SELinux: Les Bases

Table of Contents

Présentation

SELinux (**S**ecurity **E**nhanced **Linux**) est un système de contrôle d'accès obligatoire (Mandatory Access Control) qui s'appuie sur l'interface Linux Security Modules fournie par le noyau Linux. Concrètement, le noyau interroge **SELinux** avant chaque appel système pour savoir si le processus est autorisé à effectuer l'opération concernée.

Chargé dans le noyau, le module de sécurité Linux effectue trois tâches, en fonction des règles chargées à partir de l'espace utilisateur (c'est-à-dire la stratégie):

- Accorder ou refuser l'autorisation d'accès aux processus demandant à exécuter une action sur des objets
- Accorder ou refuser l'autorisation pour les modifications de contexte d'objets et de processus.
- Décider quel contexte donner aux nouveaux objets et processus lors de leur création.

Les autorisations **SELinux** sont données en plus des autorisations UNIX classiques. Une action aura lieu uniquement si les deux autorisations sont accordées.

Comment SELinux décide quoi autoriser

Le module de noyau **SELinux** permettra une opération si et seulement si:

- Une règle d'autorisation (**allow** ou **allowaudit**) correspond aux types et aux classes des éléments impliqués.
- Aucune des règles de contraint n'est violée

Remarques:

- Les décisions sont mises en cache dans le cache de vecteur d'accès.
- Dans la politique ciblée, les contraintes sont très basiques, seuls les types ont une signification

SElinux avantages et inconvénients

Les avantage de SELinux

- **Resolution**: accorde les autorisations de l'application si nécessaire, sans plus
- **Ciblage:** Laisse chacun faire ce qu'il veut, à l'exception de quelques applications potentiellement exploitables.
- **Emprisonnement:** l'application n'est pas susceptible d'échapper à son état de permission limitée
- **Flexibilité:** la machine peut être configurée à d'autres fins, telles que le contrôle de l'accès aux informations pour les employeurs.
- **Alerte:** l'administrateur peut détecter un comportement inattendu au tout début d'une attaque (l'adversaire «regardant autour de lui»).

Problèmes avec SELinux

- Compliqué
- Documentation inutile (c'est un euphémisme)
- … et donc très difficile à apprendre
- Le piratage informatique peut créer d’énormes failles de sécurité
- Peut paralyser des applications sans que l'utilisateur comprenne pourquoi
- Est présenté aux utilisateurs finaux avec un message «faites-nous confiance, nous sommes les experts»

- ... et laisse très peu de choix sauf si vous voulez plonger dans

Description du paquet SELinux

Contenu du paquet SELinux

- **Le noyau de sécurité du noyau:** Le LSM (Linux Security Modules)
- **“The example policy”:** règles de sécurité de base utilisées
- **Modules Policy:** règles spécifiques à certaines applications
- **Extension de système de fichiers** pour autoriser des attributs supplémentaires (le contexte) pour chaque fichier.
- **Utilitaires et démons de l'espace utilisateur** interagissant directement avec le LSM.
- **Utilitaires de maintenance** (essentiels pour configurer **SELinux**, mais n'interagissent pas avec le noyau, tels que le compilateur de règles).
- **utilitaires communs:** ls, ps, id, find, etc.

Est-ce que j'ai SELinux?

Si un répertoire `/selinux` est présent et s'il contient quelque chose, **SELinux** est chargé dans le noyau.

Essayer également la commande **sestatus** . Voici ce que l'on obtient par défaut sur Fedora Core 9:

```
sestatus

SELinux status: enabled
SELinuxfs mount: /selinux
Current Mode: Enforcing
Mode from config file: enforcing
Policy Version: 22
Policy from config file: targeted
```

Concepts

La politique

- **Policy** - Ensemble de déclarations et de règles indiquant au noyau **SELinux** du noyau ce qui est autorisé et comment se comporter dans différentes situations.
- **Politique ciblée** - Politique basée sur le paradigme, selon laquelle seules quelques applications sélectionnées doivent être limitées par **SELinux**. Toute autre activité repose sur la bonne vieille sécurité UNIX

- **Politique stricte** - Une politique qui tente de contrôler toutes les activités avec **SELinux**

La politique courante Est une politique ciblée.

