SELinux

Timothée Ravier
siosm@floss.social - tim.siosm.fr/cours - github.com/travier

5e année cycle ingénieur, filière STI
Option Sécurité des Systèmes Embarqués et du Cloud (2SEC)

2024 - 2025

Concepts fondateurs

La triade DIC (CIA triad)

- Les fondamentaux :
  - Confidentialité : qui a accès à une information ?
  - Intégrité : qui peut écrire, modifier ou créer de l'information ?
  - Disponibilité : est-ce qu'une ressource est disponible ?
- Les extensions :
  - Authenticité : comment établir une identité ?
  - Auditabilité : comment tracer les opérations effectuées ?
  - Non répudiation : est-il possible de nier avoir fait une action ?

Le principe du moindre privilège

> « Tous les programmes et utilisateurs d'un système doivent fonctionner en utilisant le minimum de privilèges nécessaires pour mener à bien leurs fonctions. »
>
> --- Jerome Saltzer, *Communications of the ACM*

Pour mettre en place un tel modèle d'opération, il faut :

- séparer les privilèges
- minimiser les privilèges
- confiner ou cloisonner les programmes privilégiés

Modèles de contrôle d'accès

DAC : Contrôle d'Accès Discrétionnaire

- Le DAC est le modèle utilisé par défaut sous Linux
- Chaque utilisateur a le contrôle des fichiers et programmes lui appartenant :
  - création / modification / suppression
- Les programmes lancés disposent des même droits que l'utilisateur
- Les utilisateurs peuvent donner des droits sur leur ressources à leur *discrétion* :
  - chmod / chown
- Le niveau de sécurité dépend de la bonne séparation entre utilisateurs et du niveau de sécurité de chaque application

Les limites du DAC

- Il est possible d'utiliser les *capabilities* et différents utilisateurs pour séparer les privilèges accordés aux applications
- Mais cette séparation n'est pas toujours assez fine : de nombreuses *capabilities* sont équivalente à `root` (cf. section sur les *capabilities*)
- Il n'est pas possible de séparer des programmes s'exécutant sous un même identifiant UNIX

> Si programme s'exécutant sous l'identité `root` est compromis, l'attaquant obtient les droits `root` et l'ensemble du système est compromis

Contrôle d'accès obligatoire ou Mandatory Access Control (MAC)

- Les droits d'accès aux objets du système ne sont plus à la discrétion des utilisateurs
- L'administrateur du système définit une politique de sécurité qui décrit quel ressource est accessible à quel utilisateur

Contrôle d'accès obligatoire sous Linux

- Intervient après le contrôle d'accès discrétionnaire standard
- Implémenté sous forme de *Linux Security Module (LSM)*

What roles do DAC (file permissions), ACL and MAC (SELinux) play in Linux file security?

Linux Security Modules

- Majeurs :
  - SELinux (utilisé par défaut sur Fedora, RHEL/CentOS, Android)
  - AppArmor (utilisé par défaut sur Ubuntu, Debian)
  - SMACK (utilisé par défaut sur Tizen)
  - TOMOYO
  - grsecurity RBAC
- Mineurs :
  - LoadPin
  - Yama
  - SafeSetID
  - Landlock

SELinux

SELinux ?

- Security-Enhanced Linux
- Implémentation d'un contrôle d'accès obligatoire
- Développé initialement par la NSA
- Premier MAC intégré dans le noyau Linux (2.6+)
- Implémentation du contrôle d'accès dans le noyau
- Outils en espace utilisateur pour manipuler la politique

Concepts

- Basé sur l'architecture *Flask* :
  - Moniteur de référence dans le noyau
  - Stocke la politique de contrôle d'accès
  - Prends les décision de contrôle d'accès au niveau des appels systèmes (contrôle à *grain fin*)

Politique de sécurité

- Description **exhaustive** de l'ensemble des opérations autorisées sur un système
- Pour cela, on distingue :
  - Les sujets : entités actives du système (processus)
  - Les objets : entités passives du système (fichier, socket, etc.)
- La politique décrit quelles actions les sujets peuvent faire sur les objets ou sujets du système

Exemple règles de politique SElinux

`toto.te` :

```
policy_module(toto, 0.0.1)

require {
  type httpd_t;
  type httpd_sys_script_t;
  type user_tmp_t;
}

type toto_data_t;
files_type(toto_data_t);

allow httpd_t toto_data_t:file read_file_perms;
allow httpd_t toto_data_t:dir search_dir_perms;

allow httpd_sys_script_t user_tmp_t:file read_file_perms;
```

