Documentation CLIP

1401

Cages RM

Version	Date	Auteur	Commentaires
1.2	27/11/2008	Vincent Strubel	Montage en écriture de */etc/admin* dans les vues UPDATE. Correction de typo dans 3.2 : *syslog-ng* est lancé dans la cage $AUDIT_{clip}$, et non dans $ADMIN_{clip}$. A jour pour CLIP_v03.00.23.
1.1.1	15/10/2008	Vincent Strubel	Ajout de *NXF_NO_SP* au contexte réseau de RM_B. A jour pour CLIP v03.00.21.
1.1.0	04/09/2008	Vincent Strubel	Ajout de CLSM_PRIV_PROCFD au *busybox* de la racine des cages.
1.0.2	06/08/2008	Vincent Strubel	Ajout du montage de */usr/lib* et de *nano* dans la vue ADMIN.
1.0.1	30/07/2008	Vincent Strubel	Convention plus lisible pour les références.
1.0	21/07/2008	Vincent Strubel	Version initiale, à jour pour CLIP v03.00.05.

Table des matières

Introduction

Le présent document détaille la construction et le fonctionnement des cages RM d'un système CLIP-RM, et de leur différentes vues.

Les cages RM sont des cages spécifiques, intégrées en complément des cages CLIP standard sur les systèmes de type CLIP-RM. Chaque cage RM est destinée à une utilisation interactive, dans un environnement logiciel riche, en accédant à des informations d'un niveau de sensibilité donné. Les différentes cages RM accèdent à des niveaux de sensibilité différents, et sont entièrement cloisonnées entre elles, en confidentialité aussi bien qu'en intégrité. La configuration CLIP-RM typique comporte deux cages RM, RM_B et RM_H, qui traitent respectivement des informations de niveau "bas" et "haut". La cage RM_B accède directement au réseau de déploiement du poste CLIP-RM, tandis que RM_H n'accède qu'à des serveurs de service RM_H, au travers d'un VPN IPsec dédié.

Chaque cage RM est sous-découpée en quatre **vues**, correspondant à autant de rôles fonctionnels au sein de la cage : ADMIN pour l'administration locale de la cage, AUDIT pour la collecte de ses journaux, UPDATE pour l'application de ses mises à jour, et USER pour son utilisation interactive. Les quelques composants de la cage (processus ou fichiers) qui ne sont rattachés à aucune de ces vues sont rassemblés sous le terme de **racine** de la cage.

Les cages RM se distinguent aussi des cages CLIP par le fait qu'elles sont installées à partir d'une distribution de paquetages binaires "rm", différente de la distribution standard "clip" installée dans le socle et les cages CLIP. Ainsi, les mises à jours secondaires des cages RM ne sont pas installées par la cage $\text{UPDATE}_{\text{clip}}$ (qui réalise néanmoins le téléchargement des paquetages correspondants), mais bien par la vue UPDATE de la cage, tandis que les mises à jours primaires sont installées par une procédure spécifique, décrite en section 6, et distincte de la mise à jour primaire de CLIP.

1 Racine de la cage

La racine d'une cage RM est extrêmement limitée, et n'est active qu'au démarrage et à l'arrêt de la cage. Aucun processus ne s'exécute en dehors des vues pendant le fonctionnement normal de la cage.

1.1 Configuration

1.1.1 Configuration *vserver*

La configuration de la cage RM_X est fournie par l'arborescence */etc/jails/rm_X*, installée par le paquetage *app-clip/clip-vserver*. Les principaux éléments de cette configuration sont les suivants (cf. [CLIP_1202]) :

- numéro de contexte *vserver* supérieur à 1000, contrairement aux cages CLIP qui ont des numéros strictement inférieurs à 1000 : 1001 pour RM_H, 1002 pour RM_B.
- racine : */vservers/rmX* : */vserver/rm_h* pour RM_H, */vserver/rm_b* pour RM_B
- commande initiale non définie (c'est-à-dire définie à <*invalid*>), la cage étant démarrée par une séquence *vsctl setup* plutôt que directement par *vsctl start*.
- pas d'adresse IP fixe, l'adresse (unique) de la cage est définie à partir d'un paramètre de configuration dynamique du poste lors de son démarrage. Ces paramètres sont importés depuis le fichier de configuration */etc/admin/conf.d/net*. En particulier, les adresses de RM_H et RM_B sont calculées à partir de RMH_ADDR / RMH_MASK et RMB_ADDR / RMB_MASK, respectivement.
- Les capacités maximales autorisées dans la cage sont celles généralement autorisées dans les cages CLIP (cf. [CLIP_1202]), complétées de *CAP_NET_BIND_SERVICE* et *CAP_SYS_CHROOT*, qui sont nécessaires au fonctionnement du démon *sshd* et de *jailmaster* (*CAP_SYS_CHROOT* uniquement), soit en résumé :
  - *CAP_CHOWN*
  - *CAP_DAC_OVERRIDE*
  - *CAP_DAC_READ_SEARCH*
  - *CAP_FOWNER*
  - *CAP_FSETID*
  - *CAP_KILL*
  - *CAP_SETGID*
  - *CAP_SETUID*
  - *CAP_NET_BIND_SERVICE*
  - *CAP_SYS_CHROOT*
- De même, les drapeaux de contexte *vserver* sont les drapeaux par défaut listés dans

[CLIP_1202], soit :

- *INFO_INIT*
- *HIDE_VINFO*
- *HIDE_MOUNT*
- *HIDE_NETIF*
- *VIRT_CPU*
- *VIRT_MEM*
- *VIRT_LOAD*
- *VIRT_UPTIME*

- Le contexte réseau associé à une cage RM_B dispose du drapeau spécifique à CLIP *NXF_NO_SP*, lui permettant d'accéder au réseau sans politique de sécurité IPsec. Une cage RM_H ne dispose pas de ce drapeau.

1.1.2 Montages

Montages propres à la racine

Une partition physique est réservée sur le disque dur pour chaque cage RM de chaque installation CLIP d'un poste (cf. [CLIP_1304]). Ces partitions sont montées dans le *namespace VFS* du socle CLIP sur les racines des cages RM. Les autres montages composant chaque cage RM sont réalisés par deux scripts de démarrage, *clip_viewers* et *clip_servers* (*app-clip/clip-vserver*). Le premier de ces scripts réalise uniquement le montage, dans le *namespace VFS* du socle CLIP, des systèmes de fichiers attribués aux administrateurs RM. Ces systèmes de fichiers sont montés à l'aide de l'option *loop* depuis des fichiers images de la partition */home* du système. Il est ainsi possible de stocker ces données sur cette partition commune aux deux installations CLIP d'un poste (cf. [CLIP_1304]) sans pour autant exposer directement le système de fichiers de */home* dans les cages RM[1]. Le script *clip_servers* réalise quant à lui tous les autres montages, dans le *namespace VFS* de la cage uniquement, par l'intermédiaire d'une commande *vsctl*, qui lit et traite (dans cet ordre) deux fichiers de configuration par cage :

- */etc/jails/rm_X/fstab.internal* : définit les montages internes (de type *bind* en général) à l'arborescence de la cage, avec des sources et des destinations relatives à la racine de la cage.
- */etc/jails/rm_X/fstab.external* : définit les montages externes à l'arborescence de la cage, avec des sources relatives à la racine du socle CLIP, et des destinations relatives à celle de la cage RM.

