SELinux: Ecrire un module SELinux (bases)

Table of Contents

Les bases

Un module SELinux est un groupe de déclarations et de règles injectées dans le noyau (il peut être déchargé)

Il couvre généralement les règles de sécurité pour une application donnée

Stratégie simple de génération de module

- Définir un type pour l'exécutable de l'application
- Définir un autre type, qui sera le domaine dans lequel l'application s'exécute (il sera également utilisé pour les fichiers utilisés par l'application)
- Exécuter une première fois l'application en mode permissif. Les accès qui seraient refusés seront alors enregistrés (comme le processus doit s'exécuter dans un domaine non défini ailleurs, tout accès possible aux objets existants est refusé par défaut)
- Utiliser **audit2allow** pour créer des règles correspondant aux messages du journal de refus
- Ajuster les règles si nécessaire

Préparation de l'environnement

- Créer un répertoire pour travailler
- Faire un lien symbolique vers le makefile de développement

```
ln -s /usr/share/selinux/devel/Makefile
```

Construire un module initial

Préparer un fichier source de module initial avec un suffixe .te

L'en-tête de module doit ressembler à cela

```
module haifux 1.0.0;

require{
  type unconfined_t;
  class process {transition sigchld};
  class file {read x_file_perms};
}
```

- La première ligne déclare le nom et la version du module
- La clause require indique quels types et permissions (par classe) le module devrait déjà exister (avant son chargement)

Modèle de module minimal

```
module haifux 1.0.0;
require {
  type unconfined_t;
  class process transition;;
}
type haifux_t;
type haifux_exec_t;
role unconfined_r types haifux_t;
type_transition unconfined_t haifux_exec_t : process haifux_t;
```

Le module haifux.te ci-dessus :

- définit deux nouveaux types, **haifux_t** et **haifux_exec_t** .
- indique également au noyau SELinux que, si un processus du domaine **unconfined_t** exécute un exécutable de type **haifux_exec_t** , il doit continuer dans le domaine **haifux_t**.

Compilation et chargement

Compiler et charger avec make

Pour compiler le module, lancer **make** dans le répertoire de travail.

La commande make créé trois fichiers

- **haifux.pp** est le binaire compilé du module
- **haifux.if** vide à la création il peut contenir les clauses require
- **haifux.fc** contient des informations sur quels fichiers doivent avoir quel contexte.

make install s'assurera que ces contextes sont permanents (survivent au ré-étiquetage).

Les fichiers .fc ressemblent au format de file_context . Une ligne typique serait:

```
/home/eli/myapp.sh - gen_context (system_u: object_r: myapp_exec_t, s0)
```

Afin de charger le module, lancer **make load** en tant que root.

Compiler et charger sans make

```
m4 mymodule-with-macros.te> mymodule.te (s'il y a des macros à ouvrir)
checkmodule -M -m mymodule.te -o mymodule.mod
semodule_package -o mymodule.pp -m mymodule.mod
semodule -i mymodule.pp
```

* La commande semodule charge le binaire du module et doit être exécutée en tant que root.

- Si le module est déjà chargé, il sera mis à jour.
- La compilation de make implique des macros stardard automatiquement mais, la commande m4 ci-dessus ne connaît pas les macros spécifiques à SELinux. Il faut donc les copier dans le module lui-même (une macro doit être définie dans le code qui l'utilise).

Pour supprimer le module utiliser la commande: semodule -r mymodule (en tant que root)

Test de la politique

Par exemple l'application hello.c , qui sera compilée en bonjour

```
    #include <stdio.h>
```

```
    int main() {
       printf ("Bonjour le monde \n");
       return 0;
    }
```

test de hello en mode permissif: basculer dans le mode permissif **(setenforce 0)** et changer du type de hello avec **chcon**

```
setenforce 0
chcon -t haifux_exec_t hello
```

```
./hello
    Bonjour le monde
```

Test en mode enforcing: basculer dans le mode enforcing **(setenforce 1)** et tester hello qui échoue

```
setenforce 1
./hello
bash: ./hello: permission refusée
```

En mode enforcing l'exécution de hello a été refusée car aucune permission n'est définie pour ce type.

```
grep haifux /var/log/audit/audit.log | less
```

La plupart des entrées ont été créées en mode permissif. En mode enforcing, les choses se sont arrêtées lors du premier refus, il faut donc régler les permissions.

