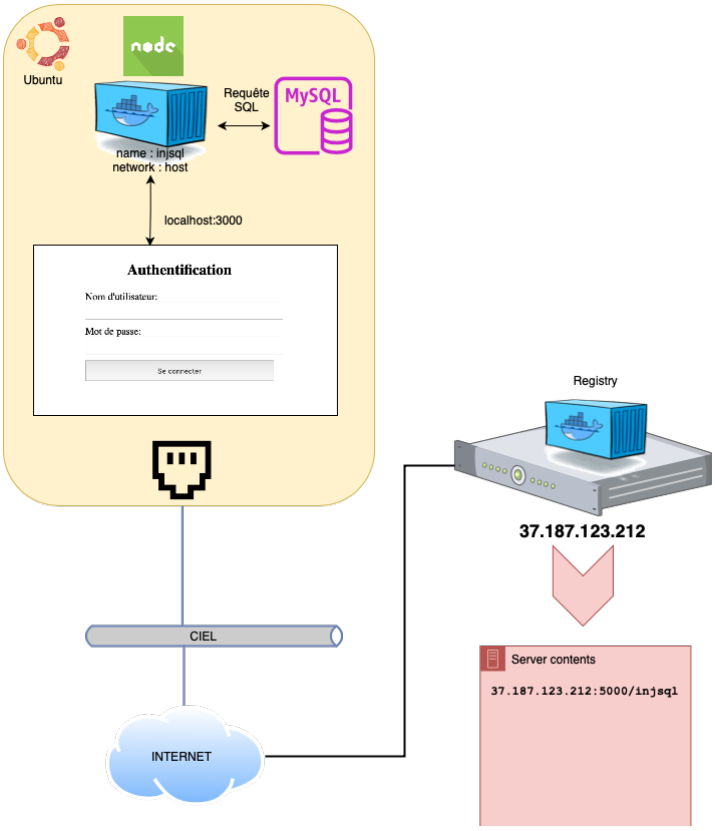
INJECTION SQL : INTRODUCTION



Création d'une base et d'une table en utilisant phpmyadmin

1. **Connectez-vous** sur phpmyadmin (http://localhost/phpmyadmin)
2. **Créez** une base « users ».
3. **Créez** une table « credentials » composée de 3 colonnes (ID en clé primaire, username(varchar 20) et password(varchar 20)).
4. **Créez** 3 enregistrements de votre choix.

Création d'un conteneur Docker « serveur nodejs »

Configuration de Docker pour autoriser l'accès au registre privé 37.187.123.212:5000 en http.

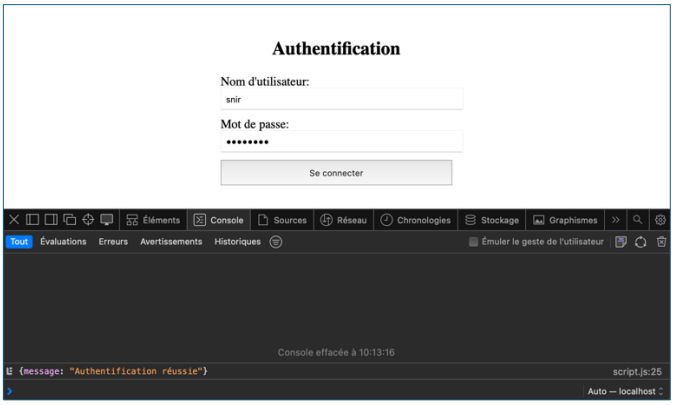
1. **Créez** le fichier daemon.json :
   `sudo nano /etc/docker/daemon.json`
2. **Éditez** le contenu suivant :
   `{"insecure-registries" : ["37.187.123.212:5000"]}`.
3. **Relancez** le service Docker :
   `systemctl restart docker`
4. **Connectez-vous** sur le registre privé :
   `docker login 37.187.123.212:5000` (le username et le password seront donnés par l'enseignant)

Création du conteneur nodeJs :

1. **Saisissez** : `docker run -it --network host --name injsql 37.187.123.212:5000/injsql`

Test de l'application :

1. **Lancez** un navigateur sur l'url : http://localhost:3000
2. **Visualisez** la console du navigateur.



1. **Testez** les identifiants d'un utilisateur ajouté dans la base et **vérifiez** le message affiché dans la console.
2. **Testez** le cas d'un mauvais identifiant.

