Documentation CLIP

1202

Patch Vserver

Version	Date	Auteur	Commentaires
1.2	14/10/2008	Vincent Strubel	Ajout du contrôle des politiques de sécurité IPsec en fonction du contexte réseau (2.2.4). A jour pour *clip-kernel-2.6.22.24-r14* (CLIP v.03.00.21).
1.1	29/09/2008	Vincent Strubel	Actualisation des fichiers du */proc* exposés dans les cages (ajouts des fichiers nécessaires à *java* et au contrôle de consommation).
1.0.2	31/07/2008	Vincent Strubel	Passage au format OpenOffice.
1.0.1	18/09/2007	Vincent Strubel	Ajout de remarques : verrous de fichiers et *inotify*. A jour pour *clip-kernel-2.6.19.7-r15, vsctl-1.0.10, clip-libvserver-4.0.4* et *pam_jail-1.0.1*.
1.0	08/08/2007	Vincent Strubel	Version initiale, à jour pour *clip-kernel-2.6.19.7-r10, vsctl-1.0.9, clip-libvserver-4.0.3* et *pam_jail-1.0*.

Table des matières

Introduction

A la différence du patch *CLIP-LSM* décrit dans [CLIP_1201], développé spécifiquement pour CLIP, le patch *Vserver* est constitué de code "sur étagère" ([VSERVER]), qui est simplement mis en œuvre dans CLIP, moyennant quelques adaptations. Ce patch apporte un mécanisme de virtualisation de niveau "*userland*" pour Linux, comparable aux *jails* FreeBSD ou aux *zones* Solaris. Le principe d'une virtualisation *userland* est de partitionner l'environnement utilisateur (processus) en plusieurs entités distinctes partageant de manière transparente un même noyau. Un tel mécanisme est par construction plus léger qu'une virtualisation complète (cas de *VMWare* par exemple) ou une paravirtualisation (cas de *Xen*). *Vserver* est développé comme un patch distinct de la branche de développement principale du noyau Linux, mais met en œuvre autant que possible des mécanismes de contrôle d'accès présents dans le noyau standard (capacités POSIX, *namespaces, chroot*, *rlimits*). Il est maintenu à jour vis-à-vis de la dernière version stable du noyau Linux.

Ce principe de virtualisation *userland* constitue la base du principal mécanisme de cloisonnement mis en œuvre dans un système CLIP. Le présent document décrit le principe de fonctionnement de la virtualisation *Vserver* (section 1), sa mise en œuvre dans CLIP (section 2), ainsi que les trois interfaces de configuration de cages *Vserver* spécifiques à CLIP, la bibliothèque *libclipvserver* (section 3), l'utilitaire *vsctl* (section 4) et le module *pam_jail* (section 5). Ces descriptions correspondent à la version du patch *Vserver* déployée dans CLIP (version *2.2.**), avec, sauf mention explicite du contraire, la configuration retenue dans CLIP, qui exclut en particulier la compatibilité avec les appels systèmes "*legacy*" de versions plus anciennes (option *CONFIG_VSERVER_LEGACY* non sélectionnée).

1 Description du patch *vserver*

1.1 Contextes de sécurité *vserver*

L'entité de base d'une virtualisation *Vserver* est le **contexte de sécurité** (*security context*), commun à un ensemble de processus. Un contexte est identifié par un entier unique, son *xid* (ainsi que par un nom en toutes lettres[1]), et représenté en mémoire noyau par une structure *struct vx_info*, comportant notamment, en plus du *xid*, les champs suivants :

- **Un masque maximal de capacités POSIX** (*capability bounding set*), qui définit les seules capacités que peut posséder un processus du contexte (avec un fonctionnement similaire à */proc/sys/kernel/cap-bound*, mais propre au contexte)
- **Un masque de capacités POSIX utilisables** (*bcap*), définissant celles des capacités POSIX qui peuvent être mises en œuvre dans le contexte. Concrètement, le contrôle d'accès réalisé par un noyau Linux patché *vserver*, lors d'un appel système dans un contexte autre que ADMIN, consiste à vérifier, d'une part que la tâche appelante possède la capacité requise dans son masque de capacités effectif, et d'autre part que cette même capacité est incluse dans le masque de capacités utilisables du contexte auquel appartient la tâche. Soit pour une capacité *CAP* :

```
capable(CAP) := [ current->cap_effective & (1<<CAP)]
                        && [ current->(ctx)->bcap & (1<<CAP)]
```

Il est important de souligner la différence entre ce masque de capacités utilisables et le masque de capacités maximal évoqué plus haut : le premier n'affecte que l'évaluation des capacités, mais non leur attribution, tandis que le deuxième modifie directement le calcul des capacités attribuées à chaque tâche du contexte. Les capacités comprises dans le masque maximal mais pas dans le masque utilisable d'un contexte sont attribuées aux processus *root* de ce contexte (elles sont présentes dans les champs *cap_effective* et *cap_permitted* des *struct task_struct* correspondantes), mais ne sont pas "vues" par un test de capacités (par exemple *capable()*). Leur présence n'est exploitée que dans l'évaluation des capacités de contexte (voir ci-dessous). Le masque maximal est normalement laissé à sa valeur par défaut par les outils *util-vserver*, soit la même valeur que le *cap-bound* principal lors de la création du contexte. Dans ce cas, la réduction des privilèges de *root* dans un contexte de sécurité *vserver* repose entièrement sur le masque de capacités utilisables.

On notera par ailleurs que ce masque n'est pas appliqué lorsque le contexte est dans l'état intermédiaire *STATE_SETUP* (cf. Tableau 2), qui correspond typiquement à la configuration initiale du contexte : la tâche créatrice d'un contexte est automatiquement enfermée dans ce contexte dès sa création, mais elle peut encore lancer des appels systèmes privilégiés pour configurer certaines fonctionnalités du contexte (masques de capacité, *rlimits*, etc.), avant d'abandonner définitivement ces privilèges en mettant à zéro le drapeau *STATE_SETUP*. De

[1] A la différence des *jails* FreeBSD, et de manière similaire aux *zones* Solaris, ces identifiants sont établis explicitement à la demande du créateur de contexte, et non dynamiquement par le noyau (cas des *jid* FreeBSD)

manière similaire, le masque maximal de capacités n'est appliqué (au lieu du *cap-bset* global du système) qu'une fois le drapeau *STATE_SETUP* mis à zéro. Ce masque maximal est par ailleurs appliqué directement (sans attendre un *exec()*) à la tâche qui met à zéro le drapeau *STATE_SETUP*.

- **Un ensemble de capacités de contexte** (cf. Tableau 1). Chacune de ces capacités donne un accès limité à un sous-ensemble des opérations contrôlées par une capacité POSIX qui ne figurerait pas elle-même dans le masque de capacités effectives du contexte (par exemple : l'ouverture de *socket raw* – contrôlée normalement par *CAP_NET_RAW* – est permise pour le protocole *icmp* uniquement par la capacité de contexte *VXC_RAW_ICMP*). Un processus est autorisé à réaliser les opérations d'un tel sous-ensemble dès lors qu'il possède la capacité POSIX d'origine[2] dans son masque effectif (même masquée par le masque effectif du contexte) et que le contexte possède lui-même la sous-capacité de contexte. Le modèle de contrôle d'accès pour une ressource relevant de la capacité POSIX C_{pos} et de la capacité de contexte C_{ctx} est ainsi :

```
vx_capable(C_pos, C_ctx) := [ current->cap_effective & (1<<C_pos)]
                            && [ current->(ctx)->ccaps & (1<<C_ctx)]
```

- **Un ensemble de drapeaux de contexte** contrôlant les options d'émulation *Vserver,* par exemple émulation du *hostname* ou de l'horloge (cf. Tableau 2).
- **Un processus *"init"***, normalement le premier processus lancé dans ce contexte, qui sera vu comme ayant un *pid* égal à 1 par les autres processus du contexte, et jouera le rôle de *"child reaper"* pour les processus orphelins du contexte.
- **Un *namespace VFS* et le *filesystem* associé**, qui correspondent à un partitionnement de l'espace de nommage des montages du système de fichiers virtuel (*VFS*). Ce partitionnement permet d'effectuer des montages limités à un contexte spécifique, et en particulier de réaliser dans chaque contexte autre que le contexte initial un montage de type *bind* "récursif" d'un répertoire arbitraire comme nouvelle racine du système de fichiers[3].
- **Un *namespace proxy***, référençant un ensemble d'espaces de nommage (dont le *namespace* VFS évoqué plus haut, mais aussi les *namespaces IPC* et *utsname* introduits dans Linux-2.6.19, et le *namespace pid* dans les noyaux 2.6.20 ou plus récents). Ces espaces de nommage dédiés au contexte limitent de manière simple l'accès aux différentes ressources qui leurs sont associées, pour les processus du contexte. Un processus d'un contexte donné ne peut ainsi accéder qu'aux objets *IPC*[4] appartenant à son espace de nommage, et donc pas aux objets

[2] La présence de cette capacité effective est testée directement dans le masque *cap_effective* de la *struct task_struct* associée au processus, et non à l'aide de la fonction *capable()*. Ainsi, le masque de capacités utilisables du contexte n'est pas pris en compte dans cette évaluation. Les capacités de contextes correspondent typiquement à des capacités POSIX qui sont autorisées (incluses dans le masque maximal) mais pas utilisables directement (exclues du masque utilisable) dans le contexte.

[3] Ce remontage de la racine, combiné à un *chroot* des processus qui rejoignent le contexte, se traduit par un confinement à une sous-arborescence du système de fichier, similaire dans ses effets à un simple *chroot*, mais avec une volatilité moindre que ce dernier. En particulier, le cloisonnement *VFS* réalisé par *Vserver* résiste aux différentes attaques connues permettant à *root* de s'échapper trivialement d'une prison *chroot*.

[4] L'accès indirect par un espace de nommage ne concerne que les *IPC System V*, et non les *IPC Posix*, qui ne font l'objet d'aucun traitement spécifique par *vserver* – dans la mesure où l'accès à ces ressources est contrôlé par les systèmes de

accessibles depuis un autre contexte (partitionnement strictement disjoint).

- **Un ensemble de statistiques** propres aux processus du contexte, par exemple consommation CPU (cf. [VSERVERDOC], *#Context_Accounting*)
- **Un ensemble de limites quantitatives d'accès aux ressources**, correspondant à des limites *rlimits* (complétées de quelques limites complémentaires) imposées à tous les processus du contexte. Chaque limite est composée d'une limite "molle" et d'une limite "dure", accompagnées de champs stockant la valeur courante pour le contexte, les valeurs maximale et minimale et le nombre de "*hits*" de la limite depuis le démarrage du contexte.
- **Une politique d'ordonnancement** conditionnant l'accès au temps CPU par les processus du contexte.

Capacité	Mot-clé *vsctl*	Capacité POSIX associée	Description
SET_UTSNAME	***set_utsname***	*SYS_ADMIN*	Autorise *setdomainname()* et *sethostname()*.
SET_RLIMIT	***set_rlimit***	*SYS_RESOURCE*	Autorise *setrlimit()*.
RAW_ICMP	***raw_icmp***	-	Autorise la création de *socket icmp raw*.
SYSLOG	***syslog***	*SYS_ADMIN*	Autorise la lecture de */proc/kmsg*.
SECURE_MOUNT	***secure_mount***	*SYS_ADMIN*	Autorise les montages VFS avec l'option *nodev*, et les démontages VFS.
SECURE_REMOUNT	***secure_remount***	*SYS_ADMIN*	Autorise les remontages VFS avec l'option *nodev*.
BINARY_MOUNT	***binary_mount***	*SYS_ADMIN*	Autorise les montages binaires et réseau.
QUOTA_CTL	***quota_ctl***	*SYS_ADMIN*	Autorise les *ioctl* de configuration des quotas.
ADMIN_MAPPER	***admin_mapper***	*SYS_ADMIN*	Autorise les *ioctl* de configuration du *device-mapper.*
ADMIN_CLOOP	***admin_loop***	*SYS_ADMIN*	Autorise la création de *loop devices*, et la lecture des clés associées aux *cryptoloop*.

Tableau 1: Capacités de contexte des contextes de sécurité *vserver.*

Un processus confiné dans un contexte de sécurité ne peut modifier aucune des propriétés de ce contexte (sauf si celui-ci possède un drapeau *SETUP*, qui est normalement mis à zéro avant la mise en service effective du contexte). Par ailleurs, deux contextes de sécurité sont définis pour des rôles spécifiques :

- **Le contexte d'administration** (***ADMIN***, *xid 0*) est le seul à pouvoir créer un contexte (sous réserve de disposer par ailleurs des capacités POSIX *CAP_SYS_ADMIN* et *CAP_CONTEXT*), ou à pouvoir confiner un processus dans un contexte existant. Il s'agit du contexte par défaut, dans lequel le système est démarré (c'est-à-dire le contexte du véritable processus *init*, premier

fichiers sous-jacents.

lancé par le noyau).

- **Le contexte d'audit** ***(WATCH****, xid 1*) est un concept purement observateur, qui seul peut voir directement l'ensemble des processus et structures labellisées, tous contextes confondus, mais qui ne dispose en revanche que d'un accès en lecture à ces contextes, n'ayant pas le droit par exemple d'envoyer un signal à un processus d'un autre contexte. L'accès en lecture peut lui-même être bloqué en sélectionnant l'option *CONFIG_VSERVER_PRIVACY* à la compilation du noyau (option non retenue à ce stade dans CLIP).

Outre les processus (*struct task_struct*), un certain nombre de structures internes au noyau sont labellisées, par inclusion du *xid* du contexte de sécurité dans lequel ces structures sont créées. Les structures concernées sont principalement les suivantes :

- Sockets
- Pseudo-terminaux UNIX98 (*pts*) (*mais pas BSD*)
- Fichiers et verrous de fichiers (y compris dans les systèmes de fichiers virtuels, notamment dans le *procfs*). La labellisation des *inodes* est représentée en utilisant les bits de poids fort des champs UID et GID (cf. [VSERVERDOC], *#Filesystem_XID_Tagging*), ce qui permet de la sauvegarder sur le disque sans modification du format. On notera que cette labellisation des fichiers sur le disque n'est utilisée qu'à des fins de comptabilité (principalement pour l'implémentation des limites de ressources et des quotas par contexte).
- Utilisateurs. Les structures *struct user_struct* sont labellisées, ce qui permet la définition de limites de ressources par utilisateur par contexte.
- Les *devices* de type *loop*.

