Introduction à SELinux

Les principes

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, du *rôle* et du *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.

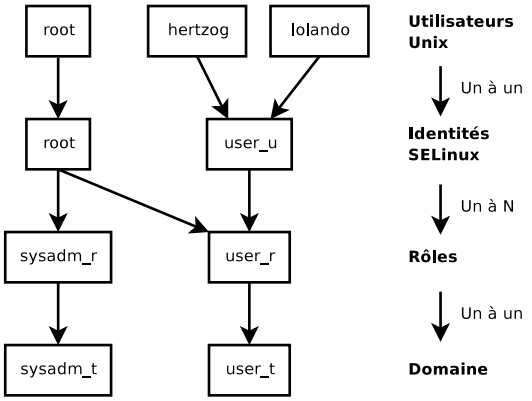

Figure 14–3
Contextes de sécurité et utilisateurs Unix

En pratique, au moment de la connexion, l'utilisateur se voit attribuer un contexte de sécurité par défaut (en fonction des rôles qu'il a le droit d'assumer). Cela fixe le domaine dans lequel il évolue. S'il veut changer de rôle et de domaine associé, il doit employer la commande **`newrole -r role_r -t domaine_t`** (il n'y a généralement qu'un seul domaine possible pour un rôle donné et le paramètre `-t` est donc souvent inutile). Cette commande demande à l'utilisateur son mot de passe afin de l'authentifier. Cette caractéristique empêche tout programme de pouvoir changer

de rôle de manière automatique. De tels changements ne peuvent avoir lieu que s'ils sont prévus dans l'ensemble de règles.

Bien entendu les droits ne s'appliquent pas universellement à tous les *objets* (fichiers, répertoires, sockets, périphériques, etc.), ils peuvent varier d'un objet à l'autre. Pour cela, chaque objet est associé à un *type* (on parle d'étiquetage). Les droits des domaines s'expriment donc en terme d'opérations autorisées (ou non) sur ces types (donc implicitement sur tous les objets qui sont marqués avec le type correspondant).

Par défaut, un programme exécuté hérite du domaine de l'utilisateur qui l'a démarré. Mais pour la plupart des programmes importants, les règles SELinux standard prévoient de les faire fonctionner dans un domaine dédié. Pour cela, ces exécutables sont étiquetés avec un type dédié (par exemple **ssh** est étiqueté avec `ssh_exec_t`, et lorsque le programme est démarré il bascule automatiquement dans le domaine `ssh_t`). Ce mécanisme de changement automatique de domaine permet de ne donner que les droits nécessaires au bon fonctionnement de chaque programme et est à la base de SELinux.

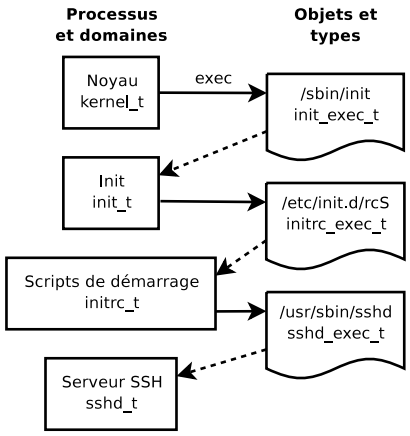

Figure 14–4
Transitions automatiques entre domaines

La mise en route

Le support de SELinux est intégré dans les noyaux standards fournis par Debian et les programmes Unix de base ont le support SELinux activé. Il est donc relativement simple d'activer SELinux.

La commande **`aptitude install selinux-basics selinux-policy-refpolicy-targeted`** installera automatiquement les paquets nécessaires pour configurer un système SELinux. L'ensemble de règles *targeted* ne restreint les accès que pour certains services très exposés. Les sessions utilisateurs ne sont pas restreintes et il n'y a donc que peu de risques que SELinux bloque des opérations légitimes des utilisateurs. En revanche,

COMPLÉMENTS

Domaine et type sont équivalents

En interne, un domaine n'est qu'un type mais un type qui ne s'applique qu'aux processus. C'est pour cela que les domaines sont suffixés par `_t` tout comme le sont les types affectés aux objets.