la politique est consommée de la façon suivante:

- La politique est compilée dans l'espace utilisateur
- Le préprocesseur de macro m4 est utilisé avant la compilation (facultatif)
- Le binaire de stratégie initial est chargé par init au démarrage
- Les modules de politique (binaires) peuvent être chargés et déchargés à tout moment

Le contexte

- **SELinux** marque chaque processus, fichier, pipe, socket, etc. avec une information appelée contexte .
- **SELinux** autorise ou refuse les actions basées sur des règles disant «un processus de contexte X peut le faire et ainsi de suite en relation avec quelque chose avec le contexte Y».
- Le contexte n'a aucun lien avec l'ID utilisateur UNIX classique, l'ID de groupe ou autre.
- En particulier: les jeux su, sudo et suid-bit ne changent pas le contexte. Pour **SELinux**, on reste ce qu'on était avant.
- En bref: dans **SELinux**, le contexte est primordial.
- Le contexte comprend trois parties: l'utilisateur, le rôle et le type.
- Dans une politique courante, 99% des décisions sont prises en fonction du type uniquement.
- Lorsque le contexte s'applique à un processus, le type s'appelle «le domaine ».
- Il n'y a pas de différence pratique entre un type et un domaine
- Les trois composants ne sont que des noms. Les règles de politique leur donne une signification.
- En particulier, si un objet a le même type qu'un domaine de processus, cela ne signifie quelque chose que si la politique le dit explicitement (c'est généralement le cas).
- Tous les utilisateurs, rôles et types peuvent être appliqués à n'importe quel objet (avec les autorisations), car ce ne sont que des noms.

Mode contraignant ou permissif

- **Enforcing Mode** - Le noyau refuse toute action pour laquelle **SELinux** refuse l'autorisation.
- **Mode permissif** - **SELinux** écrit uniquement les messages du journal de refus, mais le noyau ignore ses refus (seules les autorisations UNIX classiques prennent effet)

Par défaut, tout système sain démarrera en **enforcing mode**,

Pour changer de mode ,utiliser la commande **setenforce** en tant que roout

- **setenforce 0** pour passer en mode permissif
- **setenforce 1** pour passer en mode enforcing

Pour démarrer le système en mode permissif: Utiliser le paramètre d'amorçage du noyau « stricting = 0».

Les rôles

La politique **SELinux** limite les utilisateurs pouvant obtenir quels rôles

Il est courant, mais pas nécessaire, que chaque utilisateur de **SELinux** puisse avoir un rôle unique.

Le rôle limite les domaines (types) dans lequel que son propriétaire peut entrer

RBAC (Contrôle d'accès basé sur les rôles): limite les autorisations des utilisateurs en attribuant des rôles, ce qui limite leur variété de types et, partant, leurs actions.

```
seinfo -r # affichera tous les rôles connus du système
```

La stratégie Linux courante est Type Enforced (TE), les rôles et les utilisateurs ont donc peu d'importance:

- Lors de la connexion (pas su), le processus shell se voit attribuer un utilisateur **SELinux** et un rôle, généralement **unconfined_u** et **unconfined_r**. Celles-ci resteront probablement tout au long de la session pour tous les processus enfants.
- Les processus créés par init ou crond sont susceptibles de recevoir **system_u** et **system_r**

Objets et classes SELinux

Le terme «objet» dans **SELinux** désigne les fichiers, les répertoires, les descripteurs de fichier, les tubes, les sockets, les interfaces réseau, etc.

Un objet est la chose qu'un processus demande la permission de faire quelque chose

Il y a plus de 70 classes d'objets **SELinux**

Chaque classe définit les autorisations applicables

Il existe une classe “process”, mais dans le jargon, un processus n'est généralement pas considéré comme un objet.

Les mécanismes de Sécurité

On distingue les mécanismes de niveau (**level=MLS**) des mécanismes de catégorie (**MCS**)

Ces mécanismes sont destinés à empêcher les utilisateurs de divulguer des informations par erreur.