Cette labellisation permet de gérer les statistiques d'accès par contexte, les limites quantitatives d'accès aux ressources, mais aussi la visibilité de ces ressources par les processus. Ainsi, un processus appartenant au contexte "A" ne pourra voir (par *ps* ou lecture directe de */proc*) que les autres processus appartenant au contexte A (sauf dans le cas du contexte **AUDIT** décrit plus bas), et ne pourra envoyer des signaux ou des commandes *ptrace* qu'à ces processus. Ce processus ne "verra" par ailleurs (par *netstat* par exemple) que les sockets labellisées par ce même contexte, c'est-à-dire créées par des processus de ce contexte. De manière similaire, le système de fichier *devpts* ne "montrera" aux processus utilisateurs que les pseudo-terminaux labellisés par le même contexte. On obtient ainsi un partitionnement similaire à celui obtenu à travers le mécanisme de *namespaces*, pour celles des ressources noyau auxquelles l'accès n'est pas encore conditionné à l'appartenance à un tel espace de nommage. La principale différence tient au fait que l'unicité de l'espace de nommage sous-jacent laisse des "trous" dans l'énumération des ressources dans chaque contexte, ce qui introduit des canaux cachés de communication inter-contextes, à faible débit.

On notera bien que le contrôle de la visibilité intrinsèque des ressources – et ce aussi bien par un mécanisme de *namespace* que par la labellisation directe – n'exclut pas l'accès à ces mêmes ressources à travers une référence indirecte, portée par un autre objet lui-même visible. Ainsi, alors que le cloisonnement en visibilité ne permet pas à un contexte de se connecter à une socket UNIX anonyme créée dans un autre contexte, il ne bloque en revanche pas l'accès à une socket UNIX nommée, indépendamment de son contexte de création, dès lors que le fichier associé est visible. Cette propriété

permet de créer des canaux de communication maîtrisés entre contextes, par exposition explicite de certaines ressources.

Le partitionnement en visibilité est par ailleurs complété par la virtualisation de certaines informations vis-à-vis des différents contextes, auxquels le noyau peut présenter, de manière généralement configurable à travers les drapeaux de contexte, une vision adaptée de différentes ressources. C'est le cas par exemple d'un certain nombre de limites matérielles (espace disque, mémoire disponible) qui peuvent être représentées dans un contexte non-privilégié comme les limites logicielles affectées à ce contexte par le contexte ADMIN (voir plus bas), ou encore du temps système, pour lequel un biais spécifique à chaque contexte peut être configuré depuis le contexte ADMIN. On notera la différence entre simple isolation (une entité labellisée ne peut pas voir une entité labellisée différemment) et virtualisation (une entité labellisée voit une version artificiellement adaptée d'une ressource commune).

Remarque 1 : Simple isolation et canaux cachés

La simple isolation des ressources crée des canaux auxiliaires permettant éventuellement la transmission d'information entre contextes, dans la mesure où la visibilité ou non d'une ressource constitue en soi une information. Un exemple de tel canal auxiliaire est celui associé au système de fichier devpts*, dans lequel les pseudo-terminaux sont numérotés linéairement de manière commune à tous les contextes. La présence de "trous" dans la numérotation vue par un contexte, révélatrice de l'ouverture des terminaux correspondants par un autre contexte, permet ainsi de faire fuir une quantité limitée d'information entre ces deux contextes. Par défaut, la numérotation (*pid*) des processus constitue un canal caché de même nature et à débit légèrement plus important. Deux approches sont envisageables pour bloquer de tels canaux. La première consiste à introduire une randomisation des noms ou numéros affectés à chaque type de ressource, afin de bruiter d'éventuelles communications inter-contextes. Cette approche était précédemment mise en oeuvre pour les* pid *grâce à une fonctionnalité de* grsecurity*, qui n'est cependant plus disponible dans la version 2.1.10, actuellement mise en oeuvre dans CLIP, de ce patch. La deuxième approche, préférable car plus complète et généralement moins impactante en termes de performances, consiste à faire intervenir un espace de nommage explicite dans l'accès à chaque ressource, comme cela est déjà fait pour les montages VFS ou les IPC. On notera au demeurant qu'un espace de nommage pour les* pid *a été introduit dans la version 2.6.20 du noyau Linux.*

Drapeau	Mot-clé *vsctl*	Type	Description
INFO_PRIVATE	***private***	-	Interdire l'entrée dans ce contexte depuis le contexte ADMIN.
INFO_INIT	***fakeinit***	-	Montrer un faux processus *init*.
SCHED_HARD	***sched_hard***	-	Appliquer les limites dures d'ordonnancement (bloquer le contexte lorsqu'il a épuisé son quota CPU).
SCHED_PRIO	***sched_prio***	-	Ajuster la priorité du contexte en fonction de son utilisation du CPU.
SCHED_PAUSE	***sched_pause***	-	Contexte bloqué (retiré en bloc de l'ordonnanceur).
VIRT_MEM	***virt_mem***	-	Virtualiser les informations sur la mémoire disponible.
VIRT_UPTIME	***virt_uptime***	-	Virtualiser l'*uptime*.
VIRT_CPU	***virt_cpu***	-	Virtualiser les informations sur l'utilisation du CPU.
VIRT_LOAD	***virt_load***	-	Virtualiser la charge système.
VIRT_TIME	***virt_time***	-	Introduire un biais par contexte dans l'heure système.
HIDE_MOUNT	***hide_mount***	-	Masquer les entrées de */proc/<tgid>/mounts*.
HIDE_NETIF	***hide_netif***	-	Masquer les interfaces réseau qui ne sont pas utilisables dans le contexte réseau courant (interfaces dont aucune des adresses n'est autorisée pour le contexte réseau).
HIDE_VINFO	***hide_vinfo***	-	Masquer les informations de contexte dans */proc/<tgid>/vinfo* et */proc/<tgid>/ninfo*.
STATE_SETUP	***state_setup***	IOD	Etat intermédiaire : le contexte est créé par une tâche, mais n'est pas joignable par d'autres tâches, et les masques de capacités ne sont pas appliqués à la tâche créatrice. Cette dernière est autorisée à migrer dans le contexte même si celui-ci porte le drapeau *INFO_PRIVATE*. Certains appels systèmes ne peuvent être lancés que dans cet état (cf. Annexe A).
STATE_INIT	***state_init***	IOD	Etat intermédiaire : le processus *init* du contexte n'est pas attribué, le vrai processus *init* du système est visible dans */proc*.
STATE_ADMIN	***state_admin***	OD	Autorise l'administration du contexte depuis le contexte ADMIN.
SC_HELPER	***sc_helper***	I	Appeler l'utilitaire défini par le *sysctl kernel.vshelper* lors des changements d'état (démarrage, arrêt) du contexte.
REBOOT_KILL	***reboot_kill***	-	Tuer tous les processus du contexte en cas d'appel à *reboot*.
PERSISTENT	***persistent***	-	Contexte persistant : le contexte n'est pas supprimé lorsque son dernier processus se termine.
FORK_RSS	***fork_rss***	-	Bloquer les appels *fork* lorsque la limite *rlimit RSS* est atteinte.
IGNEG_NICE	***igneg_nice***	-	Ignorer silencieusement les tentatives d'augmenter la priorité d'une tâche du contexte (en l'absence de ce drapeau, une erreur est retournée).

Tableau 2: Options des contextes de sécurité *vserver*.

Abréviations des types : O – le drapeau peut être retiré, mais pas rétabli après cela ; I – le drapeau est à usage interne uniquement ; D – le drapeau est attribué par défaut à un nouveau contexte.

1.2 Contextes réseau *vserver*

Indépendamment[5] de ces contextes de sécurité, *Vserver* permet d'associer à des processus un **contexte réseau**, caractérisé par un identifiant entier (*nid*), représenté par une structure *struct nx_info* en mémoire noyau. Outre le *nid* et le nom du contexte, les principaux champs d'une *struct nx_info* sont les suivants :

- **Un tableau de quatre adresses IPv4**, dont une au moins (la première) doit être non-nulle. Ces adresses sont les seules autorisées comme adresses sources de paquets émis par le contexte, et adresses destinations de paquets reçus par celui-ci. Lors de l'émission d'un paquet, son adresse source est choisie parmi ces quatre adresses en fonction des contraintes de routage, la première adresse étant utilisée par défaut. Cette première adresse est par ailleurs attribuée automatiquement (par un *bind* implicite) à toutes les *sockets* du contexte qui n'auraient pas fait l'objet d'un appel *bind()* explicite sur l'une des autres adresses non-nulles du tableau, ou qui feraient un *bind(INADDR_ANY)*[6] ou *bind(INADDR_LOOPBACK)*. Elle de plus utilisée automatiquement comme adresse destination de tous les paquets envoyés sur la boucle locale du système. Enfin, lorsque l'option de compilation *CONFIG_VSERVER_REMAP_SADDR* a été sélectionnée lors de la configuration du noyau, cette adresse est aussi imposée comme adresse source des paquets émis sur la boucle locale.
- **Un tableau de quatre masques réseau** associés un à un aux adresses IPv4 décrites ci-dessus.
- **Une adresse *broadcast v4_bcast*** qui peut être utilisée au lieu de la première adresse autorisée dans le traitement de certains appels *bind()*. Cette adresse est par défaut égale à la première adresse du contexte dans les implémentations actuelles de *vserver*.
- **Un bitmask de drapeaux de contexte**, permettant de définir certaines options du contexte (cf. Tableau 3).
- **Un bitmask de capacités de contexte**. Aucune capacité de contexte réseau n'est utilisée par le patch *Vserver* actuel (*v2.2*).

On notera que le fait d'appartenir à un contexte réseau n'interdit pas en soi la création d'un autre contexte avec des adresses autorisées différentes de celles du contexte courant. Il suffit pour cela de disposer de la capacité POSIX *CAP_NET_ADMIN*. Le cloisonnement réseau réalisé par un contexte réseau n'est ainsi valable que s'il est combiné avec la réduction de privilèges imposée par un contexte de sécurité. La démarche généralement adoptée pour la création des contextes *vserver* consiste donc à créer en premier lieu un contexte réseau, à s'enfermer dedans (de manière alors encore réversible), puis à créer un contexte de sécurité comportant un masque de capacité POSIX suffisamment restrictif, et enfin à s'enfermer dans ce contexte, cette dernière opération ayant pour effet de verrouiller aussi bien le contexte réseau que le contexte de sécurité. Le *xid* du contexte de sécurité et le *nid* du contexte

[5] "Indépendamment" signifie ici qu'aucun lien n'est établi a priori entre contextes de sécurité et réseau, qui sont deux propriétés distincte de chaque processus. Il est du ressort de l'outil créant une instance *Vserver* de définir aussi bien un contexte de sécurité qu'un contexte réseau.

[6] Plus précisément, le comportement lors d'un *bind(INADDR_ANY)* diffère selon qu'une seule ou plusieurs des quatre adresses du contexte sont définies. Dans le premier cas, l'appel effectué correspond à un *bind* sur la seule adresse du contexte. Lorsque le contexte possède plusieurs adresses autorisées, en revanche, le *bind* donne l'adresse *0* (*INADDR_ANY*) à la socket, mais en lui imposant une adresse *broadcast* égale à la première adresse du contexte.

réseau sont généralement choisis égaux pour plus de simplicité.

On peut par ailleurs souligner le fait que l'attribution d'adresses IP à un contexte réseau *vserver* est totalement décorrélée de la configuration des interfaces réseau du système. Rien n'interdit de n'attribuer à un contexte que des adresses qui ne sont pas configurées sur la ou les interfaces du système. Dans ce cas, aucun trafic ne pourra être émis ou reçu, y compris sur la boucle locale, depuis ce contexte. A contrario, l'attribution à un contexte d'une adresse configurée sur l'une des interfaces réseau du système permettra au contexte d'émettre et de recevoir du trafic réseau sur cette interface, mais aussi automatiquement sur la boucle locale (même si l'adresse n'est pas configurée sur cette dernière).

Drapeau	Mot-clé *vsctl*	Type	Effet
INFO_PRIVATE	***private***	-	(*non implémenté*) Interdire l'entrée dans le contexte depuis le contexte ADMIN. N'intervient que dans l'articulation *SETUP/PRIVATE* pour l'instant.
STATE_SETUP	***Setup***	IO	(*non utilisé par défaut*) Etat intermédiaire permettant à une tâche ADMIN de migrer dans un contexte portant le drapeau *INFO_PRIVATE*. Contrairement aux contextes de sécurité, cet état intermédiaire n'est pas utilisé par défaut.
STATE_ADMIN	***admin***	OD	Le contexte peut être administré par le contexte de sécurité ADMIN.
SC_HELPER	***sc_helper***	I	Appeler l'utilitaire définit par le *sysctl kernel.vshelper* lors des changements d'état (ajout ou retrait d'une interface) du contexte.
PERSISTENT	***persistent***	-	Contexte persistant : le contexte n'est pas supprimé lorsque son dernier processus se termine.
NO_SP	***no_sp***	-	Contexte autorisé à accéder au réseau sans contrôle par une politique de sécurité IPsec. Spécifique à CLIP, cf. 2.2.4.

Tableau 3: Options des contextes réseau *vserver*.

Abréviations des types : O – le drapeau peut être retiré, mais pas rétabli après cela ; I – le drapeau est à usage interne uniquement ; D – le drapeau est attribué par défaut à tout nouveau contexte.

1.3 Fonctionnalités complémentaires

Outre les fonctions principales d'isolation ou de virtualisation des ressources, *Vserver* met à la disposition des administrateurs un certain nombre d'outils supplémentaires, permettant d'améliorer le niveau de confinement des différents contextes.