Politique

- Nombreuses déclarations ou définitions : identités, rôles, types
- Nombreuses règles : d'autorisation d'accès, de transitions de types
- Définition des contraintes (vérification à la compilation)
- Utilisation de macros (fonctions) pour factoriser les cas courants

Politique

- Initialement écrit dans le *Policy Language*
- Migration progressive vers le *SELinux Common Intermediate Language (CIL)*
- Même concepts, syntax différente

Where does CIL play in the SELinux system?

Type Enforcement (TE)

- SELinux associe à chaque ressource du système un contexte de sécurité (*Security Context* ou *SC*)
- Lorsqu'ils sont associés à des objets du système on les appelle aussi *label*
- Les décisions de contrôle d'accès prises par le noyau prennent en compte ces contextes de sécurité
- Dans le cas de SELinux, ce contrôle est appelé *Type Enforcement*

Contexte de sécurité

- Ensemble d'attributs sous forme de texte :
  - `system_u:system_r:httpd_t` : contexte du processus apache
  - `user_u:object_r:user_home_t` : contexte de fichier d'un répertoire utilisateur

Contexte de sécurité (processus)

- Attribué à la création
- Ne peut évoluer que lors d'un appel à `execve`
- `id -Z`
- `ps -Z`

Contextes de sécurité (fichiers)

- Appelés *labels*
- stocké dans les attributs étendus du fichier (*xattr*)
- peut être modifié à volonté
- `ls -Z`
- `mkdir -Z`
- `cp -Z`
- `mv -Z`
- `find -context`

Exemple de définition de labels SELinux

`toto.fc` :

```
/var/www/html/toto         -d  gen_context(system_u:object_r:toto_data_t,s0)
/var/www/html/toto(/.*)?   --  gen_context(system_u:object_r:toto_data_t,s0)
```

Contexte de sécurité

- Trois attributs principaux : `user:role:type`
  - User : identité SELinux (!= DAC) propriétaire du contexte
    - `unconfined_u`, `system_u`, `user_u`, `staff_u`, `sysadm_u`
  - Role : modèle du Role Based Access Control (RBAC)
    - Rôle courant de l'identité pour un sujet
    - `unconfined_r`, `system_r`, `user_r` `staff_r`, `sysadm_r`
    - Role unique `object_r` pour tous les objets
  - Type : modèle du Type Enforcement (TE)
    - Domaine dans lequel s'exécute un sujet : `unconfined_t`, `httpd_t`
    - Type associé à un objet : `user_home_t`, `passwd_file_t`

Type

- Un type est un identifiant
- Pas de signification intrinsèque : il est contraint aux règles de la politique
- Regroupe les sujets et objets ayant les même autorisations
- On parle de :
  - type pour un objet
  - domaine pour un sujet
- Exemples : `unconfined_t`, `system_t`, `user_t`

Convention de nommage

- `applicationd_t` : domaine associé à un démon
- `application_t` : domaine associé à un processus classique
- `application_exec_t` : fichier binaire de l'application
- `application_conf_t` : fichier de configuration
- `application_log_t` : fichier de log
- `application_file_t` : fichiers divers
- `application_lib_t` : bibliothèque
- `application_tmp_t` : fichier temporaire
- `applicationd_unit_file_t` : fichier d'unit systemd

Un objet/processus ne possède qu'un seul type mais un type peut être associé à plusieurs objets

Types

Chaque type doit être déclaré avant utilisation :

```
# Sujet
type init_t;
type sshd_t;

# Objet
type sshd_exec_t;
type admin_paswd_exec_t;
```

Interaction

Chaque règle SELinux autorisant une interaction est constituée :

- d'un contexte de sécurité source (processus)
- qui effectue une opération :
  - class : `file`, `dir`, `process`, `socket` (type de l'opération)
  - permission : `read`, `write`, `unkink`, `accept` (action)
- sur un contexte de sécurité destination (fichier, socket, processus, etc.)