Par exemple, l'application de messagerie peut être empêchée de lire des fichiers sensibles

Pour connaire quel mécanisme activé

```
cat /selinux/mls
1
```

Ces mécansimes sont implémentés avec les règles **mlsconstraint** dans les fichiers mls et mcs du répertoire source de la politique

MLS et **MCS** est le quatrième élément du contexte (s0 dans l'exemple ci-dessous)

```
ls -Z
drwxrwxr-x eli eli unconfined_u: object_r: user_home_t: s0 mydir
-rw-rw-r-- eli eli unconfined_u: object_r: user_home_t: s0 myfile
```

Étiquetage de fichier

Le contexte est stocké pour chaque fichier en tant qu'attributs sur un système de fichiers étendu, XFS (man attr).

La politique inclut des règles qui déterminent les types de fichiers à la création.

Les systèmes de fichiers qui ne peuvent pas porter d'attributs étendus obtiennent un contexte uniforme, en fonction des options de l'opération de montage et des fichiers de configuration système (par exemple, VFAT, NFS, Samba, ISO)

tar ne stocke et n'extrait pas de contextes sauf si des indicateurs explicites sont donnés

Le ré-étiquetage des fichiers peut être effectué avec plusieurs utiliatires Les contradictions entre les règles de stratégie et le ré-étiquetage des fichiers de configuration sont possibles et dangereuses:

L'utilitaire setfiles

définit le contexte dans tous les fichiers, en fonction d'un fichier de configuration (généralement `/etc/selinux/targeted/contexts/files/file_contexts`).

restorecon fait la même chose que setfiles , mais est destiné à quelques fichiers seulement (principalement pour corriger de petites incohérences)

L'utilitaire semanage fcontext

Permet de modifier de manière permanente le contexte des fichiers et des répertoires (expression régulière).

L'installation d'un module de stratégie peut également modifier les contextes de fichier de manière permanente

On peut utiliser **chcon** pour modifier le contexte de certains fichiers sans modifier les fichiers de configuration, mais cette modification est temporaire jusqu'au prochain changement d'étiquette.

Règles et syntaxe

La règle "allow"

allow Source Target:Class Permission; accorder une autorisation à un processus de domaine (type) Source sur des objets de type Cible et classe Classe

```
allow unconfined_t mytype_t:file read ;
```

signifie “autoriser les processus dans le domaine (type) unconfined*t permission de lecture sur les fichiers de type mytype*t” '

L'utilitaire **audit2allow** permet d'écrire les règles d'autorisation

Autres règles

- **auditallow** - Exactement comme allow , mais effectue des entrées dans le journal (comme dans les refus)
- **dontaudit** - Ceci n'accordera pas la permission, mais n'enregistrera rien non plus
- **neverallow** - Pas vraiment une règle, mais indique au compilateur de règles de quitter avec une erreur, si les autorisations spécifiées sont accordées par d'autres règles. Utilisé comme une protection supplémentaire contre les bugs dans la politique

À l'exception du mot-clé d'ouverture, ces règles précédents ont la même syntaxe que allow

En cas de contradiction entre les règles, la dernière règle prend effet.

Les transitions de type

Les transitions de type permettent d'affecter un autre type à un un domaine source

Les transitions de type ne sont pas des règles de permission. Au contraire, elles indiquent à **SELinux** d'effectuer une action.

Les Transitions de type pour les objets

Chaque objet créé a un contexte par défaut

Par exemple, les fichiers et les répertoires sont créés par défaut avec le contexte de leur répertoire parent.

Cependant il est souhaitable que le type du nouvel objet dépende de celui qui l'a créé.

Par exemple: Si le serveur X crée un fichier dans le répertoire /tmp , il doit être de type **xdm_tmp_t** , mais si un processus «utilisateur normal» le fait, il doit s'agir de **user_tmp_t**.

Pour permettre cela on utilise la directive **type_transition Source Source: Class new_type** : tout objet de la classe Class, créé par un processus du domaine (type) Source et obtenant par défaut le type Target, obtiendra le type new_type à la place)

Par exemple :

```
type_transition sshd_t tmp_t: fichier sshd_tmp_t;
```

Signifie que si un processus s'exécute dans le domaine sshd_t (très probablement le ssh deamon) crée un fichier normal ordinaire qui aurait dû recevoir le type **tmp_t** (probablement dans le répertoire /tmp), il devrait recevoir l'attribut **sshd_tmp_t** à la place .