Un petit nombre de montages, parmi ceux spécifiés dans ces deux fichiers, sont réservés à la racine de la cage RM. Ces montages sont résumés dans le Tableau 1. Les autres montages sont traités dans les sections réservées aux différentes vues dont ils constituent les arborescences.

Le seul point remarquable des montages de la racine RM concerne les fichiers périphériques. La racine

[1] Une telle exposition directe créerait un canal de communication entre cages RM, par la variation de l'espace disponible sur le système de fichiers partagé. Voir aussi [CLIP_1202].

de la cage possède son propre répertoire */dev*, monté en lecture seule depuis une partition du socle[2]. De même, chaque vue de la cage possède son propre répertoire */dev*, lui aussi monté en lecture seule depuis le socle CLIP. Le paquetage fournissant les fichiers *devices* contenus par ces répertoires est un paquetage primaire CLIP (*clip-layout/rm-devices*), plutôt qu'un paquetage RM, dans la mesure où seule la fonction de mise à jour du coeur CLIP dispose au sein du système des privilèges nécessaires à la création de *devices* (*CAP_SYS_MKNOD*).

Source	Point de montage	Type	Options
Montages réalisés dans le socle (à partir de */etc/fstab*)			
/dev/RMX_PART	*/*	*ext3*	*rw,nodev*
Montages de */etc/jails/rm_X/fstab.external*			
/mounts/vsdev/rm_X/jail_devs	*/dev*	*bind*	*ro,nosuid,noexec,noatime*
/mounts/vsdev/rm_X/update_devs	*/update_root/dev*	*bind*	*ro,nosuid,noexec,noatime*
/proc	*/proc*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
none	*/tmp*	*tmpfs*	*rw,nosuid,nodev,noexec,noatime, size=16m,mode=1777*

Tableau 1: Montages de la racine d'une cage RM (RM_X).

Les *devices* attribués à la racine de la cage sont les suivants :

- *zero, null, full*
- *urandom*, ainsi qu'un lien symbolique *random* pointant sur *urandom*[3]
- un lien symbolique *log* pointant vers la socket *log* créée à la racine de la vue UPDATE (cf. 3), ce qui permet de ne pas créer de *socket log* directement dans */dev*, qui est en lecture-seule.
- un répertoire */dev/mapper* dans lequel le socle peut temporairement copier des périphériques *device mapper* correspondant aux systèmes de fichiers chiffrés des utilisateurs (cf. 5.1, [CLIP_1302] et [CLIP_DCS_15088])

On notera cependant que la racine de la cage a aussi accès aux *devices* des différentes vues, sous */admin/dev*, */audit/dev*, */update/dev* et */user/dev*. Par ailleurs, les *devices* de la vue UPDATE de la cage sont aussi montés sous */update_root/dev* (montage distinct de celui réalisé sur */update/dev*, qui est visible dans la vue UPDATE), de manière à offrir un accès au périphérique *veriexec* à l'utilitaire *verictl* lorsqu'il est invoqué *chrooté* dans */update_root*, plutôt qu'*/update*, au démarrage de la cage (cf. 1.2.1).

Montages partagés entre vues

Plusieurs montages de la cage sont partagés entre ses différentes vues. Bien que ces montages ne soient pas à proprement parler utilisés par la racine de la cage, ils sont résumés ici préalablement à la description des différentes vues. Il s'agit plus particulièrement des montages suivants :

- */etc/core* : ce répertoire du socle CLIP est monté sous le même nom dans chacune des vues. Il

[2] A la différence du montage en lecture-écriture des répertoires privés des administrateurs RM, ce montage en lecture seule depuis un même système de fichiers du socle dans les différentes cages RM ne crée pas de canal de communication entre ces dernières.

[3] Le véritable *device random* (source d'aléa "matériel", éventuellement bloquante) n'est jamais exposé dans les cages, afin d'éviter qu'une cage ne puisse épuiser la "réserve d'entropie" de l'ensemble du système.

fournit les différents paramètres communs à l'ensemble du poste, en particulier la liste des comptes utilisateurs[4] et la géométrie de l'écran. Afin d'éviter de créer par ce biais un canal de communication entre cages RM, ce montage est réalisé non seulement en lecture-seule, mais aussi avec les attributs *noatime* et *nolock* (cf. [CLIP_1201]) qui interdisent la modification des attributs du montage qui ne sont pas contrôlés par l'option *read-only* (cf. [CLIP_1202]).

- */etc/shared* : ce répertoire contient les fichiers de configuration du coeur de la vue UPDATE (paquetages primaires RM) qui sont partagés par les différentes vues, par exemple les fichiers de configuration de la *glibc* RM (*/etc/services, /etc/shells*, etc.). Ce répertoire est propre à chaque cage RM, et monté sous le même nom dans chacune de ses vues depuis le répertoire */update_root/etc/shared*. Comme pour */etc/core*, des liens symboliques sont créés dans les répertoires */etc* des différentes vues, pointant vers les fichiers de */etc/shared*, par exemple :

```
/etc/services   =>   /etc/shared/services
```

- */etc/admin* : ce répertoire contient les fichiers de configuration de la cage RM qui sont modifiables par l'administrateur RM (cf. 2.2). Il est monté sous le même nom dans toutes les vues de la cage, depuis le répertoire */admin_priv/etc.admin* de l'arborescence privée ADMIN. Ce montage est en lecture-écriture dans la vue ADMIN, et en lecture seule dans les autres vues.
- Les exécutables et bibliothèques des différentes vues sont, à l'exception des racines propres à chaque vue, partagés depuis l'arborescence de la vue UPDATE, dont les répertoires */lib*, */usr* (ou seulement */usr/lib*) et */usr/local* (ou */usr/local/lib*) sont exposés en lecture seule dans les différentes vues, en fonction du besoin d'accès de chaque vue à ces exécutables et bibliothèques.

Fichiers de la racine

L'ensemble des fichiers propres à la racine (outre les périphériques) est installé par deux paquetages : *baselayout-rm*, qui fournit l'arborescence de répertoires de la cage, et *busybox-rm*, qui fournit les utilitaires propres à la racine. Ce dernier installe un unique exécutable, */bin/busybox*, qui apporte les fonctionnalités de différents utilitaires de base UNIX en fonction du nom sous lequel il est appelé, ainsi qu'un ensemble de liens symboliques vers *busybox* correspondant aux noms des différents utilitaires émulés. Le paquetage *busybox-rm* est produit lorsque le paquetage *sys-apps/busybox* est compilé avec la variable d'environnement *DEB_NAME_SUFFIX* (cf. [CLIP_1101]) positionnée à *"rm"*, ce qui sélectionne automatiquement une configuration adaptée à la cage. Cette configuration est limitée à un *shell* compatible *ash*, aux quelques utilitaires mis en oeuvre pendant le démarrage et l'arrêt de la cage : *kill, mktemp, test*, et à l'utilitaire *fuser*, utilisé pour le démontage des montages de session au sein de la cage (cf. [CLIP_1302]). Le *shell busybox* utilise le drapeau *O_MAYEXEC*, spécifique à CLIP ([CLIP_1201]), lors de l'ouverture de fichiers, ce qui interdit l'exécution ou le "sourçage" de scripts depuis des montages portant l'option *noexec*.