Régler les autorisations

Dans un nouveau répertoire de travail (na pas oublier de créer un lien symbolique vers le fichier Make) :

- Laisser **audit2allow** écrire les règles, en fonction des refus d'autorisation (il est nécessaire de filtrer les entrées de journal pertinentes, sinon le module ouvrira toutes les portes à toute tentative tentée sur le système d'où l'utilisation de grep dans notre exemple):

```
grep haifux /var/log/audit/audit.log | \
audit2allow -m haifux> haifux.te
```

- Insérer les déclarations de type du “minimal module” dans celle générée par audit2allow et supprimer les dans la clause require.
- Examiner attentivement le nouveau fichier de règles.
- Compiler et charger comme précédemment

Tout devrait bien fonctionner maintenant en mode enforcing

Déboggage

Repérer les messages d'erreurs AVC et SELINUX_ERR :

```
tail -f /var/log/audit/audit.log | \
grep -E '^type=(AVC|SELINUX\_ERR)'
```

- Les messages AVC se produisent lorsque les autorisations seront refusées
- Les messages SELINUX_ERR impliquent des tentatives pour enfreindre les restrictions de rôle et d'utilisateur.

En mode permissif, ces opérations sont effectuées de toute façon

Si le démon d'audit est désactivé, ces messages iront dans `/var/log/messages`

Messages d'erreurs courantes

Il faut indiquer au compilateur quels types sont définis et lesquels existent déjà.

Si on utilise un type sans le définir ni le demander, une erreur de compilation est retournée

```
haifux.te":24:ERROR 'unknown type haifux_exec_t' at token ';' on line 1028
```

Ou s'il manque un cours:

```
haifux.te":26:ERROR 'unknown type haifux_exec_t' at token ';' on line 1030:
```

Ou si une autorisation est manquante dans les déclarations de classe :

```
haifux.te":45:ERROR 'permission sigchld is not defined for class process' at
token ';' on line 1049:
```

Si on appelle une classe qui n'existe pas la charge du module échouera avec quelque chose comme:

```
libsepol.print_missing_requirements: haifux's global requirements were not
met: type/attribute haifux_t
```

Si on défini un type qui existe déjà):

```
libsepol.scope_copy_callback: unconfined: déclaration en double dans le
module: type/attribut unconfined_t
```

Effet de prison

- Le processus ne peut pas sortir de son domaine, sauf autorisation explicite
- Si on démarre avec un type privé, une telle permission ne peut exister à notre insu
- Si le processus exécute un autre exécutable, il sera exécuté sous le même domaine (en fonction des autorisations, particulièrement **execute_no_trans**
- On peut exiger une transition vers un autre domaine que l'on a créé
- Les processus dans les deux domaines ne peuvent rien toucher, sauf autorisation explicite

Utilisation des macros

Les exemples de règles sont accompagnés de nombreuses macros, qui regroupent des déclarations et des règles pour former un groupe cohérent pour les humains.

Certaines des macros sont documentées dans la section «Configuration de la stratégie SELinux» par la NSA et ailleurs.

Les utilitaires de génération automatique de modules sont plus susceptibles d’utiliser des macros. Ils peuvent être trouvés dans les fichiers source de la politique

Macros: les plus utilisées

- **domain_auto_trans (sshd_t, shell_exec_t, user_t)** : transition de domaine automatique avec les autorisations incluses
- **file_type_auto_trans (sshd_t, tmp_t, sshd_tmp_t)** - transition de type pour les fichiers, autorisations incluses