SELinux

SElinux est un mécanisme de contrôle d'accès obligatoire (MAC) pour les distributions GNU/Linux. C'est un développement dans la continuité de FLASK

Répertoires

/sys/fs/selinux Le système de fichier SELinux qui interface avec le service de sécurité du kernel.

/etc/selinux La configuration pour selinux

/var/lib/selinux/<SELINUXTYPE>/module Modules de stratégie SELINUX

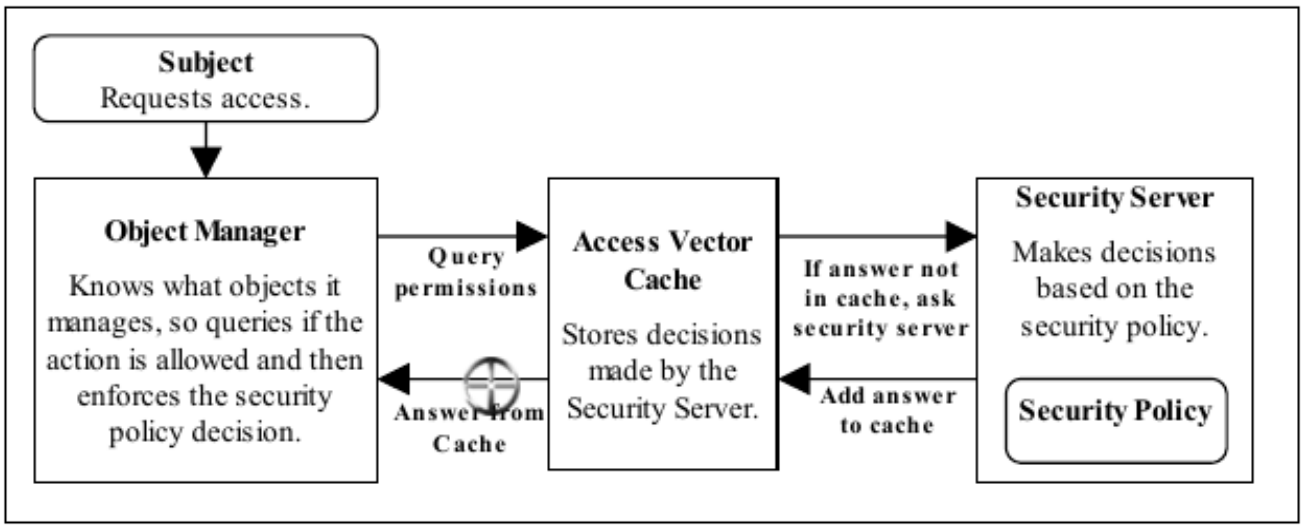
Utilité

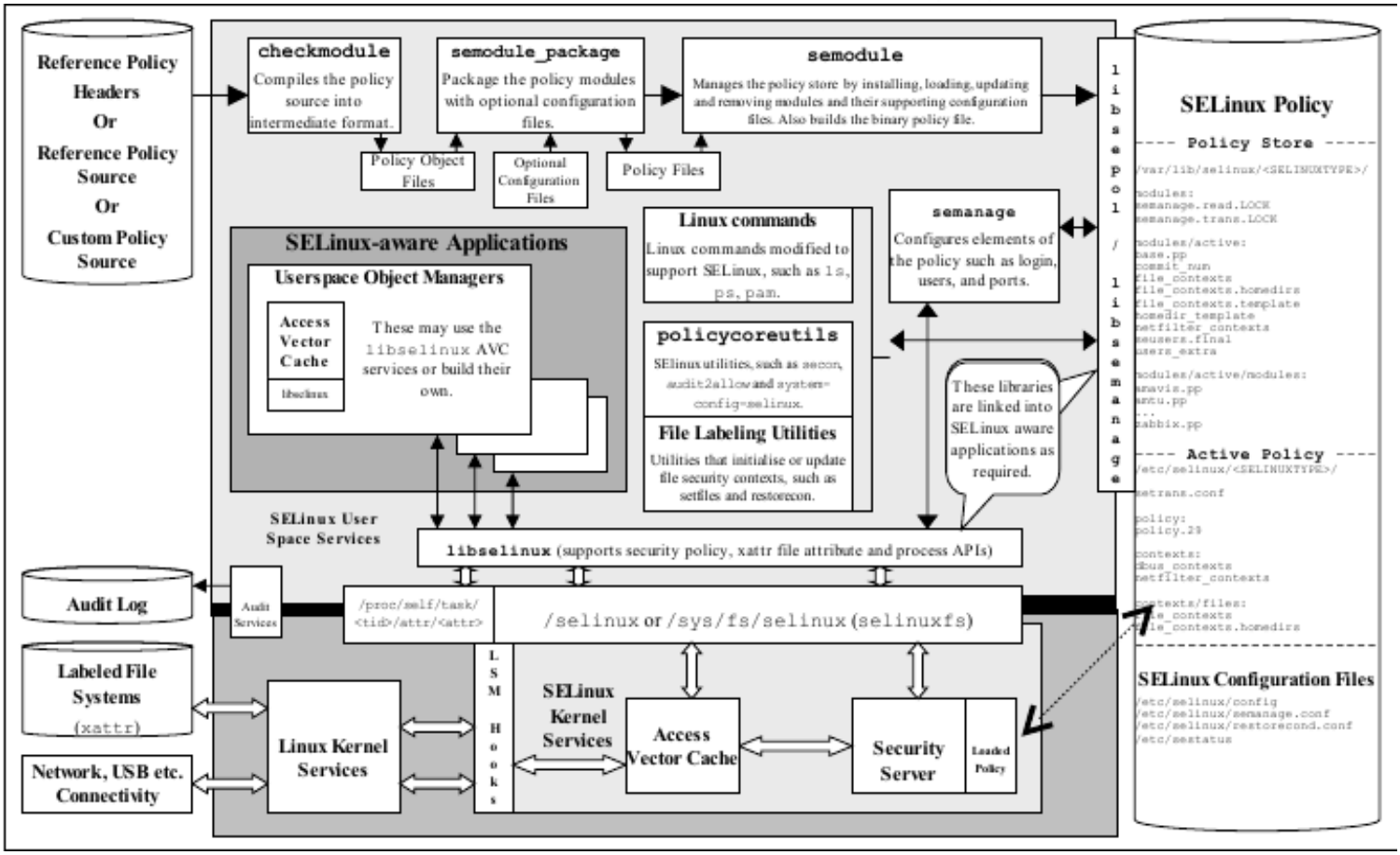
- Si SELinux est activé, la stratégie définie quels accès aux ressource et les opérations sur ces ressources (ex : read, write) sont autorisés. C'est un mécanisme MAC
- Le concept de stratégie, l'implémentation et les tests avec une stratégie de sécurité définie ou les pre-requis sont importants.
- SELinux pour confiner une application avec son propre domaine et lui permettre d'avoir les privilèges minimum pour faire son job. Si l'application nécessite un accès aux réseaux ou d'autres application (ou leur données), alors cet accès est donné.
- Si une application 'fait quelque chose" qui n'est pas permis par stratégie, SELinux peut stopper ces actions
- Si une application fait quelque chose qui est permis par stratégie, SELinux peut contenir tout dégât qui peut être fait intentionnellement ou non. Par exemple, si une application est autorisée à supprimer tout ses fichiers de données ou entrées de base de données et qu'un bug, virus ou utilisateur gagne ces privilèges, il peut faire la même chose, cependant, si cette stratégie confine l'application et les données, toutes les autres données ne seront pas touchées.
- Les sessions login utilisateur peuvent être confinés dans leur propre domaine. Cela permet aux clients qu'ils lancent d'avoir seulement les privilèges nécessaires. Cela confine/limite également tout dommage ou fuite de données.
- Certaines applications sont difficiles à confiner parce qu'elles sont généralement conçues pour avoir un accès total à toutes les ressources. SELinux peut généralement gérer ce problème en fournissant des services sandboxing.
- SELinux ne stoppe pas les fuites mémoire, ou débordement de tampon, cependant il peut contenir les dommages qui peuvent être faits.
- SELinux ne stop pas tous les virus/malware dans le système, cependant il devrait limiter les dégats ou fuites qu'ils causent.
- SELinux ne stop pas les vulnérabilités, cependant, il peut limiter leurs effets.
- Il est facile d'ajouter de nouvelles règles à une stratégie SELinux en utilisant des outils tels que audit2allow(1) si un utilisateur a les permissions nécessaires.
- SELinux ne peut pas stopper ce qui est autorisé par stratégie, donc un bon design est important.

Composants du cœur SELinux

- Un sujet qui doit être présent pour déclencher une action à prendre par un objet.
- Un gestionnaire d'objet qui connaît les actions requises pour la ressource particulière et peut forcer ces actions
- Un serveur de sécurité qui prend les décisions au regard des droits du sujet pour effectuer l'action requise sur l'objet, basé sur les règles de stratégie de sécurité

- Une Stratégie de sécurité qui décris les règles utilisant le language de stratégie SELinux
- Un cache de vecteur d'accès qui améliore les performances système.





Le schéma ci-dessus montre un diagramme plus complexe du kernel et userspace avec les services support qui sont utilisés pour gérer l'environnement SELinux. En partant du bas :

- Dans l'implémentation actuelle de SELinux le serveur de sécurité est embarqué dans le kernel et les stratégies sont chargées depuis le userspace via une série de fonctions contenues dans la librairie libselinux. Le gestionnaire d'objet (OM) et le cache de vecteur d'accès (AVC) peuvent résider dans :

  kernelspace Cette gestion d'objet est pour les services kernel tel que les fichiers, répertoires, socket, IPC, etc. et sont fournis par des hooks dans le sous-système SELinuyx via le framework LSM. Le service AVC du kernel est utilisé pour maintenir en cache les réponses du serveur de sécurité.

userspace Cette gestion d'objet est fournie avec l'application ou service qui nécessite un support pour MAC et sont des applications "SELinux-aware". Par exemple, X-Windows, D-bus, PostgreSQL, nscd, et passwd. Généralement, ces OM utilisent les services AVC construits dans la librairie SELinux, cependant il peuvent fournir leur propre AVC si besoin.

- La stratégie de sécurité SELinux et ses fichiers de configuration sont contenus dans /etc/selinux. Ce répertoire contient le fichier de configuration principal qui a le nom du la stratégie à charger et le mode d'application de la stratégie au chargement.

SELinux supporte une stratégie modulaire, ce qui signifie qu'une stratégie n'a pas à être une grande stratégie, mais peut être constru

-

Pour être capable de construire la stratégie, une source de stratégie est requise, qui peut être fournie de 3 manières :

- Un code source écrit en langage de stratégie SELinux.
- En utilisant la statégie référence, qui a des macros haut-niveau pour définir les règles.
- En utilisant CIL (Common Intermediate Language).

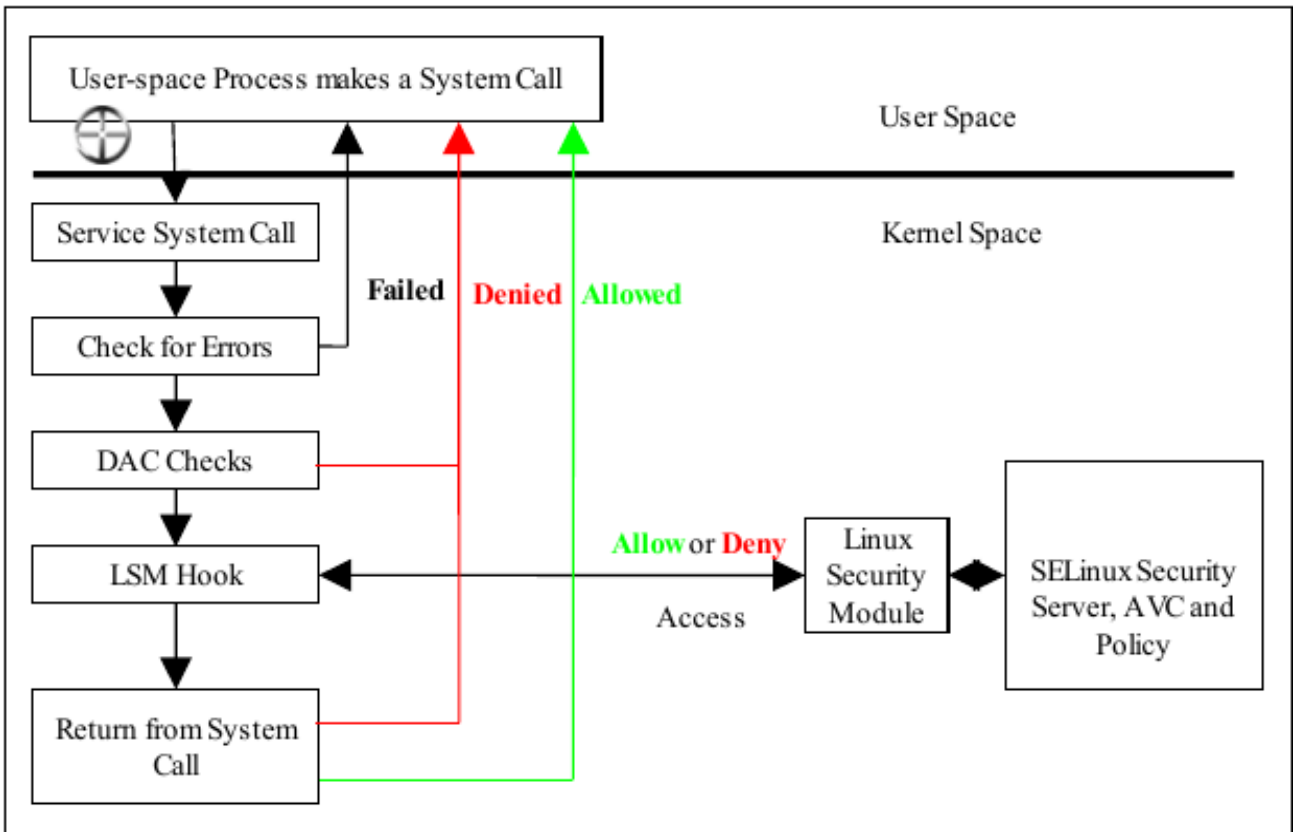
- Pour pouvoir compiler les sources de stratégie puis les charger dans le serveur de sécurité, un certains nombre d'outils sont requis
- Pour permettre à un administrateur de gérer les stratégies, l'environnement SELinux et les systèmes de fichier labélisés, outils et commandes gnu/linux modifiés sont utilisées.
- Pour s'assurer que les évènements de sécurité sont loggés, gnu/linux a un service d'audit qui capture les violations de stratégie.
- SELinux supporte les services réseaux.

Mandatory Access Control (MAC)

MAC est un type de contrôle d'accès dans lequel le système d'exploitation est utilisé pour contraindre un utilisateur ou processus (le sujet) de l'accès ou un opération sur un objet (tel qu'un fichier, mémoire, etc.). Chaque sujet et objet a un jeu d'attributs de sécurité qui peuvent être interrogés par l'os pour vérifier si l'opération demandée peut être effectuée ou non. Pour SELinux :

- Les sujets sont des processus
- Les objets sont des ressources système tels que des fichiers, sockets, etc.
- Les attributs de sécurité sont des contextes de sécurité
- Le serveur de sécurité dans le kernel autorise l'accès ou non en utilisant la stratégie de sécurité qui décris les règles définies.

Noter que le sujet ne peut pas décider de bypasser les règles de stratégie définie par la stratégie MAC avec SELinux. En contraste avec DAC de Linux, qui gouverne également la capacité des sujets à accéder aux objets, cependant il permet aux utilisateurs de décider de la stratégie. Les étapes dans les décisions DAC et MAC :



SELinux supporte 2 formet de MAC :

Type Enforcement Où les traitements lancés dans les domaines et les actions sur les objets sont contrôlés par la stratégie. C'est l'implémentation utilisée pour MAC dans SELinux avec RBAC

Multi-Level Security C'est une implémentation basée sur le modèle BLP, et utilisée par les organisations où différents niveaux d'accès sont requis pour que les informations restreintes soient séparées des informations classifiée pour maintenir la confidentialité.

Les services MLS/MCS sont utilisés pour maintenir une séparation d'applications, par exemple :

- Les machines virtuelles utilisent les catégories MCS pour permettre à chaque VM de se lancer dans son propre domaine.
- Les périphériques Android utilisent les catégories MCS pour qu'une application qui tourne à la demande d'un utilisateur ne puisse pas lire ou écrire des fichiers créé par la même application à la demande d'un autre utilisateur.

Utilisateurs SELinux

Les utilisateur dans gnu/linux sont généralement associés avec des utilisateurs humain (comme Alice et Bob) ou des fonctions d'opérateur/système (comme admin), bien que celà puisse être implémenté dans SELinux, les noms d'utilisateur SELinux sont généralement des groupes ou des classes d'utilisateur. Par exemple, tous les utilisateurs système standard peuvent être assignés à un nom d'utilisateur user_u et l'équipe d'administration sous staff_u.

Il y a un utilisateur SELinux spécial définis qui ne doit jamais être associé à un utilisateur gnu/linux vu qu'il est une identité spéciale pour les processus et objets système, cet utilisateur est system_u.

Le nom d'utilisateur SELinux est le premier composant d'un contexte de sécurité et par convention les noms d'utilisateur SELinux se terminent en '_u', cependant ce n'est pas obligatoire.

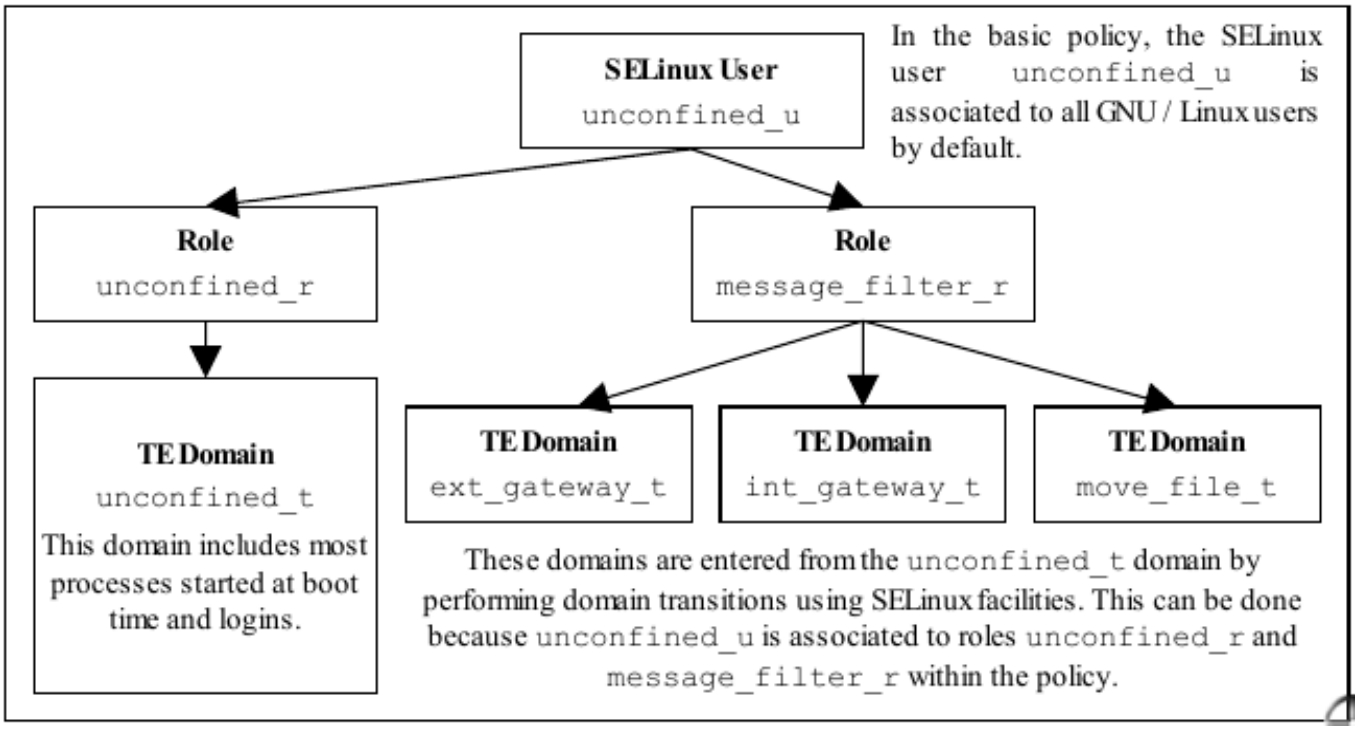
Il est possible d'ajouter des contraintes et limites dans les utilisateurs SELinux.