Les paquetages *busybox-rm* et *baselayout-rm*, bien qu'ils soient rattachés au coeur de la cage (qui comprend aussi la racine de la vue UPDATE), présentent la particularité d'être des paquetages primaires du socle CLIP plutôt que de RM, donc rattachés à la distribution CLIP et mis à jour en même temps que le coeur CLIP. Cette spécificité est rendue nécessaire par la non-exposition de la racine RM dans la cage SECURE_UPDATE_RM associée (6), ce qui interdit sa modification par la mise à jour primaire RM. Un seul exemplaire de chacun de ces deux paquetages est installé dans le socle CLIP pour l'ensemble des cages RM du système, les fichiers étant multiplexés entre les différentes racines lors de la génération des paquetages binaires, grâce à l'utilisation de la variable *CLIP_VROOTS* (cf.

[4] Partie publique uniquement, c'est-à-dire sans les empreintes de mots de passe associées à certains des comptes.

[CLIP_1101]).

Coeur et paquetages secondaires de la cage

Le coeur d'une cage RM est constitué de la combinaison de deux éléments :

- la racine de la cage, qui, comme mentionné au paragraphe précédent, est installée par des paquetages primaires du socle CLIP.
- la racine de la vue UPDATE, installée sous */update_root* et ses sous-répertoires[5], qui est quand à elle installée par des paquetages primaires RM.

On notera qu'à la différence du coeur CLIP, les coeurs RM restent accessibles en écriture depuis le socle CLIP (mais pas depuis les cages CLIP) et depuis la racine de la cage RM.

Les paquetages secondaires de la cage sont quant à eux installés dans la sous-arborescence */usr/local* de la vue UPDATE, et dans les racines des autres cages. Ces différents répertoires sont montés depuis l'arborescence "privée" de la vue UPDATE, */update_priv*. Enfin, à l'instar des cages CLIP ([CLIP_1304]), chaque vue RM dispose de sa propre arborescence privée, qui n'est exposée que dans la racine de la cage et dans la vue elle-même, et qui contient au moins un répertoire */var* privé.

1.1.3 Configuration *veriexec* de la cage

Chaque cage RM se voit associer un contexte *veriexec* ([CLIP_1201]), créé par le script */etc/init.d/veriexec* (*app-clip/veriexec*), mais activé par le script */etc/init.d/clip-servers* (*app-clip/clip-vserver*). Ce dernier attribue les drapeaux de niveau (cf. [CLIP_1202]) suivants au contexte, lorsque la cage est active :

- *VRXLVL_ACTIVE*
- *VRXLVL_ENFORCE_MNTRO*

A la différence des cages CLIP ([CLIP_1304]), ces drapeaux autorisent la modification du niveau du contexte (uniquement depuis le socle CLIP), ainsi que la modification, depuis la cage elle-même, des entrées *veriexec* du contexte. La première de ces propriétés permet au socle de désactiver entièrement *veriexec* dans le contexte de la cage lors de l'arrêt de celle-ci (1.2.3), tandis que la seconde permet à la cage, qui gère ses propres mises à jour de paquetages secondaires, de tenir à jour les entrées *veriexec* associées à ces paquetages. Le niveau de la cage n'est pas défini, au contraire des cages CLIP, dans le fichiers */etc/veriexec/ctxlevels* du socle, mais dans le fichier */etc/jails/rm_X/veriexec.lvl-up*.

Le masque de capacités du contexte *veriexec* définit les capacités attribuables par *veriexec* dans la cage. Ces capacités sont égales à celles autorisées de manière générale dans la cage, soit :

- *CAP_CHOWN*
- *CAP_DAC_OVERRIDE*
- *CAP_DAC_READ_SEARCH*
- *CAP_FOWNER*
- *CAP_FSETID*

[5] En particulier, les répertoires */bin, /sbin, /lib, /etc* (y compris */etc/shared*) et */usr* (sauf */usr/local*) de la vue.

- *CAP_KILL*
- *CAP_SETGID*
- *CAP_SETUID*
- *CAP_NET_BIND_SERVICE*
- *CAP_SYS_CHROOT*

De même, le masque de privilèges CLSM du contexte *veriexec* réduit les privilèges attribuables dans la cage à :

- *CLSM_PRIV_PROCFD*
- *CLSM_PRIV_NETSERVER*
- *CLSM_PRIV_NETCLIENT*
- *CLSM_PRIV_NETOTHER*
- *CLSM_PRIV_VERICTL*

Les entrées *veriexec* suivantes sont définies dans le contexte de la cage (avec des chemins relatifs à la racine de la vue UPDATE) :

- */usr/local/bin/mozilla-firefox/firefox-bin, /usr/local/bin/mozilla-thunderbird/thunderbird-bin, /usr/local/lib/java/bin/java_vm* et */usr/local/bin/ssh* se voient attribuer le privilège *CLSM_PRIV_NETCLIENT*.
- */admin_root/sbin/sshd* se voit attribuer *CAP_SYS_CHROOT, CAP_NET_BIND_SERVICE* et le privilège *CLSM_PRIV_NETSERVER* lorsqu'il est exécuté par *root*.
- */sbin/verictl* se voit attribuer le privilège *CLSM_PRIV_VERICTL* lorsqu'il est exécuté par *root*.
- */bin/jailmaster* et */usr/sbin/syslog-ng* se voient attribuer le privilège *CAP_SYS_CHROOT* lorsqu'ils sont exécutés par *root*.

Par ailleurs, le */bin/busybox* de la racine de la cage se voit attribuer, lorsqu'il est exécuté par *root*, le privilèges *CLSM_PRIV_PROCFD*, qui lui permet de fonctionner lorsqu'il est lancé sous l'"identité" *fuser* (qui lit les */proc/<pid>/fd/** des différents processus de la cage).

1.2 Démarrage et fonctionnement de la cage

1.2.1 Création et configuration de la cage

Toutes les cages RM du système sont démarrées par un unique script, */etc/init.d/clip_servers* (*app-clip/clip-vserver*). La liste des cages à lancer est lue dans la variable CLIP_JAILS, définie dans le fichier */etc/conf.d/clip*. Cette variable constitue un paramètre système, qui n'est en aucun cas modifiable par l'administrateur du poste. Pour chaque cage, la séquence de démarrage doit non seulement assurer la configuration du contexte *vserver* associé à la cage, mais aussi du contexte *veriexec* correspondant, ce dernier n'étant pas, contrairement aux cages CLIP, peuplé et activé par le script de démarrage *veriexec*[6].