1.3.1 Attributs supplémentaires pour les fichiers du */proc*

Le patch *vserver* introduit des attributs supplémentaires afin de gérer la visibilité de fichiers individuels de ce système de fichiers a priori commun à tous les contextes (en dehors des répertoires propres à un processus). Ces attributs sont stockés dans un bitmask *vx_flags* ajouté aux structures *struct proc_inode* et *struct proc_dir_entry.* Trois drapeaux peuvent être configurés dans ce champ :

- *IATTR_HIDE* : lorsque ce drapeau est positionné, le fichier est invisible dans tous les contextes.
- *IATTR_ADMIN* : rend le fichier visible dans le contexte ADMIN, lorsque *IATTR_HIDE* est positionné. Ce drapeau est sans effet lorsque *IATTR_HIDE* n'est pas positionné.
- *IATTR_WATCH* : rend le fichier visible dans le contexte WATCH, lorsque *IATTR_HIDE* est positionné. Ce drapeau est sans effet lorsque *IATTR_HIDE* n'est pas positionné.

Tous les fichiers du *procfs*, à l'exception des répertoires par processus et de leur contenu, sont initialisés au démarrage avec les drapeaux *IATTR_HIDE* et *IATTR_ADMIN* positionnés. Les attributs peuvent ensuite être modifiés fichier par fichier à l'aide de la commande *VCMD_set_iattr* de l'appel système *sys_vserver* (cf. 1.4.1), par exemple en utilisant l'utilitaire *vsattr* (cf. 4.5).

1.3.2 Montages *bind* en lecture seule

L'option de montage en lecture seule "*ro"* est normalement associée au superblock d'un système de fichiers, et par conséquent non applicable aux montages virtuels de type *bind,* pour lesquels le *superblock* reste celui du système de fichier sous-jacent. Le patch *Vserver* ajoute un drapeau *MNT_RDONLY* aux structures *struct vfsmount* (*include/linux/mount.h*), et ajoute un test du *vfsmount* sous-jacent dans toutes les opérations d'écriture (y compris modification des attributs, etc.) génériques et spécialisées pour les systèmes de fichiers principaux sous Linux : *ext2*, *ext3*, *reiserfs* (version 3), *xfs*, *jfs* et *nfs*. Ces tests rendent valide le passage d'une option "*ro"* à un montage de type *bind*, à quelques différences près par rapport à un *superblock* en lecture seule (cf. remarques ci-dessous). Cette fonctionnalité permet notamment de partager de manière sécurisée une même sous-arborescence du système de fichier entre deux arborescences de contextes *vserver* distincts.

Remarque 2 : Montages* bind *en lecture seule et* sockets *UNIX

L'implémentation des montages bind *en lecture seule dans* Vserver *n'interdit pas la création de fichiers par l'intermédiaire d'un* bind() *de* socket PF_UNIX. *Cette faille pourrait dans le contexte de CLIP permettre une attaque sur la disponibilité du service de mise à jour par exemple, par la création de fichiers sur lesquels* dpkg *refuserait d'écrire lors d'une installation de paquetage de mise à jour. Par ailleurs, elle pourrait remettre en cause le cloisonnement en confidentialité entre deux cages coopérant pour faire fuir de l'information,*

à condition qu'un répertoire commun soit monté en lecture seule par un montage bind *dans les deux cages : une cage pourrait alors créer un serveur écoutant sur une* socket *UNIX créée dans ce répertoire, à laquelle la deuxième cage pourrait librement venir se connecter.*

Afin de contrer la menace en disponibilité mentionnée plus haut, et d'éviter une future vulnérabilité potentielle en confidentialité, un patch spécifique (issu du groupe de patches clip-patches*) est appliqué au noyau CLIP afin d'interdire le* bind() *de* socket *UNIX sur un montage* bind *en lecture seule.*

***Remarque 3 : Montages* bind *en lecture seule et modification du* atime**

Une autre différence entre montages en lecture seule de type bind *(propriété du* vfsmount*) et de type standard (propriété du* superblock*) concerne la gestion de l'horodatage de dernier accès par l'attribut* atime *des* inodes*. Celui-ci n'est pas mis à jour lors d'un accès à un inode dont le* superblock *porte l'option* "ro"*; il l'est en revanche lorsque seul le montage VFS est marqué MNT_RDONLY (sauf option supplémentaire* noatime*, qui constitue elle aussi une propriété du* vfsmount*). Cette exception ne constitue pas vraiment une menace en intégrité (seul un attribut a priori non sensible est modifiable), mais pourrait éventuellement être utilisée comme canal auxiliaire pour faire fuir de l'information entre contextes différents. Le cas échéant, cette menace peut être couverte soit par la disjonction des systèmes de fichiers sous-jacents aux deux cages, qui garantit qu'un contexte ne peut accéder qu'aux* inodes *d'un système de fichiers qui lui est propre et non visible des autres contextes non-privilégiés, soit par l'ajout de l'option* noatime *aux montages effectivement partagés.*

Remarque 4 : Montages partagés entre contextes et verrous sur les fichiers

L'interdiction des accès en écriture à un montage n'interdit pas de déclarer des verrous (par fcntl()*,* lockf() *ou* flock()*) sur les fichiers de ce montage. De tels verrous peuvent porter atteinte à la disponibilité de certains services (mais uniquement les services utilisant eux-mêmes de tels verrous, sauf si le support des verrous "mandataires" est aussi activé dans les options de montage). Par ailleurs, ils peuvent être mis en oeuvre afin de créer un canal de communication auxiliaire entre deux contextes traitant des informations de niveaux de sensibilité différents, lorsque ces contextes partagent un montage en lecture seule. Il est de ce fait important d'ajouter à de tels montages partagés l'option* 'nolock'*, supportée spécifiquement par le noyau et les utilitaires CLIP (cf. [CLIP_1201]).*

***Remarque 5 : Montages partagés entre contextes et* inotify**

De manière similaire, les appels systèmes inotify *(cf.* inotify(7)*) permettent de créer des canaux de communications entre contextes partageant un montage (en signalant par exemple les ouvertures en lecture). L'option de montage* 'nolock' *décrite dans [CLIP_1201] bloque aussi ces canaux cachés, en interdisant l'ajout d'une veille* inotify *sur un fichier appartenant à un montage portant cette option. Cependant, la virtualisation des fonctionnalités* inotify *en fonction du contexte vserver pourrait offrir une solution plus satisfaisante.*

1.3.3 Mécanisme de *copy-on-write* pour les fichiers

Le patch *Vserver* fournit aussi une alternative plus souple au partage d'arborescences à l'aide de montages *bind* en lecture seule. Un mécanisme de *copy-on-write* spécifique permet d'exposer un même fichier en lecture-écriture dans plusieurs contextes, en créant une copie privée de l'*inode* associé lors de la première écriture sur le fichier au sein d'un contexte donné. Cette fonctionnalité, qui n'est pas mise en œuvre à ce stade dans CLIP, repose sur une utilisation originale de l'attribut d'*inode* *S_IMMUTABLE*, supporté par les principaux systèmes de fichiers Linux, complété par un deuxième attribut, *S_IUNLINK*, spécifique à *vserver*. L'attribut *S_IUNLINK* permet spécifiquement un appel *unlink()* sur un fichier marqué comme *IMMUTABLE*, tous les autres accès en écriture demeurant par ailleurs interdits par *S_IMMUTABLE*. Lors d'une tentative d'écriture sur un *inode* portant ces deux drapeaux, une copie de ce dernier est créée et liée au fichier par lequel l'accès a été réalisé, à la place de l'*inode* d'origine. Il est ainsi possible d'exposer des liens durs vers un tel *inode* dans les arborescences de plusieurs contextes, en garantissant la possibilité d'écritures dans chaque contexte sans pour autant nécessiter de dédoubler les *inodes a priori*. Les attributs *S_IUNLINK* et *S_IMMUTABLE* peuvent être manipulés à l'aide de l'appel système *sys_vserver* (par exemple avec l'utilitaire *vsattr*), ou plus classiquement par des *ioctl IOC_SETFLAGS* (spécifiques à chaque type de système de fichiers) sur les fichiers concernés (par exemple avec l'utilitaire *chattr*). Le paquetage *util-vserver* fournit de plus les outils permettant une réunification des *inodes* initialement partagés par liens durs mais devenus disjoints suite à des écritures. Cette réunification est possible dès lors que les contenus des *inodes* sont à nouveau égaux, par exemple suite à des écritures similaires mais non synchronisées dans différents contextes.

1.3.4 Barrière *chroot*

Le patch Vserver introduit un nouvel attribut de fichier *S_BARRIER* (pour les systèmes de fichiers *ext2*, *ext3*, *xfs*, *reiserfs-3* et *jfs*). La présence de cet attribut sur un répertoire interdit tout accès à ce dernier par un contexte autre que ADMIN. Cet attribut a vocation à être placé sur le répertoire commun à toutes les instances *vserver* (typiquement */versers*, les instances étant contenues individuellement dans */vservers/<nom de l'instance>*). Il permet de contrer une tentative de sortie du *chroot* par le *root* de la cage (par exemple suite à un deuxième appel *chroot* à l'intérieur de la cage). On notera cependant que, d'une part, cette protection n'est que partielle (n'étant efficace que contre une sortie du chroot par appels successifs à *chdir("..")* depuis un répertoire situé à l'intérieur de la cage), et est d'autre part redondante dès lors que les instances *vserver* sont configurées de manière à utiliser les fonctionnalités de cloisonnement de *namespace VFS*. En effet, le remontage de */vservers/<nom de l'instance>* comme racine du *namespace* d'une instance interdit en soi un *chdir("..")* à la racine de la cage. Cette fonctionnalité est à ce stade employée dans CLIP à des fins de défense en profondeur, l'attribut barrière étant positionné sur la racine du système, ainsi que sur */vservers* lorsque ce répertoire est présent. Ce mécanisme pourrait encore être étendu à d'autres répertoires du socle CLIP.

1.4 Interfaces utilisateur

Les différents mécanismes introduits par le patch *vserver* sont configurables et analysables depuis des processus utilisateurs à travers un certain nombre d'interfaces.

1.4.1 Appel système *sys_vserver*

Les appels systèmes constituent la principale interface de configuration des différentes fonctionnalités *vserver*. Un unique numéro d'appel système, 273, est réservé dans le noyau Linux officiel (sans patch *vserver*) à une utilisation par le projet *vserver*. Le patch *vserver* définit un appel *sys_vserver* associé à ce numéro, appel qui est lui-même multiplexé en fonction de son premier paramètre entre les différentes commandes *vserver*. Ces commandes permettent aussi bien la supervision (lecture d'informations de version et de configuration, ou encore de statistiques de fonctionnement) que l'administration (création et paramétrage de contextes, ajout ou modification d'attributs de fichiers).

La capacité "POSIX" (spécifique à *vserver*) *CAP_CONTEXT* est systématiquement exigée lors d'un appel à *sys_vserver*. Des capacités supplémentaires, *CAP_LINUX_IMMUTABLE*, *CAP_SYS_ADMIN*, *CAP_NET_ADMIN* ou *CAP_SYS_RESSOURCE*, peuvent être exigées pour le traitement de certains types de commandes. A ces exigences viennent aussi s'ajouter selon les commandes des contraintes sur le contexte dont est issu l'appel , ainsi qu'éventuellement sur le contexte modifié par l'appel. Ces différents éléments sont résumés dans l'Annexe A.

1.4.2 Interfaces *proc*

Vserver fournit par ailleurs un certain nombre d'informations de supervision dans */proc*. Deux répertoires globaux, */proc/virtual* et */proc/virtnet*, regroupent les informations générales sur les contextes de sécurité et réseau, respectivement. Chacun de ces répertoires contient un fichier global *info,* ainsi qu'un sous-répertoire par contexte définit dans le noyau. La lecture du fichier *info* global donne accès aux principales informations de configuration du patch *vserver* : numéros de version et d'appel système, masque d'options de configuration. Chaque sous-répertoire contient à son tour un ensemble de fichiers donnant accès en lecture aux informations statiques concernant le contexte (paramètres de configuration : masques de capacités et d'options), mais aussi aux données dynamiques correspondantes : nombre de tâches et de sockets, paquets émis et reçus, utilisation des limites et biais de virtualisation. Ces informations globales sont complétées par deux fichiers par processus, */proc/<tgid>/vinfo* et */proc/<tgid>/ninfo*, rappelant les paramètres statiques des contextes de sécurité et réseau auxquels appartient la tâche de *tgid* <*tgid*>.

La visibilité des répertoires globaux et des fichiers qu'ils contiennent est contrôlée par les attributs *vx_flags* des fichiers du *procfs* (cf. 1.3.1). Ces répertoires et fichiers sont normalement accessibles uniquement par le contexte ADMIN. En revanche, la visibilité des fichiers */proc/<tgid>/{v,n}info* est contrôlée indépendamment pour chaque contexte, par la présence ou non du drapeau *VXF_HIDE_VINFO* dans le bitmask d'options du contexte de sécurité. On notera que l'activation de ce drapeau masque uniquement le contenu de ces fichiers, et non leur présence (sauf dans une adaptation spécifique à CLIP, cf. 2.2).

1.4.3 Attributs des fichiers

Bien que cela ne constitue pas un mode de configuration classique, on notera que certains paramètres utilisés par les fonctions *vserver* sont représentés dans des champs standards des *inodes* des différents systèmes de fichiers du système, plus spécifiquement leurs *uid*, *gid* et attributs. Ces paramètres sont normalement ajustés à l'aide de commandes spécifiques de l'appel système *vserver*, mais restent aussi modifiables par les fonctions standard du noyau Linux, par exemple par *ioctl()* pour modifier les attributs d'un *inode*.

1.4.4 Journalisation

Lorsque l'option de configuration *CONFIG_VSERVER_WARN* a été activée lors de la compilation du noyau (ce qui est le cas des noyaux CLIP), *vserver* journalise à l'aide de l'interface standard *printk* un certain nombre d'événements anormaux :

- Tentative d'accéder à des fichiers masqués du */proc*.
- Commande non reconnue dans l'appel système *sys_vserver*.
- Anomalie dans le changement de *child_reaper* lorsque le processus "*init*" d'un contexte se termine.
- Accès bloqué à un fichier du fait de la barrière *chroot* (cf. 1.3.4) ou de la labellisation d'*inode*.
- Accès non autorisé aux attributs d'un fichier du *procfs* depuis un contexte non ADMIN.
- Tentative de créer un *thread* noyau depuis un contexte non ADMIN.