RBAC

Pour contrôler l'accès aux domaines, TE SELinux utilise RBAC. Cette fonctionnalité permet aux utilisateurs de SELinux d'être associés à un ou plusieurs rôles, où chaque rôle est ainsi associé à un ou plusieurs types de domaine.

Le nom du rôle SELinux est le second composant d'un contexte de sécurité, et par convention les rôle SELinux se terminent par '_r'.

Il est possible d'ajouter des contraintes et limites sur les rôles.



Type Enforcement

SElinux utilise un style spécifique de technologie TE pour forcer le contrôle d'accès obligatoire. Pour SELinux cela signifie que tous les sujets et objets ont un identifiant de type associé avec eux qui peut être utilisé pour forcer les règles établies par stratégie.

L'identifiant de type SELinux est une simple chaîne de longueur variable qui est définie dans la stratégie et associée à un contexte de sécurité. Il est également utilisé dans la majorité des règles et déclaration du langage SELinux utilisés pour construire un stratégie qui va, une fois chargée dans le serveur de sécurité, forcer la stratégie via les gestionnaires d'objet.

Parce que l'identifiant de type (ou simplement type) est associé à tous les sujets et objets, il peut parfois être difficile de distinguer le type actuellement associé. Il revient de comprendre comment ils sont alloués dans la stratégie elle-même et comment ils sont utilisé par les services SELinux.

Basiquement si l'identifiant de type est utilisé pour référencer un sujet il réfère à un processus Linux ou une collection de processus (un domaine ou un type de domaine). Si l'identifiant de type est utilisé pour référencer un objet alors il spécifie son type d'objet (ex : type de fichier).

Bien que SELinux réfère à un sujet comme étant un processus actif qui est associé à un type de domaine, le périmètre d'un domaine TE SELinux peut varier largement. Par exemple dans la stratégie simple construite dans le répertoire basic-selinux-policy des sources, tous les processus dans le système tournent dans le domaine unconfined_t, et donc tout processus est de type unconfined_t (qui signifie qu'il peut faire ce qu'il veut dans les limites de la stratégie DAC Linux vu que tous les accès sont autorisés par SELinux).

C'est seulement quand des déclarations de stratégie additionnelles sont ajoutées à cette stratégie simple que les zone commencent à être confinées. Par exemple, un passerelle externe est lancée dans son propre domaine isolé (ext_gateway_t) qui ne peut pas interférer avec un processus unconfined_t (excepté pour lancer ou transiter le processus gateway dans son propre domaine). Ce scénario est similaire à la stratégie targeted livrée en standard dans Hed Hat Fedora où la majorité des processus en userspace tournent dans le domaine unconfined_t.

Le type SELinux est le troisième composant d'un contexte de sécurité et par convention les types SELinux se terminent par '_t'.

Contraintes

Il est possible d'ajouter des contraintes sur les utilisateurs, rôles, types, et plages MLS, par exemple dans un environnement TE, la manière dont les sujets sont autorisés à accéder à un objet se fait via une règle allow, par exemple :

```
allow unconfined_t ext_gateway_t : process transition;
```

Cela status qu'un processus tournant dans le domaine unconfined_t a la permission de transiter un processus au domaine ext_gateway_t. Cependant on pourrait souhaiter contraindre cela et statuer que cela se produit seulement si le rôle du domaine source est le même que le rôle du domaine cible :

```
constrain process transition ( r1 == r2 );
```

Qui status qu'une transition de processus peut seulement se produire si le rôle source est le même que le rôle cible, cependant un contrainte est une condition qui doit être satisfaite pour une ou plusieurs permission à autoriser (une contrainte impose des restrictions additionnelles aux règles TE). Noter que la contrainte est basée sur une classe d'objet (un processus dans ce cas), et une ou plusieurs de ses permissions.

Limites

Il est possible d'ajouter des limites aux utilisateurs, rôles, et types, cependant actuellement les types sont forcés par le kernel en utilisant la règle typebounds.

Contexte de sécurité

SELinux exige un contexte de sécurité à associer avec tout processus (ou sujet) et objet qui sont utilisé par le serveur de sécurité pour décider si l'accès est autorisé ou non comme définis par la stratégie.

Le contexte de sécurité est également connu comme un label de sécurité, ou simplement label qui peut créer la confusion vu qu'il y a de nombreux types de labels dépendants du contexte.

Dans SELinux, un contexte de sécurité est représenté comme un chaîne de longueur variable qui définis l'utilisateur SELinux, son rôle, un identifiant de type et une plage de sécurité MCS/MLS optionnel :

```
user:role:type[:range]
```

user L'identité de l'utilisateur SELinux. Peut être associé à un ou plusieurs rôles que l'utilisateur SELinux est autorisé à utiliser

role Le rôle SELinux. Cela peut être associé à un ou plusieurs types que l'utilisateur SELinux est autorisé à accéder

type Quand un type est associé avec un processus, il définis quels processus (ou domaines) l'utilisateur SELinux (le sujet) peut accéder.

range Ce champ peut également être appelé level et est seulement présent si la stratégie supporte MCS ou MLS. L'entrée peut consister de :

- Un niveau de sécurité simple qui contient le niveau de sensibilité et 0 ou plusieurs catégories (ex : s0, S1 :c0, s7 :c10.c15
- Une plage qui consiste de 2 niveaux de sécurité (un faible et un élevé) séparés par un '-' (ex : s0 - s15 :c0.c1023)

Cependant, noter que :

1 les décisions d'accès au regard du sujet utilisent tous les composant du contexte de sécurité.

2 Les décisions d'accès à un objet utilise les composant comme suit :

a) L'utilisateur est soit un jeu d'utilisateur spécial appelé system_u ou définis à un utilisateur SELinux du processus créant. Il est possible d'ajouter des contraintes sur les utilisateurs dans la stratégie basée sur leur classe d'objet (un exemple de cela est la stratégie de référence UBAC)

b) Le rôle est généralement définis à un rôle SELinux interne spécial object_r, bien que la stratégie supporte les transitions de rôle dans toute classe d'objet. Il est ainsi possible d'ajouter des contraintes dans les rôles dans la stratégie basée sur leur classe d'objet.

L'exemple ci-dessous montre les contextes de sécurité pour les processus, répertoires et fichier (noter que la stratégie ne supporte pas MCS ou MLS)

```
LABEL    PID TTY CMD
unconfined_u:unconfined_r:unconfined_t 2539 pts/0 bash
unconfined_u:message_filter_r:ext_gateway_t 3134 pts/0 secure_server
unconfined_u:message_filter_r:int_gateway_t 3138 pts/0 secure_server
unconfined_u:unconfined_r:unconfined_t 3146 pts/0 ps
```

Exemple de contexte de sécurité d'objet :

```
system_u:object_r:in_queue_t /usr/message_queue/in_queue
system_u:object_r:out_queue_t /usr/message_queue/out_queue
```

Il y a les contextes de sécurité d'objet de file de message pris depuis la commande ls -Z :

```
/usr/message_queue/in_queue:
unconfined_u:object_r:in_file_t Message-1
unconfined_u:object_r:in_file_t Message-2
/usr/message_queue/out_queue:
unconfined_u:object_r:out_file_t Message-10
unconfined_u:object_r:out_file_t Message-11
```

Sujets

Un sujet est une entité active généralement sous la forme d'une personne, processus, ou périphérique qui cause l'information de transiter dans les objets, ou changent l'état système.

Dans SELinux un sujet est un processus actif et a un contexte de sécurité qui lui est associé, cependant un processus peut également être référé à un objet en fonction du contexte dans lequel il est pris, par exemple :

1. Un processus en cours de fonctionnement est un sujet parce il créé un flux d'informations ou peut changer l'état système
2. Le processus peut également être référé à un objet parce que chaque processus a une classe objet associé appelé process. Cet objet process définis quelles permissions la stratégie est autorisée à donner ou refuser dans le processus actif.

Dans SELinux les sujets peuvent être :

trusted Généralement ce sont des commandes, applications, etc. qui ont été écrits ou modifiés pour supporter SELinux. Cependant, il peut également couvrir toute application qui est trusté considéré comme faisant partie du système. Bien que - en fonction de votre niveau de paranoïa - la meilleur stratégie est de ne rien truster tant que la conformité avec la stratégie n'a pas été vérifiée. Généralement ces application trustées se lancent soit dans leur propre domaine (ex le service audit peut être sous auditd_t) ou groupés ensemble (ex semanage et semodule sont groupés sous semanage_t).

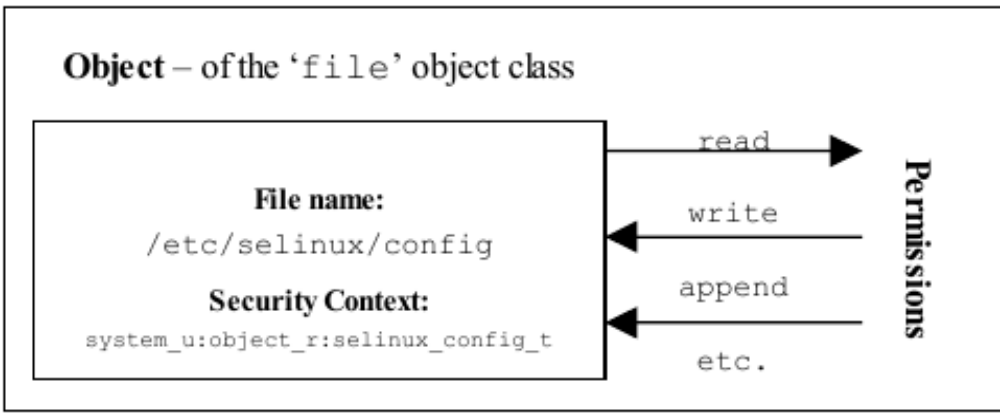
untrusted Tout le reste

Objets

Dans SELinux un objet est une ressource telle que des fichiers, sockets, pipes, ou interfaces réseaux qui sont accédés via les processus (les sujets). Ces objets sont classés en accord avec la ressource qu'ils fournissent avec les permissions d'accès significatif pour leur but (ex : lire, reçevoir et écrire), et assigné à un contexte de sécurité.

Classes objet et permissions

Chaque objet consiste d'un identifiant de classe qui définis son but (ex : file, socket) avec un jeu de permissions qui décrivent quels services l'objet peut gérer. Quand un objet est instantié, un nom et un contexte de sécurité lui sont alloués.



L'objectif de la stratégie est de permettre à l'utilisateur de l'objet (le sujet) d'accéder aux permissions minimum nécessaires pour accomplir la tâche.

Ces classes d'objet et leur permissions associées sont construits dans le kernel GNU/Linux et les gestionnaires d'objet en userspace et sont donc généralement mis à jours par ceux qui écrivent les stratégies.

Ces classes d'objet consistent de classes objet kernel (pour gérer les fichiers, sockets, etc) plus les classes d'objet en userspace pour les gestionnaires d'objet userspace (pour les services tels que X-Window ou dbus). Le nombre de classes d'objet et leur permissions peut varier en fonction des fonctionnalité configurées dans GNU/Linux.

Autoriser un processus à accéder aux ressources

C'est un simple exemple qui tente d'expliquer 2 points :

1. Comment un processus reçois la permission d'utiliser une ressource
2. En utilisant la classse objet 'process', montrer qu'un processus peut être décris comme un processus ou un objet.

Une stratégie contient de nombreuses règles et déclaration, la majorité sont les règle allow qui autorise les permissions d'accès à des processus sur des ressources.

La règle allow suivante illustre une processus qui peut être également un objet vu qu'il permet aux processus dans le domaine unconfined_t, la permission transition de l'application gateway externe dans la domaine ext_gateway_t une fois qu'il a été exécuté :

```
allow Rule | source_domain | target_type : classs | permission
```

```
allow unconfined_ ext_gateway_t : process transition;
```

allow Défini une règle allow

unconfined_t Le domaine source (ou sujet) dans ce cas le shell qui souhaite exécuter l'application gateway

ext_gateway_t L'objet cible, l'instance objet du processus gateway

process La classe objet de la cible

transition La permission est donné au domaine source dans l'objet cible. Dans ce cas, le domaine a la permission transition dans l'objet process ext_gateway_t

labeliser les objets

Dans un système GNU/Linux, labéliser les objets est géré par le système et généralement caché aux utilisateurs. Vu que les processus et les objets sont créés et détruits, soit :

1. Ils héritent leur labels du processus ou objet parent
2. Le type de stratégie, rôle et plage de transition permettent un label différent.
3. Les applications gérant SELinux peuvent forcer un nouveau label (avec approbation des stratégies) en utilisant les fonctions de l'API libselinux
4. Un gestionnaire d'objet (OM) peut forcer un label par défaut qui peut soit être construit dans l'OM ou obtenu via un fichier de configuration
5. Utiliser un identifiant de sécurité initial (ou initial SID). Il sont définis dans toute stratégie de base et monolitiques et sont utilisé pour définis soit un contexte initial durant le processus de boot, ou si un objet nécessite un défaut.

Le langage de stratégie SELinux supporte les déclaration de labélisation pour les services fichier et réseau.

La labélisation des systèmes de fichier qui implémentent les attributs étendus sont supportés par SELinux en utilisant :

1. La déclaration fs_use_xattr dans la stratégie pour identifier quels système de fichier utilisent les attributs étendus. Cette déclaration est utilisée pour informer le serveur de sécurité de la manière de labéliser le système de fichier.
2. Un fichier 'file contexts' qui définis quels contexts initiaux devraient être pour chaque fichier et répertoires dans le système de fichier.
3. Une méthode pou initialiser le système de fichier avec des attributs étendus. fixfiles et setfiles sont utilisés pour celà. Il y a également des commandes comme chcon, restorecon, et restorecond.

Les attributs étendus contenant le contexte SELinux d'un fichier peut être vu par les commande ls -Z et getfattr.

Copier et déplacer des fichiers

En assumant que les permissions ont été données correctement par la stratégie, les effets dans le contexte de sécurité d'un fichier quand il est copié ou déplacé sont :

Copie d'un fichier - prend le label du nouveau répertoire

Déplacement d'un fichier - Conserve le label du fichier

Cependant, si le service restorecond fonctionne et que le fichier restorecond.conf est configuré correctement, d'autres contextes de sécurité peuvent être associés au fichier dans ces cas. Noter qu'il y a également la commande install qui supporte -Z pour spécifier le contexte.

Labéliser des sujets

Dans un système GNU/Linux, les processus héritent du contexte de sécurité du processus parent. Si le nouveau processus lancé a la permission de changer son contexte, une transition de type est autorisé. Le langage de stratégie supporte plusieurs déclaration pour assigner les composant aux contextes de sécurité tel que des déclarations **user**, **role**, et **type**, et gère leur scope **role_allow**, et **constrain**, et gère leur transition **type_transition**, **role_transition** et **range_transition**.

Réutilisation d'objet

GNU/Linux créé des instances des objets et gère les informations qu'ils contiennent (read, write, modify, etc.) sous le contrôle des processus, et à certaines étapes ces objets peuvent être supprimés ou relâchés permettant à la ressource d'être de nouveau disponible.

GNU/Linux gère la réutilisation d'objet en s'assurant que quand une ressource est ré-allouée elle est correcte. Cela signifie que quand un processus relâche une instance d'objet (par exemple relâcher de la mémoire allouée dans le pool, supprimer une entrée répertoire ou fichier), Il peut y avoir des informations restantes. Si c'est un problème, le processus lui-même devrait effacer ou détruire les informations avant de relâcher l'objet.

Calculer les contextes de sécurité

SELinux utilise des déclarations de langage de stratégie et les fonctions de libselinux pour calculer un contexte de sécurité via le serveur de sécurité.

Quand les contextes de sécurité sont calculés, le kernel, outils en userspace et versions de stratégie peuvent influencer la sortie. À cause des patches qui ont été appliqués pendant des années qui donnent une grande flexibilité dans le calcul des contextes. Le contexte de sécurité est calculé pour un objet en utilisant les composants suivants : un contexte source, un contexte cible et une classe objet.

Calcule de contexte de sécurité pour les objets Kernel

La tâche initiale commence avec le contexte de sécurité kernel, mais le processus 'init' va typiquement faire une transition dans son propre contexte unique (ex : init_t), quand le binaire init est exécuté après que la stratégie ait été chargée. Certains programmes init se ré-exécutent eux-même après avoir chargé la stratégie, alors que dans d'autres cas le chargement de la stratégie initiale est effectuée par le script initrd/initramfs avant de monter le vrai root et exécuter le vrai init.

Les processus héritent de leur contexte de sécurité comme suit :

1. Lors d'un fork, un processus hérite du contexte de sécurité de son créateur/parent
2. Lors de l'exécution, un processus peut transiter vers un autre contexte de sécurité basé sur les déclarations de stratégie : type_transition, range_transition, role_transition, default_user, default_role, default_range et default_type ou si un processus compatible SELinux appel setexeccon(3) si autorisé par la stratégie avant d'invoquer exec.
3. À tout moment, un processus compatible SELinux peut invoquer setcon(3) pour basculer son contexte de sécurité (si permis) bien que cette pratique soit généralement découragée.

Le comportement par défaut pour le label de fichier (actuellement les inodes qui consistent des classes suivantes :

1. Le composant utilisateur est hérité du processus créateur
2. Le composant rôle est généralement par défaut object_r
3. Le composant type est par défaut au type du répertoire parent si aucune règle type_transition n'est spécifiée dans la stratégie
4. Le composant range/level est par défaut au niveau bas/courant du processus créateur si aucune règle range_transition n'est spécifié dans la stratégie

Les applications compatibles SELinux peuvent changer ce comportement par défaut en appelant setfscreatecon(3) avant de créer le fichier, si permis par la stratégie.

Pour les fichiers existants le label est déterminé depuis la valeur xattr associée avec le fichier. S'il n'y a pas de valeur xattr définie dans le fichier, le fichier est traité comme s'il était labélisé avec le contexte de sécurité par défaut pour le système de fichier. Par défaut, c'est le SID "file" initial, qui est mappé à un contexte par la stratégie. Ce défaut peut être écrasé via l'option de montage defcontext=.

Descripteurs de fichier

Hérite du label de son créateur/parent.

Systèmes de fichier

Les systèmes de fichier sont labélisé en utilisant la déclaration de langage de stratégie kernel fs_use quand ils sont monté, et sont basés sur le nom du type de système de fichier (ex : ext4) et leur comportement (ex : xattr). Par exemple si la stratégie spécifie :

fs_use_task pipefs system_u :object_r :fs_t :s0

lorsque le système de fichier pipefs est monté, le hook de sécurité LSM SELinux selinux_set_mnt appelle security_fs_use qui va :

a) Rechercher le nom du système de fichier dans la stratégie (pipefs)
b) Si présent, obtenir sont comportement (fs_use_task)
c) Puis obtenir le contexte de sécurité alloué (system_u :object_r :fs_t :s0)

si le comportement est définis par fs_use_task, alors le système de fichier sera labélié comme suit :

Notes :

1. Les systèmes de fichier qui supportent les attributs étendus xattr peuvent être identifiés via la commande mount, le mot clé seclabel est présent
2. Il y a des points de montages pour allouer différents types de contexte : context=, fscontext=, defcontext= et rootcontext=

nfsv4

Si le label NFS est implémenté via le support xattr, la création des inodes sont créés comme décris ci-dessus

sockets INET

Si un socket est créé par l'appel socket(3), il est labélisé comme suit :

1. Le composant utilisateur hérite du processus créateur
2. Le composant rôle hérite du processus créateur
3. Le composant type hérite du processus créateur si aucune règle type_transition n'est spécifié dans la stratégie
4. Le composant range/level hérite du processus créateur si aucune règle range_transition n'est spécifié dans la stratégie

IPC

Hérite le label de son processus d'envoie. Cependant si l'envois d'un message est non labélisé, calcule un nouveau label basé sur le processus courant et met le message en queue comme suit :

1. Le composant utilisateur hérite du processus émetteur
2. Le composant rôle hérite du processus émetteur
3. Le composant type hérite du processus émetteur si aucune règle type_transition n'est spécifiée dans la stratégie
4. Le composant range/level hérite du processus émetteur si aucune règle range_transition n'est spécifiée dans la stratégie

Sémaphores

Hérite du label de son créateur/parent

Mémoire partagée

Hérite du label de son créateur/parent

Clés

Hérite du label de son créateur/parent

Transition d'objet et de domaine

La déclaration type_transition est utilisée pour :

1. La transition d'un processus d'un domaine à un autre (une transition de domaine)
2. La transition d'un objet d'un type à un autre (une transition d'objet)

Transition de domaine

Une transition de domaine se produit lorsqu'un processus dans un domaine démarre un nouveau processus dans un autre domaine sous un contexte de sécurité différent. Il y a 2 manière de définir une transition de domaine :

1. En utilisant une déclaration type_transition, où l'appel système exec va exécuter automatiquement la transition de domaine pour les programmes non compatible SELinux. C'est la méthode la plus commune :

```
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

2. Les applications compatible SELinux peuvent spécifier le domaine du nouveau processus en utilisant libselinux. Pour cela, l'application doit avoir la permission setexec :

```
allow crond_t self : process setexec;
```

Cependant, avant toute transition de domaine la stratégi doit spécifier que :

1. Le domaine source a la permission de transiter dans le domaine cible
2. Le binaire de l'application doit être exécutable dans le domaine source
3. Le binaire de l'application doit avoir un point d'entrée dans le domaine cible

La déclaration type_transition suivante est un exemple pour expliquer le processus de transition :

```
type_transition | source_domain | target_type | class | target_domain;
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

Cette déclaration type_transition status que lorsqu'un processus dans le domain unconfined_t exécute un fichier labélisé secure_services_exec_t, le processus doivent être changé en ext_gateway_t si permis par la stratégie.