SELinux Object Classes and Permissions Reference

Opération

- Opération = droit d'un contexte sujet sur un autre contexte
- Permission autorisée sur une classe d'objet
- Exemple :
  - `file { open read execute }`
  - `socket  { bind listen }`

Règle de contrôle d'accès

```
allow <type_source> <type_cible>:<class> { <permissions> };
```

```
# Les sujets avec le type sysadm_t peuvent exécuter
# les fichiers avec le type ssh_exec_t
allow sysadm_t ssh_exec_t:file { getattr read execute };

# Les sujets avec le type httpd_t peuvent lire
# les fichiers avec le type http_sys_content_t
allow httpd_t http_sys_content_t:file { getattr read };
```

Type transitions pour les processus

- Par défaut, le contexte de sécurité d'un nouveau processus est celui de son parent
- Possibilité de changer de contexte avec une *transition* lors d'un appel à `execve`
- Permet de confiner chaque service dans son domaine respectif

Transition de type : sujet

```
# Lorsque le type user_t crée exécute le fichier ayant le type
# ssh_exec_t il transite automatiquement vers user_ssh_t
type_transition user_t ssh_exec_t:process user_ssh_t;

# Lorsque le type user_t crée exécute le fichier ayant le type
# passwd_exec_t il transite automatiquement vers passwd_t
type_transition user_t passwd_exec_t:process passwd_t;

# Règles supplémentaires correspondantes
allow user_t ssh_exec_t : file { read getattr execute };
allow user_ssh_t ssh_exec_t : file entrypoint;
allow user_r user_ssh_t : process transition;
```

SELinux en pratique

SELinux & Android

- Utilisé par Android depuis la version 4.4 (KitKat)
- Protection complète depuis la version 5.0 (Lollipop)
- Impact significatif sur le durcissement du système

- Honey, I Shrunk the Attack Surface, Adventures in Android Security Hardening, Nick Kralevich, Black Hat 2017 : vidéo
- State of SELinux, Paul Moore, LSS 2017

Red Hat Enterprise Linux, CentOS Stream, Fedora

- Targeted policy
- Les démons système sont confinés
- Les utilisateurs interactifs ne sont pas confinés par défaut (`unconfined`)
- Possibilité de confiner certains utilisateurs

Refpolicy

- Strict policy
- Tous les utilisateurs sont confinés
- La politique doit décrire l'intégralité des interactions autorisées
- Disponible principalement sous Gentoo Hardened (et partiellement sous Debian et Arch Linux)
- Restrictif et complexe

Politique et état de SELinux

- Configuration de SELinux et politique actuelle stockées dans `/etc/selinux`
- `sestatus -v` : état général
- Etat actuel de SELinux dans le noyau : `/sys/fs/selinux/`
- Etat actuel de la politique stocké dans `/var/lib/selinux/`
- `seinfo -v` : informations sur la politique

Audit

- SELinux utilise le sous système d'audit du noyau
- Démon `auditd` en espace utilisateur
- Logs dans `/var/log/audit/audit.log` ou dans le Journal
- Log par défaut tous les accès refusés
- Outils `audit2allow` et `audit2why` pour générer des règles de politique à partir de ces logs

Exemple de messages AVC

```
type=AVC msg=audit(1515257682.718:409): avc: denied { getattr } for
  pid=4320
  comm="httpd"
  path="/home/fedora/public_html/test"
  dev="sda1"
  ino=262285
  scontext=system_u:system_r:httpd_t:s0
  tcontext=unconfined_u:object_r:httpd_user_content_t:s0
  tclass=file
  permissive=0

type=AVC msg=audit(1515257956.228:527): avc: denied { setuid } for
  pid=4646
  comm="sudo"
  capability=7
  scontext=user_u:user_r:user_t:s0
  tcontext=user_u:user_r:user_t:s0
  tclass=capability
  permissive=0
```

Règle d'audit : `auditallow` et `dontaudit`

```
# Audite l'accès en lecture au type shadow_t par sysadm_t
auditallow sysadm_t file:read shadow_t;

# Ne pas auditer certains accès aux répertoires pour le type staff_t
# Utilisation de - pour retirer le type security_file_type de file_type
dontaudit staff_t { file_type -security_file_type }: dir { getattr search read lock};
```

`semodule -DB` : désactiver les règles `dontaudit`

Politique modulaire

- Politique source stockée sous forme d'une collection de fichiers texte
- Politique séparée entre une base et de nombreux modules
- « Compilation » de la base et des modules pour former la politique finale, sous forme binaire
- Seule la forme finale est chargée par le noyau
- On écrit rarement une politique de zéro
- On ajoute ou modifie un module existant