[6] Plus précisément, */etc/init.d/veriexec* crée le contexte *veriexec* associé à chaque cage RM, mais n'y charge aucune entrée, et le laisse inactif. Ceci permet de charger initialement les entrées *veriexec* de la cage avec le *verictl* de celle-ci, qui ne

Afin de pouvoir réaliser ces différentes opérations, le script réalise une configuration en plusieurs étapes, en créant tout d'abord la cage par une commande *vsctl setup* ([CLIP_1202]), avant d'y lancer une série de commandes de configuration, puis enfin les services de la cage. La configuration du contexte *veriexec* est facilitée par un script */bin/verictl.sh* installé à la racine de la vue UPDATE. Ce script réalise le chargement ou le déchargement de toutes les entrées *veriexec* définies dans un répertoire donné. Il peut être appelé avec les arguments suivants :

```
verictl.sh <commande> [<chemin>]
```

avec :

- <*commande*> l'argument *-l* pour procéder au chargement des entrées, ou *-u* pour procéder à leur déchargement.
- <*chemin*> le chemin de base du répertoire où chercher des entrées *veriexec*. Le script considère comme des entrées tous les fichiers du répertoire <*chemin*>*/etc/verictl.d/*.

La séquence complète de démarrage d'une cage RM est ainsi la suivante :

- Import de l'adresse et du préfixe IP de la cage. A ce stade, une seule adresse est attribuée à chaque cage[7]. En pratique, les adresses et préfixes de toutes les cages RM du système sont importées avant le lancement de la première de ces cages. Aucune cage n'est lancée en cas d'échec de l'un de ces imports.
- Suppression d'éventuels fichiers de *sockets syslog* laissés à la racine des vues de la cage (cf. 3) lors de l'arrêt précédent.
- Génération d'un *cookie* d'authentification *vsctl*, stocké dans l'environnement du script.
- Création de la cage par une commande *vsctl setup* utilisant ce même *cookie*, en passant comme argument supplémentaire l'adresse à attribuer au contexte réseau de la cage.
- Lancement, par une commande *verictl enter -c /update_root*, de *verictl.sh chrooté* dans */update_root*, avec l'argument *-l* et sans chemin, afin de charger les entrées *veriexec* des paquetages primaires RM (depuis */update_root/etc/verictl.d/**). Il est nécessaire de réaliser le *chroot* dans */update_root* plutôt que dans */update*, afin de disposer d'un accès en écriture aux montages correspondant aux fichiers décrits par ces entrées, accès qui est exigé pour le chargement d'entrées *veriexec* (cf. [CLIP_1201]).
- Lancement par une deuxième commande *verictl enter* de *verictl.sh chrooté* cette fois dans */update*, avec l'argument *-l* et le chemin */usr/local*, afin de charger les entrées *veriexec* des paquetages secondaires RM.
- Activation, dans le socle CLIP, du contexte *veriexec* de la cage, en lui attribuant le niveau défini dans */etc/jails/rm_X/veriexec.lvl-up*.
- Lancement, par une commande *vsctl enter*, d'un démon *syslog-ng* à la racine de la cage. Ce démon s'enferme ensuite dans */audit*, et assure ainsi le service AUDIT de la cage. Il doit naturellement être lancé avant tous les autres services de la cage. Son fonctionnement est décrit plus en détail en section 3.

dispose alors pas encore du privilège *CLSM_PRIV_VERICTL*.

[7] Le script *clip_servers* ne vérifie pas l'absence d'intersection entre les réseaux attribués aux différentes cages. Cependant, une telle vérification est réalisée préalablement à son appel par les scripts de configuration réseau du poste (cf. [CLIP_1501]).

- Lancement, par une commande *vsctl enter*, du démon *jailmaster* de la cage, afin de lancer les services des trois autres vues. Le comportement de ce démon au lancement est décrit plus en détail au paragraphe 1.2.2 ci-dessous.
- Terminaison du processus *vsctl setup* qui maintenait la cage active jusqu'au lancement de *syslog-ng* et de *jailmaster*, par une commande *vsctl endsetup* utilisant le même *cookie*.

1.2.2 Lancement des services

Le démon *jailmaster* est un développement spécifique à CLIP, qui assure le lancement des services UPDATE et ADMIN d'une cage RM, avant de s'enfermer dans la vue USER de manière à y assurer le service USER. Il est installé sous */bin/jailmaster* dans la vue UPDATE, par un paquetage primaire RM (*app-clip/jailmaster*). Il n'est cependant pas exécuté dans cette vue, mais directement à la racine de la cage (sous le chemin */update/bin/jailmaster*). Au lancement, *jailmaster* se "démonise" en fermant tous ses descripteurs de fichiers ouverts et en se détachant de son terminal de contrôle, puis crée deux fils, qui s'enferment par *chroot()* respectivement dans */admin* et */update*, avant d'exécuter respectivement */sbin/sshd* dans */admin* et */sbin/crond* dans */update*. Il crée ainsi les services ADMIN et UPDATE dans ces deux vues. Le processus père s'enferme ensuite dans */user*, où il assure lui-même le service USER, en créant une *socket unix /var/run/start*, sur laquelle il se met en attente de connexions issues de la cage $USER_{clip}$. Le traitement de ces connexions est décrit plus en détail dans la section 5. Le démon *jailmaster* journalise ses traitements, en particuliers les éventuelles erreurs, et les lancement de sessions utilisateur, à travers l'interface *syslog* de la cage RM. Il utilise pour cela la *socket /dev/log* de la racine RM, qu'il ouvre après s'être détaché de son terminal de contrôle et avant de créer ses fils.

Le lancement par le script de démarrage *clip_servers* des deux démons *syslog-ng* et *jailmaster* dans une cage RM assure ainsi le démarrage des quatre vues de cette cage. Cette séquence de démarrage est résumée dans la Figure 1.

