Importation des modules

```
from flask import Flask, request, jsonify
import subprocess
import shlex
import logging
import re
```

- **Flask** : Framework web léger pour créer l'API.
- **subprocess** : Permet d'exécuter des commandes système de manière sécurisée.
- **shlex** : Sécurise la gestion des arguments pour éviter l'injection de commandes.
- **logging** : Permet d'enregistrer les tentatives suspectes et les erreurs.
- **re** : Utilisé pour valider l'entrée utilisateur avec une expression régulière.

Configuration du journal de sécurité

```
logging.basicConfig(filename="security.log", level=logging.WARNING,
                    format='%(asctime)s - %(levelname)s - %(message)s')
```

- `filename="security.log"` : Stocke les logs dans un fichier `security.log`.
- `level=logging.WARNING` : Enregistre uniquement les avertissements et erreurs (pas les messages d'information normaux).
- `format='%(asctime)s - %(levelname)s - %(message)s'` : Ajoute une date/heure aux logs pour faciliter l'analyse.

Définition des commandes autorisées

```
ALLOWED_COMMANDS = {"ls", "whoami", "uptime"}
```

- Utilisation d'un **ensemble (`set`)** de commandes autorisées pour empêcher l'exécution arbitraire de commandes système dangereuses comme `rm -rf /`.

Expression régulière pour valider l'entrée

```
COMMAND_PATTERN = re.compile(r"^[a-z]+$")
```

Autorise uniquement des commandes composées de **lettres minuscules** (`ls`, `whoami`, `uptime`).

- **Évite l'injection de caractères spéciaux** comme `;`, `&`, `|` qui pourraient permettre des attaques en chaîne.

Définition de l'API

```python
@app.route('/run', methods=['GET'])
def run_command():
```

- Définit un **point d'accès (`/run`)** qui accepte uniquement les requêtes `GET`.

Récupération et validation de la commande

```python
cmd = request.args.get('cmd', '').strip()
```

- `request.args.get('cmd', '')` : Récupère la commande passée dans l'URL (`/run?cmd=ls`).
- `.strip()` : Supprime les espaces avant/après pour éviter des manipulations.

```python
if cmd not in ALLOWED_COMMANDS or not COMMAND_PATTERN.match(cmd):
    logging.warning(f"Tentative d'exécution de commande interdite : {cmd}")
    return jsonify({"error": "Commande non autorisée"}), 403
```

Vérifie si la commande est dans `ALLOWED_COMMANDS` et respecte la **regex**.

- Si ce n'est pas le cas :
  - Enregistre l'incident dans `security.log`.
  - Retourne un message d'erreur `403 Forbidden`.

Exécution sécurisée de la commande

```python
try:
    # Exécution sécurisée avec timeout
    output = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT, timeout=3)
    return jsonify({"output": output.decode().strip()})
```

`shlex.split(cmd)` : Sépare proprement la commande en arguments pour éviter l'injection.

- **`check_output()`** :
  - Exécute la commande en capturant la sortie.
  - Redirige les erreurs (`stderr`) vers la sortie standard (`stdout`).
  - Définit un `timeout=3` pour éviter que la commande ne bloque le serveur.
- Retourne la sortie de la commande dans un format JSON.

Gestion des erreurs

```python
except subprocess.CalledProcessError:
    return jsonify({"error": "Erreur lors de l'exécution de la commande"}), 400
```

- Capture les erreurs si la commande échoue (ex. : `ls /dossier_inexistan`

```
except subprocess.TimeoutExpired:
    return jsonify({"error": "Commande trop longue"}), 408
```

Empêche les commandes longues de bloquer le serveur en limitant le temps d'exécution à 3 secondes.

```
except Exception as e:
    logging.error(f"Erreur inattendue : {str(e)}")
    return jsonify({"error": "Une erreur est survenue"}), 500
```

- Capture toute autre erreur imprévue et l'enregistre dans `security.log`.
- Retourne un message générique `500 Internal Server Error` pour éviter de donner des infos sensibles à un attaquant.

Lancement de l'application Flask

```
if __name__ == '__main__':
    app.run(debug=False, host='0.0.0.0', port=5000)
```

- **`debug=False`** : Désactive le mode debug en production pour éviter les fuites d'information.
- **`host='0.0.0.0'`** : Rend l'application accessible depuis n'importe quelle machine du réseau.
- **`port=5000`** : Définit le port d'écoute du serveur Flask.

🛡️ Résumé des améliorations de sécurité :

☑ **Empêche l'exécution arbitraire de commandes** grâce à une **liste blanche**.
☑ **Valide strictement les entrées utilisateur** avec une **expression régulière**.
☑ **Évite les injections de commandes** en utilisant `shlex.split()`.
☑ **Protège contre les blocages** avec un `timeout` de 3 secondes.
☑ **Masque les erreurs techniques** pour éviter les fuites d'informations sensibles.
☑ **Enregistre les tentatives suspectes** dans un fichier de **logs**.

Ton code est maintenant **solide et sécurisé** contre les attaques courantes ! 🚀🔐