la base de données.

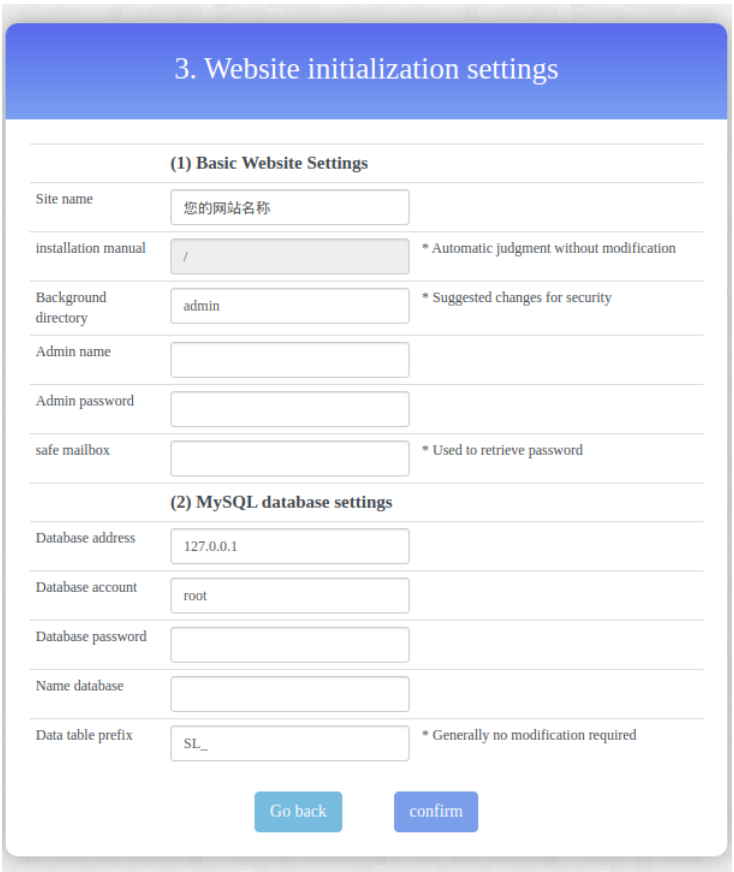

FIGURE 5 – Initialisation du site

Les information à remplir sont :

- **Admin name** : admin
- **Admin password** : admin
- **Database password** : root
- **Name database** : db2

Malheureusement, on rencontre jusqu'au moment un problème de connexion à la base de données. En effet, après avoir spécifié toutes les informations, dont la correctitude est assurée, la connexion nous est refusée. En examinant le fichier `install.php`, on voit que l'erreur vient de la fonction `mysqli_connect` dont le rôle est de se connecter à la base de données mais échoue à le remplir.

4 Conclusion

Pour résumer, la faille étudiée lors de ce projet est une SQL injection. Cette dernière est fréquemment rencontrée dans des sites web basés sur l'interaction php-sql. Ce projet nous a permis de comprendre le danger que peut présenter une telle faille ainsi que les bonnes pratiques à mettre en place comme précaution. C'était une occasion de nous familiariser avec l'architecture typique d'un tel système d'information, notamment les bouts de code en php.

Références

[1] KINGFLYME'S BLOG. *The POC of S-CMS(sql-injection)-CVE-2018-19331*. 2018. URL : https://kingflyme.blogspot.com/2018/11/the-poc-of-s-cmssql-injection.html.

[2] CVE DETAILS. *CVE 2018-19331*. 2018. URL : https://www.cvedetails.com/vulnerability-search.php.

[3] Wikipedia The free ENCYCLOPEDIA. *SQL injection*. 2019. URL : https://en.wikipedia.org/wiki/SQL_injection.

[4] Secrétariat général de la défense et de la sécurité NATIONALE. *Recommandations pour la sécurisation des sites web*. 2013. URL : https://www.ssi.gouv.fr/administration/guide/recommandations-pour-la-securisation-des-sites-web/.