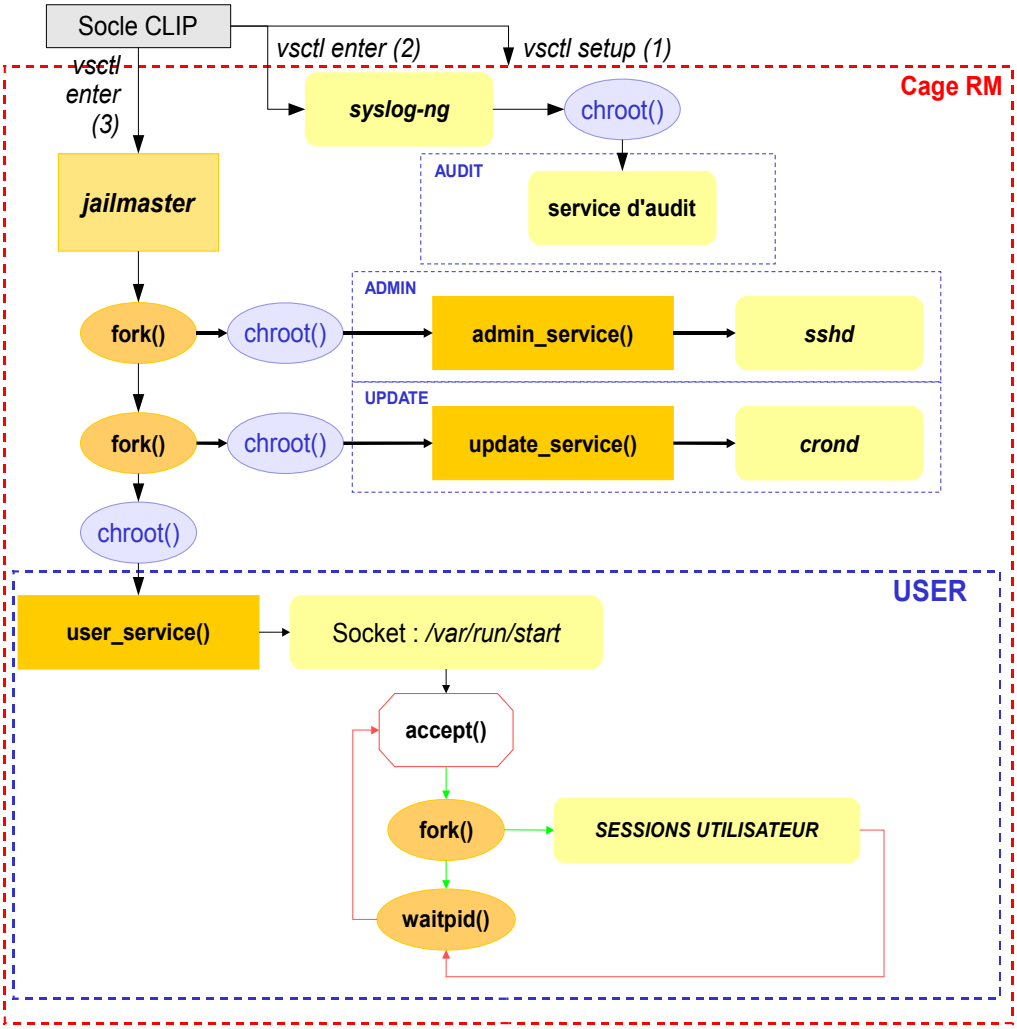

Figure 1: Lancement des différentes vues d'une cage RM.

1.2.3 Arrêt de la cage

L'arrêt des cages RM est réalisé par la fonction *stop()* du script de démarrage *clip_servers*. Pour chaque cage RM RM_X, le script effectue les opérations suivantes, dans cet ordre :

- Désactivation de *veriexec* dans la cage, en lui attribuant le niveau défini dans */etc/jails/rm_X/veriexec.lvl-down*, soit normalement *VRXLVL_INACTIVE*.
- Déchargement des entrées *veriexec* du contexte de la cage, en lançant deux fois *verictl.sh* dans la cage avec l'option *-u*, respectivement *chrooté* dans */update_root* (pour décharger les entrées des paquetages primaires) et dans */update* (pour décharger celles des paquetages secondaires).
- Terminaison de la cage par un appel *vsctl stop*

La désactivation de *veriexec* dans la cage, et la suppression des entrées correspondantes, permettent d'envisager de relancer la cage par un nouvel appel à */etc/init.d/clip_servers start,* bien que cette possibilité ne soit pas exploitée à ce jour (voir remarque ci-dessous).

Remarque 1 : interruption des cages RM pour mise à jour

La possibilité d'interrompre le fonctionnement des cages RM, pour les relancer ensuite, vise en particulier à permettre une mise à jour des paquetages primaires RM pendant le fonctionnement normal du système, en lançant les cages SECURE_UPDATE_RM (cf. 6) pendant une interruption des cages RM. Cependant, le système CLIP n'intègre pas à ce jour de mécanisme permettant le déclenchement d'une telle interruption après le téléchargement par $UPDATE_{clip}$ d'une nouvelle configuration primaire RM. Un mécanisme de ce type serait souhaitable à terme, en portant une attention particulière au risque d'interruption brutale des sessions utilisateur.

2 Vue ADMIN

La vue ADMIN d'une cage RM est dédiée à l'administration de certains paramètres propres à la cage, et partagés par ces différentes vues. Elle est utilisée de manière interactive, sous un compte unique *_admin*, à travers une connexion *ssh* sur la boucle locale, ouverte depuis la vue USER de la même cage.

2.1 Configuration de la vue

Montages

Les montages constituant l'arborescence de la vue ADMIN sont résumés dans le Tableau 2. Ils présentent les caractéristiques suivantes :

- La vue dispose d'une racine dédiée, montée depuis l'arborescence UPDATE (ce qui permet sa mise à jour comme un paquetage secondaire RM).
- La vue dispose de son répertoire */dev* dédié en lecture-seule, contenant un ensemble de *devices* adaptés à son fonctionnement, en particulier les pseudo-terminaux indispensables à son utilisation interactive. Plus précisément, ces périphériques sont :
  - *zero, null, full*
  - *urandom*, ainsi qu'un lien symbolique *random* pointant sur *urandom*
  - un lien symbolique *log* pointant vers la *socket log* créée à la racine de la vue (cf. 3), ce qui permet de ne pas créer de *socket log* directement dans */dev*, qui est en lecture-seule.
  - Les pseudo-terminaux *tty* et *ptmx*, complétés par un montage du système de fichiers *devpts* sur */dev/pts*.
- Le système de fichiers *proc* est monté dans la cage afin de permettre son utilisation interactive (il est en particulier indispensable au fonctionnement de *sshd*). On notera que ce système de fichiers est monté par un *bind* en lecture seule depuis le */proc* du socle CLIP, plutôt que par un montage direct de type *proc*, afin de pouvoir décorréler les options de montage[8]. On se souviendra par ailleurs que ce système de fichiers est particulièrement "allégé" dans une cage CLIP, et réduit pour l'essentiel aux répertoires */proc/<pid>* des processus de la cage ([CLIP_1202]). L'option *grsecurity* GRKERNSEC_CHROOT_FINDTASK ([CLIP_1203]) assure de plus un cloisonnement supplémentaire entre vues du */proc*, avec pour résultat que seuls les */proc/<pid>* correspondants aux processus de la vue sont visibles dans celle-ci.
- Un système de fichiers spécifique est monté depuis la partition */home* du système CLIP sur le répertoire */home/admin*, qui correspond au *$HOME* de l'utilisateur de la vue, *_admin*. Ce montage permet de partager ce système de fichiers, qui contient principalement les clés publiques *ssh* des utilisateurs autorisés à se connecter dans la vue ADMIN depuis USER, entre les deux installations CLIP présentes sur le disque, et ainsi de conserver ces clés publiques lors

[8] En effet, les options de montage de *proc* sont communes à tous les montages de ce système de fichiers. Ainsi, si plusieurs montages de type *proc* sont réalisés successivement avec des options différentes, tous les montages porteront au final les options du dernier montage.

d'une mise à jour du coeur CLIP par exemple (cf. [CLIP_1304]). Ce montage est réalisé à travers un montage *loop*, plutôt que par un *bind* direct, afin d'éviter la création d'un canal de communication entre cages RM (comme détaillé en 1.1.2).