Les entrées correspondantes dans le journal noyau sont toutes précédées du mot-clé "*vxW:*", et créées avec le niveau de priorité *KERN_WARNING*. Le cas échéant, le *xid* du contexte d'origine de l'anomalie est inclus dans l'entrée.

2 Mise en oeuvre dans un système CLIP

2.1 Principe des "cages"

Les cages, qui forment l'entité de base du cloisonnement lourd mis en place dans CLIP, sont réalisées à l'aide de doubles contextes *vserver*, réseau et de sécurité, généralement complétés du contexte *veriexec* correspondant (cf. [CLIP_1201]). Ces cages disposent d'adresses propres (normalement une seule), qui correspondent ou non selon les cages à des adresses configurées sur la ou les interfaces réseau du système.

2.1.1 Capacités et options des contextes

Les capacités et options attribuées aux contextes *vserver* varient légèrement d'une cage à l'autre, mais reposent néanmoins sur un substrat commun :

- Le masque maximal de capacités POSIX est laissé à sa valeur initiale par défaut, c'est-à-dire au masque global *cap-bset* du système lors de la création du contexte. Sous CLIP, ce masque comporte toutes les capacités sauf (voir [CLIP_1204] pour une définition de ces capacités):
  - CAP_SETPCAP
  - CAP_LINUX_IMMUTABLE
  - CAP_SYS_RAWIO
  - CAP_SYS_MODULE
- Le masque de capacités POSIX utilisables comporte en général un sous-ensemble de capacités POSIX dont la portée est limitée au contexte. Ces capacités sont celles associées aux privilèges *userland* du super-utilisateur, par exemple le contournement des droits UNIX sur les fichiers, le changement d'identité ou l'envoi de signaux à n'importe quel processus (du même contexte), mais pas aux privilèges spécifiquement "noyau", qui permettraient de modifier des ressources intrinsèquement partagées du système, comme la configuration réseau par exemple. Ces capacités "*userland*" sont les suivantes :
  - CAP_CHOWN
  - CAP_DAC_OVERRIDE
  - CAP_DAC_READ_SEARCH
  - CAP_FSETID
  - CAP_FOWNER
  - CAP_KILL
  - CAP_SETGID
  - CAP_SETUID

Les capacités *CAP_SYS_CHROOT* et *CAP_NET_BIND_SERVICE* ne sont pas attribuées automatiquement à tous les contextes, mais peuvent l'être à ceux qui en ont effectivement besoin, sans que cela n'impacte de manière significative la sécurité du cloisonnement. On notera en revanche que la cage *X11* (cf. [CLIP_1304]) se voit à titre exceptionnel attribuer des capacités plus "lourdes" : *CAP_SYS_TTY_CONFIG* et *CAP_SYS_BOOT*.

- Les options du contexte de sécurité sont ajustées de manière à masquer autant d'informations que possible sur le système hôte. En revanche, les options liées à la gestion quantitative des ressources (ordonnancement en particulier) ne sont pas activées. Les drapeaux généralement sélectionnés sont :
  - INFO_INIT
  - HIDE_VINFO
  - HIDE_MOUNT
  - HIDE_NETIF
  - VIRT_CPU
  - VIRT_MEM
  - VIRT_LOAD
  - VIRT_UPTIME
- Les *rlimits* par contexte ne sont pas utilisées, pas plus que les politiques d'ordonnancement. La gestion en disponibilité des ressources ne constitue pas à ce stade un objectif de sécurité de CLIP.

2.1.2 Systèmes de fichiers

Chaque cage dispose d'une arborescence dédiée dans le système de fichier, avec sa propre racine. L'arborescence peut soit correspondre à un système de fichier dédié (cas des cages RM par exemple dans CLIP-RM), soit être construite par projection de sous-ensembles de l'arborescence principale, à l'aide de montages *bind*, en lecture seule ou non. Dans le cas de montages *bind* partagés entre plusieurs cages, l'option *noatime* est ajoutée afin d'éviter de créer des canaux auxiliaires de communication entre contextes. Chaque cage dispose d'un */dev*[7] statique, monté en lecture seule[8] depuis un répertoire du socle CLIP dédié à cette cage. Ce répertoire contient systématiquement les fichiers suivants :

- *null, full* et *zero.*
- *urandom.*
- Un lien symbolique *random* pointant vers *urandom*. Le véritable *device random* n'est pas exposé dans les cages afin de ne pas leur permettre de vider le *pool* d'entropie du noyau.
- Des liens symboliques *fd* vers */proc/self/fd* et *stdin, stdout, stderr*, vers *fd/0, fd/1, fd/2* respectivement.

[7] Ou de plusieurs répertoires */dev* construits selon le même principe, lorsque la cage est subdivisée en plusieurs vues (cf. [CLIP_1401])

[8] Les quelques *devices* auxquels ils peut être nécessaire d'accéder en écriture, comme *null*, ignorent les options de montages.

Selon les cages, ces fichiers sont complétés de :

- *ptmx*, *tty* ainsi qu'un montage de type *devpts* sur le répertoire *pts*, pour les cages ayant besoin de mettre en œuvre des pseudos terminaux. Il est rappelé que le système de fichiers *devpts* est spécifiquement labellisé par *vserver*, de manière à interdire le partage d'un pseudo-terminal entres contextes.
- *veriexec*, lorsque la cage est chargée d'installer des mises à jour.

Les attributs du système de fichiers */proc* sont ajustés de telle sorte que les seuls fichiers visibles (hors répertoires /proc/<tgid>) dans les contextes non privilégiés soient les suivants :

- */proc/acpi/battery* et ses sous-répertoires *BAT0* et *BAT1*, ainsi que leurs contenus (fichiers *alarm, state* et *info* dans chaque répertoire). Ces fichiers sont nécessaires au suivi de l'état de charge des éventuelles batteries du poste CLIP depuis la cage $USER_{clip}$.
- */proc/ac_adapter/AC/state*, nécessaire au suivi de l'état de charge des postes CLIP portables (alimentation sur batterie ou sur secteur) depuis la cage $USER_{clip}$.
- */proc/version*, qui contient le numéro de version du noyau, et qui est nécessaire au fonctionnement de nombreux applicatifs, en particulier *java*.
- */proc/stat*, qui contient un certains nombre de statistiques sur le système (nombre et charge des CPU, compteurs d'interruptions, nombre de processus, etc.), et qui est nécessaire au fonctionnement de la machine virtuelle *java*, éventuellement déployée dans les cages $USER_{clip}$ ou RM.
- */proc/meminfo*, qui contient les informations d'utilisation de la mémoire, et qui est nécessaire au fonctionnement de la machine virtuelle *java*, éventuellement déployée dans les cages $USER_{clip}$ ou RM.

Les fichiers */proc/veriexec* et */proc/devctl* sont par ailleurs visibles dans le contexte WATCH. Le système de fichiers */proc* est monté dans les cages en lecture seule[9] et avec les options *nosuid,noexec,nodev*.

2.1.3 Fonctionnalités *vserver* non exploitées sous CLIP

Un certain nombre de fonctionnalités du patch *vserver* ne sont pas exploitées dans les cages CLIP à ce stade. Il s'agit principalement :

- Des limites (*rlimits* étendues) d'accès aux ressources par contexte. Cette option pourrait à terme servir dans CLIP à protéger la disponibilité, en particulier du socle du système mais aussi des autres cages, vis-à-vis d'une cage compromise.
- Du mécanisme d'ordonnancement par *tocken bucket*, permettant de limiter l'accès d'un contexte non privilégié au temps CPU. L'ordonnancement par *tocken bucket* peut prendre une forme extrêmement stricte, dans laquelle aucun processus d'un contexte ne peut obtenir de temps CPU une fois épuisé le quota CPU du contexte, ou alternativement une forme plus souple, dans laquelle l'utilisation du CPU par un contexte affecte uniquement la priorité de ses processus. Ce mécanisme pourrait lui aussi contribuer à assurer des objectifs en disponibilité

[9] Ce qui nécessite un montage *bind* du */proc* du socle sur celui de la cage, plutôt qu'un montage direct de type */proc* dans la cage, dont les options affecteraient tous les autres montages */proc* du système.

dans CLIP. On notera que la mise en place de quotas stricts est potentiellement très impactante en termes de réactivité pour un poste client : le bureau est entièrement gelé à chaque fois que le contexte correspondant épuise son quota de temps CPU.

- Des statistiques par contexte, qui pourraient à terme être exploitées par un outil de supervision dédié.
- Des limites d'utilisation disque par contexte. Ce mécanisme pourrait être utilisé dans CLIP pour interdire la saturation de la partition de journaux par un contexte autre que la cage $AUDIT_{clip}$.
- De la gestion des quotas d'utilisation du disque par utilisateurs par contexte. Dans la mesure où des limites similaires sont automatiquement imposées dans CLIP par l'utilisation de partitions utilisateur et temporaires de taille fixe et propres à chaque session utilisateur, ce mécanisme ne présente aucune utilité apparente pour CLIP.
- Du système de *copy-on-write* pour les *inodes* décrit en 1.3.3. Il n'est à ce stade pas prévu de mettre en œuvre ce mécanisme dans CLIP.
- De la "barrière *chroot*", décrite 1.3.4, qui est effectivement utilisée dans CLIP, mais dont l'usage pourrait être étendu à l'ensemble des répertoires utilisés uniquement par le socle.

2.2 Adaptations spécifiques à CLIP du patch *vserver*

Outre les fonctionnalités complémentaires apportées par les patches *Grsecurity* et *CLIP-LSM*, un certain nombre de mécanismes spécifiques à *Vserver* sont légèrement adaptés dans CLIP, afin d'améliorer leur sécurité ou de supporter des fonctionnalités nécessaires au fonctionnement de CLIP.

2.2.1 *Sockets* UNIX et montages *bind* en lecture seule

Comme décrit en 1.3.2, l'appel système *bind()* sur les *sockets* de type *PF_UNIX* ne respecte pas l'option lecture-seule ajoutée par *vserver* aux montages de type *bind*. Dans la mesure ou une telle exception risquerait de porter atteinte à certains objectifs de sécurité de CLIP (en particulier, disponibilité du service de mise à jour), elle est corrigée par un patch spécifique, qui interdit la création d'un fichier par *bind()* d'une *socket PF_UNIX* sur un montage *VFS* en lecture-seule.

2.2.2 Visibilité des fichiers */proc/<tgid>/vinfo* et */proc/<tgid>/ninfo*

Un patch spécifique à CLIP vient masquer entièrement ces fichiers, plutôt que leur seul contenu, au sein des contextes portant le drapeau *HIDE_VINFO*.

2.2.3 Envoi de signaux depuis le contexte ADMIN

L'envoi de signaux est normalement impossible entre contextes *vserver* différents, même lorsque le contexte émetteur est ADMIN. Au sein d'un système CLIP, l'envoi d'un signal depuis un processus du contexte ADMIN vers un processus d'un autre contexte est permis, sous réserve que les autres contraintes pour l'envoi de signaux (imposées par le LSM CLIP, cf. [CLIP_1201]) soient satisfaites. Cette exception est en particulier nécessaire à la mise en œuvre du démon d'ouverture de session graphique *XDM*, qui est exécuté dans le contexte ADMIN mais doit gérer des processus fils enfermés

dans deux cages distinctes (*USER*$_{clip}$ et *X11*, cf. [CLIP_1304]). L'envoi de signaux dans le sens inverse, depuis un contexte non privilégié vers le contexte ADMIN, peut être autorisé pour certains signaux (*SIGUSR*) par le LSM CLIP, lorsque aussi bien l'émetteur que le récepteur disposent de privilèges spécifiques. Cet envoi de signaux en retour est lui-aussi nécessaire à la mise en œuvre de *XDM*, afin de permettre la synchronisation entre ce dernier et le serveur graphique.

Par ailleurs, une modification est apportée au traitement des appels *kill(-1, ...)*, de telle sorte que cet appel, lorsqu'il est réalisé par le processus *init (pid == 1)* du système, entraîne l'envoi d'un signal à tous les processus du système, et non plus seulement à ceux du contexte ADMIN. Cette modification, couplée à une adaptation de l'exécutable *init*, permettrait la mise en œuvre du privilège CLSM *CLSM_PRIV_IMMORTAL*, pour protéger la disponibilité de certains démons comme le démon de collecte de journaux.

2.2.4 Contrôle des politiques de sécurité IPsec par contexte

Les politiques de sécurité IPsec (SP) sont principalement associées aux cages d'un système CLIP par l'intermédiaire des adresses autorisées des contextes réseaux correspondants à ces cages (cf. [CLIP_1501]). Cependant, une modification spécifique à CLIP du noyau, apportée par le *patch 2002_vserver_nx_nosp.patch* du module *clip-patches*, apporte une protection supplémentaire de cette association entre cages et SP. Ce *patch* définit un drapeau de contexte réseau supplémentaire, *NXF_NO_SP*, qui autorise un contexte réseau non-ADMIN à émettre et recevoir des flux réseau non locaux auxquels n'est associée aucune politique de sécurité. Il modifie la pile IP de telle sorte que la fonction de recherche de politique de sécurité IPsec (*net/xfrm_policy.c:__xfrm_lookup()*) renvoie une erreur lorsque la *socket* associée à une opération appartient à un contexte réseau non-ADMIN ne disposant pas du drapeau *NXF_NO_SP*, et qu'aucune politique de sécurité n'est associée à l'opération. Une telle erreur fait échouer l'envoi ou la réception ayant déclenché la recherche de politique, et est journalisée comme un avertissement *vserver* (cf. 1.4.4). Ainsi, une cage *vserver* d'un système CLIP ne peut avoir accès au réseau sans contrôle par une politique de sécurité IPsec que si elle y est spécifiquement autorisée par sa configuration. Ces vérifications s'appliquent uniquement aux flux non locaux, et en aucun cas à ceux passant par la boucle locale réseau.