EN PRATIQUE

Connaître le contexte de sécurité

Pour connaître le contexte de sécurité appliqué à un processus, il faut employer l'option Z de **ps**.

```
$ ps axZ | grep vstfpd
system_u:system_r:ftpd_t:s0
 ➥ 2094  ?Ss  0:00 /usr/sbin/vsftd
```

Le premier champ contient l'identité, le rôle, le domaine et le niveau MCS, séparés par des double points. Le niveau MCS (*Multi-Category Security*) est un paramètre intervenant dans la mise en place d'une politique de protection de la confidentialité, laquelle restreint l'accès aux fichiers selon leur degré de confidentialité. Cette fonctionnalité ne sera pas abordée dans ce livre.

Pour connaître le contexte de sécurité actuellement actif dans un terminal de commande, il faut invoquer **`id -Z`**.

```
$ id -Z
user_u:system_r:unconfined_t:s0
```

Enfin, pour connaître le type affecté à un fichier, on peut employer **`ls -Z`**.

```
$ ls -Z test /usr/bin/ssh
-rw-r--r--  rhertzog rhertzog
 ➥ user_u:object_r:user_home_t:s0
 ➥ test
-rwxr-xr-x  root     root
 ➥ system_u:object_r:ssh_exec_t:s0
 ➥ /usr/bin/ssh
```

Signalons que l'identité et le rôle associé à un fichier n'ont pas d'importance particulière, ils n'interviennent jamais. Mais par souci d'uniformisation, tous les objets se voient attribuer un contexte de sécurité complet.

cela permet d'apporter un surcroît de sécurité pour les services systèmes fonctionnant sur la machine.

Le paquet **`selinux-policy-refpolicy-strict`** contient un ensemble de règles plus strictes et va confiner les opérations permises aux utilisateurs.

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` 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 (`/boot/grub/menu.lst`) pour rajouter les paramètres désirés. Au prochain démarrage, SELinux sera actif.

La gestion d'un système SELinux

Chaque ensemble de règles SELinux est modulaire, et leur 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, et pour cela c'est la commande **`semanage`** qu'il faudra utiliser.

Ces deux commandes permettent donc de modifier la configuration SELinux courante qui est stockée dans `/etc/selinux/refpolicy-targeted/` ou `/etc/selinux/refpolicy-strict/`. Contrairement à ce qui se pratique d'habitude avec les fichiers de configuration de `/etc/`, tous 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 les répertoires `/usr/share/selinux/refpolicy-targeted/` ou `/usr/share/selinux/refpolicy-strict/`. Pour activer un de ces modules dans la configuration courante, il faut employer **`semodule -i`** ***`module.pp`***. L'extension *pp* signifie *policy package* que l'on pourrait traduire par « paquet de règles ».

POUR ALLER PLUS LOIN

Documentation trop rare

SELinux ne dispose pas réellement de documentation de référence. Pour aller au-delà de cette introduction, il faudra consulter les documentations SELinux de différentes distributions Linux (notamment Fedora et Gentoo) ainsi que celles fournies par des contributeurs impliqués dans son développement. Russell Coker est un des développeurs Debian les plus actifs sur le sujet de SELinux et son blog contient régulièrement des articles sur le sujet.

- http://docs.fedoraproject.org/selinux-faq-fc5/
- http://www.gentoo.org/proj/en/hardened/selinux/selinux-handbook.xml
- http://www.crypt.gen.nz/selinux/faq.html
- http://wiki.debian.org/SELinux
- http://etbe.coker.com.au/tag/selinux/

En se référant à toutes ces documentations, il faut faire attention au fait que les informations contenues peuvent être dépassées et ne pas s'appliquer telles quelles au cas de Debian.