- La vue dispose, au contraire des autres vues, d'un accès en lecture-écriture au montage */etc/admin*, ce qui permet la modification des fichiers de configuration stockés dans ce répertoire.

Source	Point de montage	Type	Options
Montages réalisés dans le socle CLIP (par */etc/init.d/clip_viewers*)			
/home/admin.rmX	*/home/adminrmX*	*ext2*	*loop,rw,nosuid,nodev,noexec,noatime*
Montages de */etc/jails/rm_X/fstab.internal* Les sources sont relatives à la racine de la cage RM, les points de montages à la racine de la vue.			
/update_priv/admin_root	*/*	*bind*	*ro,nodev,noatime*
/update/etc/shared	*/etc/shared*	*bind*	*ro,nosuid,nodev,noexec,noatime*
/update/lib	*/lib*	*bind*	*ro,nosuid,nodev,noatime*
/update/usr/lib	*/usr/lib*	*bind*	*ro,nosuid,nodev,noatime*
/admin_priv/home	*/home*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/admin_priv/var	*/var*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/admin_priv/etc.admin	*/etc/admin*	*bind*	*rw,nosuid,nodev,noexec,noatime*
Montages de */etc/jails/rm_X/fstab.external* Les sources sont relatives à la racine du socle CLIP, les points de montages à la racine de la vue.			
/home/adminrmX	*/home/admin*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/etc/core	*/etc/core*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
/mounts/vsdev/rm_X/admin_devs	*/dev*	*bind*	*ro,nosuid,noexec,noatime*
/proc	*/proc*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
<none>	*/tmp*	*tmpfs*	*rw,nosuid,nodev,noexec,noatime, size=16m,mode=1777*
<none>	*/dev/pts*	*devpts*	*rw,nosuid,noexec,noatime,nolock, gid=5,mode=0620*

Tableau 2: Montages de la vue ADMIN.

Paquetages

Les seuls exécutables de la cage sont ceux fournis par sa racine dédiée, et plus précisément par les trois paquetages qui y installent des fichiers :

- *busybox-admin*, version de *sys-apps/busybox* adaptée à un compartiment d'administration (avec la même configuration que celui déployé dans $ADMIN_{clip}$, cf. [CLIP_1304]) fournit un *shell* compatible *ash*, avec auto-complétion et gestion de l'historique, ainsi que les quelques utilitaires simples permettant l'édition des fichiers de configuration, en particulier un éditeur *vi*. Le *shell busybox* utilise le drapeau *O_MAYEXEC*, spécifique à CLIP ([CLIP_1201]), lors de l'ouverture de fichiers, ce qui interdit l'exécution ou le "sourçage" de scripts depuis des montages portant l'option *noexec*.

- *openssh* installe dans la vue sa partie serveur, c'est-à-dire */sbin/sshd*, et ses fichiers de configuration. On notera que ceux-ci sont installés, du fait de la configuration de ce paquetage secondaire, dans */usr/local/etc/sshd*, qui est bien un répertoire propre à la vue ADMIN, distinct en particulier du */usr/local* des vues UPDATE et USER.
- *nano-admin* (*app-editors/nano*) installe l'éditeur de texte */bin/nano,* comme alternative à *vi.*

2.2 Fonctionnement de la vue

Après son démarrage par *jailmaster*, la vue se met en attente de connexions utilisateur depuis la vue USER. Le seul processus actif en dehors de ces connexions est le démon *sshd*.

Sessions d'administration

Le démon *sshd* de la vue est configuré statiquement de la manière suivante :

- Écoute sur le port 22, toutes adresses confondues (l'adresse d'écoute étant de toutes manières fixée par la configuration de la cage).
- Protocole SSHv2 uniquement. Les clés d'authentification du serveur sont générées à la première installation du paquetage *net-misc/openssh,* et stockées dans */usr/local/etc/ssh/* au sein de l'arborescence de la cage (montage en lecture seule). Une mesure spécifique appliquée lors de l'installation du poste garantit que les clés d'authentification du serveur sont les mêmes pour les deux installations CLIP normalement présentes sur le poste.
- Authentification limitée au compte *_admin*, et uniquement à l'aide de la méthode *PubkeyAuthentication* (cf. [CLIP_1302]), avec des clés publiques autorisées définies dans */home/admin/.ssh/authorized_keys* dans l'arborescence de la cage. Le répertoire */home/admin* de la cage étant monté depuis le fichier */home/admin.rmX* du socle, ces clés sont partagées entre les deux installations CLIP du poste.
- Mise en oeuvre de la séparation de privilèges. Ce mode de fonctionnement est détaillé en [PRIVSEP] et [PRIV].

Le filtrage réseau du système (cf. [CLIP_1501]) n'autorise par ailleurs les connexions *ssh* que sur la boucle locale de la cage (c'est-à-dire sur la boucle locale et uniquement avec une adresse source et destination égale à l'adresse de la cage), interdisant en particulier des connexions distantes ou issues d'autres cages locales.

Les connexions à ce serveur sont réalisés automatiquement lorsqu'un utilisateur membre du groupe *rm_admin* ouvre une session dans la vue USER de la cage, comme cela est décrit en 5. Ces connexions se limitent à un *shell* unique sous l'identité *_admin*.

Fichiers de configuration modifiables

Les fichiers de configuration susceptibles d'être modifiés par l'administrateur local RM sont ceux du répertoire */etc/admin* qui sont attribués à l'utilisateur *_admin*, et inscriptibles par ce dernier au sens des permissions discrétionnaires. Ces fichiers sont beaucoup moins nombreux que ceux de la cage $ADMIN_{clip}$, et pour l'heure limités à :

- */etc/admin/hosts* : fichier de résolution de noms statique, correspondant à */etc/hosts* pour toutes

les vues.

- */etc/admin/resolv.conf* : fichier de résolution de noms dynamique, correspondant à */etc/resolv.conf* pour toutes les vues.
- */etc/admin/clip_install/clip_install_rm_apps.conf* : paramètres de l'outil d'installation de paquetages (prise en compte des paquetages à fort impact) pour les paquetages secondaires RM. Voir aussi [CLIP_DCS_13006] et 4.
- */etc/admin/clip_install/clip_install_rm_core.conf* : paramètres de l'outil d'installation de paquetages (prise en compte des paquetages à fort impact) pour les paquetages primaires RM. Voir aussi [CLIP_DCS_13006] et 6.

3 Vue AUDIT

La seule fonction de la vue AUDIT est la collecte des journaux des différentes vues de la cage RM, et leur transmission au démon de collecte principal du système, exécuté dans la cage $AUDIT_{clip}$ ([CLIP_1304]). A la différence de cette dernière, la vue AUDIT ne permet pas la consultation interactive des journaux, ce qui permet de réduire son arborescence au strict minimum.