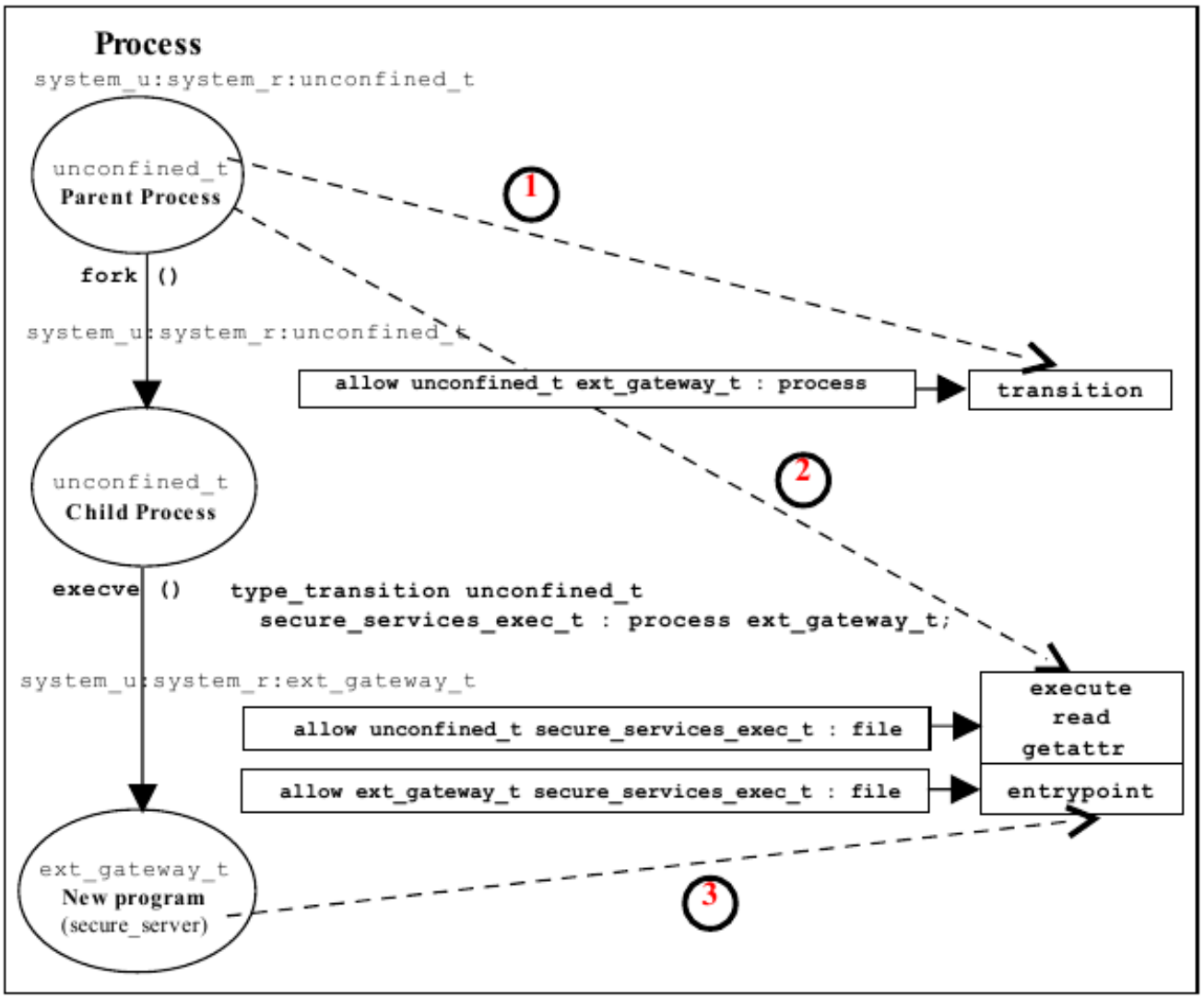
Cependant, comme statué plus haut, pour être capable de transiter dans le domaine ext_gateway_t, les permissions minimum suivantes doivent être donnés dans la stratégie en utilisant les règles alow, où :

1. Le domaine nécessite la permission de transiter dans le domaine ext_gateway_t :

   allow unconfined_t : process transition ;
2. Le fichier exécutable doit être exécutable dans le domaine unconfined_t, et donc nécessite également que le fichier soit lisible : allow unconfined_t secure_services_exec_t : file { execute read getattr } ;
3. Le fichier exécutable doit avoir un point d'entrée dans le domaine ext_gateway_t :

   allow ext_gateway_t secure_services_exec_t : file entrypoint ;

Comme affiché ci-dessous où unconfined_t fork un processus enfant, le nouveau programme transite dans ext_gateway_t. Noter qu'à cause de la déclaration type_transition utilisée, la transition est automatiquement géré par le kernel.



Règles de type forcé

En construisant les modules ext_gateway.conf et int_gateway.conf l'intention était d'avoir ces transitions et leur domaines respectifs via les déclarations type_transition. La déclaration ext_gateway_t serait :

```
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

et la déclaration int_gateway_t serait :

```
type_transition unconfined_t secure_services_exec_t : process int_gateway_t;
```

Cependant, en liant ces 2 modules dans la stratégie, l'erreur suivante se produit :

```
semodule -v -s modular-test -i int_gateway.pp -i ext_gateway.pp
Attempting to install module 'int_gateway.pp':
Ok: return value of 0.
Attempting to install module 'ext_gateway.pp':
Ok: return value of 0.
Committing changes:
libsepol.expand_terule_helper: conflicting TE rule for (unconfined_t,
secure_services_exec_t:process): old was ext_gateway_t, new is int_gateway_t
libsepol.expand_module: Error during expand
libsemanage.semanage_expand_sandbox: Expand module failed
```

```
semodule: Failed!
```

Celà se produit parce que les règles de type forcé ne permettent qu'un seul type par défaut pour une source et cible donnée. Dans la cas ci-dessus il y a 2 déclaration type_transition avec la même source et cible, mais différents domaines par défaut. Le module ext_gateway.conf a les déclarations suivantes :

```
allow unconfined_t ext_gateway_t : process { transition };
allow unconfined_t secure_services_exec_t : file { read execute getattr };
allow ext_gateway_t secure_services_exec_t : file { entrypoint };
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

et le module int_gateway_t a les déclarations suivantes :

```
allow unconfined_t int_gateway_t : process { transition };
allow unconfined_t secure_services_exec_t : file { read execute getattr };
allow int_gateway_t secure_services_exec_t : file { entrypoint };
type_transition unconfined_t secure_services_exec_t : process int_gateway_t;
```

Alors que les règles allow sont valides pour permettre les transitions, les 2 déclarations type_transition ont des types par défaut différents (ou domaines cible), qui cassent la règle de type forcé.

Pour résoudre cela, il a été décidé :

1. Conserver la règle type_transition pour le type par défaut de ext_gateway_t, et autoriser le processus serveur à être exécuté depuis unconfined_t, en lançant simplement la commande depuis le prompts comme suit : **secure_server 99999**
2. Utiliser la commande runcon pour s'assurer que la gateway interne se lance dans le domaine corecte en lançant runcon depuis le prompte comme suit : **runcon -t int_gateway_t -r message_filter_r secure_server 1111**

La commande runcom utilise libselinux pour vérifier le contexte courant et définir le nouveau contexte.

d'autres manières de résoudre ce problème sont :

1. Utiliser la commande runcon pour lancer les 2 gateway pour transiter dans leur domaine respectifs. Les déclarations type_transition sont donc non requis.
2. Utiliser des noms différents pour les fichiers exécutables du serveur et s'assurer qu'ils ont un type différent
3. Implémenter la stratégie sur le modèle de la section templace macro.

Transition d'objet

Une transition d'objet se produit lorsqu'un nouvel objet nécessite un label différent de celui de son parent. Par exemple un fichier créé qui nécessite un label différent de son répertoire parent. la déclaration type_transition permet de le faire :

```
type_transition ext_gateway_t in_queue_t:file in_file_t;
```

Les détails suivants d'une transition d'objet utilisée dans le module ext_gateway.conf où par défaut, les fichiers sont labélisés in_queue_t quand ils sont créés par l'application gateway vu que ce label est attaché au répertoire parent :

```
ls -Za /usr/message_queue/in_queue
drwxr-xr-x root root unconfined_u:object_r:in_queue_t .
drwxr-xr-x root root system_u:object_r:unconfined_t ..
```

Cependant il est requis que les fichiers dans ce répertoire soient labélisés in_file_t. Pour celà les fichiers créé doivent être relabélisés dans in_file_t en utilisant un type_transition :

```
type_transition | source_domain | target_domain : object | default_type;
```

```
type_transition ext_gateway_t in_queue_t : file in_file_t;
```

Cette déclaration type_transition status que lorsqu'un processus tournant dans le domaine ext_gateway_t (le domaine source) souhaite créer un objet fichier dans le répertoire qui est labélisé in_queue_t, le fichier devrait être relabélisé in_file_t si permis par la stratégie.

Cependant, comme décris plus haut, pour être capable de créer le fichier, les permissions minimum suivantes doivent être données dans la stratégie en utilisant les règle allow, où :

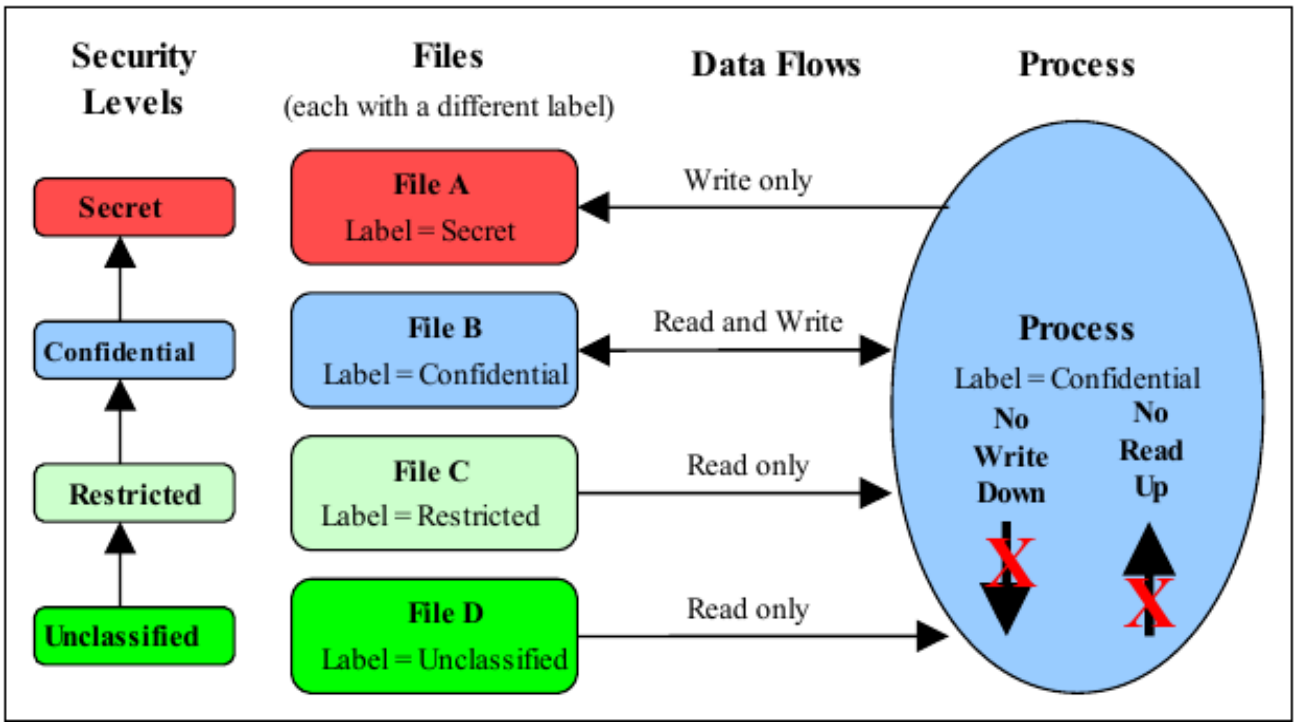
1. le domaine source a la permission d'ajouter des entrées fichier dans le répertoire :
allow ext_gateway_t in_queue_t : dir { write search add_name) ;
2. Le domaine source a la permission de créer des entrées fichiers :
allow ext_gateway_t in_file_t : file (write create getattr) ;
3. La stratégie peut ainsi s'assurer que les fichiers créé dans in_queue sont relabélisés :
type_transition ext_gateway_t in_queue_t : file in_file_t ;
Un exemple de sortie d'un répertoire :
ls -Za /usr/message_queue/in_queue
drwxr-xr-x root root unconfined_u :object_r :in_queue_t .
drwxr-xr-x root root system_u :object_r :unconfined_t ..
-rw-r–r– root root unconfined_u :object_r :in_file_t Message-1
-rw-r–r– root root unconfined_u :object_r :in_file_t Message-2

MLS et MCS

Comme définis dans la section MAC et TE, SELinux supporte également MLS et MCS en ajoutant une entrée optionnelle level ou range dans le contexte de sécurité. Cette section donne une brève introduction à MLS et MCS.

Le schéma ci-dessous montre un diagramme simple où les niveaux de sécurité représentent la classification des fichiers dans un serveur de fichier. Les niveaux de sécurité sont strictement hiérarchiques et conformes au modèle Bell-La & Padula dans le fait qu'un processus (tournant au niveau Confidential) peut lire/écrire à son niveau courant, mais ne peut que lire dans les niveaux inférieurs ou seulement écrire dans les niveaux supérieurs.

Celà s'assure de la confidentialité vu que le processus peut copier un fichier au nieau secret, mais ne peut jamair relire son contenu sauf si le processus s'élève à ce niveau. Le processus ne peut pas écrire les fichiers à des niveaux inférieurs au niveau confidential.



Pour accomplir ce niveau de contrôle, les extensions MLS de SELinux utilisent les contraintes similaires à ceux décris dans la section contraintes des types forcés, excepté que la déclaration est appelée mlsconstrain.

Cependant, comme la vie n'est pas simple :

1. Les processus et objets peuvent avoir une plage qui représente les niveaux de sécurité faible et élevés
2. Le niveau de sécurité peut être plus complexe, en cela que c'est une sensibilité hiérarchique et 0 ou plusieurs catégories
3. Permettre à un processus d'accéder à un objet est géré par des règles de dominance appliqués aux niveaux de sécurité.
4. Les processus trustés peuvent avoir des privilèges qui les autorisent à bypasser les règles BLP.
5. Certains objets ne supportent pas les fonctions de lecture/écriture séparés vu qu'ils nécessitent de lire/répondre dans les ce cas comme les réseaux.

Les sections qui suivent discutent du format d'un niveau de sécurité et de la plage, et la manière dont elles sont gérées par les mécanismes de contrainte dans SELinux en utilisant les règles de dominance.

Niveaux de sécurité

Les éléments ci-dessous montrent les composants qui créent un niveau de sécurité et comment 2 niveaux de sécurité depuis une plage de pour le quatrième et options [:range].

Le niveau de sécurité consiste d'une sensibilité ou 0 ou plusieurs entrées catégories : **sensitivity [: category ;...]**. Noter que SELinux utilise les niveaux, la sensibilité et la catégorisation dans les déclarations de langage, cependant les termes suivants peuvent également être utilisés : labels, classification, et compartiment.

Dans une plage, low, pour un processus, un sujet ou un objet constitue le niveau ou la sensibilité courante. SystemLow est le niveau ou la classification la plus faible pour le système. (pour SELinux, c'est généralement s0, noter qu'il n'y a pas de catégories).

Dans une plage, high, pour un processus ou un sujet est l'autorisation. Pour un objet, c'est la plage maximum. SystemHigh est le niveau ou la classification la plus haute (pour SELinux c'est généralement s15 :c0,c255)

Les composants suivants sont utilisés pour définir les niveaux de sécurité MLS/MCS dans le contexte de sécurité :

```
user:role:type:sensibilité[:catégorie,...] - sensibilité[:catégorie,...]
```

où :

Sensibilité Les niveaux de sensibilité sont hiérarchiques avec (traditionnellement) s0 étend le plus faible. Ces valeurs sont définies en utilisant la déclaration sensitivity. Pour définir leur hiérarchie, la déclaration dominance est utilisée. Pour les systèmes MLS la sensibilité la plus haute est la dernière définie dans la déclaration dominance. Traditionnellement le maximum pour les système MLS est s15. Pour les système MCS il y a seulement une sensibilité définie, s0.

Catégorie Les catégories sont optionnelles et forment une liste non-ordonnée et non-liées de compartiments. Ces valeurs sont définies en utilisant la déclaration category, où par exemple c0.c3 représente une plage et c0,c3,c7 représente une liste. Traditionnellement les valeurs sont entre c0 et c255.

level Le niveau est une combinaison de valeurs de sensibilité et catégorie qui forment le niveau actuel de sécurité. Ces valeurs sont définies en utilisant une déclaration level.

Traduire les niveaux

En écrivant des stratégie pour MS/MCS on utilise généralement une forme abbréviée tel que s0, s1, etc. pour représenter la sensibilité et c0, c1, etc. pour représenter les catégories. Cela permet de conserver de l'espace dans les attributs étendus des fichiers, et également en mémoire. Donc pour que ces labels puissent être représentés au format compréhensible, un service de traduction est fournis via le fichier setrans.conf qui est utilisé par le services mcstransd. Par exemple s0 = Unclassified, s15 = Top Secret et c0 = Finance, c100 = Spy Stories. La commande semanage peut être utilisée pour utiliser cette traduction.

Gérer les niveaux de sécurité via les règles de dominance

Comme statué plus haut, autoriser un processu à accéder à un objet est géré par des règles de dominance appliqués aux niveaux de sécurité. Ces règle sont les suivantes :

L1 domine L2 Si la sensibilité de L1 est égal ou supérieur à la sensibilité de L2 et que les catégories de L1 sont les même ou un super-jeu des catégories de L2

L2 domine L1 Si la sensibilité de L1 est égal ou inférieur à la sensibilité de L2 et que les catégories de L1 sont un sous-jeu des catégories de L2

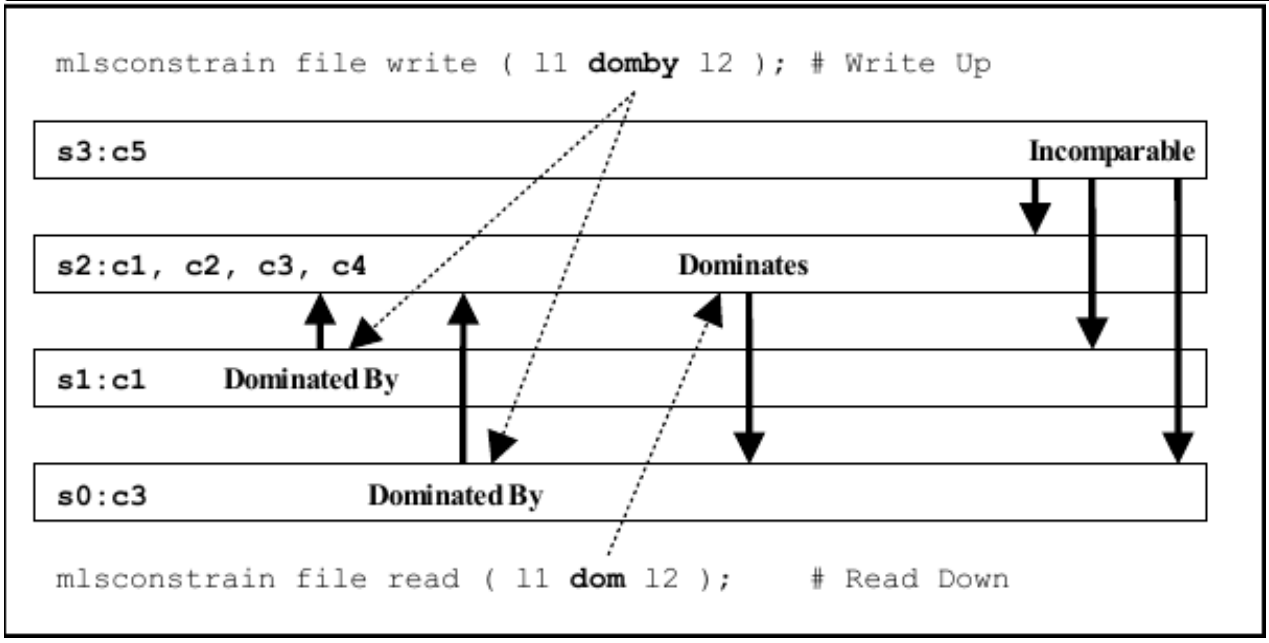
L1 équivaut à L2 Si la sensibilité de 1 et égal à L2 et les catégories de L1 et L2 sont les même.

L1 est incomparable à L2 Si les catégories de L1 et L2 ne peuvent être comparés.

Pour illustrer l'utilisation de ces règles, la table ci-dessous liste les attributs de niveau de sécurité pour montrer des fichiers qui ont été labélisés comme s3 :c0. Le processus qui accède à ces fichiers (ex : un éditeur) tourne avec une plage s0 - s3 :c1.c5 et a accès aux fichiers en gras.

Vu que la déclaration dominance MLS est utilisée pour forcer la hiérarchie de sensibilité, les niveaux de sécurité suivent maintenant la séquence (faible = s0 à élevé = s3) avec les catégories étant des listes de compartiments. Pour autoriser le processus à accéder aux fichiers dans son scope et avec les règles de dominance, le processus sera contraint en utilisant la déclaration mlsconstrain.

	Category →	c0	c1	c2	c3	c4	c5	c6	c7
s3	**Secret**	**s3:c0**					**s3:c5**	**s3:c6**	
s2	**Confidential**		**s2:c1**	**s2:c2**	**s2:c3**	**s2:c4**			**s2:c7**
s1	**Restricted**	**s1:c0**	**s1:c1**						**s1:c7**
s0	**Unclassified**	**s0:c0**			**s0:c3**				**s0:c7**
↑ Sensitivity	↑ **Security Level** (sensitivity:category) aka: classification	↑ **File Labels** ↑ A process running with a `range` of `s0 - s3:c1.c5` has access to the files within the grey boxed area.							



1. Pour autoriser write-up, le niveau source (l1) doit être dominé par le niveau cible (l2)
2. Pour autoriser read-down, le niveau source (l1) doit dominer le niveau cible (l2)

Cependant, dans le monde réel la stratégie de référence MLS n'autorise pas le write-up sauf si le processus a un privilège spécial (en ayant le type de domaine ajouté à un attribut), bien qu'il autorise le read-down. Le défaut est d'utitilesr l1 et l2. Le fichier source policy/mls montre ces déclaration mlsconstrain.

Certification Common Criteria

Bien que le processus de certification Common Criteria est au delà du scope de ce document, Il est à souligner que des versions de logiciel RedHat, tournant sur des plateformes hardware spécifiques avec SELinux/MLS ont passé le processus d'évaluation Common Criteria. Noter que pour l'évaluation et le déploiement de logiciel et hardware sont liés, quand une mise à jours est nécessaire, une mise à jours de la certifiation doit être obtenue.

Types de stratégie SELinux

Cette section décris les différents types de stratégie. Le type de stratégie SELinux peut être décris de plusieurs manières.

Stratégie exemple

La stratégie exemple est le nom utilisée pour décrir le source utilisé pour construire une stratégie monolitique produite par le NSA et désormais remplacé par le stratégie de référence.

Stratégie référence

Le stratégie référence est le startégie standard utilisée pour construire des stratégies SELinux, et son but est de fournir une arborescence simple avec une documentation support qui peut être utilisé pour construire des stratégie pour différents cas tel que confiner des services importants, supporter MLS/MCS et bloquer les systèmes pour que tous les processus soient sous le contrôle de SELinux.

La stratégie référence est utilisée par toutes les distributions Linux, cependant, chaque distribution effectue des propres changements.

Fonctionnalité de stratégie basé sur le nom ou le type

Généralement une stratégie est installée avec un nom donné tel que targeted, mls, refpolicy ou minimum qui tente de décrire ses fonctionnalités. Ce nom devient ainsi l'entrée dans :

1. Le répertoire pointant à l'emplacement de la stratégie (ex : si le nom est targeted, la stratégie est installée dans /etc/selinux/targeted)
2. L'entrée SELINUXTYPE dans /etc/selinux/config quand c'est la stratégie active (ex : si le nom est targeted, l'entrée est SELINUXTYPE=targeted).

Stratégie Monolitique

Une stratégie monolitique est une stratégie SELinux qui est compilée depuis un fichier source appelé par convention policy.conf. Il n'utilise pas les déclaration de module chargeable et est prévu pour les systèmes embarqués.

Stratégie à module chargeable

L'infrastructure à module chargeable permet de gérer une stratégie sur une base modulaire, il y a un module de stratégie de base qui contient tous les composants du corp de la stratégie, et 0 ou plusieurs modules qui peuvent être chargés et déchargés si requis. Il y a plusieurs composants qui forment l'infrastructure :

1. Le source de la stratégie qui est construite pour une stratégie modulaire avec un module de base et des modules chargeables optionnels
2. Des utilitaires pour compiler et lier les modules et les placer dans un magasin de stratégie
3. Des utilitaires pour gérer les modules et les fichiers de configuration associés dans le magasin de stratégie

Stratégie optionnelle

L'infrastructure de stratégie à module supporte également une déclaration optionnelle qui permet de définir des règsles qui sont activées eulement dans la stratégie binaire une fois les conditions satisfaites.

Stratégie conditionnelle

Les stratégie conditionnelles peuvent être implémentées dans des stratégies monolitiques ou à module et permet d'activer des parties de la stratégie en fonction de l'état d'un flag en temps-réel. Les flags sont maintenus dans le kernel et peuvent être changés avec setsebool.

Les déclaration de langage de stratégie conditionnelle sont des déclaration bool qui définissent l'identifiant de flag et son status initial, et la déclaration if qui permet à certaines règles d'être exécutées en fonction de l'état des valeur booléennes.

Stratégie binaire

Également connu sous le nom de stratégie kernel, c'est le fichier de stratégie qui est chargé dans le kernel et est localisé à /etc/selinux/<SELINUXTYPE>/policy/policy.<version>.

Versions de stratégie

SELinux a une base de stratégie (définie dans libsepol) qui décris le format des données gérée dans une stratégie binaire, cependant, si de nouvelles fonctionnalités sont ajoutées à SELinux, il peut y avoir un changement dans la base. sestatus affiche la version maximum supportée par le kernel.

Mode permissif et forcé

SELinux a 3 modes d'opération majeurs :

Enforcing SELinux impose la stratégie chargée

Permissive SELinux charge la stratégie, mais ne l'impose pas. Généralement utilisé pour les tests

Disabled SELinux n'est pas activé.

Ces flags sont définis dans /etc/selinux/config. Il y a une autre méthode pour exécuter des domaines spécifique en mode permissif en utilisant la déclaration permissive. Cela peut être utilisé dans un module utilisateur ou semanage génère le module approprié et le charge en utilisant l'exemple suivant : **semanage permissive -a unconfined_t**

Il est également possible de définir un mode permissif dans le gestionnaire d'objet userspace en utilisant libselinux. sestatus affiche le mode courant, cependant, il n'affiche pas le domaine individuel ou les modes du gestionnaire d'objet.

Auditer les évènements SELinux

Pour SELinux il y a 2 types principaux d'évènements d'audit :

1. Les évènement AVC - Ils sont générés par le sous-système AVC en résultat à des accès refusés, ou quand des évènements spécifique sont demandés.
2. Évènements d'applications - Ils sont générés par les services kernel SELinux et les applications compatibles SELinux.

Les messages d'évènement et d'audit sont généralement stockés dans un de ces logs :

1. Le évènement SELinux du boot kernel sont loggés dans /var/log/dmesg

2. /var/log/messdages contient les messages générés par SELinux avant que le système d'audit soit chargé

3. /var/log/audit/audit.log contient les évènements qui prennent place une fois le service d'audit chargé.

Notes

a) Il n'est pas obligatoire pour les applications SELinux d'auditer les évènements ni de les logger.

b) Le format des messages d'audit n'a pas besoin de se conformer à un format, cependant, si possible les applications devraient utiliser la fonction audit_log_user_avc_message(3)

c) Le fonctions de librairie SELinux affichent des messages également sur stderr par défaut. Cependant celà peut être changé avec selinux_set_callback(3).

Évènements d'audit AVC

Cette section décris le format général des messages d'audit AVC dans audit.log.

type type peut être AVC pour les évènements kernel, ou USER_AVC pour les évènement des gestionnaires d'objet userspace. Une fois l'évènement AVC loggé, le type SYSCALL peut suivre et contient des informations supplémentaires

msg Contient le mot clé 'audit' avec un numéro de référence

avc Soit denied soit granted

pid|comm Si c'est une tâche, log le pid et le nom de l'exécutable

capability Si c'est un évènement de capability, log l'identifiant

path|name|dev|ino Si c'est en évènement de système de fichier, log les informations spécifique

laddr|lport|faddr|fport Si c'est un évènement socket, log les adresses/port source/destination

path Si c'est un évènement de fichier socket, log le chemin

saddr|src|daddr|dest|netif Si un évènement réseau, log les adresses/port source/destination

sauid|hostname|addr|terminal Identifiant d'association de sécurité IPSec

resid|restype Type et ID de ressource X-Windows

scontext Le contexte de sécurité de la ressource ou sujet

tcontext Le contexte de sécurité de la cible ou objet

tclass La classe objet de la cible ou objet

Évènements d'audit SELinux général

Cette section montre une sélection d'évènements d'audits de service non AVC pris depuis audit.log. Pour une liste d'entrée "type=" valides, les fichiers include suivant devraient être consultés : include/libaudit.h et include/linux/audit.h.

Noter qu'il peut y avoir plusieurs évènements générés pour le même évènement. Par exemple, le serveur de sécurité kernel génère une évènement MAC_POLICY_LOAD pour indiquer que la stratégie a été rechargée, mais chaque gestionnaire d'objet userspace peut également générer un USER_MAC_POLICY_LOAD pour indiquer qu'il a également traité l'évènement.

MAC_POLICY_LOAD, USER_MAC_POLICY_LOAD - ces évènements sont générés quand la stratégie est rechargée :

```
type=MAC_POLICY_LOAD msg=audit(1336662937.117:394): policy loaded auid=0 ses=2
type=SYSCALL msg=audit(1336662937.117:394): arch=c000003e syscall=1 success=yes exit=4345108 a0=4
a1=7f0a0c547000 a2=424d14 a3=7fffe3450f20 items=0 ppid=3845 pid=3848 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0
egid=0 sgid=0 fsgid=0 tty=pts2 ses=2 comm="load_policy" exe="/sbin/load_policy"
subj=unconfined_u:unconfined_r:load_policy_t:s0-s0:c0.c1023 key=(null)
```

```
type=USER_MAC_POLICY_LOAD msg=audit(1336662938.535:395): pid=0 uid=0 auid=4294967295 ses=4294967295
subj=system_u:system_r:xserver_t:s0-s0:c0.c1023 msg='avc: received policyload notice (seqno=2) :
exe="/usr/bin/Xorg" sauid=0 hostname=? addr=? terminal=?'
```

MAC_STATUS - Généré quand le mode SELinux est changé :

```
type=MAC_STATUS msg=audit(1336836093.835:406): enforcing=1 old_enforcing=0
auid=0 ses=2
type=SYSCALL msg=audit(1336836093.835:406): arch=c000003e syscall=1 success=yes exit=1 a0=3 a1=7fffe743f9e0
a2=1 a3=0 items=0 ppid=2047 pid=5591 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0
ses=2 comm="setenforce" exe="/usr/sbin/setenforce" subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
key=(null)
```

MAC_CONFIG_CHANGE - Généré quand setsebool a été lancé pour changer un booléen.

```
type=MAC_CONFIG_CHANGE msg=audit(1336665376.629:423): bool=domain_paste_after_confirm_allowed val=0 old_val=1
auid=0 ses=2
type=SYSCALL msg=audit(1336665376.629:423): arch=c000003e syscall=1 success=yes exit=2 a0=3 a1=7fff42803200
a2=2 a3=7fff42803f80 items=0 ppid=2015 pid=4664 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0
fsgid=0 tty=pts0 ses=2 comm="setsebool" exe="/usr/sbin/setsebool"
subj=unconfined_u:unconfined_r:setsebool_t:s0-s0:c0.c1023 key=(null)
```

MAC_UNLBL_STCADD - Généré quand un label statique non-mappé est ajouté

```
type=MAC_UNLBL_STCADD msg=audit(1336664587.640:413): netlabel: auid=0 ses=2
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 netif=lo src=127.0.0.1
sec_obj=system_u:object_r:unconfined_t:s0-s0:c0,c100 res=1
type=SYSCALL msg=audit(1336664587.640:413): arch=c000003e syscall=46 success=yes exit=96 a0=3 a1=7fffde77f160
a2=0 a3=666e6f636e753a72 items=0 ppid=2015 pid=4316 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0
fsgid=0 tty=pts0 ses=2 comm="netlabelctl" exe="/sbin/netlabelctl"
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key=(null)
```

SELINUX_ERR - Généré par le serveur de sécurité kernel suite à des privilège de anon-webapp_t supérieur à celui donné au processus qui a lancé le thread

```
type=SELINUX_ERR msg=audit(1311948547.151:138): op=security_compute_av reason=bounds
scontext=system_u:system_r:anon_webapp_t:s0-s0:c0,c100,c200 tcontext=system_u:object_r:security_t:s0
tclass=dir perms=ioctl,read,lock
type=SELINUX_ERR msg=audit(1311948547.151:138): op=security_compute_av reason=bounds
scontext=system_u:system_r:anon_webapp_t:s0-s0:c0,c100,c200 tcontext=system_u:object_r:security_t:s0
tclass=file perms=ioctl,read,write,getattr,lock,append,open
```

Généré par le serveur de sécurité kernel quand une application compatible SELinux tente d'utiliser setcon pour créer un nouveau thread

```
type=SELINUX_ERR msg=audit(1311947138.440:126): op=security_bounded_transition result=denied
oldcontext=system_u:system_r:httpd_t:s0-s0:c0.c300
newcontext=system_u:system_r:anon_webapp_t:s0-s0:c0,c100,c200
type=SYSCALL msg=audit(1311947138.440:126): arch=c000003e syscall=1 success=no exit=-1 a0=b a1=7f1954000a10
a2=33 a3=6e65727275632f72 items=0 ppid=3295 pid=3473 auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48
egid=48 sgid=48 fsgid=48 tty=(none) ses=4294967295 comm="httpd" exe="/usr/sbin/httpd"
subj=system_u:system_r:httpd_t:s0-s0:c0.c300 key=(null)
```

USER_ROLE_CHANGE - newrole(1) a été utilisé pour définir un nouveau rôle qui n'est pas valide

```
type=USER_ROLE_CHANGE msg=audit(1336837198.928:429): pid=0 uid=0 auid=0 ses=2
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='newrole:
old-context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 new-context=?: exe="/usr/bin/newrole"
hostname=? addr=? terminal=/dev/pts/0 res=failed'
```

Support de la poly-instanciation

GNU/Linux supporte la poly-instanciation des répertoires qui peuvent être utilisé par SELinux via PAM. La poly-instanciation des objets est également supporté pour les sélections X-Windows. Noter que les sockets ne sont pas encore supportés. Pour clarifier le support de la polyinstanciation :

1. SELinux a des fonctions libselinux et une règle de stratégie pour supporter la poly-instanciation
2. La poly-instanciation des répertoires est une fonction de GNU/Linux
3. La poly-instanciation des sélections X-Windows et des propriétés est une fonction du gestionnaire d'objet XSELinux et le support du service XACE

Objects poly-instanciés

Déterminer un contexte poly-instancié pour un objet est supporté par SELinux en utilisant la déclaration type_member et les fonctions avc_compute_member et security_compute_member. Ce n'est pas limité à des classes d'objets spécifique, cependant seul les objets dir, x_selection et x_property sont actuellement supportés.

Support de la poly-instanciation dans PAM

PAM supporte la poly-instanciation (namespaces) des répertoires à la connexion en utilisant les services d'arborescence partagée/espaces de noms disponibles dans GNU/Linux (voir namespace.conf) Noter que PAM et les services d'espace de nom sont compatibles SELinux.

L'installation par défaut de F-20 n'active pas les répertoires poly-instanciés par défaut, cependant cette section montre la configuration requise pour activer cette fonctionnalité et quelques exemples.

Pour implémenter les répertoires poly-instanciés PAM exige que les fichiers suivants soient configurés :

1. Une entrée pour le module pam_namespace. F-20 a déjà ces entrées dans /etc/pam.d/gdm-password.
2. Les entrées sont ajoutées dans /etc/security/namespace.conf et définissent les répertoires à poly-instancier par PAM.

Une fois ces fichiers configurés et qu'un utilisateur se log, pam_namespace départage l'espace de nom cournat de son parent et monte les espaces de nom en accord avec les règles définies dans namespace.conf. F-20 inclus également le script /etc/security/namespace.init. qui est utilisé pour initialiser l'espace de nom chaque fois qu'une nouvelle instance de répertoire est définis. Ce script reçois 4 paramètres : le chemin du répertoire poly-instancié, le chemin de l'instance du répertoire, un flag pour indiquer si une nouvelle instance, et un nom d'utilisateur. Si une nouvelle instance est définie, les permissions du répertoire sont définis et restorecon(8) est lancé pour définir le contexte de fichier.

Fichier de configuration namespace.conf

Chaque ligne dans namespace.conf est formaté comme suit :
polydir instance_prefix method list_of_uids
où :

polydir Chemin absolu du répertoire à poly-instancier. $USER et $HOME sont remplacés

instance_prefix Un préfixe utilisé pour construire le chemin pour le répertoire poly-instancié. $USER et $HOME sont remplacés

method Détermine a méthode de la poly-instanciation :

user La poly-instanciation est basée sur les noms d'utilisateur

level La poly-instanciation est basée sur le nom d'utilisateur et le niveau MLS

context La poly-instanciation est basée sur le nom d'utilisateur et le contexte de sécurité

Liste de noms d'utilisateurs qui n'ont pas de répertoires poly-instanciés. Si vide, tous les utilisateurs poly-instanciés. Si la liste est précédée par '-', seul les utilisateurs dans la liste ont des répertoires poly-instanciés.

Exemple de configuration

Cette section montre 2 exemple de configuration namespace.conf, le premier utilise la méthode user, et l'autre context. Noter que tant que la poly-instanciation est activée, les noms de chemins complet ne sont pas visible, c'est seulement quand la poly-instanciation est désactivée que les répertoires deviennent visible.

Exemple 1 - method=user :

```
#polydir instance-prefix method list_of_uids
/tmp /tmp-inst/ user root,adm
/var/tmp /var/tmp/tmp-inst/ user root,adm
$HOME $HOME/$USER.inst/ user
```

Se connecter en tant qu'utilisateur normal. Le processus PAM va construire les répertoires poly-instanciés suivant :

```
# /tmp
/tmp/tmp-inst/myuser
# /var/tmp
/var/tmp/tmp-inst/myuser
# $HOME
/home/myuser/myuser.inst/myuser
```

Exemple 2 - method=context

```
#polydir instance-prefix method list_of_uids
/tmp /tmp-inst/ context root,adm
/var/tmp /var/tmp/tmp-inst/ context root,adm
$HOME $HOME/$USER.inst/ context
```

Se connecter en tant qu'utilisateur normal. Le processus PAM va construire les répertoires poly-instanciés suivant :

```
# /tmp
/tmp/tmp-inst/unconfined_u:unconfined_r:unconfined_t_myuser
# /var/tmp
/var/tmp/tmp-inst/unconfined_u:unconfined_r:unconfined_t_myuser
# $HOME
/home/myuser/myuser.inst/unconfined_u:unconfined_r:unconfined_t_myuser
```

Support dans la stratégie référence

Les modules files.te et files.if supportent la poly-instanciation. Il y a également un booléen allow_polyinstanciation qui peut être utilisé pour activer cette fonctionnalité durant le login. Défaut : false.

Processus de login PAM

Les application utilisées pour fournir des services de login (comme gdm ou ssh) utilisent PAM pour fournir les services suivants :

Gestion de compte : Ce service gère les services tels que l'expiration des mots de passe, et les services de login autorisés

Gestion de l'authentification Authentifie l'utilisateur ou le sujet et définis les accréditifs

Gestion des mots de passe Gère les mises à jours de mot de passe

Gestion de session Gère les service qui doivent être invoqués avant que le processus de login se complète et/ou quand le processus de login de termine.

Il y a également des fichiers de configuration PAM liés à SELinux dans la section file de /etc/security/sepermit.conf. Les services core sont fournis par PAM, cependant d'autres moules peuvent être écris pour gérer les services spécifiques tel que le support pour SELinux. Les modules SELinux utilisent libseliinux pour obtenir la configuration :

pam_selinux_permit.so Autorise à des utilisateurs prédéfinis la capacité de se logger sans fournir de mot de passe

pam_selinux.so open Définis un contexte de sécurité pour l'utilisation au login.

pam_selinux.so close Réinitialise le contexte des programme login au contexte par défaut

LSM et SELinux

LSM est le framework de sécurité Linux qui autorise à des mécanisme de contrôle d'accès tier d'être liés dans le kernel GNU/Linux. Actuellement il y a 5 services qui utilisent LSM :

1. SElinux
2. AppArmor
3. SMACK
4. Tomoyo
5. Yama

L'idée de base derrière LSM est de :

- Insérer des hooks de fonction de sécurité et des structures de données de sécurité dans les divers services kernel.
- Autoriser l'enregistrement et l'initialisatio des services pour les modules tiers de sécurité
- Les attributs de sécurité de processus sont disponibles aux services userspace en étendant le système de fichier /proc avec un espace de nom de sécurité
- Supporter des systèmes de fichier qui utilisent les attributs étendus
- Consolider les capacités Linux dans un module optionnel

Il devrait être noté que LSM ne fournis aucun service de sécurité, seulement les hooks et structures pour supporter des modules tiers. S'il n'y a pas de module tiers chargé, les capacités deviennent le module par défaut autorisant le modèle DAC standard

Les services kernel où LSM implémente les hooks et structures sont :

```
exécution de programme
opérations de fichier
Réseaux de domaine Unix
Opérations de gestion de clé
Sémaphores
Syslog
Opérations de système de fichier
Opérations sur les tâches
Opérations sur les sockets
Opérations IPC
Capability
Audit
Opérations sur les inodes
Messagerie Netlink
Opérations XFRM
Segments mémoire
```

Fichiers /proc/self/attr/ utilisé par les services kernel et libselinux :

current (-rw-rw-rw) - Contient le contexte de sécurité du processus

exec (-rw-rw-rw) - Définis le contexte de sécurité pour le prochain appel exec

fscreate (-rw-rw-rw) - Définis le contexte de sécurité des nouveaux fichiers créés

keycreate (-rw-rw-rw) - Définis le contexte de sécurité pour les clés qui sont en cache dans le kernel

prev (-r–r–r-) - Contient le précédent contexte de sécurité

sockcreate (-rw-rw-rw) - Définis le contexte de sécurité pour les nouveaux sockets créés

Support réseau

SELinux supporte les types suivans de label réseau :

Label interne C'est où les objets réseaux sont labélisés et gérés en interne dans une seule machine : SECMARK et NetLabel

Réseau labélisé Lorsque les labels sont passé aux systèmes distant où ils peuvent être interprétés : IPSec et CIPSO

Il y a 2 options de capability de stratégie qui peuvent être définis dans la stratégie en utilisant la déclaration policycap qui affecte la configuration réseau :

network_peer_controls Toujours activé dans la stratégie référence

always_use_network Cette capacité est normalement à false. À true, SECMARK et NetLabel sont toujours activés même s'il n'y a pas de règles configurés. Celà force la vérification de la classe packet pour protéger le système.

Les paramètres de capability de stratégie sont disponible en userspace via les systèmes de fichiers SELinux. Pour supporter le label paire et CIPSO, les outils NetLabel doivent être installés (package netlabel_tools)

Pour supporter IPSec labélisé, les outils IPSec doivent être installés (package ipsec-tools)

Il est également possible d'utiliser un service IPSec labélisé alternatif, LibreSwan

Il est important de noter que le kernel doit être configuré pour supporter ces services. Le package iproute a la commande de statistique socket ss(8) qui affiche le contexte SELinux des processus réseaux et les sockets réseaux. Noter que les contextes sockets sont pris depuis les inodes associés au socket et non de la structure socket kernel.

SECMARK

SECMARK utilise le framework NetFilter du kernel. NetFilter inspecte automatiquement tous les paquets entrants et sortants et peut placer des contrôles dans les interfaces, les adresse IP et ports avec un suivi de connexion. L'extension SECMARK permet d'ajouter des contextes de sécurité aux paquets (SECMARK) ou aux session (CONNSECMARK).

Le framework NetFilter inspecte et tag les paquets avec des labels tel que définis dans iptables puis utilise le framework de sécurité (ex : SELinux) pour forcer les règles de stratégie. La structure de base est comme suit :

- Un table appelée 'security table' est utilisée pour définir les paramètres qui identifient et marque les paquets qui peuvent ainsi être suivis à mesure que le paquet voyage dans le sous-système réseaux. Ces marques sont appelées SECMARK et CONNSECMARK.
- Un SECMARK est placé sur un paquet s'il matche une entrée dans la table de sécurité qui applique un label qui peut ensuite être utilisé pour forcer la stratégie dans le paquet.

- Un CONNSECMARK maque tous les paquets dans une session avec le label approprié qui peut ensuite être utilisé pour forcer la stratégie.

Exemple d'entrée dans la table security. Marquer tous les paquets :
iptables -t security -A INPUT -i -lo -p tcp -d 127.0.0.0/8 -j SECMARK –selctx system.user :object_r :msg_filter.default_packet :s0
Ces règles remplacent le contexte précédent avec la gateway interne ou externe si le port 9999 ou 1111 est trouvé :
iptables -t security -A INPUT -i lo -p tcp –dport 9999 -j SECMARK –selctx system.user :object_r :msg_filter.ext_gateway.packet :s0
iptables -t security -A INPUT -i lo -p tcp –sport 9999 -j SECMARK –selctx system.user :object_r :msg_filter.ext_gateway.packet :s0
iptables -t security -A INPUT -i lo -p tcp –dport 1111 -j SECMARK –selctx system.user :object_r :msg_filter.int_gateway.packet :s0
iptables -t security -A INPUT -i lo -p tcp –sport 1111 -j SECMARK –selctx system.user :object_r :msg_filter.int_gateway.packet :s0
iptables -t security -A INPUT -m state –state ESTABLISHED,RELATED -j CONNSECMARK –save
Flux de sortie
iptables -t security -A OUTPUT -o lo -p tcp -d 127.0.0.0/8 -j SECMARK –selctx system.user :object_r :msg_filter.default_packet :s0
iptables -t security -A OUTPUT -i lo -p tcp –dport 9999 -j SECMARK –selctx system.user :object_r :msg_filter.ext_gateway.packet :s0
iptables -t security -A OUTPUT -i lo -p tcp –sport 9999 -j SECMARK –selctx system.user :object_r :msg_filter.ext_gateway.packet :s0
iptables -t security -A OUTPUT -i lo -p tcp –dport 1111 -j SECMARK –selctx system.user :object_r :msg_filter.int_gateway.packet :s0
iptables -t security -A OUTPUT -i lo -p tcp –sport 1111 -j SECMARK –selctx system.user :object_r :msg_filter.int_gateway.packet :s0
iptables -t security -A OUTPUT -m state –state ESTABLISHED,RELATED -j CONNSECMARK –save

NetLabel - Fallback labeling

le labeling peut optionnellement être implémenté dans un système si IPSec ou CIPSO labelisé n'est pas utilisé (falback labeling). Si IPSEC ou CIPSO labelisé est utilisé, il a précédence. Le contrôle du paire réseau a été étendu pour supporter une classe object additionnelles 'peer' qui est activé par défaut dans la stratégie F-20 si network_peer_controls dans /sys/fs/selinux/policy_capabilities est à 1.

NetLabel - CIPSO

Pour permettre de passer des niveaux de sécurité dans un réseau entre des systèmes MLS, Le protocole CIPSE est utilisé. C'est définis dans draft-ietf-cipso-ipsecurity-01. Le protocole définis comment les labels de sécurité sont encodés dans l'en-tête IP.

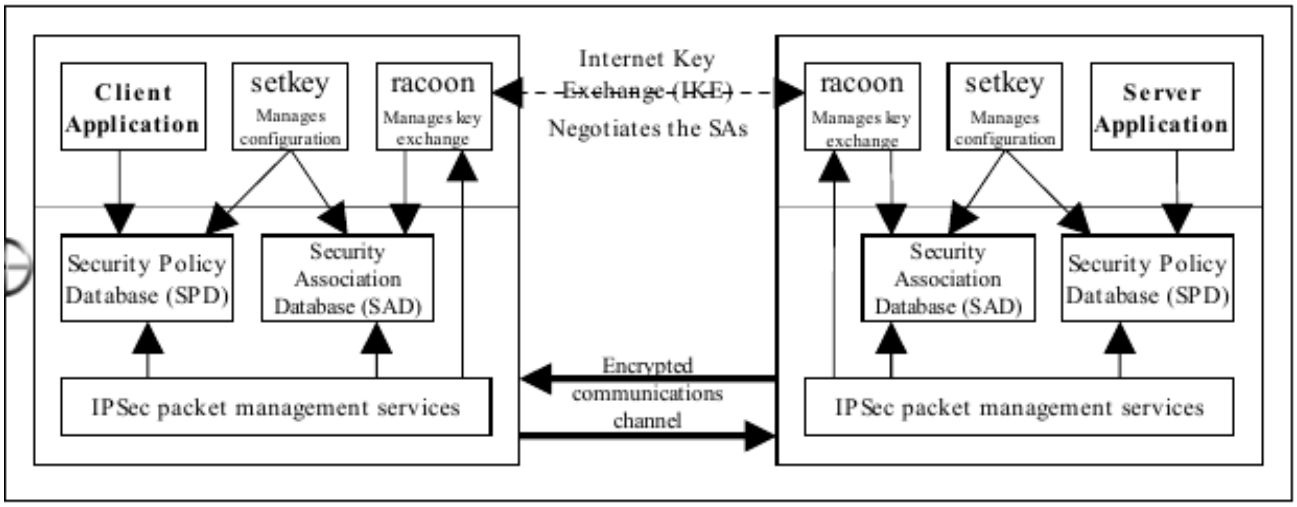
Noter que seul le composant de niveau du contexte de sécurité est passé sur le réseau. L'exception est le mode loopback. Le protocole est implémenté par le service NetLabel et peut être utilisé par d'autres modules de sécurité qui utilisent l'infrastructure LSM. L'implémentation NetLabel supporte :

1. 1 bit de type de tag qui autorise un maximum de 256 niveaux et 240 catégories
2. Une option non-traductible où les labels sont passés depuis/vers les systèmes de manière inchangée.
3. Une option de traduction où la sensibilité et la catégories peuvent être mappés pour les systèmes qui ont des définitions différentes.

IPsec labélisé

IPSec labélisé a été construit dans les service IPSec GNU/Linux. La figure ci-dessous montre les composants de base qui forment le service basé sur les outils IPSec généralement utilisés pour chiffrer un tunnel entre 2 machines ou une session de transport chiffré. Les

extensions décrive comment le contexte de sécurité est configuré et négocié entre les 2 systèmes (appelé Associations de Securité (SA) dans la terminologie IPSec).



Basiquement il se produit :

1. La base de stratégie de sécurité (SPD) définis les caractéristiques de communication sécurisé à utiliser entre les 2 systèmes. C'est complété en utilisant setkey
2. Le SA a ses paramètres de configuration tel que les protocoles utilisé pour sécuriser les paquets, les algorithmes de chiffrement et la durée de validité des clés dans la base d'association de sécurité (SAD). Pour IPSec labélisé le contexte de sécurité est également définis dans SAD. Les SA peuvent être négociés entre les 2 systèmes en utilisant racoon ou pluto qui vont automatiquement populer SAD et manuellement avec setkey.
3. Une fois les SA négociés et validés, le lien devrait être actif.

Noter que ces SA sont unidirectionnels, ainsi quand 2 systèmes communiquent, un système aura un SA. SAout pour traiter les paquets sortant, et un autre SAin pour traiter les paquets entrants. L'autre système devrat également créer 2 SA pour traiter ses paquets.

Chaque SA partage les même paramètres cryptographiques tels que les clés et protocoles ESP et AH.

La classe objet utilisée pour l'association d'un SA est 'association' et les permissions disponibles comme suit :

polmatch Matche le contexte SPD (-ctx) à un domaine SELinux (qui est contenu dans l'entrée -ctx SAD)
recvfrom Reçu depuis une association IPSec
sendto Envoyé à une association IPSec
setcontext Définis le contexte d'une association IPSec à la création.

En lançant IPSec labélisé il est recommandé que les systèmes utilisent le même type/version de stratégie pour éviter tout problème.

Exemples de configuration

Il y a 2 solutions disponibles :

Outils IPSec - setkey et racoon
LibreSwan - ipsec et pluto

Tous fonctionnent de la même manière mais utilisent différents fichiers de configuration. Le point qu'ils ont en commun est qu'ils démarrent une session en utilisant IKE, setkey doit être utilisé pour initialiser les labels dans la SPD dans chaque machine

Un autre point à noter est que si racoon et pluto sont interopérables, les valeurs d'attributs d'association de sécurité sont différents :

- racoon a cette valeur à 10
- pluto est configurable avec par défaut 32001. Pour intéropérer avec racoon, ipsec.conf doit avoir : **secctx_attr_value = 10**

L'exemple suivant montre la configuration setkey commune pour définir les entrées SPD et en exemple de configuration supportant racoon et pluto :

Ajouter un label/contexte au SPD pour la boucle locale :

```
flush;
spdflush;
spdadd 127.0.0.1 127.0.0.1 tcp -ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.ext_gateway.process:s0"
-P out ipsec esp/transport//require;
spdadd 127.0.0.1 127.0.0.1 tcp -ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.ext_gateway.process:s0"
-P in ipsec esp/transport//require;
spdadd 127.0.0.1 127.0.0.1 tcp -ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.int_gateway.process:s0"
-P out ipsec esp/transport//require;
spdadd 127.0.0.1 127.0.0.1 tcp -ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.int_gateway.process:s0"
-P in ipsec esp/transport//require;
```

Configuration racoon :

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";
path script "/etc/racoon/scripts";

sainfo anonymous
{
 lifetime time 1 hour ;
 encryption_algorithm 3des, blowfish 448, rijndael ;
 authentication_algorithm hmac_sha1, hmac_md5 ;
 compression_algorithm deflate ;
}
```

Configuration LibreSwan/pluto :

```
version 2.0
config setup
 plutorestartoncrash=false
 protostack=netkey
 plutodebug="all"
 secctx_attr_value = 32001
conn labeled_loopback_test
 auto=start
 rekey=no
 authby=secret
 type=transport
 left=127.0.0.1
 right=127.0.0.1
 ike=3des-sha1
 phase2=esp
 phase2alg=aes-sha1
 loopback=yes
 labeled_ipsec=yes
 policy_label=unconfined.user:msg_filter.role:msg_filter.ext_gateway.process:s0
 leftprotoport=tcp
 rightprotoport=tcp
```

Support des machines virtuelles

Le support de SELinux est disponible dans KVM/QEMU et Xen. Actuellement le principal support SELinux pour la virtualisation se fait via libvirt qui est une API de virtualisation open-source utilisée pour charger dynamiquement les VM. Les extensions de sécurité ont été ajoutés dans le projets Svirt et l'implémentation pour QEMU/KVM. Les sections qui suivent donnent une introduction à QEMU/KVM, puis libvirt.

Support QEMU/KVM

KVM est un module kernel qui utilise le kernel Linux comme hyperviseur et utilise un QEMU modifié pour supporter l'émulation hardware. Le support SELinux pour les VM est implémenté par le sous-système libvirt qui est utilisé pour gérer les images VM en utilisant un VMM, et comme KVM est basé sur Linux il a le support SELinux par défaut. Il y a également des module de stratégie référence pour supporter l'infrastructure générale.

Support libvirt

Le projet Svirt a ajouté des hooks de sécurité dans la librairie libvirt qui est utilisée par le service libvirtd. Le fournisseur de VM peut implémenter des mécanismes de sécurité qui nécessite d'utiliser un pilote libvirt spécifique qui va charger et gérer les images. L'implémentation SELinux supporte 4 méthodes de label des images VM, processus et leur ressources associées supporté dans le module modules/services/virt.* de la stratégie référence. Pour supporter le labeling, libvirt nécessite une stratégie MCS ou MLS vu que le level du contexte de sécurité est utilisé (user :role :type :level).

Labéliser les images VM

Le **labeling dynamique**, Le mode par défaut, lorsque chaque VM est lancé dans son propre domaine configuré dynamiquement et donc isole les VM entre-elles (chaque fois que la VM est lancée un label MCS différent et unique est généré). Ce mode est implémenté comme suit :

a) Un contexte initial pour le processus est obtenu depuis /etc/selinux/<SELINUXTYPE>/contexts/virtual_domain_context (défaut : system_u :system_r :svirt_tcg_t :s0)

b) UN contexte initial pour le fichier image est obtenu depuis /etc/selinux/<SELINUXTYPE>/contexts/virtual_image_context (defaut : system_u :system_r :svirt_image_t :s0 qui autorise la lecture/écriture des fichiers image)

c) Quand l'image est utilisée pour démarrer la VM, un niveau MCS est généré et ajouté au contexte du processus et au contexte du fichier image. Le processus et les fichiers images sont transités dans le contexte via setfilecon et setexeccon.

Si l'image disque doit être partagé, le niveau dynamiquement alloué sera généré par chaque instance du processus VM, cependant il y aure une seule instance de l'image disque (**image partagée**).

La ressource XML pour le contenu de <disk> doit inclure <shareable/>. Avec la stratégie targeted de F-20, l'option shareable donne une erreur, il est donc nécessaire d'activer la mémoire partagée :

setsebool -P virt_use_execmem on

Maintenant que l'image a été configurée comme partageable, le processus d'initialisation commence :

a) Un contexte initiale pour le processus est obtenu depuis /etc/selinux/<SELINUXTYPE>/contexts/virtual_domain_context (défaut : system_u :system_r :svirt_tcg_t :s0)

b) Un contexte initial pour le lable du fichier image est obtenu depuis /etc/selinux/<SELINUXTYPE>/contexts/virtual_image_context. Défaut : system_r :system_r :svirt_image_t :s0)

c) Quand l'image est utilisée pour démarrer la VM un niveau MCS aléatoire est généré et ajouté au contexte du processus (mais pas le fichier image). Le processus est transité au contexte approprié par les appels setfilecon et setexeccon.

Il est possible de définir les **labels statiques**, cependant en conséquence l'image ne peut pas être clonée en utilisant le VMM. C'est la méthoide utilisée pour configurer les VM dans les systèmes MLS via qu'il y a un label connu qui définis le niveau de sécurité. Avec cette méthode il est également possible de configurer 2 ou plusieurs VM avec le même contexte de sécurité pour qu'elles puissent partager des ressources.

Dans la ressource XML, ajouter :

```
...
</devices>
<seclabel type='static' model='selinux' relabel='no'>
 <label>system_u:system_r:avirt_t:s0:c1022,c1023</label>
</seclabel>