On notera qu'à ce stade, une politique de sécurité quelconque, y compris passante en clair, suffit à satisfaire les conditions d'autorisation des flux en l'absence de *NXF_NO_SP*. Ainsi, ce mécanisme ne garantit pas absolument le chiffrement des flux issus et à destination des cages correspondantes, mais assure uniquement la médiation de ces flux par la SPD. Il complète ainsi à plus bas niveau le contrôle des SP réalisé à l'aide du filtrage *netfilter* (cf. [CLIP_1501]), et protège essentiellement le système d'une erreur ou *race-condition* dans le démon de gestion des SP (typiquement *spmd*) qui laisserait le système temporairement ou définitivement dépourvu des SP indispensables à la sécurité de ses communications réseau.

3 Bibliothèque *libclipvserver*

La bibliothèque *libclipvserver* (paquetage *clip-libs/clip-libvserver*) est un développement spécifique[10] au projet CLIP. Elle offre une interface simple pour un sous-ensemble de commandes *sys_vserver*, permettant essentiellement la création de cages et la migration de tâches vers des cages existantes. A ce stade, la bibliothèque utilise les commandes *sys_vserver* suivantes (voir l'annexe A pour une liste exhaustive des commandes supportées par l'appel) :

- *ctx_create*
- *ctx_migrate*
- *enter_space*
- *set_bcaps*
- *set_ccaps*
- *set_cflags*
- *net_create*
- *net_migrate*
- *net_add*
- *set_nflags*
- *set_iattr*

Les fonctions exportées par la bibliothèque sont déclarées dans un fichier d'en-tête unique, *clip/clip-vserver.h*. Ces fonctions sont décrites ci-après. Toutes retournent une valeur nulle en cas de succès, et –1 en cas d'échec, auquel cas un message d'erreur est affiché sur la sortie d'erreur standard de l'appelant, et la variable *errno* est positionnée à une valeur significative.

clip_new_namespace()

Cette fonction créée un nouveau jeu d'espaces de nommage (*VFS, UTS* et *IPC*) par un appel *clone()*. Seul le fils créé par cet appel (utilisant les nouveaux espaces de nommages) retourne de la fonction, tandis que le père se met en attente et se termine en même temps que le fils. On notera que cette fonction ne prend pas en charge le montage de la racine ou l'appel *chroot* nécessaires, après création d'un nouveau *namespace VFS*, au cloisonnement dans une sous-arborescence du système de fichier : au retour de la fonction, l'espace de nommage *VFS* conserve la même racine que celui de l'appelant.

clip_new_context(*xid, bcaps, ccaps, cflags*)

Cette fonction crée un nouveau contexte de sécurité d'identifiant *xid*, et lui attribue les masques de

[10] Cette bibliothèque offre une abstraction des appels systèmes *vserver* comparable à celle fournie par la bibliothèque *libvserver* du paquetage *sys-cluster/util-vserver*. Cependant, *libclipvserver* est sensiblement plus simple, ne couvrant qu'un sous-ensemble limité de commandes *vserver*, et ce pour une seule version du patch *vserver*, alors que l'essentiel de la complexité de *libvserver* est due à des couches de compatibilité, permettant l'utilisation de nombreuses version différentes du patch noyau. Par ailleurs, *libclipvserver* offre aussi une interface de plus haut niveau, permettant la création d'une cage complète par l'appel d'une unique fonction.

capacités POSIX utilisables, de capacités de contextes et d'options respectivement représentés par *bcaps, ccaps* et *cflags*. La fonction retourne immédiatement une erreur si un contexte de sécurité d'identifiant *xid* existe déjà. Au retour sans erreur de la fonction, le processus appelant est enfermé dans ce nouveau contexte. On notera que, quelle que soit la valeur de *cflags*, les drapeaux *STATE_SETUP* et *STATE_INIT* sont systématiquement mis à zéro. Le nouveau contexte est donc verrouillé au retour de la fonction, et aucune modification de ses masques n'est plus possible.

clip_new_net_context(*nid, addr, mask, nflags*)

Cette fonction crée un nouveau contexte réseau d'identifiant *nid*, et lui attribue le masque d'options *nflags*, et l'unique adresse (autorisée et de *broadcast*) *addr*, avec le masque de sous-réseau *mask*. La fonction retourne immédiatement une erreur si un contexte réseau d'identifiant *nid* existe déjà. Au retour sans erreur de la fonction, l'appelant est "enfermé" dans le nouveau contexte. Le drapeau *STATE_SETUP* du nouveau contexte est systématiquement mis à zéro. Il est rappelé (cf. 1.2) que cet "enfermement" demeure aisément contournable (du moins par *root*) tant que l'appelant ne s'enferme pas dans un contexte de sécurité.

clip_new_net_context_multi(*nid, addrs, masks, nflags*)

Cette fonction est similaire à *clip_new_net_context()*, à ceci près qu'elle prend en argument non pas une seule adresse, mais un tableau d'au maximum quatre adresses (et autant de masques). Chaque tableau doit être terminé par un champ nul s'il compte moins de quatre éléments. La première adresse doit être non-nulle, et est définie comme adresse *broadcast* du contexte. Si plus de quatre adresses sont passées en argument, seules les quatre premières sont prises en compte.

clip_enter_namespace(*xid*)

Fait migrer le processus appelant vers les espaces de nommage du contexte de sécurité d'identifiant *xid*, s'il existe, sans pour autant rejoindre ce même contexte. L'appelant acquiert donc un accès aux espaces de nommage propre à ce contexte, sans perdre les privilèges liés à son appartenance au contexte ADMIN (il n'est en particulier pas soumis au masque de capacités POSIX utilisables du contexte).

clip_jailself(*xid, bcaps, cflags, addr, path*)

Cette fonction crée une cage élémentaire, d'identifiant *xid*, de masques de capacités utilisables *bcaps* et d'options de contexte de sécurité *cflags*, confinée dans le système de fichier à l'arborescence située au-dessus de *path*, et limitée à l'adresse (unique) *addr*. Des valeurs par défaut sont utilisées pour les autres paramètres de la cage : aucune capacité de contexte, pas d'options de contexte réseau, et un masque de sous-réseau égal à *255.255.255.0.* Les opérations réalisées par cette fonction sont dans l'ordre :

- création et paramétrage d'un nouveau contexte réseau (équivalent à un appel à *clip_new_net_context()*)
- création d'un nouveau jeu d'espaces de nommage (équivalent à un appel à *clip_new_namespace()*)
- confinement du nouvel espace de nommage *VFS* à la racine *path*, par montage *bind* récursif de *path* sur */*, puis *chroot* du processus appelant dans *path*.

- création et paramétrage d'un contexte de sécurité (équivalent à un appel à *clip_new_context()*)

L'ensemble de ces opérations est détaillé dans la Figure 1.

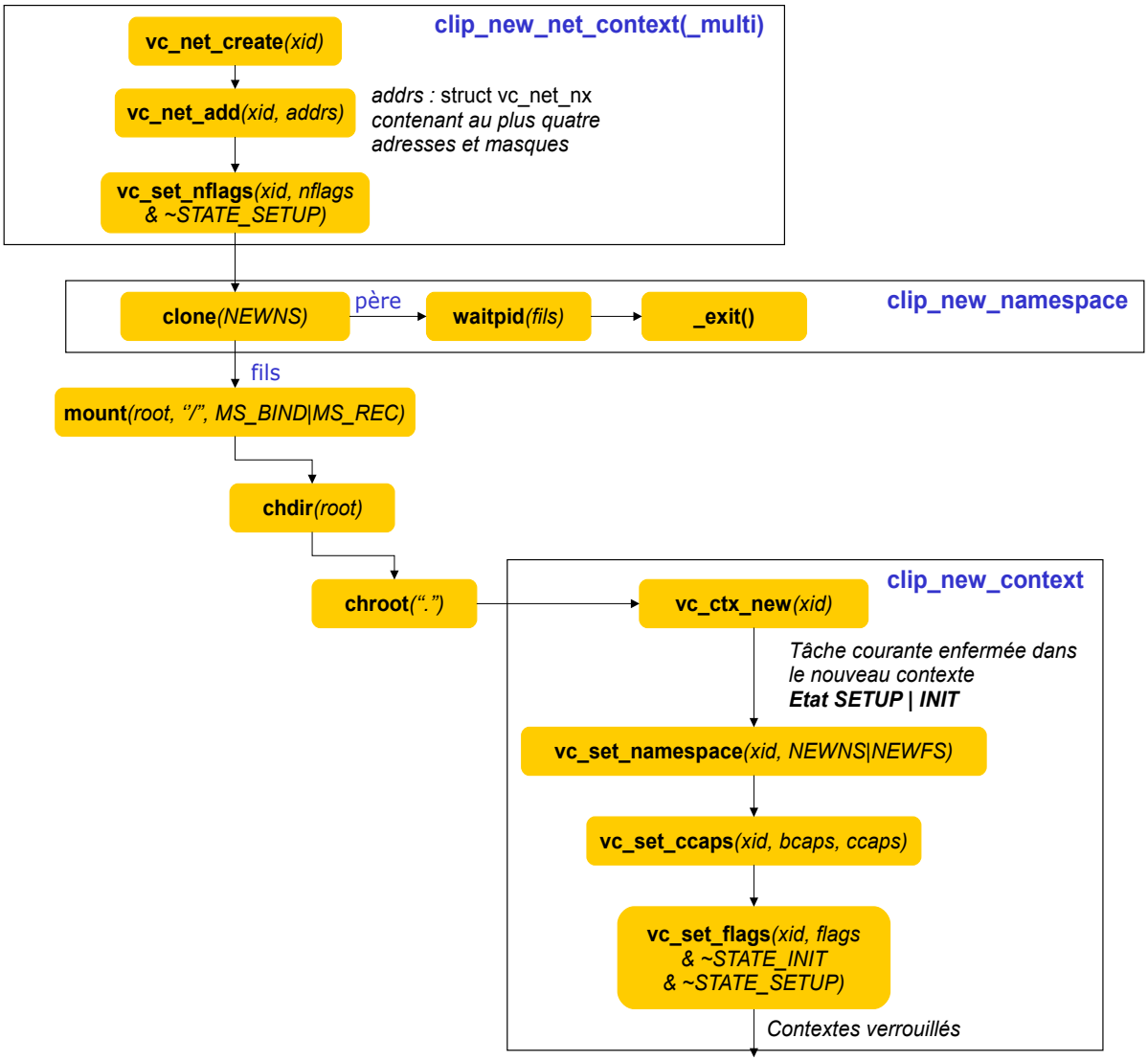

Figure 1: Création et paramétrage d'une cage vserver par libclipvserver.

Les fonctions individuelles de la *libclipvserver*, *clip_new_net_context()*, *clip_new_namespace()* et *clip_new_context()*, sont représentées par des encadrés. L'ensemble correspond à un appel *clip_jailself()*.

On notera qu'une erreur intermédiaire dans ces traitements peut laisser l'appelant dans une cage incomplète au retour de l'appel : contexte réseau uniquement, ou bien contexte réseau et espaces de nommages (confinés ou non). Le paramétrage de la cage est délibérément simpliste, afin de ne pas compliquer l'API. Si besoin est, une cage plus complexe peut être construite par des appels aux fonctions plus élémentaires de la bibliothèque, comme cela est fait par exemple dans *vsctl* (cf. 4.1).

clip_enter_context(*xid*)

Enferme le processus appelant dans une cage (complète, et non juste le contexte de sécurité) existante, en migrant successivement vers les espaces de nommages associés au contexte de sécurité d'identifiant *xid*, puis vers le contexte réseau d'identifiant *xid*, puis enfin vers le contexte de sécurité d'identifiant *xid*. L'appel échoue si l'un des deux contextes (de sécurité et réseau) d'identifiant *xid* n'existe pas. On notera que, tout comme pour *clip_jailself(),* l'appelant peut se retrouver dans une cage incomplète lorsque la fonction rend une erreur.

clip_vlogin()

Cette fonction (fortement inspirée de l'utilitaire *vlogin* du paquetage *sys-cluster/util-vserver*) assure la mise en place d'un *proxy* de terminal de contrôle pour l'appelant. Ce *proxy* ne met en œuvre aucune fonctionnalité spécifique à *vserver*, mais complète la sécurisation des cages en leur permettant d'utiliser un terminal dédié comme terminal de contrôle, plutôt que le terminal du créateur de la cage (ou d'un processus créé hors de la cage qui s'est ensuite enfermé dans celle-ci). La fonction *clip_vlogin* réalise les opérations suivantes :

- Passage du terminal de contrôle en mode *raw*.
- Ouverture d'un pseudo-terminal maître-esclave UNIX98.
- Création d'un processus fils par *fork()*.
- Le processus fils ferme le terminal de contrôle hérité du père, ainsi que la partie maître du nouveau pseudo-terminal, puis prend la partie esclave de ce dernier comme terminal de contrôle, avant de retourner de la fonction *clip_vlogin*.
- Le processus père ne retourne jamais de la fonction. Il ferme le descripteur esclave du nouveau pseudo-terminal, puis se met en attente sur le descripteur maître de ce dernier, son terminal de contrôle et les signaux *SIGWINCH, SIGCHLD* et *SIGTERM*. Il relaie les lectures et écritures depuis et vers le pseudo-terminal esclave vers son propre terminal de contrôle, et synchronise les géométries des deux terminaux si besoin est (*SIGWINCH*). Il termine son fils avant de se terminer lui-même sur réception de *SIGTERM*, et se termine automatiquement en cas de terminaison de son fils *(SIGCHLD).* Dans les deux cas, le paramétrage d'origine du terminal de contrôle est rétabli avant la terminaison.

La fonction *clip_vlogin* peut être appelée avant la création d'une cage (création réalisée alors par le processus fils, qui seul retourne de l'appel), ou l'entrée dans une cage existante, afin d'éviter de conserver dans la cage un descripteur de fichier ouvert correspondant à un terminal utilisé en dehors de la cage. Ceci est particulièrement utile lorsque la commande lancée dans la cage est interactive (*shell*), et plus encore lorsque cette commande est lancée depuis une console du système (avec un */dev/ttyX* comme terminal de contrôle). Elle est en revanche largement superflue lorsque le processus lancé dans la cage est un démon, qui se détache immédiatement de tout terminal de contrôle (sauf à supposer le démon piégé et malicieux).

clip_set_iattr(*file, set, unset*)

Cette fonction ajoute à l'*inode* lié à au fichier *file* les attributs *vserver* spécifiés dans le masque *set*, et retire ceux spécifiés dans le masque *unset*. Une erreur est renvoyée lorsqu'un attribut est défini dans ces deux masques à la fois.