Module SELinux

- Composé de trois fichiers :
  - `.te` : ensemble des règles SELinux à appliquer
  - `.fc` : liste des associations entre chemin de fichiers et contexte (*fc : file contexts*)
  - `.if` : interface, ou ensemble de « fonctions » proposé pour utilisation dans d'autres modules
- Création d'un module à l'aide de `/usr/share/selinux/devel/Makefile`
- Installation avec `semodule --install --priority=500`

Exemple de module SELinux

`toto.te` :

```
policy_module(toto, 0.0.1)

require {
  type httpd_t;
  type httpd_sys_script_t;
  type user_tmp_t;
}

type toto_data_t;
files_type(toto_data_t);

allow httpd_t toto_data_t:file read_file_perms;
allow httpd_t toto_data_t:dir search_dir_perms;

allow httpd_sys_script_t user_tmp_t:file read_file_perms;
```

Exemple de module SELinux

`toto.fc` :

```
/var/www/html/toto         -d  gen_context(system_u:object_r:toto_data_t,s0)
/var/www/html/toto(/.*)?   --  gen_context(system_u:object_r:toto_data_t,s0)
```

Ecrire un module SELinux

1. `sepolgen` : génère un canevas à partir de template pour les modules couramment ajoutés
2. « Compilation » du module
3. Chargement du module dans le noyau
4. Tests de l'application
5. Lecture des logs d'erreur d'audit
6. Conversion des logs d'erreurs en règle `allow` et ajout au module
7. Goto 2

Fin : Désactivation du mode permissif

Gestion des contextes

- `restorecon` : restaure le contexte associé à un fichier par la politique :
  - `-R` : récursif
  - `-v` : verbeux
  - `-F` : restaure aussi le rôle et l'utilisateur SELinux
- `chcon` : modification arbitraire du contexte d'un fichier
  - `-u`, `-r`, `-t` : défini l'utilisateur, rôle, type
  - `-R` : récursif
  - `--reference=file` : utilise le contexte d'un autre fichier comme référence
- `fixfiles onboot` : restaure les contextes associés à tous les fichiers du système au prochain boot

Manipulation de la politique

- `semanage` : gestion de la politique SELinux
  - login : association entre les utilisateurs Linux et SELinux
  - user : utilisateurs SELinux
  - port : association contextes et ports réseau
  - module : modules de la politique
  - fcontext : associations contextes et fichiers
  - boolean : gestion des booléens
  - dontaudit : active/désactive les règles *dontaudit*
  - etc.

Booléens

- Une fois chargée, la politique est immuable
- Possibilité d'ajouter des conditions pour autoriser dynamiquement certaines opérations sans avoir à changer la politique
- Commandes `getsebool` & `setsebool` ou `semanage boolean`

Recherches dans la politique

- `sesearch` : recherche de règles dans la politique
- Liste tous les fichiers que le contexte `httpd_t` peut lire :

```
$ sesearch -v --allow -s httpd_t -c file -p read
```

- Liste tous les contextes pouvant écrire dans les fichiers `shadow_t` :

```
$ sesearch -v --allow -t shadow_t -c file -p write
```

- Liste toutes les règles associées au booléen `samba_enable_home_dirs` :

```
$ sesearch -v --allow -b samba_enable_home_dirs
```

Inspection de la politique

- `sepolicy` :
  - booleans : informations sur les booléens
  - communicate : est-ce que deux domaine peuvent interagir entre eux ?
  - generate : générer des modules à partir de template
  - gui : lance les outils graphiques
  - interface
  - manpage : générer les pages de man de SELinux
  - network : relation entre ports et types SELinux
  - transition : est-ce qu'un domaine peut transiter vers un autre ?

Désactiver SELinux ?

Désactiver SELinux ?

- Pour désactiver complètement SELinux, il faut :
  - Ajouter `selinux=0` sur la ligne de commande du noyau
  - Redémarrer le système
- Plus de MAC, fonctionnement classique DAC
- Avant de réactiver SELinux, il faut relabéliser intégralement le système de fichiers (`fixfiles onboot`) et redémarrer

Désactiver SELinux ?

```
# setenforce 0
```

- Mode *Permissive*, qui n'applique plus les contrôle mais log uniquement les accés refusés
- Le mode permissif ne doit pas être activé sur un système entier en production
- Si un démon ne fonctionne pas correctement avec SELinux, il faut passer uniquement ce démon en permissif :

```
# semanage permissive -l
# semanage permissive -a myapp_t
# semanage permissive -d myapp_t
```

Désactiver SELinux ?

