Documentation du script d'audit SELinux avec audit2allow

J'ai conçu ce script pour **auditer les refus SELinux** liés à des processus spécifiques et générer automatiquement des **règles permissives locales** avec `audit2allow`. Il me permet de centraliser les informations sur les refus SELinux et de produire un rapport clair pour chaque processus.

---

1. Pré-requis

Avant d'exécuter ce script, je m'assure que les éléments suivants sont installés sur ma machine :

- **SELinux** activé sur le système.

Vérifier la présence des paquets SELinux

Debian/Ubuntu n'installe pas SELinux par défaut. On commence par installer les paquets nécessaires :

```
sudo apt update
sudo apt install selinux-basics selinux-policy-default auditd -y
```

Explications :

- `selinux-basics` : permet de configurer et d'activer SELinux facilement.
- `selinux-policy-default` : fournit la politique par défaut.
- `auditd` : journalise les alertes SELinux.

Activer SELinux au démarrage

SELinux se configure via `/etc/selinux/config`. J'édite le fichier :

```
sudo nano /etc/selinux/config
```

Je modifie les lignes suivantes :

```
SELINUX=enforcing
SELINUXTYPE=default
```

- `enforcing` : SELinux bloque réellement les accès interdits.
- `permissive` : SELinux ne bloque rien mais journalise les violations (utile pour tester avant `enforcing`).

Je auvegarde et ferme le fichier.

Activer SELinux pour cette session

Je peux forcer l'activation sans redémarrer (utile pour tester) :

```
sudo selinux-activate
sudo reboot
```

> Le reboot est nécessaire pour que le noyau charge SELinux.

Vérifier que SELinux est actif

Après le redémarrage, je peux vérifier :

```
sestatus
```

Je devrais voir :

```
SELinux status:                 enabled
Current mode:                   enforcing
Mode from config file:          enforcing
```

A retenir

- Si je mets `permissive` au début, SELinux va juste journaliser les violations sans tuer les processus.
- Si je passes en `enforcing`, SELinux bloquera réellement les accès non autorisés (utile pour tester ton script TP).
- Les logs SELinux sont dans `/var/log/audit/audit.log`.

- **auditd** pour collecter les logs de sécurité. Installation rapide sous Debian :

  ```
  sudo apt install auditd
  ```

- **policycoreutils** pour utiliser `audit2allow` :

  ```
  sudo apt install policycoreutils-python-utils
  ```

---

2. Couleurs dans le script

Pour améliorer la lisibilité de l'affichage, j'ai défini des variables de couleur :

Variable	Couleur	Utilisation
`RED`	Rouge	Pour afficher les refus SELinux
`GREEN`	Vert	Pour indiquer qu'aucun refus n'a été détecté
`YELLOW`	Jaune	Pour signaler la génération d'une règle SELinux
`BLUE`	Bleu	Pour les titres et étapes
`NC`	Aucune	Pour réinitialiser la couleur

---

3. Lecture du fichier de politique

Je passe le fichier de politique SELinux en **argument** lors de l'exécution du script :

```
./script.sh mon_fichier_politique.txt
```

- Si aucun fichier n'est fourni, le script s'arrête avec un message d'erreur en rouge.
- Ce fichier contient la liste des processus à auditer, sous la forme :

  ```
  nom_processus:chemins_associes
  ```

- Je lis chaque ligne et je stocke les informations dans un **tableau associatif Bash**, avec le nom du processus comme clé et les chemins associés comme valeur.

---

4. Affichage des processus disponibles

Après lecture du fichier, je liste tous les processus présents dans la politique avec leurs chemins associés :

```
Processus disponibles dans mon_fichier_politique.txt :
- httpd:/var/www/html
- mysqld:/var/lib/mysql
```

Cela me permet de vérifier rapidement que le fichier est correct.

---

5. Sélection d'un processus à scanner

J'offre la possibilité de scanner :

- **Tous les processus** si l'utilisateur laisse la réponse vide.
- **Un processus spécifique** en entrant son nom.

Cette étape rend le script flexible et évite de traiter tous les processus si je veux cibler uniquement un service.

---

6. Initialisation du fichier de rapport

Je crée un fichier `rapport.log` pour **conserver tous les résultats** de l'analyse. Chaque exécution écrase l'ancien rapport pour éviter toute confusion.

---

7. Analyse des refus SELinux

Pour chaque processus sélectionné :

1. Je récupère les logs de refus SELinux avec la commande :

```
ausearch -m avc -c "$process"
```

2. Si aucun refus n'est détecté :
    - J'affiche un message vert.
    - J'écris le message dans le rapport.
3. Si des refus sont détectés :
    - J'affiche un message rouge.
    - J'ajoute les détails au rapport.
    - Je génère une **règle SELinux locale** permissive avec `audit2allow` :

```
echo "$REFUS" | audit2allow -M "${process}_local"
```

    - Le fichier généré est `${process}_local.pp`.

---

8. Génération des règles SELinux

La règle générée permet d'ajouter au **module local** les autorisations nécessaires pour que le processus fonctionne sans déclencher de refus SELinux. Je signale cette étape en jaune pour attirer l'attention sur le fichier `.pp` généré.

---

9. Conclusion de l'analyse

À la fin du script, j'affiche un message bleu pour indiquer :

- La fin de l'analyse.
- L'emplacement du rapport complet.

Exemple :

```
Analyse terminée. Rapport complet enregistré dans rapport.log.
```

---

10. Points clés et bonnes pratiques

- Toujours exécuter le script avec un **fichier de politique valide**.
- Vérifier que `auditd` est actif pour capturer les logs.
- Les règles générées doivent être testées avant d'être appliquées en production.
- Le fichier de rapport permet de **tracer l'historique des refus** pour audits futurs.