4 Utilitaires du paquetage du paquetage *app-clip/vsctl*

Le paquetage *app-clip/vsctl* regroupe un certain nombre d'utilitaires spécifiques à CLIP, permettant la manipulation de cages *vserver*. Il reprend un nombre limité de fonctionnalités du paquetage *sys-cluster/util-vserver*, sans la complexité de ce dernier. La gestion des appels systèmes *vserver* est réalisée à travers l'interface *libclipvserver* détaillée en section 3. Le principal utilitaire du paquetage est *vsctl*, un outil comparable à la commande *vserver* de *util-vserver*, permettant la création et la suppression de cages *vserver*, ainsi que l'entrée dans des cages existantes. Le paquetage est complété par *nsmount* et *vsattr*, deux utilitaires plus spécifiques permettant respectivement de réaliser des montages *VFS* dans un *namespace* particulier et de modifier les attributs de fichiers *vserver*.

4.1 Commandes *vsctl*

Une ligne de commande *vsctl* prend la forme suivante :

```
vsctl [options] <cage> <commande>
```

Les options supportées sont décrites en 4.3. <*cage*> est le nom de la cage, qui détermine le répertoire */etc/jails/*<*cage*> utilisé pour la configuration de la cage. La structure de ce répertoire de configuration est détaillée en 4.2. <*commande*> est l'une des commandes décrites ci-dessous.

start

Démarrer la cage <*cage*>. La séquence d'opérations réalisées pour créer et configurer la cage est similaire à celle employée dans la fonction *clip_jailself* de la *libclipvserver* (cf. 3), à ceci près que plusieurs adresses et masques de sous-réseau sont supportés pour le contexte réseau, et que les capacités de contexte sont configurables. De plus, une étape supplémentaire de montages et démontages *VFS* est intercalée entre la création des espaces de nommage de la cage et celle de son contexte de sécurité, ce qui permet de créer une arborescence de fichiers complexe et réservée à la cage (sans "polluer" le *namespace VFS* du contexte ADMIN), mais aussi de nettoyer le *namespace* de la cage de ceux des montages hérités du contexte ADMIN dont cette cage n'a pas besoin. A l'issue de la configuration, un exécutable est lancé dans la cage. Cet exécutable est défini statiquement dans la configuration de la cage, et exécuté sous l'identité *root* sans argument et avec un environnement réduit à la variable *PATH* (définie à *bin:/sbin:/usr/bin:/usr/sbin*). L'ensemble de ces opérations est résumé dans la Figure 2.

stop

Tente d'arrêter une cage en envoyant le signal *SIGTERM,* puis une seconde après *SIGKILL*, à tous les processus de cette cage. L'implémentation de cette commande est similaire à une commande *enter* (voir ci-dessous) suivie d'appels *kill –TERM –1* et *kill –KILL –1*.

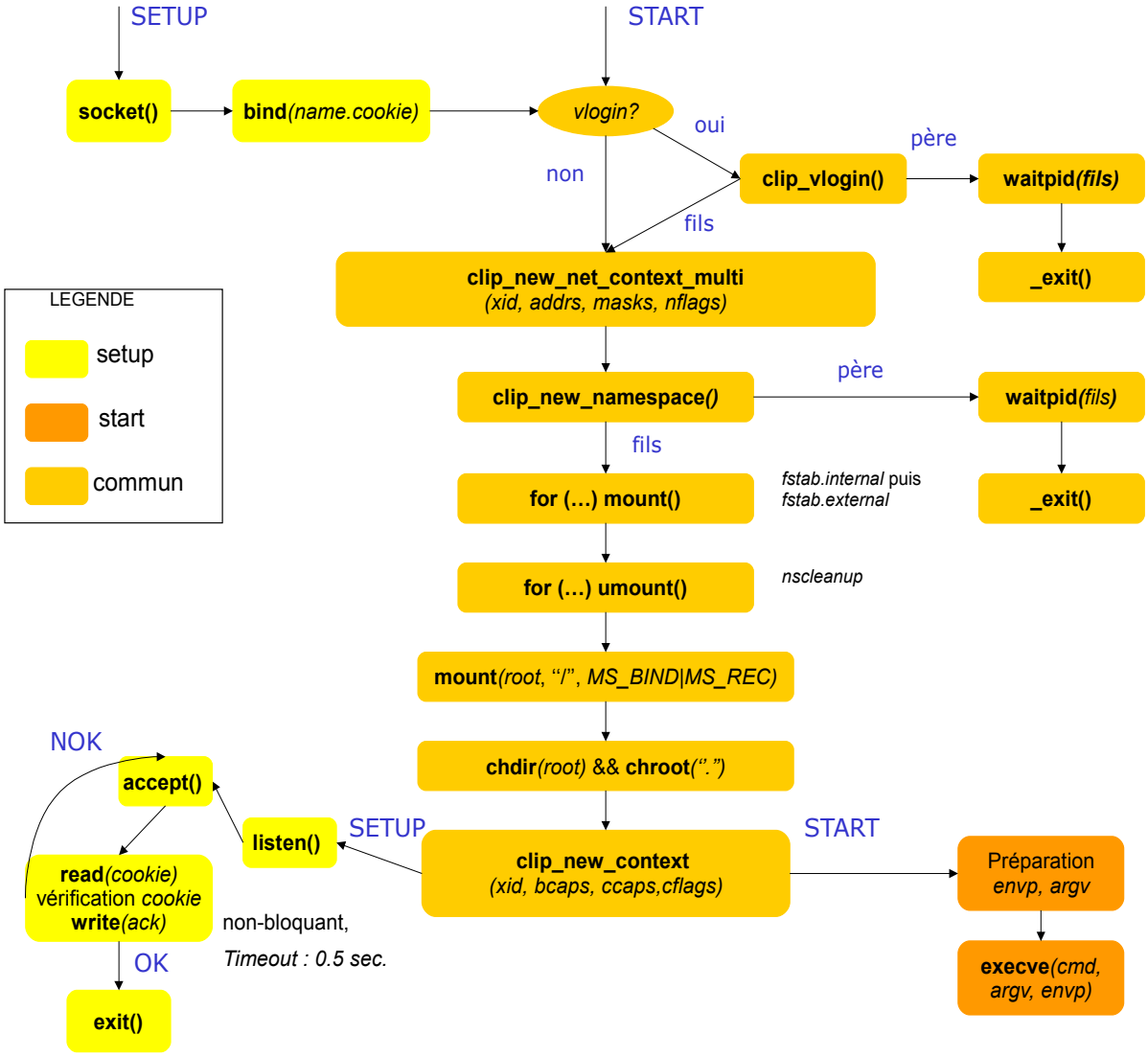

Figure 2: Traitement d'une commande start ou setup par vsctl.

enter

Entrer dans une cage existante et y exécuter une commande arbitraire. La cage doit avoir été créée auparavant par une commande *start* ou *setup*, et ne pas s'être terminée par terminaison de tous ses processus. Cette commande offre plus de souplesse que *start* quant à la commande qui peut être lancée. Les différentes options de la ligne de commande *vsctl* (cf. 4.3) permettent notamment de préciser l'exécutable invoqué dans la cage (l'exécutable invoqué par défaut étant celui défini dans le ficher *cmd* du répertoire de configuration de la cage), ses arguments, son environnement, l'identité sous laquelle il est lancé. Il est de plus possible de spécifier un chemin relatif à la racine de la cage, dans lequel

l'exécutable sera enfermé par appel *chroot*[11] après l'entrée dans la cage elle-même. Ce double cloisonnement, par *chroot* à l'intérieur d'une cage *vserver*, correspond au mécanisme de vues mis en œuvre principalement au sein des cages RM.

setup

Configurer une cage sans y lancer d'exécutable. Cette commande est traitée de manière largement similaire à *start* (cf. Figure 2), à deux différences près :

- Une *socket* UNIX est créée dans */var/run* avant le lancement de la procédure de configuration de la cage.
- Une fois la cage configurée, *setup* n'invoque pas un exécutable de cette dernière, mais se met en attente sur la *socket* précédemment créée. Sur réception d'une connexion valide à cette *socket*, le processus se termine.

L'attente sur une *socket* UNIX permet de "maintenir en vie" les contextes réseau et de sécurité nouvellement créés, sans pour autant y lancer de processus autre que *vsctl*. Le cadre d'emploi typique consiste à maintenir la cage active à l'aide de *vsctl setup* le temps de lancer plusieurs commandes de configuration spécifiques à l'aide de *vsctl enter*, puis de démarrer un démon maître pour la cage à l'aide là encore de *vsctl enter*, avant de terminer le processus *setup*.

L'accès à la *socket* est contrôlé par un *cookie*, chaîne de caractères arbitraires de vingt octets qui est lue au démarrage de *vsctl* dans la variable d'environnement *VSCTL_MAGIC_COOKIE*. La commande renvoie immédiatement une erreur si cette variable n'est pas définie. Les quatre premiers octets de ce *cookie* sont utilisés pour construire le nom de la socket *vsctl*, qui est de la forme :

```
/var/run/vsctl.<nom cage>.HEX(cookie[0...3])
```

avec HEX la fonction de représentation hexadécimale octet par octet (deux caractères par octet). Lors d'une tentative ultérieure de connexion à cette *socket*, l'initiateur de la connexion doit écrire l'intégralité du *cookie* sur celle-ci pour que la connexion soit considérée comme valide. Après lecture du *cookie*, *vsctl setup* écrit un unique caractère, 'Y', sur la *socket* afin de signaler l'acceptation de la connexion avant de se terminer. Dans le cas contraire (*cookie* erroné, ou impossibilité de lire un *cookie* complet dans un délai d'une demi-seconde), 'N' est écrit sur la *socket* et *vsctl setup* poursuit son attente.

endsetup

Terminer un processus *vsctl setup*, en se connectant sur la *socket* UNIX d'écoute de ce dernier et en y écrivant le *cookie* lu dans la variable d'environnement *VSCTL_MAGIC_COOKIE*. En cas de succès de la connexion (le processus *setup* ayant renvoyé le caractère 'Y'), cette commande assure aussi la suppression du fichier lié à la *socket* de connexion, suppression qui ne peut pas être réalisée par le processus *setup* dans la mesure où ce fichier n'est généralement pas accessible dans l'arborescence de la cage.

[11] L'appel à *chroot* étant réalisé par *vsctl*, qui dispose des capacités nécessaires, avant de lancer l'exécutable, le chemin de ce dernier doit être donné relativement à la racine du *chroot*.

cookie

Générer un *cookie* et l'afficher sur la sortie standard. Le cookie est obtenu par lecture de vingt octets sur */dev/urandom*.

La séquence type d'opérations associée à une configuration par *setup* / *enter* est ainsi (en faisant abstraction du traitement des cas d'erreur) :

```
cookie=`vsctl <cage> cookie`
VSCTL_MAGIC_COOKIE="$cookie" vsctl <cage> setup
vsctl <cage> enter -- commande1   # configuration
vsctl <cage> enter -- commande2   # démon maître
VSCTL_MAGIC_COOKIE="$cookie" vsctl <cage> endsetup
```

mount

Réaliser uniquement les montages spécifiés par la configuration de la cage, dans le *namespace VFS* courant (contexte ADMIN). Les montages réalisés sont ceux définis dans les fichiers *fstab.external* et *fstab.internal* du répertoire de configuration (cf. 4.2), dans l'ordre de lecture. Cette commande est destinée au test de l'utilitaire en phase de développement, et n'est supportée que si *vsctl* a été compilé avec l'option de configuration *--enable-testing*.

umount

Démonter les montages réalisés par la commande *mount*. Les opérations de démontage sont réalisées dans le *namespace VFS* courant, dans l'ordre inverse de leur montage. Cette commande constitue le pendant de la commande *mount* ci-dessus, et n'est supportée que si *vsctl* a été compilé avec l'option de configuration *--enable-testing*.

4.2 Fichiers de configuration *vsctl*

La configuration d'une cage est définie dans un ensemble de fichiers contenus dans le répertoire */etc/jails/<nom de la cage>*, où *<nom de la cage>* correspond au premier argument de la ligne de commande *vsctl*. Certains de ces fichiers sont obligatoires, et *vsctl* refusera de configurer une cage pour laquelle un de ces fichiers n'est pas présent dans le répertoire de configuration. Les autres fichiers sont facultatifs : un avertissement sera émis en leur absence, mais *vsctl* poursuivra son exécution avec des paramètres par défaut correspondant à un fichier vide. On notera que ces contraintes ne portent que sur les opérations de création de la cage, correspondant aux commandes *start* et *setup*. Les autres commandes *vsctl* n'utilisent pas l'ensemble des fichiers (*context* étant généralement le seul fichier lu), et sont par conséquent moins contraintes.