- `/etc/selinux/config` ne permet plus de désactiver complètement SELinux

SELinux avancé

Politique Multi-Niveau (MLS & MCS)

Deux attributs supplémentaires ajoutés aux contextes de sécurité :

```
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Multi-Level Security (MLS) : Niveaux de sécurité

`unconfined_u:unconfined_r:unconfined_t:s1-s3`

- Ajout de contraintes entre différents niveau de sécurité
- Souvent utilisé pour implémenter des modèles de confidentialité
- N'est pas utilisé dans la politique *targeted*:
  - Toujours `s0`

Multi-Category Security (MCS) : Catégories

`unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023`

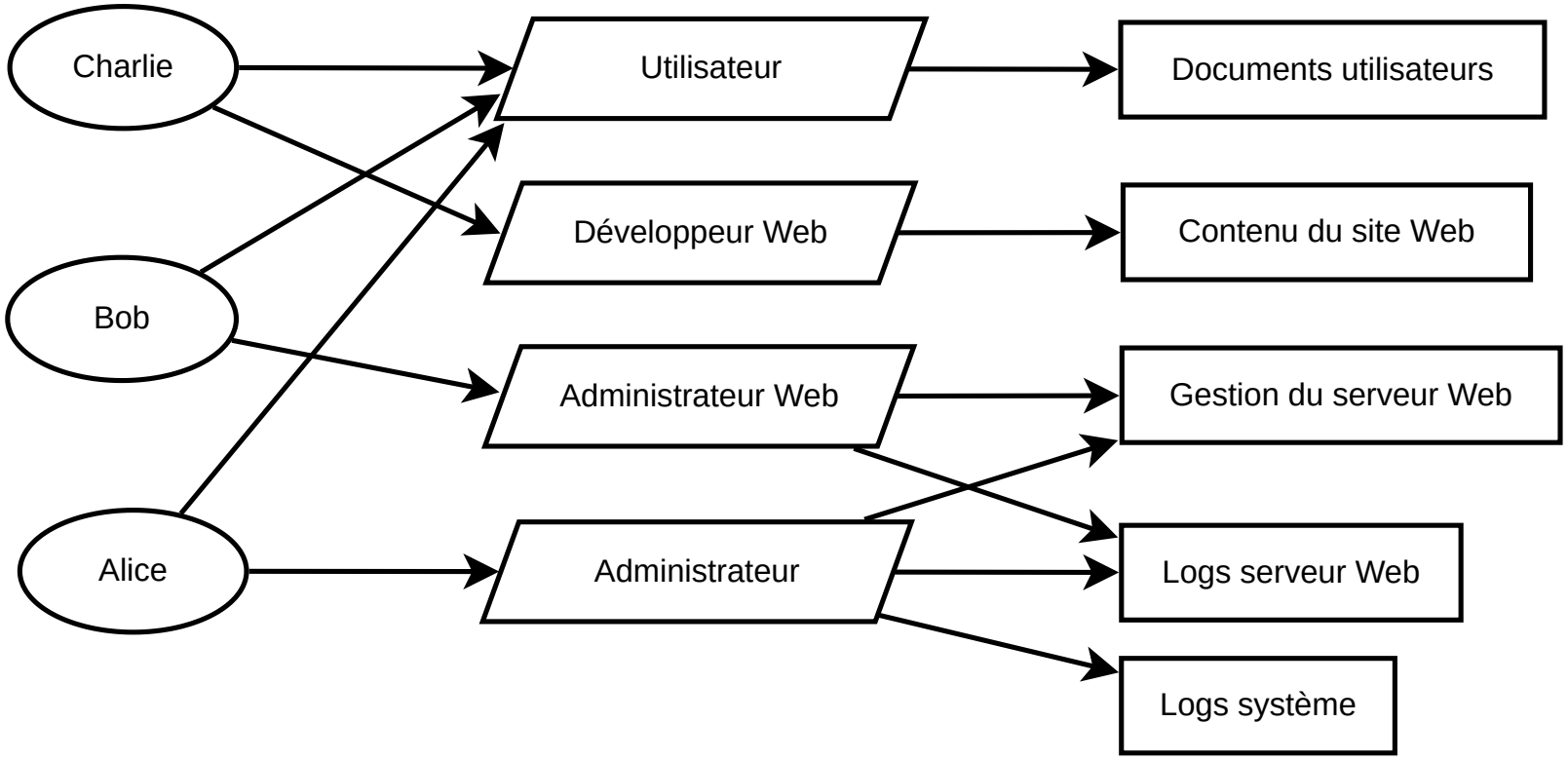
- Isole plusieurs instances d'un même type
- Le `.` définie un ensemble
  - Exemple : `c0.c16` signifie de `c0` à `c16`
- La `,` définie une liste
  - Exemple : `c21,c36,c45`
- Les deux séparateurs peuvent être combinés
  - Exemple : `c0.c16,c21,c36,c45`

Politique Multi-Niveau (MLS & MCS)

- Contrôle entre les niveaux et les catégories est réalisé via des contraintes SELinux
- Décrites dans la politique : `policy/mls`, `policy/mcs`
- Tous les types ne sont pas nécessairement contraints
- Utilisé principalement pour confiner les machines virtuelles et les conteneurs (cf. chapitre sécurité & conteneurs, sécurité & KVM)

Role Based Access Control (RBAC)

- Sous concept pour mettre en place le MAC
- Objectif : factoriser les opérations réalisables sur des objets en un rôle
- Association des utilisateurs à un ou plusieurs rôles



Factorisation des permissions

- Macros pour simplifier l'utilisation des permissions : `policy/support/*perms_set.spt`
- Exemples :
  - `r_file_perms` : permissions pour lire un fichier
    - `open getattr read lock ioctl`
  - `r_dir_perms` : permissions pour traverser un répertoire
    - `open getattr read lock search` ioctl`
  - `x_file_perms` : permissions pour exécuter un fichier
    - `open getattr execute`

Attributs

Attributs : regroupe plusieurs types pour factoriser la politique :

```
# Déclaration d'un attribut
attribute file_type;
attribute application_domain_type;
attribute application_exec_type;

# Association d'un type (déjà déclaré) à un attribut (déjà déclaré)
typeattribute shadow_t security_file_type;

# Déclaration d'un type et association à un attribut (déjà déclaré)
type sshd_exec_t executable_file_type;
```

Attributs et règles de contrôle d'accès

```
# Les sujets avec le type sysadm_t peuvent exécuter
# les fichiers dont le type a pour attribut application_exec_type
allow sysadm_t application_exec_type:file { getattr read execute ... };
```

Transition de type pour les fichiers

- Par défaut, le contexte de sécurité d'un fichier est celui de son répertoire parent
- Possibilité d'ajouter des règles pour associer des contextes différents en fonction des répertoire ou noms de fichiers

Transition de type : objets

```
# Lorsque le processus labellé par user_t accède à un fichier de type tmp_t,
# le type de cet objet transite automatiquement vers user_tmp_t
type_transition user_t tmp_t:{ dir file lnk_file sock_file fifo_file } user_tmp_t;

# Lorsque le processus labellé par sshd_t accède à un fichier de type tmp_t,
# le type de cet objet transite automatiquement vers sshd_tmp_t
type_transition sshd_t tmp_t:{ dir file sock_file } sshd_tmp_t;

# Règles supplémentaires correspondantes
allow user_t tmp_t: dir { open getattr create write };
allow user_t user_tmp_t:file rw_file_perms;

# Lorsque qu'un processus user_t créé un dossier nommé .ssh dans un dossier
# user_home_dir_t, il est labellé ssh_home_t
type_transition user_t user_home_dir_t:dir ssh_home_t .ssh;
```

Contraintes

```
# Les types non privilégiés ne peuvent pas lire shadow_t
# Utilisation de ~ pour la négation
neverallow ~can_read_shadow_passwords shadow_t:file read;
```

Points de montage

- Options à passer à `mount`
- Force les contexte pour tous les fichiers :
  - `-o context=user:role:type`
- Contexte par défaut pour les fichiers sans contexte :
  - `-o defcontext=user:role:type`
- Contexte pour le système de fichier :
  - `-o fscontext=user:role:type`

Suite

Sécurité des conteneurs