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

```
# semodule -i /usr/share/selinux/refpolicy-targeted/amavis.pp
# semodule -l
amavis  1.1.0
apache  1.4.0
apm     1.3.0
[...]
# semodule -r amavis
libsepol.sepol_genbools_array: boolean amavis_disable_trans
  ➥ no longer in policy
# semodule -l
apache  1.4.0
apm     1.3.0
[...]
```

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, et cette identité 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**.

La lecture de la page de manuel `semanage(8)` est indispensable même si la syntaxe de cette commande ne varie guère selon les concepts manipulés. On retrouvera des options communes aux différentes sous-commandes : `-a` pour ajouter, `-d` pour supprimer, `-m` pour modifier, `-l` pour lister et `-t` pour indiquer un type (ou domaine).

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__`. La commande **semanage login -a -s user_u *utilisateur*** va associer l'identité *user_u* à l'utilisateur. Enfin, **semanage login -d *utilisateur*** va retirer la correspondance affectée à l'utilisateur.

```
# semanage login -a -s user_u rhertzog
# semanage login -l
 Login Name                SELinux User              MLS/MCS Range
```

```
__default__              user_u                   s0
rhertzog                 user_u                   s0
root                     root                     s0-s0:c0.c1023
# semanage login -d rhertzog
```

semanage user -l liste les correspondances entre identité SELinux et rôles possibles. Ajouter une nouvelle identité nécessite de préciser d'une part les rôles correspondants et d'autre part un préfixe d'étiquetage qui définira le type affecté aux fichiers personnels (`/home/`*`utilisateur`*`/*`). Le préfixe est à choisir entre `user`, `staff` et `sysadm`. Un préfixe « `staff` » donnera des fichiers typés « `staff_home_dir_t` ». La commande créant une identité est **semanage user -a -R** ***rôles*** **-P** ***préfixe identité***. Enfin, une identité peut être supprimée avec **semanage user -d** ***identité***.

```
# semanage user -a -R 'staff_r user_r' -P staff test_u
# semanage user -l
                Labeling   MLS/       MLS/
SELinux User    Prefix     MCS Level  MCS Range                      SELinux Roles

root            user       s0         s0-s0:c0.c1023                 system_r sysadm_r user_r
system_u        user       s0         s0-s0:c0.c1023                 system_r
test_u          staff      s0         s0                             user_r staff_r
user_u          user       s0         s0-s0:c0.c1023                 system_r sysadm_r user_r
# semanage user -d test_u
```

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 et la deuxième 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 permettent d'influencer 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*** permet de changer 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
```

L'adaptation des règles

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

Le paquet `selinux-policy-refpolicy-dev` sera nécessaire, ainsi que `selinux-policy-refpolicy-doc`. Ce dernier contient la documentation des règles standard (`/usr/share/doc/selinux-policy-refpolicy-doc/html/`) et des fichiers exemples permettant de créer des nouveaux modules. Installons ces fichiers pour les étudier de plus près :

```
$ zcat /usr/share/doc/selinux-policy-refpolicy-doc/Makefile.example.gz
  ➥ >Makefile
$ zcat /usr/share/doc/selinux-policy-refpolicy-doc/example.fc.gz
  ➥ >example.fc
$ zcat /usr/share/doc/selinux-policy-refpolicy-doc/example.if.gz
  ➥ >example.if
$ cp /usr/share/doc/selinux-policy-refpolicy-doc/example.te ./
```

Le fichier `.te` est le plus important, il définit les règles à proprement parler. Le fichier `.fc` définit les « contextes des fichiers », autrement dit les types affectés aux fichiers relatifs à ce module. Les informations du fichier `.fc` sont utilisées lors de l'étiquetage des fichiers sur le disque. Enfin 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 le module en cours de création.

Rédiger un fichier .fc

La lecture de l'exemple ci-dessous 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.

EXEMPLE **Fichier example.fc**

```
# 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 ci-dessous, la première interface (« `myapp_domtrans` ») permet de contrôler qui a le droit d'exécuter l'application et la seconde (« `myapp_read_log` ») permet de fournir 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 standard pour attribuer des droits. Notons toutefois qu'il est possible d'employer des interfaces fournies par d'autres modules. La prochaine section en dévoilera plus sur la manière d'exprimer ces droits.