La lecture de ces différents fichiers de configuration est entièrement réalisée sous une identité effective non privilégiée (*euid* 250 et *egid* 250, en conservant *uid* 0 et *gid* 0). En l'absence d'erreur dans la lecture, les identités effectives *root* sont réengagées avant de procéder à la configuration de la cage à partir des structures de données élaborées en mémoire au cours de la lecture.

addr (obligatoire[12])

Adresses IP du contexte réseau associé à la cage, une par ligne suivie de son masque de sous-réseau, les

[12] Sauf si une adresse est passée sur la ligne de commande par l'option *–a*.

deux adresses étant exprimées sous la forme "nombre et points" et séparées par un '/', par exemple :

```
A.B.C.D/255.255.255.0
A'.B'.C'.D'/255.255.255.0
```

Le fichier doit contenir au minimum une adresse. Si plus de quatre adresses sont définies, seules les quatre premières seront utilisées pour configurer le contexte réseau. Dans tous les cas, la première ligne du fichier correspond à l'adresse principale du contexte, aussi utilisée pour configurer son adresse *broadcast*.

bcaps

Capacités POSIX utilisables dans la cage (c'est-à-dire constituant le masque de capacités POSIX utilisables du contexte de sécurité associé), à raison d'une par ligne. Chaque capacité est désignée par son nom privé du préfixe *CAP_*, par exemple *SETUID* pour *CAP_SETUID*, ou *NET_BIND_SERVICE* pour *CAP_NET_BIND_SERVICE*.

ccaps

Capacités de contexte de sécurité de la cage, à raison d'une par ligne. Les capacités sont désignées par les mêmes mots-clés que ceux utilisés par *util-vserver*, qui sont rappelés dans le Tableau 1.

cflags

Options de contexte de sécurité de la cage, à raison d'une par ligne. Les options sont désignées par les mêmes mots-clés que ceux utilisés par *util-vserver*, qui sont rappelés dans le Tableau 2.

context (obligatoire)

Numéro de contexte (aussi bien réseau que de sécurité) de la cage, sous la forme d'un nombre en notation décimale compris entre 2 et 65534 inclus (les valeurs 0, 1 correspondent au contextes ADMIN et WATCH respectivement, et 65535 est réservé comme code d'erreur).

cmd (obligatoire[13])

Chemin absolu de l'exécutable à lancer dans la cage à l'issue du traitement d'une commande *start*. Ce fichier ne permet pas de passer des arguments ou des options à cet exécutable, pas plus que l'ajout de -- <*options*> <*arguments*> à la ligne de commande de *vsctl* (cette forme n'est valide que pour la commande *enter*). Le fichier *cmd* définit aussi l'exécutable lancé par la commande *enter* en l'absence d'une option --.

Deux approches sont possibles s'il est nécessaire de passer des arguments à l'exécutable lancé au démarrage du contexte : soit utiliser comme *cmd* un script sans arguments qui appelle l'exécutable en lui passant des options et des arguments prédéfinis, soit remplacer le *start* par une séquence *setup / enter / endsetup*, la commande *enter* réalisant l'appel de l'exécutable avec les bons paramètres.

fstab.external

Montages externes à réaliser dans le *namespace VFS* de la cage avant de la démarrer. Ces montages

[13] Sauf si la cage est lancée uniquement par une commande *vsctl setup*.

sont exprimés selon un format similaire à celui de */etc/fstab*, un montage par ligne avec la syntaxe suivante :

```
<spec> <file> <type> <options>
```

avec :

- ***spec*** : la source du montage, fichier de type bloc, répertoire, système de fichier distant ou identifiant arbitraire, selon le type de montage. Lorsque *spec* est un fichier de type bloc ou un répertoire, celui-ci est désigné par son chemin absolu par rapport à la racine du système (*namespace* standard). Les montages réseau sont réalisés dans le contexte réseau de la cage.
- ***file*** : le point de montage, sous la forme d'un chemin absolu par rapport à la racine de la cage (*namespace* de la cage, contrairement à *spec*).
- ***type*** : le type de système de fichier. Les types valides sont ceux recensés dans */proc/filesystems*, ainsi que *none* pour les montages de type *bind*.
- ***options*** : options de montages, séparées par des virgules. Les options supportées et leur formulation sont les mêmes que celles utilisées par la commande *mount(8)* standard, auxquelles s'ajoute l'option *nolock* (*MS_NOLOCK*), spécifique au noyau CLIP (cf. [CLIP_1201]).

Des commentaires peuvent être inclus dans le fichier, sur des lignes complètes (pas de commentaires partiels sur une fin de ligne) commençant par un '#'. Les montages sont réalisés dans l'ordre de leur lecture (première ligne en premier, dernière ligne en dernier).

fstab.internal

Montages internes à réaliser dans le *namespace VFS* de la cage avant de la démarrer. Ce fichier utilise la même syntaxe que le fichier *fstab.external* décrit ci-dessus, à ceci près que le champ *spec*, lorsqu'il correspond à un chemin absolu de fichier ou de répertoire, est relatif à la racine de la cage, et non du système hôte. Les montages correspondants, typiquement des montages de type *bind* entre répertoires de l'arborescence de la cage, sont réalisés avant ceux de *fstab.external*.

nflags

Options de contexte réseau de la cage, à raison d'une par ligne. Les options sont désignées par les mêmes mots-clés que ceux utilisés par *util-vserver*, qui sont rappelés dans le Tableau 3.

nscleanup

Points de montage à démonter dans le *namespace VFS* de la cage, afin d'éviter de "polluer" ce dernier par des montages superflus hérités du contexte ADMIN. Le fichier contient un point de montage par ligne, exprimé sous la forme d'un chemin absolu par rapport à la racine du système hôte. Les opérations de démontage sont réalisées dans l'ordre de lecture, après les montages spécifiés par les fichiers *fstab.external* et *fstab.internal*.

root (obligatoire)

Chemin absolu de la racine *VFS* du contexte.

4.3 Options de la ligne de commande *vsctl*

vsctl supporte plusieurs options sur sa ligne de commande. Certaines de ces options ne sont supportées que par certaines commandes *vsctl*, et sont simplement ignorées lorsqu'elles sont passées avec une commande ne les supportant pas. Tout comme les fichiers de configuration (cf. 4.2), ces options sont lues et analysées sous une identité effective non privilégiée.

Options génériques

Les options qui suivent sont supportées avec toutes les commandes *vsctl*, mais aussi en l'absence de toute commande :

- **–h** : affichage d'un message d'aide et retour immédiat
- **–v** : affichage du numéro de version et retour immédiat

L'option suivante est supportée avec toutes les commandes *vsctl* :

- **–p** : ne pas réaliser les appels systèmes correspondants à l'action demandée, mais afficher ces appels et leurs arguments sur la sortie standard. Cette option est utile uniquement pour le test de l'utilitaire. Elle se substitue à un utilitaire comme *strace*, qui ne peut suivre *vsctl* lors d'un changement de contexte de sécurité.

Options des commandes autres que *setup*

- **–d** : fermer tous les descripteurs de fichier ouverts et se détacher du terminal de contrôle courant avant l'exécution de la commande spécifiée (mais après la lecture de la configuration).

Options des commandes *start, setup* et *enter*

- **–t** : créer un *proxy* de terminal de contrôle à l'aide de la fonction *clip_vlogin()* de la *libclipvserver* (cf. 3) avant de migrer vers le nouveau contexte de sécurité.

Options des commandes *start* et *setup*

- **–a <adresse>** : utiliser l'adresse et le masque (uniques) définis dans <*adresse*> plutôt que le fichier *addr* du répertoire de configuration de la cage pour configurer le contexte réseau. <*adresse*> est de la même forme qu'une ligne du fichier *addr*, soit :

```
A1.A2.A3.A4/M1.M2.M3.M4
```

L'option *-a* peut être passée jusqu'à quatre fois sur la même ligne de commande, pour définir les quatre adresses que peut supporter un contexte réseau *vserver* (cf. 1.2).

Options de la commande *enter*

- **–c <chemin>** : réaliser un *chroot* dans <*chemin*> une fois à l'intérieur de la cage, avant de lancer l'exécutable spécifié. <*chemin*> est un chemin absolu par rapport à la racine de la cage.
- **–g <gid>** : prendre <*gid*> comme identifiant de groupe *(egid, gid* et *sgid)* une fois à l'intérieur de la cage, avant de lancer l'exécutable spécifié. <*gid*> est un entier en représentation décimale.
- **–u <uid>** : prendre <*uid*> comme identifiant d'utilisateur *(euid, uid* et *suid)* une fois à l'intérieur de la cage, avant de lancer l'exécutable spécifié. <*uid*> est un entier en représentation décimale.
- **–e <env>** : paramétrer l'environnement de l'exécutable spécifié à l'aide de <*env*>. Indépendamment de la présence ou non de cette option, la variable *PATH* est systématiquement définie, à */bin:/sbin:/usr/bin:/usr/sbin* si l'exécutable doit être lancé sous l'identité *root*, et à */bin:/usr/bin:/usr/local/bin* sinon. <*env*> est une liste d'affectations de variables séparées par des ':', soit :

```
<env>="VAR1=val1:VAR2=val2:…:VARn=valn"
```

- **-- <arguments>** : spécifier la commande à lancer à l'intérieur de la cage, ainsi éventuellement que ses arguments. Cette option doit obligatoirement figurer à la fin de la ligne de commande, car <*arguments*> englobe tous les arguments situés après le motif '--'. <*arguments*> est une liste d'arguments séparés par des espaces, le premier constituant la commande à lancer dans la cage, et les suivants étant passés en argument à cette commande.

4.4 Utilitaire *nsmount*

L'utilitaire *nsmount* permet de réaliser des opérations de montage et démontage dans un *namespace VFS* autre que le *namespace* standard, référencé par le *xid* du contexte de sécurité *vserver* auquel il est associé. *nsmount* supporte deux formes d'invocation principales :

Opérations de montage

```
nsmount -x <xid> [options] <spec> <file>
```

Cette commande réalise le montage de <*spec*> sur <*file*> dans le *namespace VFS* associé au contexte de sécurité <*xid*>, avec :

- **spec** : source du montage, fichier de type bloc, répertoire, système de fichier distant ou identifiant arbitraire, selon le type de montage. Lorsque *spec* est un fichier de type bloc ou un répertoire, celui-ci est désigné par son chemin absolu par rapport à la racine de la cage (dans le *namespace* de la cage). On notera que les montages réseau sont réalisés dans le contexte réseau de l'appelant, contrairement aux montages réalisés par *vsctl*.
- **file** : point de montage (chemin absolu par rapport à la racine de la cage).
- **xid** : *xid* du contexte de sécurité référençant le *namespace* cible.

Les options suivantes sont supportées :

- **–t <type>** : précise le type du montage (par défaut, *auto*). <*type*> est l'un des types de systèmes de fichiers définis dans */proc/filesystems*.
- **–o <options>** : précise les options du montage (par défaut, *rw*). <*options*> est une liste d'options séparées par des virgules, selon la même syntaxe que celle reconnue par la commande *mount(8)*.

Opérations de démontage

```
nsmount –u -x <xid> <file>
```

Cette commande réalise le démontage de <*file*> dans le *namespace VFS* associé à <*xid*>, avec :

- **file** : point de montage (chemin absolu par rapport à la racine de la cage).
- **xid** : *xid* du contexte de sécurité référençant le *namespace* cible.

Options génériques

nsmount supporte aussi les options suivantes :

- **–h** : afficher un message d'aide et retourner immédiatement.
- **–v** : afficher le numéro de version et retourner immédiatement.

4.5 Utilitaire *vsattr*

L'utilitaire *vsattr* permet d'affecter des attributs *vserver* (barrière *chroot*, *IUNLINK* et attributs de visibilité du *procfs*) à des *inodes*, ou de supprimer ces mêmes attributs. Il est largement similaire à l'utilitaire *setattr* de *util-vserver*. Une commande *vsattr* prend la forme suivante :

```
vsattr <attribut1> [<attribut2>…] <fichier1> [<fichier2>…]
```

avec *<fichier1>, <fichier2>*, … des chemins vers des fichiers liés aux *inodes* dont on souhaite modifier les attributs (les *inodes* qui ne sont pas liés dans le *VFS* ne peuvent pas être traités par *vsattr*), et *<attribut1>, <attribut2>*, … des spécifications d'attributs. Lorsque plusieurs attributs et fichiers sont définis, tous les attributs sont appliqués à tous les fichiers. Les champs *<attribut>* sont choisis parmi les suivants :

- **--hide / --~hide** : ajouter/supprimer l'attribut *IATTR_HIDE* (cf. 1.3.1)
- **--admin / --~admin** : ajouter/supprimer l'attribut *IATTR_ADMIN*
- **--watch / --~watch** : ajouter/supprimer l'attribut *IATTR_WATCH*
- **--iunlink / --~iunlink** : ajouter/supprimer l'attribut *IATTR_IUNLINK* (cf. 1.3.3)
- **--barrier / --~barrier** : ajouter/supprimer l'attribut *IATTR_BARRIER* (cf. 1.3.4)

Par ailleurs, *vsattr* support les options génériques suivantes :

- **–h** : afficher un message d'aide et retourner immédiatement.
- **–v** : afficher le numéro de version et retourner immédiatement.
- **–p** : ne pas réaliser les appels systèmes *vserver*, mais afficher ces appels et leurs arguments sur la sortie standard.

4.6 Privilèges nécessaires aux utilitaires *vsctl*

Les différents utilitaires *vsctl* ont besoin pour leur fonctionnement des capacités POSIX :

vsctl

Il est rappelé que *vsctl* abandonne temporairement ses capacités effectives et son identité *root* pour la lecture de ses options de ligne de commande et de ses fichiers de configuration.

- Pour une commande *start* ou *setup* :
  - *CAP_CONTEXT* : appels systèmes *vserver*
  - *CAP_SYS_ADMIN* : appels *vserver* privilégiés, montages
  - *CAP_NET_ADMIN* : configuration du contexte réseau
  - *CAP_SYS_RESSOURCE* : attribution des espaces de nommage, attribution de masques de capacités et d'options
  - *CAP_SYS_CHROOT* : *chroot* sur la nouvelle racine
  - *CAP_SYS_TTY_CONFIG* (optionnel) : traitement de l'option *–t* depuis une console
- Pour une commande *enter* :
  - *CAP_CONTEXT* : appels systèmes *vserver*
  - *CAP_SYS_ADMIN* : appels *vserver* privilégiés
  - *CAP_SYS_CHROOT* (optionnel) : traitement de l'option *–p*
  - *CAP_SYS_TTY_CONFIG* (optionnel) : traitement de l'option *–t* depuis une console
  - *CAP_SETUID* (optionnel) : traitement de l'option *–u*
  - *CAP_SETGID* (optionnel) : traitement de l'option *-g*
- Pour une commande *stop* :
  - *CAP_CONTEXT* : appels systèmes *vserver*
  - *CAP_SYS_ADMIN* : appels *vserver* privilégiés
  - *CAP_KILL* : envoi des signaux *SIGTERM* et *SIGKILL* (le drapeau CLSM *CLSM_FLAG_RAISED* est aussi nécessaire si des processus privilégiés au sens de CLSM s'exécutent dans la cage, cf. [CLIP_1201])
- Pour les autres commandes :
  - *CAP_SYS_ADMIN* pour *mount* et *umount*
  - Droits d'accès discrétionnaires à la *socket* d'écoute du processus *setup* ou *CAP_DAC_OVERRIDE* pour *endsetup*

nsmount

- *CAP_CONTEXT* : appels systèmes *vserver*

- *CAP_SYS_ADMIN* : appels *vserver* privilégiés, montages

vsattr

- *CAP_CONTEXT* : appels systèmes *vserver*
- *CAP_SYS_ADMIN* : appels *vserver* privilégiés, montages
- *CAP_LINUX_IMMUTABLE* : modification des attributs d'*inode*

5 Module *pam_jail*

Le module *pam_jail* (paquetage *sys-auth/pam_jail*) est un module PAM (cf. [PAM_MWG]) permettant d'enfermer le processus appelant dans une cage *vserver* existante, en fonction des groupes auxquels appartient l'utilisateur passé dans les *credentials* PAM.