La transition de type pour les objets ne nécessite pas de règle d'autorisation supplémentaire.

Mais plusieurs autres actions nécessitent une autorisation:

- Accès en lecture-écriture au répertoire parent
- Création d'un nouveau fichier ou répertoire avec le nouveau type

Pour simplifier les choses, une macro regroupe l'instruction de transition de type avec les autorisations: **file_type_auto_trans**

En reprenant le dernier exemple, la macro-instruction suivante couvre une variété de types de fichiers (fichiers simples, répertoires, liens symboliques, etc.) et gère également les autorisations. Tout en un:

```
file_type_auto_trans (sshd_t, tmp_t, sshd_tmp_t);
```

Les Transitions de domaine pour les processus

Avec la directive **type*transition Source Target:process new*type;** lorsqu'un processus du

domaine Source exécute, appel exec(), un fichier de type Cible, le domaine du processus sera changé en new_type.

Exemple:

```
type_transition sshd_t shell_exec_t: process user_t;
```

Signifie que si un processus du domaine sshd_t exécute un exécutable de type shell_exec_t (un shell, très probablement), le processus se poursuivra dans le domaine user_t .

Pour les processus, l'instruction type_transition n'inclut pas l'autorisation.

De nombreuses autorisations doivent être explicitement déclarées: la transition elle-même, la lecture et l'exécution de l'exécutable, et bien plus encore.

La **macro domain_auto_trans** inclut l'instruction de transition de type et un grand nombre d'autorisations pertinentes (par exemple, autoriser une conduite de canal entre les deux domaines pertinents).

Donc, au lieu de l'exemple précédent, on utilisera donc l'instruction:

```
domain_auto_trans (sshd_t, shell_exec_t, user_t);
```

En l'absence d'une règle de transition correspondante, l'exécutable s'exécutera sans changer de domaine. Cela nécessite l'autorisation execute_no_trans

Utilisation des caractères géneriques

Les caractères génériques peuvent être utilisés dans la défition de la politique

- **Les accolades { et }** avec des éléments délimités par des espaces signifient “pour chaque élément”
- **Le caractère tilde** précédant une expression indique le complément de l'ensemble
- **L'astérisque** représente tous les types ou classes
- **Un signe moins** précédant un élément, dans une expression entre accolades réduit l'élément de l'ensemble

Exemples:

```
allow unconfined_t mytype_t: fichier {read getattr};
allow unconfined_t mytype_t: file *;
```

Déclarations

Déclaration des rôles

- **role ROLE types TYPE;** permet de déclarer “il est légal qu'un contexte de processus avec le rôle ROLE soit dans le domaine TYPE”

Les ensembles peuvent être utilisés pour le type, mais pas pour le rôle

Une tentative d'entrer dans un domaine avec un rôle non autorisé provoquera une erreur «contexte invalide».

Pour lister les types actuellement connus par le noyau utiliser la commande :

```
seinfo -r
```

Exemple:

```
role unconfined_r types mytype_t;
```

Déclaration des contraintes (constraints)

Chaque demande d'autorisation :

- doit obéir à toutes les contraintes actuellement actives dans le noyau.
- il nee devrait pas être nécessaire d'en déclarer dans un module de politique

Comme ce n'est pas si pertinent, prenons un exemple:

```
constrain process transition ( u1 == u2 or t1 == privuser );
constrain process transition ( r1 == r2 or t1 == privrole );
constrain dir_file_class_set { create relabelto relabelfrom } ( u1 == u2 or
t1 == privowner );
```

Il existe également **mlsconstraint** , qui limite les autorisations liées à MLS

Déclaration des types

- **type identifier attributelist ;** déclare le type avec de nom identifier et une liste d' attributs (facultative).

Exemples :

```
type mytype_t;
type crond_t, domain, privuser, privrole, privfd, privowner;
```

Compte tenu de la déclaration de type ci-dessus, si lun des attribut domain, privuser, privrole, privfd ou privowner est utilisé là où la syntaxe attend un type, cela inclura plusieurs types, y compris crond_t.

Déclaration des attributs et attribuer du texte

- **attribute myattributename;** permet de déclarer ses propres attributs
- **typeattribute mytype/_t theattribute;** permet de donner à un type un attribut dans une instruction séparée: