SELinux: Gestion d'un système SELinux (avancé)

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

SELinux s'appuie sur un ensemble de règles (policy) pour autoriser ou interdire une opération. Ces règles sont assez délicates à créer, mais heureusement deux jeux de règles standards (targeted et strict) sont fournies pour éviter le plus gros du travail de configuration.

Le système de permissions de SELinux est totalement différent de ce qu'offre un système Unix traditionnel. Les droits d'un processus dépendent de son contexte de sécurité. Le contexte est défini par l'identité de celui qui a démarré le processus, le rôle et le domaine qu'il avait à ce moment. Les permissions proprement dites dépendent du domaine, mais les transitions entre les domaines sont contrôlées par les rôles. Enfin, les transitions autorisées entre rôles dépendent de l'identité.

Mise en route

Le code de SELinux est intégré dans les noyaux standards fournis par Debian et les programmes Unix de base le gèrent sans modification. Il est donc relativement simple d'activer SELinux. La commande **apt install selinux-basics selinux-policy-default** installera automatiquement les paquets nécessaires pour configurer un système SELinux.

Le paquet **selinux-policy-default** contient un ensemble de règles standards. Par défaut, l'ensemble de règles ne restreint les accès que pour certains services très exposés. Les sessions utilisateur ne sont pas restreintes et il n'y a donc que peu de risques que SELinux bloque des opérations légitimes des utilisateurs. En revanche, cela permet d'apporter un surcroît de sécurité pour les services système fonctionnant sur la machine. Pour obtenir l'équivalent des anciennes règles « strictes », il faut simplement désactiver le module unconfined (la gestion des modules est détaillée plus loin).

Une fois les règles installées, il reste à étiqueter tous les fichiers disponibles (il s'agit de leur affecter un type). C'est une opération qu'il faut déclencher manuellement avec **fixfiles relabel**.

Le système SELinux est prêt, il ne reste plus qu'à l'activer. Pour cela, il faut passer le paramètre **selinux=1 security=selinux** au noyau Linux. Le paramètre **audit=1** active les traces SELinux qui enregistrent les différentes opérations qui ont été refusées. Enfin, le paramètre **enforcing=1** permet de mettre en application l'ensemble des règles : en effet, par défaut SELinux fonctionne en mode permissive où les actions interdites sont tracées mais malgré tout autorisées. Il faut donc modifier le fichier de configuration du chargeur de démarrage GRUB pour ajouter les paramètres désirés. Le plus simple pour cela est de modifier la variable **GRUB_CMDLINE_LINUX** dans `/etc/default/grub` et d'exécuter **update-grub**. Au démarrage suivant, SELinux sera actif.

Le script **selinux-activate** automatise ces opérations et permet de forcer un étiquetage au prochain redémarrage, ce qui évite d'avoir des fichiers non étiquetés créés alors que SELinux n'était pas

encore actif et que l'étiquetage était en cours.

Gestion d'un système SELinux

L'ensemble de règles SELinux est modulaire et son installation détecte et active automatiquement tous les modules pertinents en fonction des services déjà installés. Ainsi, le système est immédiatement fonctionnel. Toutefois, lorsqu'un service est installé après les règles SELinux, il faut pouvoir activer manuellement un module de règles. C'est le rôle de la commande semodule. En outre, il faut pouvoir définir les rôles accessibles à chaque utilisateur ; pour cela c'est la commande semanage qu'il faudra utiliser. Ces deux commandes modifient donc la configuration SELinux courante qui est stockée dans /etc/selinux/default/. Contrairement à ce qui se pratique d'habitude avec les fichiers de configuration de /etc/, ces fichiers ne doivent pas être modifiés manuellement. Il faut les manipuler en utilisant les programmes prévus pour cela.

Gestion des modules SELinux

Les modules SELinux disponibles sont stockés dans le répertoire /usr/share/selinux/default/. Pour activer un de ces modules dans la configuration courante, il faut employer **semodule -i module.pp.bz2**. L'extension pp.bz2 signifie policy package que l'on pourrait traduire par « paquet de règles » (comprimé avec bzip2).

À l'inverse, la commande **semodule -r module** retire un module de la configuration courante. Enfin, la commande **semodule -l** liste les modules qui sont actuellement installés. La commande inclut également le numéro de version du module.

```
# semodule -i /usr/share/selinux/default/abrt.pp.bz2
# semodule -l
abrt    1.5.0   Disabled
accountsd       1.1.0
acct    1.6.0
[...]
# semodule -e abrt
# semodule -d accountsd
# semodule -l
abrt    1.5.0
accountsd       1.1.0   Disabled
acct    1.6.0
[...]
# semodule -r abrt
# semodule -l
accountsd       1.1.0   Disabled
acct    1.6.0
[...]
```

semodule recharge immédiatement la nouvelle configuration, sauf si l'on utilise l'option -n. Signalons également que le programme modifie par défaut la configuration courante (celle indiquée par la variable SELINUXTYPE dans `/etc/selinux/config`) mais qu'on peut en modifier une autre grâce à l'option -s.

Gestion des identités

Chaque fois qu'un utilisateur se connecte, il se voit attribuer une identité SELinux, qui va définir les rôles qu'il va pouvoir assumer. Ces deux correspondances (de l'utilisateur vers l'identité SELinux et de cette identité vers les rôles) se configurent grâce à la commande semanage. On retrouvera des options communes aux différentes sous-commandes :

- -a pour ajouter: **semanage login -a -s user_u utilisateur** va associer l'identité user_u à l'utilisateur
- -d pour supprimer : **semanage login -d utilisateur** va retirer la correspondance affectée à l'utilisateur
- -m pour modifier
- -l pour lister : **semanage login -l** liste les correspondances existantes entre identifiants d'utilisateurs et identités SELinux. Si un utilisateur n'a pas de correspondance explicite, il aura l'identité indiquée en face de __default__. **semanage user -l** liste les correspondances entre identité SELinux et rôles possibles.
- -t pour indiquer un type (ou domaine).

Gestion des contextes de fichiers, des ports et des booléens

Chaque module SELinux fournit un ensemble de règles d'étiquetage des fichiers, mais il est également possible de rajouter des règles d'étiquetage spécifiques afin de les adapter à un cas particulier. Ainsi, pour rendre toute l'arborescence /srv/www/ accessible au serveur web, on pourrait exécuter **semanage fcontext -a -t httpd_sys_content_t "/srv/www(/.*)\?"** , puis **restorecon -R /srv/www/**.

- La première commande enregistre la nouvelle règle d'étiquetage
- la seconde restaure les bonnes étiquettes en fonction des règles enregistrées.

D'une manière similaire, les ports TCP/UDP sont étiquetés afin que seuls les démons correspondants puissent y écouter. Ainsi, si l'on veut que le serveur web puisse également écouter sur le port 8080, il faut exécuter la commande **semanage port -m -t http_port_t -p tcp 8080**.

Les modules SELinux exportent parfois des options booléennes qui influencent le comportement des règles. L'utilitaire **getsebool** permet de consulter l'état de ces options (**getsebool booléen** affiche une option et **getsebool -a** les affiche toutes).

La commande **setsebool booléen valeur** change la valeur courante d'une option. L'option -P rend le

changement permanent, autrement dit la nouvelle valeur sera celle par défaut et sera conservée au prochain redémarrage.

L'exemple ci-dessous permet au serveur web d'accéder aux répertoires personnels des utilisateurs (utile dans le cas où ils ont des sites web personnels dans ~/public_html/ par exemple).

```
# getsebool httpd_enable_homedirs
httpd_enable_homedirs --> off
# setsebool -P httpd_enable_homedirs on
# getsebool httpd_enable_homedirs
httpd_enable_homedirs --> on
```

Adaptation des règles

Comme l'ensemble des règles (que l'on nomme policy) est modulaire, on peut développer de nouveaux modules pour les applications (éventuellement spécifiques) qui n'en disposent pas encore, ces nouveaux modules venant alors compléter la reference policy.

Trois fichiers permettent de définir un nouveau module (Le paquet selinux-policy-dev sera également nécessaire) :

- **Le fichier .fc** définit les « contextes des fichiers », autrement dit les types affectés aux fichiers relatifs à ce module. Les informations du .fc sont utilisées lors de l'étiquetage des fichiers sur le disque.
- **Le fichier .if** définit l'interface du module ; il s'agit d'un ensemble de « fonctions publiques » qui permettent à d'autres modules de s'interfacer proprement avec celui en cours de création.
- **Le fichier .te** est le plus important : il définit les règles à proprement parler.

Rédiger un fichier .fc

La lecture de l'exemple qui suit suffit à comprendre la structure d'un tel fichier. Il est possible d'employer une expression rationnelle pour affecter le même contexte à plusieurs fichiers, voire à toute une arborescence.

```
# myapp executable will have:
# label: system_u:object_r:myapp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/usr/sbin/myapp         --
gen_context(system_u:object_r:myapp_exec_t,s0)
```

Rédiger un fichier .if

Dans l'exemple suivant, la première interface (myapp_domtrans) sert à contrôler qui a le droit

d'exécuter l'application et la seconde (myapp_read_log) fournit un droit de lecture sur les fichiers de logs de l'application.

Chaque interface doit générer un ensemble correct de règles comme s'il était directement placé dans un fichier .te. Il faut donc déclarer tous les types employés (avec la macro gen_require) et employer les directives standards pour attribuer des droits. Notons toutefois qu'il est possible d'employer des interfaces fournies par d'autres modules.

```
## <summary>Myapp example policy</summary>
## <desc>
##      <p>
##              More descriptive text about myapp.  The <desc>
##              tag can also use <p>, <ul>, and <ol>
##              html tags for formatting.
##      </p>
##      <p>
##              This policy supports the following myapp features:
##              <ul>
##              <li>Feature A</li>
##              <li>Feature B</li>
##              <li>Feature C</li>
##              </ul>
##      </p>
## </desc>
#

########################################
## <summary>
##      Execute a domain transition to run myapp.
## </summary>
## <param name="domain">
##      Domain allowed to transition.
## </param>
#
interface(`myapp_domtrans',`
        gen_require(`
                type myapp_t, myapp_exec_t;
        ')

        domtrans_pattern($1,myapp_exec_t,myapp_t)
')

########################################
## <summary>
##      Read myapp log files.
## </summary>
## <param name="domain">
##      Domain allowed to read the log files.
## </param>
#
interface(`myapp_read_log',`
```

```
        gen_require(`
                type myapp_log_t;
        ')

        logging_search_logs($1)
        allow $1 myapp_log_t:file r_file_perms;
')
```

Rédiger un fichier .te

Le Langage de macro m4: Pour structurer proprement l'ensemble des règles, les développeurs de SELinux se sont appuyés sur un langage de création de macro-commandes. Au lieu de répéter à l'infini des directives allow très similaires, la création de fonctions « macro » permet d'utiliser une logique de plus haut niveau et donc de rendre l'ensemble de règles plus lisible.

Dans la pratique, la compilation des règles va faire appel à l'outil m4 pour effectuer l'opération inverse : à partir des directives de haut niveau, il va reconstituer une grande base de données de directives allow. Ainsi, les « interfaces » ne sont rien que des fonctions macro qui vont être remplacées par un ensemble de règles au moment de la compilation. De même, certaines permissions sont en réalité des ensembles de permissions qui sont remplacées par leur valeur au moment de la compilation.

```
policy_module(myapp,1.0.0) 1

########################################
#
# Declarations
#

type myapp_t; 2
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t) 3

type myapp_log_t;
logging_log_file(myapp_log_t) 4

type myapp_tmp_t;
files_tmp_file(myapp_tmp_t)

########################################
#
# Myapp local policy
#

allow myapp_t myapp_log_t:file { read_file_perms append_file_perms }; 5
```

```
allow myapp_t myapp_tmp_t:file manage_file_perms;
files_tmp_filetrans(myapp_t,myapp_tmp_t,file)
```

1. Le module doit être identifié par son nom et par son numéro de version. Cette directive est requise.
2. Si le module introduit de nouveaux types, il doit les déclarer avec des directives comme celle-ci. Il ne faut pas hésiter à créer autant de types que nécessaires, plutôt que distribuer trop de droits inutiles.
3. Ces interfaces précisent que le type myapp_t est prévu pour être un domaine de processus et qu'il doit être employé pour tout exécutable étiqueté par myapp_exec_t. Implicitement, cela ajoute un attribut exec_type sur ces objets. Sa présence permet à d'autres modules de donner le droit d'exécuter ces programmes : ainsi, le module userdomain va permettre aux processus de domaine user_t, staff_t et sysadm_t de les exécuter. Les domaines d'autres applications confinées n'auront pas le droit de l'exécuter, sauf si les règles prévoient des droits similaires (c'est le cas par exemple pour dpkg avec le domaine dpkg_t).
4. Logging_log_file est une interface fournie par la reference policy qui sert à indiquer que les fichiers étiquetés avec le type précisé en paramètre sont des fichiers de logs et doivent bénéficier des droits associés (par exemple ceux permettant à logrotate de les manipuler).
5. La directive allow est la directive de base qui permet d'autoriser une opération. Le premier paramètre est le domaine du processus qui sera autorisé à effectuer l'opération. Le second décrit l'objet qu'un processus du domaine aura le droit de manipuler. Ce paramètre prend la forme « type:genre » où type est son type SELinux et où genre décrit la nature de l'objet (fichier, répertoire, socket, fifo, etc.). Enfin, le dernier paramètre décrit les permissions (les opérations qui sont autorisées).

Les permissions se définissent comme des ensembles d'opérations autorisées et prennent la forme { operation1 operation2 }. Il est également possible d'employer des macros qui correspondent aux ensembles de permissions les plus utiles. Le fichier /usr/share/selinux/devel/include/support/obj_perm_sets.spt permet de les découvrir.

Une liste relativement exhaustive des genres d'objet (object classes) et des permissions que l'on peut accorder est consultable ici

Il ne reste plus qu'à trouver l'ensemble minimal des règles nécessaires au bon fonctionnement du service ou de l'application ciblé(e) par le module. Pour cela, il est préférable de bien connaître le fonctionnement de l'application et d'avoir une idée claire des flux de données qu'elle gère et/ou génère.

Toutefois, une approche empirique est possible. Une fois les différents objets impliqués correctement étiquetés, on peut utiliser l'application en mode permissif : les opérations normalement interdites sont tracées mais réussissent tout de même. Il suffit alors d'analyser ces traces pour identifier les opérations qu'il faut autoriser. Voici à quoi peut ressembler une de ces traces :

```
avc:  denied  { read write } for  pid=1876 comm="syslogd" name="xconsole"
dev=tmpfs ino=5510  scontext=system_u:system_r:syslogd_t:s0
tcontext=system_u:object_r:device_t:s0 tclass=fifo_file permissive=1
```