Test des injections sql :

À l'aide du document : http://newtonformationsnir.fr/btsciel/InjectionsSqlPart2.pdf

1. **Retrouvez** le nombre de bases de données.
2. **Retrouvez** le nom des bases de données.
3. **Retrouvez** le nombre de tables de la base « users ».
4. **Retrouvez** le nom des tables de la base « users ».
5. **Retrouvez** le nombre de colonne de la table « credentials ».
6. **Retrouvez** le nombre d'enregistrements de la table « credentials ».

Analyse de la requête à l'aide de Burpsuite.

On utilisera Exegol pour créer un conteneur d'un environnement contenant les outils logiciel de cyber-sécurité.

1. En utilisant le document ressource http://newtonformationsnir.fr/TP/Exegol.pdf, **créez** un conteneur avec l'image « full ». Vous **nommerez** le conteneur avec votre nom (ex : bouhenic).
2. **Lancez** Firefox en tâche de fond : `firefox  &> /dev/null &`
3. **Lancez** Burpsuite : `burpsuite` et aidez-vous de la fiche de guidance : http://newtonformationsnir.fr/TP/FICHEDEGUIDANCEburpsuite.pdf
4. **Relevez** le type de requête http utilisé et le contenu des informations transmises

Utilisation de l'outil sqlmap pour relever le contenu de la base vulnérable.

En vous aidant du document « Sqlmap cheat sheet » :
http://newtonformationsnir.fr/TP/SQLmapCheatSheet.pdf

1. **Relevez** le nom des bases du serveur mysql en utilisant sqlmap.
2. **Relevez** les noms des tables de la base « users » en utilisant sqlmap.
3. **Relevez** les noms des colonnes de la table « credentials ».
4. **Relevez** le contenu de la table « credentials ».

Sécurisation de l'application web contre les injections SQL :

1. **Stoppez** le conteneur Docker : `docker stop injsql`

Nous allons modifier le code de l'application. Pour cela, nous allons cloner le code stocké sur github.

2. **Clonez** le dépôt : `git clone https://github.com/bouhenic/InjectionSql`

3. **Déplacez-vous** dans le répertoire cloné : `cd InjectionSql`

4. **Éditez** le fichier index.js : `sudo nano index.js`.

5. **Commentez** la partie du code suivant :

```
    // Requête SQL VULNÉRABLE à l'injection SQL
const query = "SELECT * FROM credentials WHERE username = '" + username + "' AND password = '" + password + "'";
console.log(query); // Affiche la requête SQL

//Avec username=snir' or 1=1--' ou ' OR 1=1-- - ou ' OR TRUE-- - ou ' OR 1=SLEEP(4)-- - (pour détecter que c'est du sql dérrière)
connection.query(query, function(err, results) {
    if (err) {
        console.error('Erreur lors de l\'exécution de la requête: ' + err.stack);
        res.status(500).json({ message: 'Erreur lors de la connexion' });
        return;
    }

    if (results.length > 0) {
        // Authentification réussie
        res.json({ message: 'Authentification réussie' });
    } else {
        // Informations d'identification invalides
        res.status(401).json({ message: 'Informations d\'identification invalides' });
    }
});
```

6. **Décommentez** la partie du code suivant :

```
    // Requête SQL pour vérifier les informations d'identification
//const query = "SELECT * FROM credentials WHERE username = ? AND password = ?";
//console.log(query);
// Remplace les placeholders par les variables sécurisées
/*connection.query(query, [username, password], function(err, results) {
   if (err) {
        console.error('Erreur lors de l\'exécution de la requête: ' + err.stack);
        res.status(500).json({ message: 'Erreur lors de la connexion' });
        return;
    }

    if (results.length > 0) {
        // Authentification réussie
        res.json({ message: 'Authentification réussie' });
    } else {
        // Informations d'identification invalides
        res.status(401).json({ message: 'Informations d\'identification invalides' });
    }
});*/
```

Ce code utilise une requête préparée.

Explication :

- **?** : Ce sont les "placeholders" dans la requête. Ils seront remplacés par les valeurs réelles de username et password.
- **[username, password]** : C'est un tableau contenant les valeurs qui vont être insérées à la place des ? dans la requête. Le moteur SQL va s'occuper d'associer les valeurs aux "placeholders" de manière sécurisée.
- **Avantage** : Même si un utilisateur tente d'injecter du SQL malveillant dans username ou password, cela ne fonctionnera pas, car les données sont traitées comme des valeurs strictes et non comme du code SQL.

Nous allons à présent reconstruire l'image Docker et Relancer un conteneur :

7. **Saisissez : `docker compose up --build`**

8. **Testez** de nouveau l'application avec des injections SQL.