DOCUMENTATION

Explications sur la reference policy

La *reference policy* évolue comme un projet libre au gré des contributions. Le projet est hébergé sur le site de Tresys, une des sociétés les plus actives autour de SELinux. Leur wiki contient des explications sur la structure des règles et sur la manière d'en créer de nouvelles.

- http://oss.tresys.com/projects/refpolicy/wiki/GettingStarted

EXEMPLE **Fichier example.if**

```
## <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;
        ')

        domain_auto_trans($1,myapp_exec_t,myapp_t)

        allow $1 myapp_t:fd use;
        allow myapp_t $1:fd use;
        allow $1 myapp_t:fifo_file rw_file_perms;
        allow $1 myapp_t:process sigchld;
')

########################################
## <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;
')
```

POUR ALLER PLUS LOIN **Langage de macro m4**

Pour structurer proprement l'ensemble des règles, les développeur de SELinux se sont appuyés sur un langage permettant de créer des 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.

Rédiger un fichier .te

Analysons le contenu du fichier `example.te` :

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

❶ Le module doit être identifié par son nom et par son numéro de version. Cette directive est requise.

❷ Si le module introduit des 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.

❸ 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 rajoute 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`).

❹ `logging_log_file` est une interface fournie par la *reference policy* qui permet d'indiquer que les fichiers étiquetés avec le type indiqué en paramètre sont des fichiers de logs et doivent bénéficier des droits associés (par exemple ceux permettant à **`logrotate`** de les manipuler).

❺ 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 théoriquement la forme `{ operation1 operation2 }`. Mais heureusement, il existe des macros qui correspondent aux ensembles de permissions les plus utiles. Le fichier `/usr/share/selinux/refpolicy-targeted/include/support/obj_perm_sets.spt` permet de les découvrir.

La page web ci-contre fournit une liste relativement exhaustive des genres d'objet (*object classes*) et des permissions que l'on peut accorder.