Un fichier de configuration, */etc/security/pam_jail.conf*, définit des associations entre noms de groupes et numéros *xid*. Lors d'un appel au module *pam_jail*, celui-ci parcourt la liste des groupes auxquels appartient l'utilisateur qui lui est transmis par les *credentials* PAM (seul les 16 premiers groupes, y compris le groupe principal sont pris en compte lorsque l'utilisateur appartient à plus de 15 groupes supplémentaires). Chacun de ces groupes est recherché dans le fichier de configuration *pam_jail.conf*. Dès qu'un groupe est trouvé dans ce fichier, le module (et avec lui, le processus appelant) migre, à l'aide de la fonction *clip_enter_context* de la *libclipvserver* (cf. 3) vers la cage dont le *xid* est associé à ce groupe et rend *PAM_SUCCESS*. Une erreur *PAM_AUTH_ERR* est renvoyée s'il est impossible de migrer vers la première cage spécifiée (par exemple à cause de privilèges insuffisants, ou parce que la cage n'existe pas). Si aucune correspondance de groupe n'est trouvée, le module ne fait aucun traitement supplémentaire, et renvoie *PAM_SUCCESS*, sauf lorsqu'il a été invoqué avec l'option *not_found_fails*, auquel cas *PAM_AUTH_ERR* est renvoyé.

Le module définit des *hooks* permettant de réaliser ce traitement dans le cadre de trois groupes PAM : *authentication*, *account management*, et *session management*. Il est important de bien choisir son implantation au sein d'une pile de modules PAM, dans la mesure où il est susceptible de modifier la vision qu'a le processus appelant du système de fichiers, et donc d'interdire le bon fonctionnement d'autres modules PAM (en interdisant par exemple l'accès à */etc/passwd* ou */etc/security/*). Par ailleurs, au regard de la nature privilégiée des opérations réalisées par ce module, il ne doit être invoqué qu'après authentification réussie de l'utilisateur par un autre module PAM. Enfin, on notera bien que le module *pam_jail* ne peut pas créer lui-même une cage, mais ne peut que migrer vers des cages déjà configurées et actives.

Le module journalise son activité par l'interface *syslog* standard, dans le domaine *authpriv*. Il supporte les options suivantes :

- **debug** : produire des traces *syslog* plus verbeuses.
- **no_jail** : réaliser la recherche des groupes dans *pam_jail.conf*, mais ne pas migrer vers les contextes trouvés. Utile uniquement pour le test du module.
- **not_found_fails** : renvoyer une erreur lorsque aucun groupe n'est trouvé dans *pam_jail.conf* pour un utilisateur.
- **proxy** : mettre en place un proxy de terminal à l'aide de la fonction *clip_vlogin* de la bibliothèque *libclipvserver* avant de migrer vers une cage.

Les opérations de migration ne sont possibles que si l'appelant dispose des capacités POSIX suivantes :

- *CAP_CONTEXT* : appels systèmes *vserver*
- *CAP_SYS_ADMIN* : appels *vserver* privilégiés
- *CAP_SYS_TTY_CONFIG* (optionnellement) : prise en compte de l'option *proxy* en console.

Annexe A Appels système *vserver*

Le tableau suivant liste les différentes commandes supportées par l'appel système *sys_vserver*. Pour chaque commande sont détaillés ses effets, les capacités requises pour la mener à bien, ainsi que les autres contraintes exigées pour le traitement de la commande. Ces dernières sont exprimées selon les conventions suivantes :

- ADMIN : le contexte de sécurité appelant doit être le contexte ADMIN.
- WATCH : le contexte de sécurité appelant doit être le contexte WATCH.
- NADMIN : le contexte réseau appelant doit être le contexte ADMIN.
- NWATCH : le contexte réseau appelant doit être le contexte WATCH.
- VXI : le contexte de sécurité cible (implicite ou passé en argument) doit exister (différent de ADMIN ou WATCH).
- NXI : le contexte réseau cible (implicite ou passé en argument) doit exister (différent de ADMIN ou WATCH).
- ZID : le contexte (de sécurité si couplé avec VXI, réseau si couplé avec NXI) cible peut être nul (ADMIN).
- V_ADMIN : le contexte appelant doit être ADMIN. Si VXI est précisé, le contexte de sécurité cible doit posséder le drapeau *STATE_ADMIN*. Si NXI est précisé, le contexte réseau cible doit posséder le drapeau *STATE_ADMIN*.
- SETUP : le contexte cible doit posséder le drapeau *STATE_SETUP*.
- !PRIVATE : le contexte cible ne doit pas posséder le drapeau *INFO_PRIVATE*.
- !SETUP : le contexte cible ne doit pas posséder le drapeau *STATE_SETUP*.
- Le "ou" logique est représenté par '|', le "et" par '&'.

Commande	Capacités	Autres contraintes	Effet
get_version	CONTEXT	-	Lecture du numéro de version
get_vci	CONTEXT	-	Lecture des options de compilation
get_rlimit_mask	CONTEXT	-	Lecture du masque *rlimit*, définissant quel type de limite (min, soft, max) est applicable à quelle *rlimit* par contexte
get_space_mask	CONTEXT	-	Lecture du masque de drapeaux *clone()* applicables à un *namespace vserver*
task_xid	CONTEXT SYS_ADMIN	ADMIN | WATCH	Lecture du *xid* d'une tâche.
reset_minmax	CONTEXT SYS_ADMIN	VXI	Remise à zéro des champs *min* et *max* d'une limite du contexte.
vx_info	CONTEXT	VXI	Lecture du *xid* et du *pid* du processus

	SYS_ADMIN		*init* du contexte courant.
get_bcaps	CONTEXT SYS_ADMIN	VXI	Lecture du masque de capacités POSIX
get_ccaps_v0	CONTEXT SYS_ADMIN	VXI	Lecture des masques de capacités POSIX et de contexte.
get_ccaps	CONTEXT SYS_ADMIN	VXI	Lecture du masque de capacités de contexte
get_cflags	CONTEXT SYS_ADMIN	VXI	Lecture du masque d'options de contexte
get_vhi_name	CONTEXT SYS_ADMIN	VXI	Lecture d'un champ *utsname* (hôte, domaine, etc) d'un contexte
get_rlimit	CONTEXT SYS_ADMIN	VXI	Lecture des trois champs (min, soft, max) d'une *rlimit* d'un contexte
ctx_stat	CONTEXT SYS_ADMIN	VXI	Lecture du nombre de tâches et du compteur de référence d'un contexte
virt_stat	CONTEXT SYS_ADMIN	VXI	Lecture des informations de virtualisation d'un contexte : uptime, biais de temps système, nombres de tâches et charges.
sock_stat	CONTEXT SYS_ADMIN	VXI	Lecture des statistiques pour un type de socket et un contexte : nombre de paquets et d'octets envoyés, reçus et en erreur.
rlimit_stat	CONTEXT SYS_ADMIN	VXI	Lecture des différents champs d'une limite du contexte courant.
task_nid	CONTEXT SYS_ADMIN	NADMIN | NWATCH	Lecture du *nid* d'une tâche.
nx_info	CONTEXT SYS_ADMIN	NXI	Lecture du *nid* d'un contexte *(redondant)*.
get_ncaps	CONTEXT SYS_ADMIN	NXI	Lecture du masque de capacités de contexte réseau.
get_nflags	CONTEXT SYS_ADMIN	NXI	Lecture du masque d'options de contexte réseau.
get_iattr	CONTEXT SYS_ADMIN	-	Lecture des attributs *vserver* d'un *inode* : *xid,* barrière *chroot*, *IUNLINK, IMMUTABLE*, attributs du *procfs*.
get_dlimit	CONTEXT SYS_ADMIN	-	Lecture des limites d'occupation de disque : valeur courante / totale en nombre d'*inodes* et espace.
get_sched	CONTEXT SYS_ADMIN	VXI	Lecture des données de l'ordonnanceur *token bucket* pour un contexte.
sched_info	CONTEXT SYS_ADMIN	VXI | ZID	Lecture de la charge d'un contexte.
wait_exit	CONTEXT SYS_ADMIN	VXI	Bloquer tant que le contexte n'est pas dans l'état *VXS_SHUTDOWN*, puis lire la commande employée pour arrêter le contexte et le code de retour de celle-ci.
ctx_create_v0 **ctx_create**	CONTEXT SYS_ADMIN	-	Créer un contexte de sécurité (v0 : sans spécifier les options initiales) et y faire migrer la tâche appelante.
ctx_migrate_v0 **ctx_migrate**	CONTEXT SYS_ADMIN	VXI & V_ADMIN & ! PRIVATE & !SETUP	Faire migrer la tâche appelante dans un contexte de sécurité (v0 : sans spécifier les options *SET_INIT* et *SET_REAPER*).

enter_space_v0 **enter_space**	CONTEXT SYS_ADMIN	VXI & V_ADMIN & ! PRIVATE	Entrer dans les *namespaces* d'un contexte (v0 : sans préciser le masque de migration).
net_create_v0 **net_create**	CONTEXT SYS_ADMIN	-	Créer un contexte réseau (v0 : sans spécifier les options initiales) et y faire migrer la tâche appelante.
net_migrate	CONTEXT SYS_ADMIN	NXI & V_ADMIN & ! PRIVATE & !SETUP	Faire migrer la tâche appelante dans un contexte réseau.
ctx_kill	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI	Envoyer un signal à un ou tous les processus d'un contexte.
set_space_v0 **set_space**	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Attribuer les *namespaces* de la tâche appelante à un contexte (v0 : au contexte de la tâche appelante).
set_ccaps_v0	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier les masques de capacités POSIX utiles et de capacités de contexte d'un contexte.
set_ccaps	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier le masque de capacités de contexte d'un contexte.
set_bcaps	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier le masque de capacités POSIX utiles d'un contexte.
set_cflags	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier le masque d'options d'un contexte.
set_vhi_name	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier l'un des noms (*domainname, hostname*, etc.) virtuels d'un contexte.
set_rlimit	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier une *rlimit* d'un contexte.
set_sched **set_sched_v2** **set_sched_v3** **set_sched_v4**	CONTEXT SYS_ADMIN SYS_RESOURCE	VXI & SETUP	Modifier les paramètres d'ordonnancement d'un contexte (différents types d'arguments).
set_ncaps	CONTEXT SYS_ADMIN SYS_RESOURCE	NXI & SETUP	Modifier les capacités de contexte d'un contexte réseau.
set_nflags	CONTEXT SYS_ADMIN SYS_RESOURCE	NXI & SETUP	Modifier le masque d'options d'un contexte réseau.
net_add	CONTEXT SYS_ADMIN SYS_RESOURCE	NXI & SETUP	Ajouter des adresses IPv4 (maximum 4) à un contexte réseau, ou définir l'adresse *broadcast* de ce contexte.
net_remove	CONTEXT SYS_ADMIN SYS_RESOURCE	NXI & SETUP	Retirer toutes les adresses IPv4 d'un contexte réseau. L'adresse *broadcast* n'est pas modifiée.
set_iattr	CONTEXT SYS_ADMIN LINUX_IMMUTABLE	-	Modifier les attributs d'un *inode* (*xid* ou attributs *IMMUTABLE, IUNLINK*, *BARRIER* ou attributs du *procfs*).
set_dlimit	CONTEXT SYS_ADMIN SYS_RESOURCE	-	Définir une limite d'utilisation disque.

add_dlimit	CONTEXT SYS_ADMIN SYS_RESOURCE	-	Ajouter une limite d'utilisation disque.
rem_dlimit	CONTEXT SYS_ADMIN SYS_RESOURCE	-	Retirer une limite d'utilisation disque.
dump_history	CONTEXT SYS_ADMIN	-	Afficher par des *printk()* l'historique des appels *sys_vserver*, tous contextes confondus. Uniquement valable si l'option de *debug VSERVER_HISTORY* a été activée à la compilation du noyau.
read_history	CONTEXT SYS_ADMIN	-	Lire une entrée de l'historique des appels *sys_vserver*, tous contextes confondus. Uniquement valable si l'option de *debug VSERVER_HISTORY* a été activée à la compilation du noyau.
read_monitor	CONTEXT SYS_ADMIN	-	Lire l'historique de décisions d'ordonnancement, tous contextes confondus. Uniquement valable si l'option de *debug VSERVER_MONITOR* a été activée à la compilation du noyau.

Annexe B Références

[CLIP_1201] *Documentation CLIP – 1201 – Patch CLIP-LSM*

[CLIP_1203] *Documentation CLIP – 1203 – Patch Grsecurity*

[CLIP_1204] *Documentation CLIP – 1204 – Privilèges Linux*

[CLIP_1303] *Documentation CLIP – 1303 – Cloisonnement graphique*

[CLIP_1304] *Documentation CLIP – 1304 – Cages et socle CLIP*

[CLIP_1401] *Documentation CLIP – 1401 – Cages RM*

[CLIP_1501] *Documentation CLIP – 1501 – Configuration réseau*

[PAM_MWG] Andrew G. Morgan, Thorsten Kukuk, *The Linux-PAM Module Writer's Guide* (version PAM 1.0)

[UTIL_VSERVER] *Util-vserver, http://www.nongnu.org/util-vserver/*

[VSERVER] *Linux Vserver, http://linux-vserver.org/*

[VSERVERDOC] *Documentation Vserver, http://linux-vserver.org/Paper*

Annexe C Liste des figures

Annexe D Liste des tableaux

Annexe E Liste des remarques