 </domain>
```

Services Sandbox

Fedora supporte les 3 type de service sandbox :

1. sandboxing non-GUI.
2. Sandboxing GUI utilisant le serveur Xephyr. Il permet l'isolation d'application X via des serveurs Xephyr imbriqués.

Ces services sandbox sont définis dans sandbox(3) et sont disponibles dans le package policycoreutils. Ils utilisent seunshare(8), qui permet de lancer des commandes dans un home alternatif. sandbox.conf permet de configurer le nom, cpu et utilisation mémoire.

Noter que les services sandbox nécessitent le support MCS vu que les catégories sont utilisées pour isoler les multiples sandbox.

3. sandboxing pour la virtualisation des application en utilisant soit QEMU/KVM ou LXC. ce service est disponible dans le package libvirt-sandbox et fournis une API et des services en ligne de commande pour démarrer des sessions.

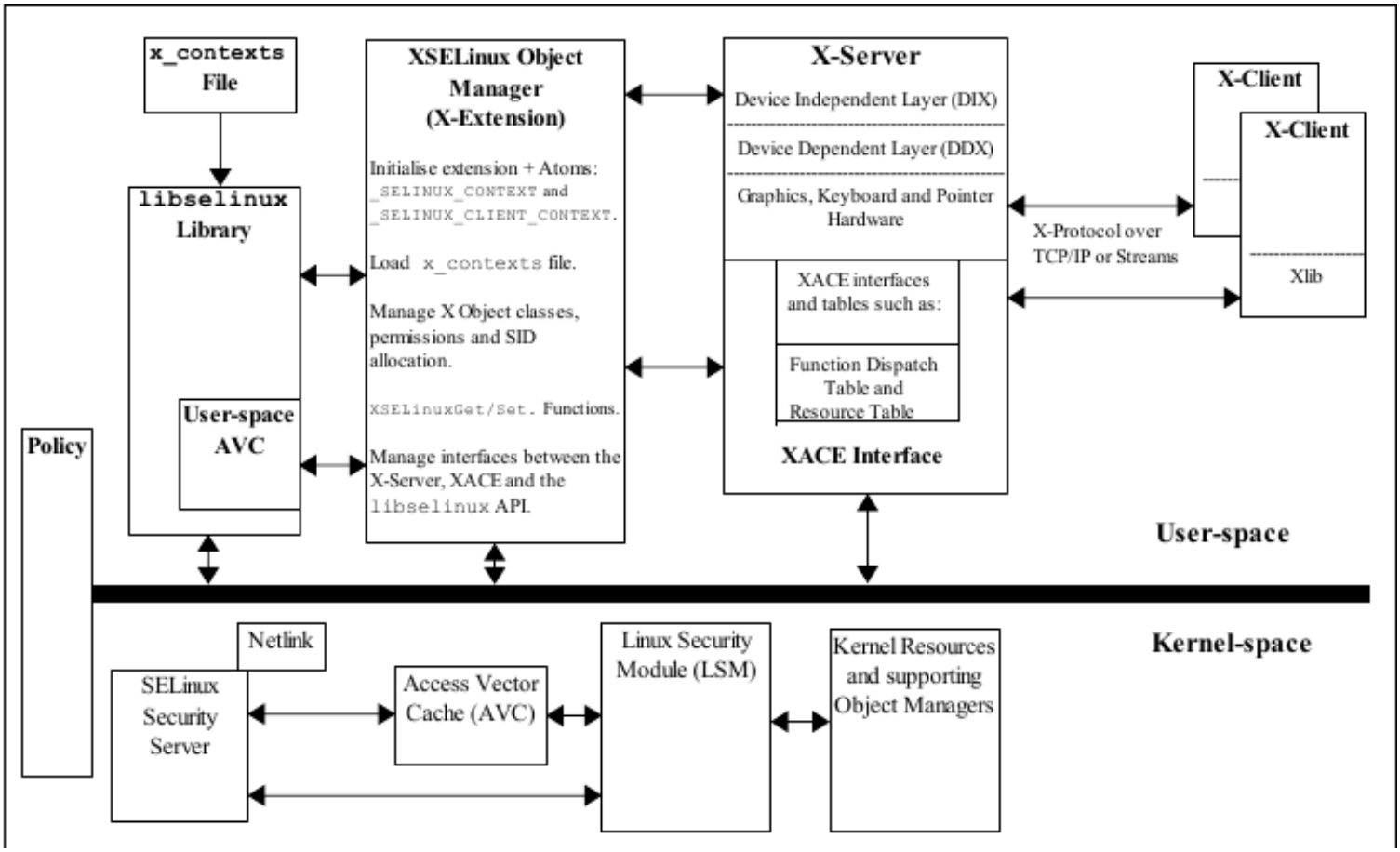
Le package est construit sur Svirt qui fournis la virtualisation avec SELinux et QEMU/KVM ou LXC pour fournir un environnement de virtualisation. Si le support de KVM n'est pas disponible dans la machine, LXC est l'alternative à utiliser.

Support X-Windows

L'implémentation XSELinux fournis un contrôle d'accès fin à la majorité des objets X-server en utilisant une extension X-Windows ajissant comme gestionnaire d'objet. Le nom de l'extension est 'SELinux'.

Dans Fedora XSELinux est désactivé dans la stratégie ciblée, mais activée dans la stratégie MLS, dû au fait que Red-Hat préfère utiliser le sandboxing avec Xephyr pour isoler les fenêtres.

Il est important de noter que l'OM X-Windows opère sur les objets bas niveau du serveur X. Un gestionnaire comme Gnome ou twm s'appuie dessus, cependant ils ne connaissent pas la stratégie forcée par SELinux.



Les composant majeur qui forment XSELinux sont :

la stratégie La stratégie référence a été mise à jours, cependant dans Fedora l'OM est activé pour mls uniquement. (xserver-object-manager).

libselinux La libairie fournie les interfaces nécessaire entre l'OM les services userspaces SELinux, et les services kernel.

fichier x_contexts Contient les informations de configuration de contexte par défaut qui sont requis par l'OM pour labéliser certains objets. L'OM le lit via selabel_lookup(3)

XSELinux Object Manager C'est une extension pour X-server qui agit comme médiateur pour les décisions d'accès entre X-server (via XACE) et le serveur de sécurité SELinux. l'OM est initialisé avant que les X-clients se connectent au X-server.

XACE X Access Control Extension peut être utilisé par d'autres extensions de sécurité de contrôle d'accès, pas seulement SELinux.

X-server Le cœur du serveur X-Windows qui gère les requêtes et réponses depuis/vers les clients X en utilisant le protocole X. l'OM XSELinux intercepte ces requêtes/réponses via XACE et force les décisions de la stratégie

X-clients Ils se connectent au serveur X et sont généralement des gestionnaires de fenêtre comme Gnome ou KDE.

Polyinstanciation

Les services OM/XACE supportent la polyinstanciation des propriétés et séléctions permettant d'être groupés dans différentes zones d'appartenance pour qu'un groupe ne connaisse pas l'existance des autres. Pour implémenter la polyinstanciation, le mot clé poly_ est utilisé dans le fichier x_context pour les séléctions et propriétés requises. Il devrait y avoir une règle type_member dans la stratégie pour forcer la séparation en calculant un nouveau context avec soit security_compute_member ou avc_compute_member

Noter que la stratégie référence actuelle n'implémente pas la polyinstanciation, mais la stratégie MLS utilise les règles mlsconstrain pour limiter le périmètre des propriétés et sélections.

Information de configuration

La stratégie référence a un booléen xserver_object_manager qui active/désactive le module de stratégie X-Server.

Le gestionnaire d'objet est traité comme une extension X-server et son opcode majeur peut être requêté avec la fonction Xlib XQueryExtension.

Si l'OM X-server doit tourner dans un mode SELinux spécifique, l'option peut être ajoutée dans le fichier xorg.conf (SELInux mode disabled|permissive|enforcing). S'il n'y a pas d'entrée, l'OM suit le mode courant

```
Section "Module"
 SubSection "extmod"
  Option "SELinux mode enforcing"
 EndSubSection
EndSection
```

Le fichier x_contexts contient les informations de contexte par défaut requis par l'OM pour initialiser le service et labéliser les objets quand ils sont créés. La stratégie nécessite également de connaître ces informaions de contexte pour forcer la stratégie ou la transition des nouveaux objets. Une entrée typique est comme suit :

object_type object_name context
selection PRIMARY system_u :object_r :clipboard_xselection_t :s0
ou pour la polyinstanciation
poly_selection PRIMARY system_u :object_r :clipboard_xselection_t :s0
object_name peut contenir '*' pour any ou ' ?' pour substitue. les entrées object_type sont client, property, poly_property, extension, selection, poly_selection et events.

Les entrées object_name peuvent être tout nom de ressource X qui sont définis dans le code source du serveur X et peuvent être trouvé dans protocol.txt et le fichier source BuildInAtoms.

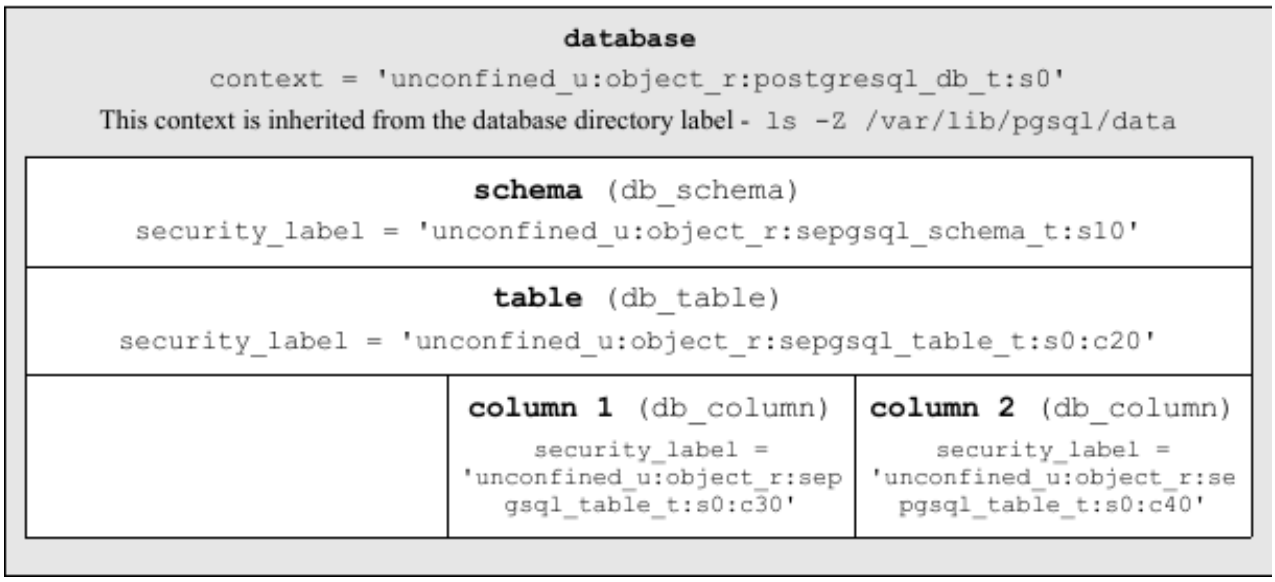
Notes :

1) La manière dont le code XSELinux foncion est que les entrées non-poly sont recherchés en premier, si une entrée n'est pas trouvée, cherche une entrée poly. La raison de ce comportement est qu'en opérant dans un environnement sécurisé tous les objets sont polyinstanciés sauf s'il y des exception pour certains objets.
2) Pour les systèmes utilisant la stratégie référence tous les clients X se connectant à distance reçoivent un contexte de sécurité depuis le fichier x_contexts.

SE-PostgreSQL

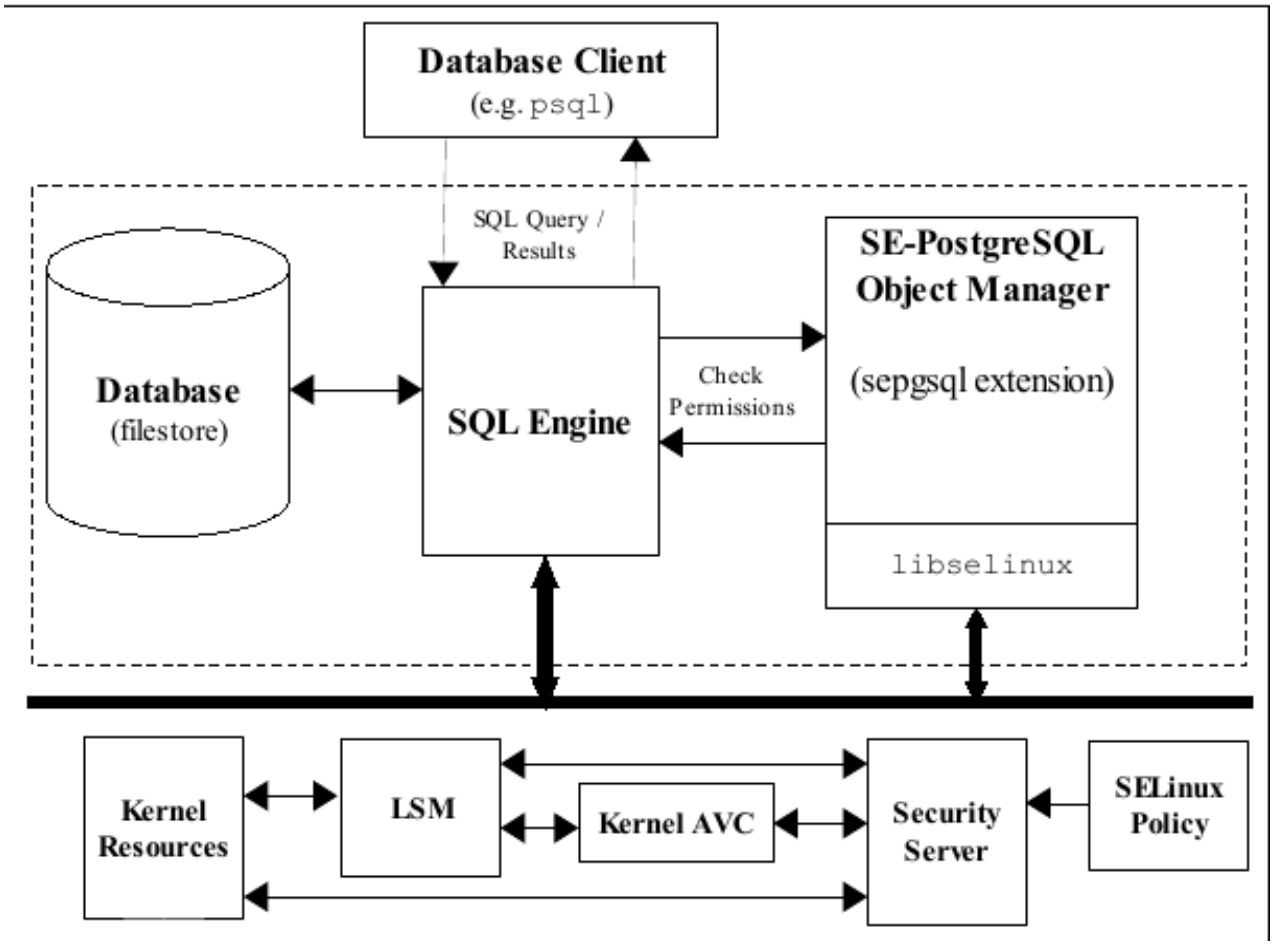
L'extension sepgsql permet de labéliser les objets de base PostgreSQL. Le labéling au niveau des entrées n'est plus supporté.

l'extension segpsql ajoute SELinux aux objets de base de données tels que les tables, colonnes, views, fonctions, schemas et séquences.



Pour utiliser SE-PostgreSQL chaque utilisateur GNU/Linux doit avoir un rôle de base de données valide (à ne pas confondre avec un rôle SELinx). L'installation par défaut ajoute automatiquement un utilisateur pgsql avec un rôle de base de données adapté.

Si un client se connecte à distance et qu'un label réseau est requis, il est possible d'utiliser IPSec ou NetLabel. Dans l'illustration ci-dessous, l'application cliente se connecte à une base et exécute les commandes SQL. vu que les commandes SQL sont traités par PostgreSQL, chaque opération effectuées sur un objet est vérifié par l'OM pour voir si l'opération est permise



La commande sql SECURITY LABEL a été ajoutée à PostgreSQL pour autoriser les fournisseurs de sécurité à labéliser ou changer un label dans les objets de base de données. Le format de la commande est :

```
SECURITY LABEL [ FOR provider ] ON
{
```

```
 TABLE object_name |
 COLUMN table_name.column_name |
 AGGREGATE agg_name (agg_type [, ...] ) |
 DATABASE object_name |
 DOMAIN object_name |
 EVENT TRIGGER object_name |
 FOREIGN TABLE object_name
 FUNCTION function_name ( [ [ argmode ] [ argname ] argtype
[, ...] ] ) |
 LARGE OBJECT large_object_oid |
 [ PROCEDURAL ] LANGUAGE object_name |
 ROLE object_name |
 SCHEMA object_name |
 SEQUENCE object_name |
 TABLESPACE object_name |
 TYPE object_name |
 VIEW object_name
} IS 'label'
```

Quelques exemples :
SECURITY LABEL ON SCHEMA test_ns IS 'unconfined_u :object_r :sepgsql_schema_t :s0 :c10' ;
SECURITY LABEL ON TABLE test_ns.info IS 'unconfined_u :object_r :sepgsql_table_t :s0 :c20' ;
SECURITY LABEL ON COLUMN test_ns.info.user_name IS 'unconfined_u :object_r :sepgsql_table_t :s0 :c30' ;
SECURITY LABEL ON COLUMN test_ns.info.email_addr IS 'unconfined_u :object_r :sepgsql_table_t :s0 :c40' ;

Fonctions SQL additionnelles

les fonctions suivantes ont été ajoutées :

sepgsql_getcon Retourne le contexte de sécurité du client

sepgsql_mcstrans_in(text con) Traduit la plage du contexte en format brut fournis par mcstransd

sepgsql_restorecon(text specfile) Définis les contextes de sécurité dans tous les objets de base de données en accord avec specfile. Normalement utilisé à l'initialisation

postgresql.conf

Le fichier postgresql.conf supporte les entrées additionnelles suivante pour permettre et gérer SE-PostgreSQL :

Cette entrée est obligatoire pour active l'extension sepgsql
shared_preload_libraries = 'sepgsql'
Permet de personnaliser les entrées sepgsql
custom_variable_classes = 'sepgsql'
Permet de lancer sepgsql en mode permissive
sepgsql.permissive = on
ACtive les messages d'audit sans regarder les paramètres de la stratégie
sepgsql.debug_audit = on
pour voir ces paramètres, la déclaration SHOW peut être utilisée :
SHOW sepgsql.permissive ;
SHOW sepgsql.debug_audit ;
SE-PostgreSQL gère ses propres entrées d'audit AVC dans les logs standard PostgreSQL dans /var/lib/pgsql/data/pg_log, et par défaut seul les erreurs sont loggés. 'sepgsql.debug_audit = on' peut être spécifié pour logger tous les évènements d'audit.

Pour supporter les opérations de base de données PostgreSQL a des tables internes dans le catalogue système qui maintient les

informations sur les bases, tables, etc. Cette section discute de la table pg_seclabel qui maintient les labels de sécurité et autres références.

objoid L'OID de l'objet pour lequel ce label est pertinent

classoid L'OID du catalogue système dans lequel cet objet apparaît

objsubid Pour un label de sécurité dans une colonne, c'est le numéro de colonne. Pour tous les autres objets cette colonne est 0.

provider Fournisseur de label associé avec ce label. Actuellement seul SELinux est supporté

lael Label de sécurité appliqué à cet objet

Ce sont des entrées prises depuis un 'SELECT * FROM pg_seclabel ;' d'une base de test :

```
objoid | classoid | objsubid | provider | label
-----+-------+-------+-------+------------------------------
16390 | 2615 | 0 | selinux | unconfined_u:object_r:sepgsql_schema_t:s0:c10
16391 | 1259 | 0 | selinux | unconfined_u:object_r:sepgsql_table_t:s0:c20
16391 | 1259 | 1 | selinux | unconfined_u:object_r:sepgsql_table_t:s0:c30
16391 | 1259 | 2 | selinux | unconfined_u:object_r:sepgsql_table_t:s0:c40
```

La première entrée est le schéma, le second la table elle-même, et les 2 dernières sont les colonnes 1 et 2. Il y a également une vue intégrée pour afficher des détails sur les labels de sécurité appelés pg_seclabels. 'SELECT * FROM pg_seclabels ;' pour l'afficher :

```
objoid | classoid | objsubid | objtype | objnamespace | objname | provider | label
-----+-------+-------+------+---------+------+-------+------------------
16390 | 2615 | 0 | schema | 16390 | test_ns | selinux | unconfined_u:object_r:sepgsql_schema_t:s0:c10
16391 | 1259 | 0 | table | 16390 | test_ns.info | selinux | unconfined_u:object_r:sepgsql_table_t:s0:c20
16391 | 1259 | 1 | column | 16390 | test_ns.info.user_name | selinux |
unconfined_u:object_r:sepgsql_table_t:s0:c30
16391 | 1259 | 2 | column | 16390 | test_ns.info.email_addr| selinux |
unconfined_u:object_r:sepgsql_table_t:s0:c40
```

Support Apache

Les serveurs Apache sont supportés par SELinux en utilisant les modules de stratégie Apache de la stratégie référence (les modules httpd), cependant il n'y a pas de gestionnaire d'objet Apache. Il y a une librairie partagée qui permet un contrôle d'accès fin en utilisant Apache avec les threads. Le module additionel est appelé mod_selinux.so et supporte le module de stratégie appelé mod_selinux.pp

mod_selinux utiliser la déclaration typebounds qui a été introduit dans la version 24 de la stratégie. Il permets les threads dans une application multi-thread comme apache d'être lié dans un jeu de permissions définis et le domaine enfant ne peut pas avoir plus de permissions que le domaine parent.

Ces composants sont connus sous le nom Apache/SELinux Plus. L'objectif de ces services addon Apache est de créer une pile web compatible SELinux. Par exemple, la pile LAPP (Linux, Apache, PostgreSQL, PHP/Perl/Python) a le support suivant :

L Linux support SELinux

A Apache a un support partiel de SELinux

P PostgreSQL supporte SELinux avec SE-PostgreSQL

P PHP/Perl/Python ne sont pas actuellement compatible SELinux, cependant PHP et Python ont un support pour les fonctions de la libselinux : php-pecl-selinux et libselinux-python.

mod_selinux

Ce module autorise une application web d'être lancé par Apache avec un contexte de sécurité basé sur la stratégie au lieu que le serveur web le traite lui-même. par exemple :

1. Un utitisateur envoie une requête HTTP à Apache qui nécessite les services d'une application web
2. Apache reçois la requête et lance l'instance de l'application web pour effectuer la tâche :

   a) Sans mod_seliunx activé le contexte de sécurité des applications web sont identique au processus apache, il n'y a pas de restriction de privilèges

   b) Avec mod_selinux activé, l'application web est lancée avec le contexte de sécurité définis dans mod_selinux.conf

L'application web quitte, redonnant le contrôle au serveur web qui répond avec la réponse HTTP.

Limites

Parce que plusieurs threads partagent le même segment mémoire, SELinux ne peut pas vérifier les flux d'informations entre ces différents threads en utilisant setcon(3) avant le kernel 2.6.28. Pour résoudre ce problème la déclaration typebounds a été introduite qui stoppe un thread enfant ayant des privilèges supérieur au thread parent. par exemple la déclaration typebounds et les règles allow :