3.1 Configuration de la vue

Les montages constituant l'arborescence de la vue AUDIT sont résumés dans le Tableau 3. Ils présentent les caractéristiques suivantes :

- La vue dispose d'une racine dédiée, montée depuis l'arborescence UPDATE (ce qui permet sa mise à jour comme un paquetage secondaire RM).
- La vue dispose de son répertoire */dev* dédié en lecture-seule, contenant un ensemble de *devices* adaptés à son fonctionnement, soit :
  - *zero, null, full*
  - *urandom*, ainsi qu'un lien symbolique *random* pointant sur *urandom*
  - un lien symbolique *log* pointant vers la *socket log* créée à la racine de la vue, ce qui permet de ne pas créer de *socket log* directement dans */dev*, qui est en lecture-seule. Cette *socket* n'est en pratique pas utilisée à ce stade, dans la mesure où le seul processus de la vue est *syslog-ng*, qui utilise son interface interne pour la journalisation.
  - aucun pseudo-terminal, en l'absence d'utilisations interactives de la vue.

Par ailleurs, à la différence des autres vues, la vue AUDIT n'intègre aucun exécutable ni bibliothèque dans son arborescence. En effet, le seul processus de la vue est exécuté à l'origine en dehors de celle-ci (cf. 3.2), et ne fait appel à aucun exécutable externe pendant son fonctionnement.

Source	Point de montage	Type	Options
Montages de */etc/jails/rm_X/fstab.internal* Les sources sont relatives à la racine de la cage RM, les points de montages à la racine de la vue.			
/update_priv/audit_root	*/*	*bind*	*ro,nodev,noatime*
/update/etc/shared	*/etc/shared*	*bind*	*ro,nosuid,nodev,noexec,noatime*
/audit_priv/var	*/var*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/admin_priv/etc.admin	*/etc/admin*	*bind*	*ro,nosuid,nodev,noexec,noatime*
Montages de */etc/jails/rm_X/fstab.external* Les sources sont relatives à la racine du socle CLIP, les points de montages à la racine de la vue.			
/etc/core	*/etc/core*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
/mounts/vsdev/rm_X/audit_devs	*/dev*	*bind*	*ro,nosuid,noexec,noatime*

Tableau 3: Montages de la vue AUDIT.

3.2 Fonctionnement de la vue

Le démon *syslog-ng* de la vue AUDIT doit ouvrir une *socket* de collecte dans chacune des vues de la cage, ainsi qu'à la racine de celle-ci. Afin de ne pas exposer des portions des arborescences de ces différents compartiments dans la vue, il est nécessaire d'adopter un principe de fonctionnement similaire à celui de $ADMIN_{clip}$, dans lequel le démon est lancé en dehors de la vue, et crée ses *sockets* de collecte avant de s'enfermer dans celle-ci. L'enfermement dans une vue étant réalisé par un simple *chroot()*, cette manipulation ne nécessite aucune modification de *syslog-ng*, qui supporte dans sa version standard une option *-C <path>*, permettant de l'enfermer par *chroot* dans *<path>*. Ainsi, le service AUDIT est démarré à la racine de la cage RM, lors de l'activation de celle-ci par *clip_servers*, en exécutant le fichier */update/usr/sbin/syslog-ng* (installé par un paquetage primaire RM) avec l'argument *-C /audit*.

Le fichier de configuration */update/etc/syslog-ng/syslog-ng.conf* utilisé par ce démon spécifie l'ouverture de quatre *sockets unix* de type *stream* pour la collecte de journaux, une dans chaque vue. Une difficulté supplémentaire est introduite par l'impossibilité d'écrire dans les répertoires */dev*, montés en lecture seule, de ces différents compartiment. Ainsi, *syslog-ng* ouvre sa *socket log* à la racine de chacun des compartiments, tandis qu'un lien symbolique dans chaque répertoire */dev* redirige automatiquement chaque ouverture de */dev/log* par un client vers */log*. Pour ouvrir ces *sockets*, *syslog-ng* utilise les répertoires accessibles en lecture-écriture depuis la racine de la cage qui sont associés au racines de vues, par exemple */update_root* plutôt que */update*, ou */update_priv/user_root* plutôt que */user*. Par ailleurs, ces quatre *sockets* sont complétées par un lien symbolique */dev/log* à la racine de la cage, pointant sur */update/log* et permettant aux processus lancés à la racine (uniquement *jailmaster*) de journaliser leurs actions.

Enfin, la configuration *syslog-ng* spécifie une seule destination pour les journaux, correspondant à la *socket unix* de type *datagram /var/run/logger* (à la racine de la cage), qui est elle même une *socket* de collecte du démon *syslog-ng* principal du système, exécuté dans la cage $AUDIT_{clip}$ (cf. [CLIP_1304]).

Le démon de la vue AUDIT transmet directement tous les journaux de niveau de priorité autre que *debug* sur cette *socket*, sans réaliser ni filtrage ni stockage local des journaux. Les journaux de niveau *debug* sont en revanche perdus.

Après son enfermement dans la vue, le démon *syslog-ng* poursuit son exécution sous l'identité *syslog:syslog*. Cette séquence de lancement est résumée dans la Figure 2.