▸ http://www.tresys.com/selinux/obj_perms_help.html

Il ne reste plus qu'à trouver l'ensemble minimal de règles nécessaires au bon fonctionnement du service ou de l'application ciblé 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
```

Pour mieux comprendre ce message, analysons le bout par bout.

Tableau 14–1 Analyse d'une trace SELinux

Message	Description
`avc:  denied`	Une opération a été refusée.
`{ read write }`	Cette opération requérait les permissions `read` et `write`.
`pid=1876`	Le processus ayant le PID 1876 a exécuté l'opération (ou essayé de l'exécuter).
`comm="syslogd"`	Le processus était une instance de la commande `syslogd`.
`name="xconsole"`	L'objet cible portait le nom de `xconsole`.
`dev=tmpfs`	Le périphérique stockant l'objet est de type `tmpfs`. Pour un disque réel, nous pourrions voir la partition contenant l'objet (exemple : « hda3 »).
`ino=5510`	L'objet est identifié par le numéro d'inode 5510.
`scontext=system_u:system_r:syslogd_t:s0`	C'est le contexte de sécurité courant du processus qui a exécuté l'opération.
`tcontext=system_u:object_r:device_t:s0`	C'est le contexte de sécurité de l'objet cible.
`tclass=fifo_file`	L'objet cible est un fichier FIFO.

COMPLÉMENTS **Pas de rôle dans les règles**

On peut s'étonner que les rôles n'interviennent à aucun moment dans la création des règles. SELinux emploie uniquement les domaines pour savoir quelles opérations sont permises. Le rôle n'intervient qu'indirectement en permettant à l'utilisateur d'accéder à un autre domaine. SELinux en tant que tel est basé sur une théorie connue sous le nom de *Type Enforcement* (Application de types) et le type (ou domaine) est le seul élément qui compte dans l'attribution des droits.

Ainsi il est possible de fabriquer une règle qui va autoriser cette opération, cela donnerait par exemple `allow syslogd_t device_t:fifo_file { read write }`. Ce processus est automatisable et c'est ce que propose la commande **`audit2allow`** du paquet `policycoreutils`. Une telle démarche ne sera utile que si les objets impliqués sont déjà correctement étiquetés selon ce qu'il est souhaitable de cloisonner. Dans tous les cas, il faudra relire attentivement les règles pour les vérifier et les valider par rapport à votre connaissance de l'application. En effet, bien souvent cette démarche donnera des permissions plus larges que nécessaires. La bonne solution est souvent de créer des nouveaux types et d'attribuer des permissions sur ces types uniquement. Il arrive également qu'un échec sur une opération ne soit pas fatal à l'application, auquel cas il peut être préférable d'ajouter une règle « `dontaudit` » qui supprime la génération de la trace malgré le refus effectif.

Compilation des fichiers

Une fois que les trois fichiers (`example.if`, `example.fc` et `example.te`) sont conformes aux règles que l'on veut créer, il suffit d'invoquer **`make`** pour générer un module dans le fichier `example.pp` (que l'on peut immédiatement charger avec **`semodule -i example.pp`**). Si plusieurs modules sont définis, **`make`** créera tous les fichiers `.pp` correspondants.

Autres considérations sur la sécurité

La sécurité n'est pas un simple problème de technique. C'est avant tout des bonnes habitudes et une bonne compréhension des risques. Cette section propose donc une revue de certains risques fréquents, ainsi qu'une série de bonnes pratiques, qui, selon le cas, amélioreront la sécurité ou réduiront l'impact d'une attaque fructueuse.

Risques inhérents aux applications web

L'universalité des applications web a entraîné leur multiplication et il est fréquent d'en avoir plusieurs en service : un *webmail*, un wiki, un groupware, des forums, une galerie de photos, un blog, etc. Un grand nombre de ces applications s'appuient sur les technologies LAMP (*Linux Apache Mysql PHP*). Malheureusement un grand nombre ont aussi été écrites sans faire trop attention aux problèmes de sécurité. Trop souvent les données externes sont utilisées sans vérifications préalables et il est possible de subvertir un appel à une commande pour qu'il en résulte une autre, simplement en fournissant une valeur inattendue. Avec le temps, les problèmes les plus évidents ont été corrigés, mais de nouvelles failles de sécurité sont régulièrement découvertes.

Il est donc indispensable de mettre à jour ses applications web régulièrement pour ne pas rester vulnérable au premier pirate (amateur ou pas) qui cherchera à exploiter cette faille connue. Selon le cas, le risque varie : cela va de la destruction des données, à l'exécution de commandes arbitraires en passant par le vandalisme du site web.

Savoir à quoi s'attendre

Ainsi donc la vulnérabilité d'une application web est un point de départ fréquent pour un acte de piraterie. Voyons quelles peuvent en être les conséquences.

VOCABULAIRE Injection SQL

Lorsqu'un programme exécutant des requêtes SQL y insère des paramètres d'une manière non sécurisée, il peut être victime d'injections SQL. Il s'agit de modifier le paramètre d'une telle manière à ce que le programme exécute en réalité une version altérée de la requête SQL, soit pour endommager les données soit pour récupérer des données auxquelles l'utilisateur ne devait pas avoir accès.

▸ http://fr.wikipedia.org/wiki/Injection_SQL

VOCABULAIRE Déni de service

Une attaque en déni de service consiste à rendre inopérant une machine ou un de ses services. Une telle attaque est parfois « distribuée », il s'agit alors de surcharger la machine avec un grand nombre de requêtes en provenance de nombreuses sources, afin que le serveur ne puisse plus répondre aux requêtes légitimes. En anglais, on parle de *(distributed) denial of service* (abrégé en DoS ou DDoS).