```
typebounds httpd_t httpd_child_t;
allow httpd_t etc_t : file { getattr read };
allow httpd_child_t etc_t : file { read write };
```

Déclare que le domaine parent (httpd_t) a les permissions file : { getattr read }. Cependant le domaine enfant (httpd_child_t) a les permission file : { read write }. Cela ne sera pas permis par le kernel parce que le parent n'a pas les permissions write, et s'assure que le domaine enfant aura toujours les même permissions ou moins de permissions que le parent.

Quand setcon(3) est utilisé pour définir un contexte différent dans un nouveau thread sans une déclaration typebounds associée, l'appel retourne 'Operation not permitted' et une entrée SELINUX_ERR est ajoutée au logs d'autid indiquant 'op=security_bounded_transition result_denied' avec les ancien et nouveau contexte.

S'il y a une déclaration typebounds valide et que le domaine enfant tente d'obtenir un privilège supérieur que le domaine parent est refusé avec une entrée SELINUX_ERR ajoutée dans les log d'audit indiquant 'op=security_compute_av reason=bounds'.

Le document d'exemple contient 2 démonstrations utilisant setcon(3) avec les threads et comment la déclaration typebounds est utilisée pour autoriser un thread à être exécuté. Il sont localisés dans libselinux/examples et sont :

a) setcon_thread1_example.c - qui appelle setcon dans le processus principal et lance un thread. Si le module setcon_example.conf est chargé et qu'une contexte "unconfined_u :unconfined_r :user_t :s0" est sélectionné, un message d'erreur est affiché : "setcon_raw - ERROR : Operation not permitted"

b) setcon_thread2_example.c - ces fonctions sont similaire, cependant il appel setcon depuis un thread

Fichiers de configuration de SELinux

Cette section explique chaque fichier de configuration SELinux. Les fichiers de configuration sont classés comme suit :

configuration globale - affecte la stratégie active et les application supportant SELinux, utilitaires et commandes

configuration de stratégie - utilisés par une stratégie active

configuration kernel - localisé sous /sys/fs/selinux et reflète la configuration courante

Migration du magasin de stratégie

Quand les distributions sont passés à la version 2.4 de libsemanage, libsepol et policycoreutils, le magasin de module de stratégie a été déplacé vers /var/lib/selinux/<SELINUXTYPE>. Une fois les librairies mis à jours, tous les magasins doivent être migrés avant que toute

commande soit exécutée. Une fois la migration complétée, il est possible de construire les stratégies contenant un mix des modules de stratégie référence, modules de langage de stratégie kernel et modules écris en CIL comme dans l'exemple suivant :

Compiler et installer une base et 2 modules écrits en langage kernel
checkmodule -o base.mod base.conf
semodule_package -o base.pp base.mod -f base.fc
checkmodule -m ext_gateway.conf -o ext_gateway.mod
semodule_package -o ext_gateway.pp -m ext_gateway.mod -f gateway.fc
checkmodule -m int_gateway.conf -o int_gateway.mod
semodule_package -o int_gateway.pp -m int_gateway.mod
semodule -s modular-test –priority 100 -i base.pp ext_gateway.pp int_gateway.pp
Compiler et installer un module écrit en CIL
semodule -s modular-test –priority 400 -i custom/int_gateway.cli
Affiche la liste des modules :
semodule -s modular-test –list-modules=full
affiche un listing standard des modules
semodule -s modular-test –list-modules=standard
Noter l'utilisation de –priority disponible après migration de semodule. Les priorités permettent d'avoir plusieurs modules avec le même nom dans le magasin de stratégie. Le module ayant la priorité la plus haute est incluse dans le binaire kernel final, les autres sont ignorés.

/etc/selinux/config

Ce fichier est obligatoire, sans lui, ou corrompu, aucune stratégie SELinux n'est chargée et SELinux est désactivé.

/etc/selinux/semanage.conf

Ce fichier contrôle la configuration et les actions de semanage et semodule.

/etc/selinux/restorecond[-user].conf

Ces fichiers contient les fichiers et répertoire que restorecond surveille afin de corriger le contexte de sécurité

/etc/selinux/newrole_pam.conf

Ce fichier est utilisé par la commande newrole et mappe les applications ou commandes en fichiers de configuration PAM. Chaque ligne contient le nom du fichier exécutable suivi par le nom d'un fichier de configuration pam qui existe dans /etc/pam.d

/etc/sestatus.conf

Utilisé par sestatus pour lister les fichiers et processus dont le contexte de sécurité doit être affiché avec -v.

/etc/security/sepermit.conf

Fichier de configuration pour le module PAM pam_sepermit

Fichiers de configuration du magasin de stratégie

/var/lib/selinux/<policy_name>/modules. Noter qu'il peut y avoir plusieurs magasins de stratégie dans un système, chaque fichier décris dans cette section est relatif à ./<policy_name>. Ces fichiers sont installés, mis à jours ou construit par semodule et semanage. Les fichiers résultant sont soit copiés dans le répertoire, ou utilisés pour reconstruire la stratégie binaire kernel dans /etc/selinux/<policy_name>/policy.

modules/ le magasin a 2 fichiers de lock utilisés par libsemanage pour gérer les magasin (semanage.read.LOCK et semanage.trans.LOCK)

modules/active/base.pp Package de stratégie de base qui contient les modules obligatoires et les composants de stratégie tels que les classe objets, déclarations de permission et SID initiaux.

modules/active/base.linked Seulement présent si save-linked vaut TRUE dans semanage.conf. Il contient les modules qui ont été liés avec semodule_link

modules/active/commit_num Fichier binaire utilisé par libsemanage pour gérer les mises à jours dans le magasin.

modules/active/file_contexts.template Contient une copie de toutes les entrées de module 'Labeling Policy File' qui ont été extraits de base.pp et les modules chargeables dans le répertoire modules/active/modules. Ces entrées sont utilisées pour construire les fichiers suivants :

homedir_template Utilisé pour produire le fichier file_contexts.homedirs qui devient ensuite le fichier contexts/files/file_contexts.homedirs

file_contexts Qui devient ensuite le fichier contexts/files/file_contexts

Noter que dans le processus de construction de semanage, ces 2 fichiers on également les fichiers file_contexts.bin et file_contexts.homedirs.bin présents dans contexts/files, dû au fait que semanage nécessite des expression régulière Perl (PCRE). Ils sont générés par sefcontext_compile.

homedir_template

file_contexts

Le format du fichier file_contexts.template est comme suit :

```
pathname_regexp [file_type] opt_security_context
```

pathname_regexp Une entrée qui définis le chemin qui peut être une expression régulière

file_type Une des entrées optionnelles suivante : -b|c|d|p|l|s|-.

opt_security_context Cette entrée peut être soit le contexte de sécurité, incluant le niveau et plage MLS/MCS, ou la valeur «none» pour indiquer que les fichiers qui matchent ne devraient pas être relabélisés

Les mots clé qui peuvent être dans ce fichier sont :

HOME_ROOT Remplacé par le répertoire racine des répertoires personnels, normalement /home.

HOME_DIR Répertoire des répertoires personnels utilisateurs, défaut /home.

USER Remplacé par le user id

ROLE Remplacé par l'entrée 'prefix' du fichier de configuration users_extra qui correspond au user id des utilisateurs SELinux.

Exemple de file_contexts.template

```
/. /.. /.config /.hcwd /.local system_u:object_r:default_t:s0
/[^/]+ - system_u:object_r:etc_runtime_t:s0
/a?quota\.(user|group) - system_u:object_r:quota_db_t:s0
/nsr(/.*)? system_u:object_r:var_t:s0
/sys(/.*)? system_u:object_r:sysfs_t:s0
...
/etc/ntop.* system_u:object_r:ntop_etc_t:s0
HOME_DIR/.+ system_u:object_r:user_home_t:s0
/dev/dri/.+ -c system_u:object_r:dri_device_t:s0
...
/tmp/gconfd-USER -d system_u:object_r:user_tmp_t:s0
...
/tmp/gconfd-USER/.* - system_u:object_r:gconf_tmp_t:s0
...
HOME_ROOT/\.journal «none»
```

modules/active/file_contexts Ce fichier devient les fichiers de stratégie contexts/files/file_contexts et est construit depui modules/active/file_contexts.template. Il est ensuite utilisé par les utilitaires de labéling de fichier pour s'assurer que les fichiers et répertoires sont labélisés en accord avec la stratégie.

modules/active/homedir_template Ce fichier est construit depuis les entrées dans le fichier file_contexts.template et utilisé par genhomedircon,semanage login, et semanage user pour générer des entrées utilisateurs dans le fichier file_contexts.homedirs

modules/active/file_contexts.homedirs Ce fichier devient le fichier de stratégie contexts/files/file_contexts.homedirs en construisant la stratégie. Il est ensuite utilisé par les utilitaires de label de fichier pour s'assurer que les répertoires home sont labélisés en accord avec la stratégie.

modules/active/netfilter_contexts & netfilter.local Ces fichiers ne sont pas utilisés pour le moment. Contiennent du code pour produire un fichier netfilter_contexts à utiliser par le services iptables.

modules/active/policy.kern Ce fichier binaire construite pas semanage ou semodule, qui devient la stratégie binaire policy/policy.[ver] qui sera chargé dans le kernel

modules/active/seusers.final & seusers Le fichier seusers.final mappe les utilisateurs GNU/Linux en utilisateurs SELinux et devient le fichier seusers. seusers.final est construit ou modifié en construisant une stratégie ou un fichier optionnel seusers est inclus dans le package de base ou par la commande semanage login.

modules/active/users_extra, users_extra.local, users.local users_extra et users_extra.local sont utilisés pour mapper un préfixe en répertoires personnels utilisateurs, où il est utilisé pour remplacer le mot clé ROLE. Le préfixe est lié à un utilisateur SELinux et devrait refléter le rôle de l'utilisateur. La commande semanage user permet d'ajouter un préfixe. users_extra contient toutes les entrées de préfixe de stratégie, et users_extra.local contient ceux générés par semanage user. users.local est utilisé pour ajouter de nouveaux utilisateurs SELinux à la stratégie sans éditer la stratégie.

modules/active/booleans.local Ce fichier est créé et mis à jours par semanage boolean et maintient les booléens

modules/active/file_contexts.local Ce fichier est créé et mis à jours par semanage fcontext et maintient les informations de contexte de fichiers et répertoires qui ne sont pas fournis par la stratégie core (ex : ne sont pas définis dans des fichiers .fc)

modules/active/interfaces.local Ce fichier est créé et mis à jours par la commande semanage interface et maintient les informations d'interface réseaux qui ne sont pas fournies par la stratégie core (ex : ne sont pas définis dans le fichier base.conf). Les nouvelles informations sont ensuite construites dans la stratégie par la commande semanage. Chaque ligne contient une déclaration netifcon.

modules/active/nodes.local Ce fichier est créé ni mis à jours par la commande semanage node et maintient des informations sur les adresses réseaux qui ne sont pas fournies par la stratégie core (ne sont pas définis dans base.conf). Ces informations sont ensuite construite dans la stratégie par la commande semanage. Chaque ligne contient une déclaration nodecon.

modules/active/ports.local Ce fichier est créé et mis à jours par semanage port et maintient les informations sur les port réseaux qui n'ont pas été fournis par la stratégie core. Chaque ligne contient une déclaration portcon

modules/active/preserve_tunables Ce fichier n'existe que si la stratégie construite préserve les personnalisations.

modules/active/disable_dontaudit Ce fichier n'existe que si la stratégie construite en ayant désactivé les règles dontaudit

modules/active/modules Ce répertoire contient des modules chargeable (<module_name>.pp ou <module_name>.pp.disabled si désactivé) qui ont été construits par la commande semodule_package et placées dans le magasin par les commande semodule ou semanage module -a.

Fichiers de configuration de stratégie

Chaque fichier dans cette section est relatif à /etc/selinux/<policy_name>. La majorité des fichiers sont installés par la stratégie référence, semanage ou semodule. Il est possible de construire des stratégie monolitiques personnalisée qui utilisent seulement les fichiers installés dans cette zone (ex : n'utilise pas semanage ou semoduleè).

policy/policy.29 Stratégie binaire chargée dans le kernel

context/files/file_contexts Pour permettre au système de fichier d'être relabelisé

context/dbus_contexts Pour permettre au service de messagerie dbus à fonctionner sous SELinux

context/x_contexts Pour autoriser le service X-Windows à fonctionner sous SELinux

fichier seusers

Le fichier seusers est utilisé par les programes login et par les utilisateurs GNU/Linux en utilisateurs SELinux. Une séquence de connexion serait :

- En utiliser le userid GNU/Linux, recherche re selinux_id dans ce fichier. S'il n'est pas trouvé, utilise l'entrée __default__
- Pour déterminer le contexte à utiliser, lit contexts/users/[seuser_id]. S'il n'est pas présent :
  - Recherche un contexte par défaut dans contexts/default_contexts. Si aucun contexte par défaut n'est trouvé, lit contexts/failsafe_context pour autoriser un contexte sûr.

Noter que l'utilisateur system_u est définis dans ce fichier, cependant il ne doit pas y avoir d'utilisateur GNU/Linux system_u dans le système.

Fichiers booleans et booleans.local

Dépréciés et généralement non présent, ces fichiers gèrent les booléens.

booleans.subs_dist

Ce fichier, si présent, permet d'allouer de nouveaux noms de booléens dans la stratégie active. Ce fichier a été ajouté parce que beaucoup d'anciennes booléennes commençaient avec 'allow' qui rendait difficile à déterminer ce qu'ils faisaient. Par exemple allow_console_login et plus compréhensible avec login_console_enabled. Si ce fichier est présent, les 2 noms peuvent être utilisés. Chaque ligne est sous la forme **policy_bool_name new_name**.

setrans.conf

Ce fichier est utilisé par mcstrands pour traduire les niveaux de stratégies MCS/MLS en labels user friendly.

secolor.conf

secolor.conf contrôle la couleur associée avec les composant d'un contexte quand les informations sont affichées par SELinux par une application gérant les couleurs.

policy/policy.<ver>

Fichier de stratégie binaire qui est chargé dans le kernel pour forcer la stratégie et est construit par checkpolicy ou semodule. Par convention l'extention du nom de fichier est la version de la base de stratégie utilisée pour construire la stratégie.

contexts/customizable_types

Ce fichier contient une liste de types qui ne sont pas relabélisés par setfiles ou restorecon. Les commandes vérifient ce fichier avant de relabéliser et exclus ceux dans cette liste sauf si -F est utilisé.

contexts/default_contexts

Ce fichier est utilisé par les application compatible SELinux pour définir un contexte de sécurité pour les processus utilisateurs (généralement les applications login) où l'identité de l'utilisateur GNU/Linux doit être connu par l'application.

contexts/dbus_contexts

Ce fichier est pour le service de messagerie dbus qui est utilisé par certaines applications comme KDE. Si SELinux est activé, ce fichier doit exister pour que ces applications puissent fonctionner.

contexts/default_type

Le fichier permet aux applications comme newrole de sélectionner un type par défaut pour un rôle si aucun n'est fournis.

contexts/failsafe_context

Le fichier failsafe_context est utilisé quand un process login ne peut déterminer un contexte par défaut à utiliser. Le contenu du fichier est utilisé pour permettre à un administrateur d'accéder au système.

contexts/initrc_context

Utilisé par la commande run_init pour permettre de démarrer les services système dans le même contexte de sécurité que init.

contexts/lxc_contexts

Ce fichier supporte le labéling des conteneurs lxc dans la librairie libvirt. Le format est le suivant :

```
process = "security_context"
file = "security_context"
```

```
content = "security_context"
```

process Une entrée qui contient le contexte de sécurité de domaine lxc
file contient le contexte de sécurité de fichier lxc
content Une entrée qui contient le contexte de sécurité de contenu lxc
sandbox_kvm_process
sandbox_lxc_process Ces entrées peuvent être présents

contexts/netfilter_contexts

Ce fichier supporte le labeling Secmark pour netfilter/iptable. Il n'est pas utilisé actuellement.

contexts/removable_context

Ce fichier contient un label par défaut pour les périphérique hot-plug qui ne sont pas définis dans contexts/files/media.

contexts/securetty_types

Utilisé par newrole pour trouver le type à utiliser avec les périphériques tty en changeant les rôles ou niveaux.

contexts/sepgsql_contexts

Ce fichier contient les contextes de sécurité par défaut pour les objets de base SE-PostgreSQL, décrit dans selabel_db

contexts/systemd_contexts

Ce fichier contient les contextes de fichiers utilisé par les tâches lancées via systemd. Le format est **service_class = security_context**.Exemple :
`runtime=system_u:object_r:systemd_runtime_unit_file_t:s0`

contexts/userhelper_context

Ce fichier contient le contexte de sécurité par défaut utilisé par les applications system-config-*, lancés en root.

contexts/virtual_domain_context

Ce fichier est utilisé par libvirt et fournis les contextes de domaine qemu disponibles dans la stratégie. Il peut y avoir 2 entrées dans ce fichier, la deuxième étant un contexte alternatif.

contexts/virtual_image_context

Ce fichier est utilisé par libvirt et fournis les contextes d'image qui sont disponibles dans la stratégie. La première entrée est le contexte de fichier image et la seconde est le contexte du contenu de l'image.

contexts/x_contexts

Ce fichier fournis les contextes de sécurité par défaut pour l'extension de sécurité SELinux, décris dans selabel_x.

contexts/files/file_contexts

Ce fichier est géré par semodule et semanage et ne devrait pas être édités. Ce fichier est utilisé par des commandes comme setfiles, fixfiles, matchpathcon, restorecon pour relabéliser parties ou tout un système de fichier.

contexts/files/file_contexts.local

Ce fichier est géré par semanage fcontext.

contexts/files/file_contexts.homedirs

Ce fichier est géré par semodule et semanage et ne devrait pas être édité. Il est généré par genhomedircon et utilisé pour définir les contextes de sécurité dans les répertoires personnels.

contexts/files/file_contexts.subs[_dist]

Ces fichiers permettent des substitutions de noms de fichier pour les fonctions matchpatchcon(3) et selabel_lookup(3).

contexts/files/media

Ce fichier est utilisé pour mapper les types de média en un contexte de fichier.

contexts/users/[seuser_id]

Ces fichiers optionnels sont nommés d'après les utilisateurs SELinux qu'ils représentent. Chaque fichier est utilisé pour assigner le contexte de sécurité correct à l'utilisateur, généralement durant le login.

logins/<linuxuser_id>

Ces fichier optionnels sont utilisés par les application de login pour obtenir un username SELinux et un niveau basé sur l'UID GNU/Linux et le nom du service.

users/local.users

Généralement le fichier local.users n'est pas présent si semanage est utilisé pour gérer les utilisateurs. Ce fichier contient les définitions des utilisateurs locaux sous la forme de déclarations user.

Langage de stratégie SELinux

Cette section est une référence des déclarations règles de langage de stratégie.

Langage de stratégie kernel

Il y a 3 types de base de fichier source de stratégie qui peuvent contenir des déclarations et règle de langage. Ces 3 type sont :

monolitique Simple fichier source qui contient toutes les déclarations. Par convention, il est appelé policy.conf et est compilé par checkpolicy.

base Fichier source obligatoire qui supporte l'infrastructure modulaire. Toute la stratégie système peut être contenu dans ce fichier, cependant il est plus courant d'avoir des fichiers sources de modules chargeable séparés. Par convention ce fichier est appelé base.conf

module Ces fichiers sources optionnels peuvent être chargés dynamiquement dans le magasin de stratégie. Par convention ces fichiers sont nommés d'après le module qu'ils représentent.

Le tableau ci-dessous affiche l'ordre dans lequel les déclarations devraient apparaître dans les fichiers sources

Base Entries	*M/O*	*Module Entries*	*M/O*
Security Classes (`class`)	m	`module` Statement	o
Initial SIDs	m		
Access Vectors (permissions)	m	`require` Statement	o
MLS sensitivity, category and level Statements	o		
MLS Constraints	o		
Policy Capability Statements	o		
Attributes	o	Attributes	o
Booleans	o	Booleans	o
Default user, role, type, range rules	o		
Type / Type Alias	m	Type / Type Alias	o
Roles	m	Roles	o
Policy Rules	m	Policy Rules	o
Users	m	Users	o
Constraints	o		
Default SID labeling	m		
`fs_use_xattr` Statements	o		
`fs_use_task` and `fs_use_trans` Statements	o		
`genfscon` Statements	o		
`portcon`, `netifcon` and `nodecon` Statements	o		

La grammaire du langage définis les déclaration et règles qui peuvent être utilisées dans les types de fichier source. Pour souligner ces règles une indication est incluse dans chaque déclaration et règle pour montrer dans quelles circonstances elle est valide dans un fichier de stratégie source :