---

11. Exemple d'exécution

```
./script.sh policy.txt
```

Exemple d'affichage :

```
Chargement de la politique depuis mon_fichier_politique.txt

Processus disponibles dans mon_fichier_politique.txt :
- httpd:/var/www/html
- mysqld:/var/lib/mysql

Voulez-vous scanner un processus spécifique ? (laisser vide pour tous) :

=== Analyse des refus SELinux ===
 Aucun refus SELinux détecté pour httpd:/var/www/html
 Refus SELinux détecté pour mysqld:/var/lib/mysql
 Règle SELinux locale générée : mysqld_local.pp

Analyse terminée. Rapport complet enregistré dans rapport.log.
```

Script d'audit SELinux avec audit2allow

```
#!/bin/bash

# ---------------------------------------
# 1. Définition des couleurs pour l'affichage
# ---------------------------------------
RED='\033[0;31m'      # Rouge pour les erreurs ou refus SELinux
GREEN='\033[0;32m'    # Vert pour indiquer qu'aucun refus n'a été détecté
```

```
YELLOW='\033[1;33m'   # Jaune pour les messages informatifs comme la génération de règles
BLUE='\033[0;34m'     # Bleu pour les titres et étapes importantes
NC='\033[0m'           # Aucune couleur, pour réinitialiser l'affichage

# ---------------------------------------
# 2. Lecture du fichier de politique fourni en argument
# ---------------------------------------
POLICY_FILE=$1          # Je récupère le premier argument, qui doit être le fichier de politique
if [[ -z "$POLICY_FILE" ]]; then
    # Si aucun fichier n'est fourni, j'affiche un message d'erreur et je quitte le script
    echo -e "${RED}Erreur : Aucun fichier de politique fourni.${NC}"
    exit 1
fi

echo -e "${BLUE}Chargement de la politique depuis $POLICY_FILE${NC}"

# ---------------------------------------
# 3. Lecture des processus depuis le fichier politique
# ---------------------------------------
declare -A PROCESSES    # Je crée un tableau associatif pour stocker les processus et leurs chemins
while IFS=: read -r name paths; do
    # Pour chaque ligne du fichier, je sépare le nom du processus et ses chemins
    PROCESSES["$name"]="$paths"
done < "$POLICY_FILE"

# J'affiche la liste des processus disponibles pour que l'utilisateur puisse vérifier
echo -e "\nProcessus disponibles dans $POLICY_FILE :"
for p in "${!PROCESSES[@]}"; do
    echo "- $p:${PROCESSES[$p]}"
done

# ---------------------------------------
# 4. Choix d'un processus spécifique à scanner
# ---------------------------------------
read -p $'\nVoulez-vous scanner un processus spécifique ? (laisser vide pour tous) : ' CHOICE
# Si l'utilisateur laisse vide, tous les processus seront analysés

# ---------------------------------------
# 5. Préparation du fichier de rapport
# ---------------------------------------
REPORT_FILE="rapport.log"
echo "" > "$REPORT_FILE"    # Je vide le fichier avant de commencer l'analyse

echo -e "\n${BLUE}=== Analyse des refus SELinux ===${NC}"

# ---------------------------------------
# 6. Boucle sur les processus pour analyser les refus
# ---------------------------------------
```

```
for process in "${!PROCESSES[@]}"; do
    # Si un processus spécifique a été choisi et qu'il ne correspond pas à celui
de la boucle, je passe au suivant
    if [[ -n "$CHOICE" && "$process" != "$CHOICE" ]]; then
        continue
    fi

    PATHS="${PROCESSES[$process]}"   # Je récupère les chemins associés au
processus

    # ---------------------------------------
    # 7. Vérification des refus SELinux pour ce processus
    # ---------------------------------------
    REFUS=$(ausearch -m avc -c "$process" 2>/dev/null)  # Je recherche les logs
AVC pour ce processus

    if [[ -z "$REFUS" ]]; then
        # Aucun refus détecté
        echo -e "${GREEN} Aucun refus SELinux détecté pour $process:$PATHS${NC}"
        echo "Aucun refus SELinux détecté pour $process:$PATHS" >> "$REPORT_FILE"
    else
        # Refus détecté, je l'affiche et je l'enregistre dans le rapport
        echo -e "${RED} Refus SELinux détecté pour $process:$PATHS${NC}"
        echo "$REFUS" >> "$REPORT_FILE"

        # ---------------------------------------
        # 8. Génération d'une règle permissive locale avec audit2allow
        # ---------------------------------------
        echo "$REFUS" | audit2allow -M "${process}_local" 2>/dev/null
        echo -e "${YELLOW} Règle SELinux locale générée :
${process}_local.pp${NC}"
    fi
done

# ---------------------------------------
# 9. Fin de l'analyse
# ---------------------------------------
echo -e "\n${BLUE}Analyse terminée. Rapport complet enregistré dans
$REPORT_FILE.${NC}"
```

Explications complémentaires

1. **Tableau associatif** `PROCESSES` Je l'utilise pour relier chaque processus à ses chemins, ce qui me permet de filtrer les logs de manière ciblée.

2. `ausearch` Cette commande me permet d'extraire les événements AVC (Access Vector Cache) correspondant à SELinux. Je filtre par processus pour éviter d'avoir trop de logs inutiles.

3. `audit2allow` Je transforme les refus détectés en module local `.pp` que je peux ensuite charger pour corriger les permissions SELinux sans toucher à la politique principale.

4. **Rapport centralisé** Le fichier `rapport.log` contient **tous les refus SELinux** et me permet de garder un historique clair et lisible.

5. **Flexibilité** L'utilisateur peut scanner un processus spécifique ou tous les processus du fichier politique, ce qui rend le script pratique pour différents besoins d'audit.