```
Monolithic | Base | Module
Yes/No | Yes/No | Yes/No
```

Règles de déclaration conditionnelle, optionnelle et requise

La grammaire spécifie quelles déclaration et règles peuvent être incluses dans les déclarations conditionnelles, optionnelles et requise. Pour souligner ces règles une indication est incluse dans chaque déclaration et règle pour montrer dans quelles circonstances chacune est valide dans un fichier de stratégie source :

```
if | optional | require
Yes/No | Yes/No | Yes/No
```

Déclarations MLS et composants MLS optionnels

Quand MLS est activé, il y a d'autres déclaration qui nécessitent le composant MLS comme argument.

Informations générales

1. Les identifiant peuvent généralement être de n'importe quelle longueur mais devraient être restreints aux caractères suivant : [a-zA-Z0-9_]
2. Un '#' indique le début d'un commentaire
3. Toutes les déclarations disponible dans la stratégie version 29 ont été incluses
4. Quand plusieurs entrées source et cible sont affichés en une seul déclaration ou règle, le compilateur étend ces déclaration ou règles individuelles.
5. Certaines déclarations peuvent être ajoutées à une stratégie via le magasin de stratégie en utilisant semanage.
6. Les mots réservés sont :

```
alias allow and attribute attribute_role auditallow
auditdeny bool category cfalse class clone
common constrain ctrue dom domby dominance
dontaudit else equals false filename filesystem
fscon fs_use_task fs_use_trans fs_use_xattr genfscon h1
h2 identifier if incomp inherits iomemcon
ioportcon ipv4_addr ipv6_addr l1 l2 level
mlsconstrain mlsvalidatetrans module netifcon neverallow nodecon
not notequal number object_r optional or
path pcidevicecon permissive pirqcon policycap portcon
r1 r2 r3 range range_transition require
role roleattribute roles role_transition sameuser sensitivity
sid source t1 t2 t3 target true type typealias
typeattribute typebounds type_change type_member types type_transition
u1 u2 u3 user validatetrans version_identifier
xor default_user default_role default_type default_range low
high low_high
```

7. La table ci-dessous indique quelles déclarations et règles sont permises dans chaque type de fichier source, et si la déclaration est valide dans une construction if/else, optional {rule_list}, ou require {rule_list}

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement[42]
allow	Yes	Yes	Yes	Yes	Yes	No
allow - Role	Yes	Yes	Yes	No	Yes	No
attribute	Yes	Yes	Yes	No	Yes	Yes
attribute_role	Yes	Yes	Yes	No	Yes	Yes
auditallow	Yes	Yes	Yes	Yes	Yes	No
auditdeny (Deprecated)	Yes	Yes	Yes	Yes	Yes	No
bool	Yes	Yes	Yes	No	Yes	Yes
category	Yes	Yes	No	No	No	Yes
class	Yes	Yes	No	No	No	Yes
common	Yes	Yes	No	No	No	No
constrain	Yes	Yes	No	No	No	No
default_user	Yes	Yes	No	No	No	No
default_role	Yes	Yes	No	No	No	No
default_type	Yes	Yes	No	No	No	No
default_range	Yes	Yes	No	No	No	No
dominance - MLS	Yes	Yes	No	No	No	No
dominance - Role (Deprecated)	Yes	Yes	Yes	No	Yes	No
dontaudit	Yes	Yes	Yes	Yes	Yes	No
fs_use_task	Yes	Yes	No	No	No	No
fs_use_trans	Yes	Yes	No	No	No	No
fs_use_xattr	Yes	Yes	No	No	No	No
genfscon	Yes	Yes	No	No	No	No
if	Yes	Yes	Yes	No	Yes	No
level	Yes	Yes	No	No	No	No
mlsconstrain	Yes	Yes	No	No	No	No

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement
mlsvalidatetrans	Yes	Yes	No	No	No	No
module	No	No	Yes	No	No	No
netifcon	Yes	Yes	No	No	No	No
neverallow	Yes	Yes	Yes[43]	No	Yes	No
nodecon	Yes	Yes	No	No	No	No
optional	No	Yes	Yes	Yes	Yes	Yes
permissive	Yes	Yes	Yes	Yes	Yes	No
policycap	Yes	Yes	No	No	No	No
portcon	Yes	Yes	No	No	No	No
range_transition	Yes	Yes	Yes	No	Yes	No
require	No	Yes[44]	Yes	Yes[45]	Yes	No
role	Yes	Yes	Yes	No	Yes	Yes
roleattribute	Yes	Yes	Yes	No	Yes	No
role_transition	Yes	Yes	Yes	No	Yes	No
sensitivity	Yes	Yes	No	No	No	Yes
sid	Yes	Yes	No	No	No	No
type	Yes	Yes	Yes	No	No	Yes
type_change	Yes	Yes	Yes	Yes	Yes	No
type_member	Yes	Yes	Yes	Yes	Yes	No
type_transition	Yes	Yes	Yes	Yes	Yes	No
typealias	Yes	Yes	Yes	No	Yes	No
typeattribute	Yes	Yes	Yes	No	Yes	No
typebounds	Yes	Yes	Yes	No	Yes	No
user	Yes	Yes	Yes	No	Yes	Yes
validatetrans	Yes	Yes	No	No	No	No

policycap

Permet d'activer/désactiver de nouvelles capacibility dans le kernel via la stratégie. La déclaration de la définition est :

```
policycap <capability>;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

default_user

Sélectionne l'utilisateur par défaut depuis le contexte source ou cible en calculant un nouveau contexte pour un objet de classe définie. La

déclaration est :
```
default_user <class> <source|target>;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

default_role

Sélectionne le rôle par défaut depuis le contexte source ou cible en calculant un nouveau contexte pour un objet de classe définie. La déclaration est :
```
default_role <class> <source|target>;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

default_type

Sélectionne le type par défaut depuis le contexte source ou cible en calculant un nouveau contexte pour un objet de classe définie. La déclaration est :
```
default_type <classe> <source|target>;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

default_range

Sélectionne la plage ou le niveau depuis le contexte source ou cible en calculant un nouveau contexte pour un objet de classe définie. La déclaration est :
```
default_range <class> <source|target> <low,high,low_high>;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

user

La déclaration user déclare un identifiant utilisateur SELinux dans la stratégie et l'associe à un ou plusieurs rôles. La déclaration permet également un niveau ou plage MLS pour contrôler le niveau de sécurité utilisateur. Il est également possible d'ajouter l'id de l'utilisateur SELinux en dehors de la stratégie avec 'semanage user' qui va associer l'utilisateur avec les rôles précédemment déclarés dans la stratégie. La déclaration est :
```
user <seuser_id> roles <role_id>;
ou pour une stratégie MCS/MLS:
user <seuser_id> roles <role_id> level <mls_level> range <mls_range>;

Monolithic | Base | Module: Yes|Yes|Yes
```

```
if | optional | require: No|No|No
```

role

La déclaration role déclare un identifiant de rôle ou associe un identifiant de rôle à un ou plusieurs types. Lorsqu'il y a plusieurs déclaration role déclarant le même rôle, le compilateur associe les types additionnels avec le rôle. La déclaration est :

```
role <role_id>;
role <role_id> types <type_id>;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|Yes
```

attribute_role

Déclare un identifiant d'attribut de rôle qui peut être utilisé pour référrer à un groupe de rôles. La déclaration est :

```
attribute_role attribute_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|Yes
```

roleattribute

Permet l'association de rôles précédemment déclarés et un ou plusieurs attribute_roles précédemment déclarés. La déclaration est :

```
roleattribute role_id attribute_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

allow

allow vérifie si une requête pour changer de rôle est permise, si c'est le cas, peut ensuite être suivie par un role_transition. La déclaration est :

```
allow from_role_id to_role_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

role_transition

Spécifie qu'une transition de rôle est requise. La déclaration est :

```
role_transition current_role_id type_id : class new_role_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

type

Déclare l'identifiant de type et des identifiants optionnels alias ou attribute associés. La déclaration est :

```
type type_id [alias alias_id] [, attribute_id];

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|No|Yes
```

attribute

Déclare un identifiant qui peut ensuite être utilisé pour référrer à un groupe d'identifiant de type. La déclaration est :

```
attribute attribute_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|Yes
```

typeattribute

Permet l'association des types précédemment déclarés à un ou plusieurs attributs déclarés. La déclaration est :

```
typeattribute type_id attribute_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

typealias

Permet l'association d'un type précédemment déclaré à un ou plusieurs identifiants alias (la déclaration type est une alternative). La déclaration est :

```
typealias type_id alias alias_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

permissive

Permet au domaine nommé d'être lancé en mode permissive au lieu de lancer tous les domaines SELinux en mode permissive. La déclaration est :
```
permissive type_id;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

type_transition

Spécifie le type par défaut à utiliser pour la transition de domaine ou la création d'objet. Noter qu'une règle allow doit être utilisée pour autoriser la transition. La déclaration est :
```
type_transition source_type target_type : class default_type object_name;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: Yes|Yes|No
```

type_change

Spécifie un type par défaut en relabélisant un objet existant. Noter qu'une règle allow doit être utilisée pour autoriser l'accès. La déclaration est :
```
type_change source_type target_type : class change_type;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: Yes|Yes|No
```

type_member

Spécifie un type par défaut en créant un objet polyinstancié. Noter qu'une règle allow doit être utilisée pour autoriser l'accès. La déclaration est :
```
member_type source_type target_type : class member_type;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: Yes|Yes|No
```

typebounds

Définis une relation hiérarchique entrée les domaines où le domaine lié ne peut pas avoir plus de permission que son domaine parent. La déclaration est :
```
typebounds bounding_domain bounded_domain;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

allow

Vérifie si l'opération entre le source_type et target_type sont permis pour la classe et les permissions définies. La déclaration est :

```
allow source_type target_type : class perm_set;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: Yes|Yes|No
```

dontaudit

Stoppe l'audit des messages de refus. Celà aide à gérer les audits en excluant des évènements connus. La déclaration est :

```
dontaudit source_type target_type : class perm_set;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: Yes|Yes|No
```

auditallow

Audit l'évènement comme un enregistrement utile pour l'audit. Noter que cette règle ne fait qu'auditer l'évènement. Une règle allow est requise. La déclaration est :

```
auditallow source_type target_type : class perm_set;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: Yes|Yes|No
```

neverallow

Spécifie qu'une règle allow ne doit pas être générée pour l'opération, même si elle a été autorisée précédemment. La déclaration est :

```
neverallow source_type target_type : class perm_set;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

class

Hérite et/ou associe les permisdsion à une classe précédemment déclarées. Les classe sont déclarée comme suit :

```
class class_id
puis
class class_id [ inherits common_set ] [ { perm_set } ]

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|Yes
```

common

Déclare un identifiant commun et associe une ou plusieurs permissions communes. La déclaration est :

```
common common_id { perm_set }

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

bool

Définis un identifiant booléen et son état initial utilisable dans une déclaration if. La déclaration est :

```
bool bool_id default_value;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|Yes
```

if

Utilisé pour former un block conditionnel de déclaration et règles qui sont forcés en fonction d'un ou plusieurs identifiants booléen. La déclaration est :

```
if (conditional_expression) { true_list } [ else { false_list } ]

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

constrain

permet de restreindre les permissions pour les classes objet spécifiés en utilisant des expressions booléenness couvrant les types, role et users source et cible. La déclaration est :

```
constrain class perm_set expression;

expression est définis comme suit:
( expression : expression )
| not expression
| expression and expression
| expression or expression
| u1 op u2
| r1 role_op r2
| t1 op t2
| u1 op names
| u2 op names
| r1 op names
| r2 op names
| t1 op names
| t2 op names

u1, r1, t1 = user, role, type source
u2, r2, t2 = user, role, type cible
```

```
op : == | !=
role_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

validatetrans

Seul les classes objets lié au fichier sont supportés par cette déclaration et est utilisée pour contrôler la capacité de changer le contexte de sécurité des objets. La déclaration est :

```
validatetrans class expression;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

mlsconstrain

Permet de restreindre les permissions pour les classes objets spécifiés en utilisant des expressions booléennes couvrant les types, rôle, utilisateurs source et cible et les niveaux de sécurité. La déclaration est :

```
mlsconstrain cluss perm_set expression;

expression est définis comme suit:
( expression : expression )
| not expression
| expression and expression
| expression or expression
| u1 op u2
| r1 role_mls_op r2
| t1 op t2
| l1 role_mls_op l2
| l1 role_mls_op h2
| h1 role_mls_op l2
| h1 role_mls_op h2
| l1 role_mls_op h1
| l2 role_mls_op h2
| u1 op names
| u2 op names
| r1 op names
| r2 op names
| t1 op names
| t2 op names

u1, r1, t1, l1, h1 = user, role, type, low level, high level source
u2, r2, t2, l2, h2 = user, role, type, low level, high level cible

op : == | !=
role_mls_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
```

```
name_list : name | name_list name

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

mlsvalidatetrans

Équivalent MLS de validatetrans et est seulement utilisé pour les classes objets liées aux fichiers, et permet de contrôler la capacité de changer le contexte de sécurité des objets. La déclaration est :

```
mlsvalidatetrans class expression;

expression est définis comme suit:
( expression : expression )
| not expression
| expression and expression
| expression or expression
| u1 op u2
| r1 role_mls_op r2
| t1 op t2
| l1 role_mls_op l2
| l1 role_mls_op h2
| h1 role_mls_op l2
| h1 role_mls_op h2
| l1 role_mls_op h1
| l2 role_mls_op h2
| u1 op names
| u2 op names
| r1 op names
| r2 op names
| t1 op names
| t2 op names
| u3 op names
| r3 op names
| t3 op names

u1, r1, t1, l1, h1 = ancien user, role, type, low level, high level
u2, r2, t2, l2, h2 = nouveau user, role, type, low level, high level
u3, r3, t3, l3, h3 = Processus user, role, type, low level, high level

op : == | !=
role_mls_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

Déclarations MLS

L'extension de stratégie MLS ajoute un composant de contexte de sécurité additionnel qui consiste des entrées suivantes :

```
user:role:type:sensitivity[:category,...]= sensitivity [:category,...]
```

Ils consistent d'une sensibilité hiérarchique obligatoire et de catégories non-hiérarchiques. La combinaison des 2 comprennent un niveau ou un niveau de sécurité. En fonction des circonstances, cela peut être un niveau définis ou une plage.

Pour que les niveaux de sécurité soient plus significatifs, il est possible d'utiliser le service setransd pour les traduire au format humain.

sensitivity

Définis la sensibilité de stratégie MLS et les alias optionnels. La déclaration est :

```
sensitivity sens_id [alias sensitivityalias_id ...];

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|Yes
```

dominance

Quand plus d'une déclaration sensitivity sont définies dans une stratégie, une déclaration dominance est requise pour définir la hiérarchie actuelle entre toutes les sensibilités. La déclaration est :

```
dominance { sensitivity_id ... }

La liste est dans l'ordre du plus faible au plus haut.

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

category

La déclaration category définis les identifiants de catégorie de stratégie MLS et les alias optionnels. La déclaration est :

```
category category_id [alias categoryalias_id ... ];

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|Yes
```

level

Permet de combiner des sensibilité et les catégories précédemment déclarées en un niveau de sécurité. Noter qu'il doit y avoir seulement une déclaration level pour chaque déclaration sensitivity. La déclaration est :

```
level sensetivity_id [ :category_id ];

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

range_transition

Principalement utilisée par le processus init ou les commandes d'administration pour s'assurer que les processus fonctionnent avec leur plage MLS correct. La déclaration est :

```
range_transition source_type target_type : class new_range;

Monolithic | Base | Module: Yes|Yes|Yes
if | optional | require: No|Yes|No
```

sid

Déclare l'identifiant SID actuel et est définis au démarrage d'un fichier source. la délaration sert également à initialiser un contexte de sécurité au SID. La déclaration est :

```
sid sid_id
sid sid_id context

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

fs_use_xattr

Utilisé pour allouer un contexte de sécurité aux systèmes de fichiers qui supportent l'attribut étendu security.selinux. Le labéling est persistant pour les systèmes de fichiers qui supportent ces attributs étendus, et le contexte de sécurité est ajouté à ces fichiers et répertoires par les commandes SELinux. La déclaration est :

```
fs_use_xattr fs_name fs_context;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

fs_use_task

Utilisé pour allouer un contexte de sécurité à un pseudo système de fichier qui supporte les services liés aux tâches tels que les pipes et les sockets. La déclaration est :

```
fs_use_task fs_name fs_context;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

fs_use_trans

Utilisé pour allouer un contexte de sécurité à des pseudo systèmes de fichier tels que les pseudo terminaux et les objets temporaires. Le contexte assigné est dérivé du processus créateur et du type de systèmed de fichier basé sur les règles de transition. La déclaration est :

```
fs_use_trans fs_name fs_context;
```

```
Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

genfscon

Utilisé pour allouer un contexte de sécurité aux systèmes de fichiers qui ne supportent pas d'autres déclaration de labéling. Généralement un système de fichier a un seul contexte de sécurité par défaut assigné par genfscon depuis '/' qui est ensuit hérité par tous les fichiers et répertoires dans ce système de fichier. /proc est l'exception où les répertoires peuvent être labélisés avec un contexte spécifique. La déclaration est :

```
genfscon fs_name partial_path fs_context

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

netifcon

Utilisé pour labéliser les objets d'interface réseaux (ex : eth0). La déclaration est :

```
netifcon netif_id netif_context packet_context

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

nodecon

Utilisé pour labéliser les objets d'adresse réseaux qui représentent des adresse IPv4 et IPv6 et des masques réseaux. La déclaration est :

```
nodecon subnet netmask node_context

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

portcon

Utilisé pour labéliser les ports udp ou tcp. La déclaration est :

```
portcon protocol port_number port_context

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

module

Cette déclaration est obligatoire pour les modules chargeables et doit être la première ligne d'un fichier source de stratégie. L'identifiant ne doit pas être en conflit avec d'autres noms de module dans la stratégie. La déclaration est :

```
module module_name version_number;
```

```
Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

require

La déclaration require est utilisée pour 2 raisons. La première est dans les fichiers sources de module pour indiquer quels composants de stratégie sont requis depuis un fichier de sources externe. Ensuite dans un fichier source de stratégie de base, si précédé par "optional", indique quels composants de stratégie sont requis depuis un fichier source externe. La déclaration est :

```
require { rule_list }
```

```
Monolithic | Base | Module: No|Yes|Yes
if | optional | require: Yes|Yes|No
```

optional

UTilisé pour indiquer quelles déclarations de startégie peuvent ou non être présent dans la stratégie compilée finale. Les déclarations seront inclus dans la stratégie seulement si toutes les déclarations optional peuvent être étendues, ce qui est généralement accomplis en utilisant une déclaration require. La déclaration est :

```
optional { rule_list } [ else { rule_list } ]
```

```
Monolithic | Base | Module: No|Yes|Yes
if | optional | require: Yes|Yes|Yes
```

iomemcon

La déclaration sid déclare l'identifiant SID actuel et est définis au début d'un fichier source de stratégie. La déclaration est :

```
iomemcon addr context;
```

```
Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

ioportcon

La déclaration est :

```
ioportcon port context;
```

```
Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

pcidevicecon

La déclaration est :

```
pcidevicecon pci_id context;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```

pirqcon

la déclaration est :

```
pirqcon irq context;

Monolithic | Base | Module: Yes|Yes|No
if | optional | require: No|No|No
```