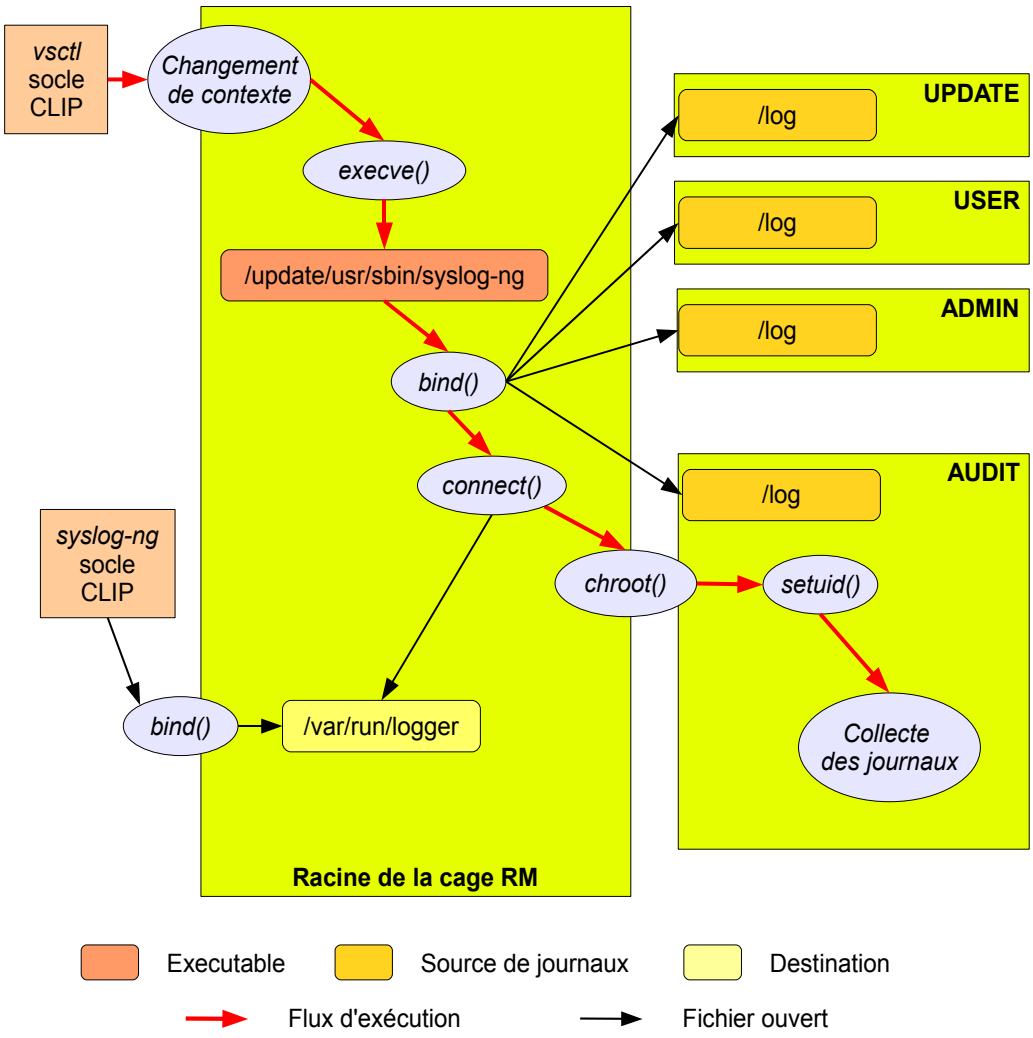

Figure 2: Lancement du service AUDIT dans une cage RM.

4 Vue UPDATE

La vue UPDATE réalise l'installation périodique des mises à jour de paquetages secondaires RM, qui lui sont mis à disposition après téléchargement par $UPDATE_{clip}$. Elle fonctionne de manière entièrement automatique, sans sessions interactives.

4.1 Configuration de la vue

Montages

Les montages constituant l'arborescence de la vue UPDATE sont résumés dans le Tableau 4. Ils présentent les caractéristiques suivantes :

- La vue dispose d'une racine dédiée, montée en lecture seule depuis un répertoire accessible en écriture depuis la racine RM. Cette racine UPDATE est rattachée au coeur de la cage, et ne contient que des fichiers installés par des paquetages primaires.
- La vue dispose de son répertoire */dev* dédié en lecture-seule, contenant un ensemble de *devices* adaptés à son fonctionnement, soit :
  - *zero, null, full*
  - *urandom*, ainsi qu'un lien symbolique *random* pointant sur *urandom*
  - un lien symbolique *log* pointant vers la *socket log* créée à la racine de la vue (cf. 3), ce qui permet de ne pas créer de *socket log* directement dans */dev*, qui est en lecture-seule.
  - Aucun pseudo-terminal, en l'absence de sessions interactives.
  - Le *device veriexec*, permettant la mise à jour de la base d'entrées de la cage lors de l'installation de mises à jour (de paquetages secondaires uniquement).
- La vue dispose d'un accès en écriture au répertoire principal d'installation des paquetages secondaires, */usr/local*, qui est aussi exposé en lecture seule dans USER, ainsi que d'un accès en écriture aux racines des autres vues. De même, la vue dispose d'un accès en écriture à */etc/admin*, afin d'y mettre à jour les éventuels fichiers de configuration modifiables localement qui sont intégrés à des paquetages RM.
- Bien que cela n'apparaisse pas dans le tableau, le répertoire */pkgs* de l'arborescence privée de la vue est aussi monté dans la cage $UPDATE_{clip}$ ([CLIP_1304]), qui y place les miroirs de paquetages téléchargés pour le compte de la cage RM ([CLIP_DCS_14009]).

Paquetages

Le coeur de la vue contient les exécutables spécifiques à celle-ci, installés par un ensemble de paquetages primaires (accompagnés de leurs dépendances, comme la *glibc*) :

- *busybox-update* fournit les utilitaires de base de la vue, en particulier des versions simples de *sh, sed*, *awk*, *grep*, etc, ainsi que le démon *crond* qui constitue le principal processus de la cage. La configuration utilisée par ce paquetage est la même que celle de la cage $UPDATE_{clip}$

([CLIP_1304]). Le *shell busybox* utilise le drapeau *O_MAYEXEC*, spécifique à CLIP ([CLIP_1201]), lors de l'ouverture de fichiers, ce qui interdit l'exécution ou le "sourçage" de scripts depuis des montages portant l'option *noexec*.

- *vericti* fournit l'outil de mise à jour des entrées *veriexec* ([CLIP_1201]), appelé par les *maintainer-scripts* des paquetages mis à jour ([CLIP_1101]).
- *dpkg, tar* et *apt* fournissent les outils d'installation de paquetages et de gestion des dépendances, complétés par *clip-install-common* et *clip-install-rm*, qui fournissent les scripts constituant la surcouche spécifique à CLIP de ces outils ([CLIP_DCS_13006]).

Source	Point de montage	Type	Options
Montages de */etc/jails/rm_X/fstab.internal* Les sources sont relatives à la racine de la cage RM, les points de montages à la racine de la vue.			
/update_root	*/*	*bind*	*ro,nodev,noatime*
/update_priv/var	*/var*	*bind*	*rw,nosuid,nodev,noatime*
/update_priv/var/shared	*/var/shared*	*bind*	*rw,nosuid,nodev,noatime,nolock, nosymfollow*
/update_priv/tmp	*/tmp*	*bind*	*rw,nosuid,nodev,noexec,noatime, mode=1777*
/update_priv/usr_local	*/usr/local*	*bind*	*rw,nodev,noatime*
/update_priv/user_root	*/user_root*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/update_priv/admin_root	*/admin_root*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/update_priv/audit_root	*/audit_root*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/update_priv/pkgs	*/pkgs*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/admin_priv/etc.admin	*/etc/admin*	*bind*	*rw,nosuid,nodev,noexec,noatime*
Montages de */etc/jails/rm_X/fstab.external* Les sources sont relatives à la racine du socle CLIP, les points de montages à la racine de la vue.			
/etc/core	*/etc/core*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
/mounts/vsdev/rm_X/update_devs	*/dev*	*bind*	*ro,nosuid,noexec,noatime*

Tableau 4: Montages de la vue UPDATE.

4.2 Fonctionnement de la vue

Le principal processus de la vue est un démon *cron*, lancé au démarrage de la cage RM par *jailmaster*. La seule tache périodique invoquée par ce démon est à ce stade la mise à jour des paquetages secondaires RM, en invoquant le script *clip-install* ([CLIP_DCS_13006]) toutes les heures piles. Ce script procède à l'installation des nouveaux paquetages secondaires disponibles dans */pkgs*, en tenant compte des paramètres locaux définis dans */etc/admin/clip_install/clip_install_rm_apps.conf* . Les sorties standard de ce script sont inscrites dans un fichier du */var/log* privé de la vue, qui n'est pas utilisé ni consultable sur un poste en production, mais peut être analysé en phase de debug ou de test.

5 Vue USER

5.1 Configuration de la vue

Montages

Les montages constituant l'arborescence de la vue USER sont résumés dans le Tableau 5. Ils présentent les caractéristiques suivantes :

- La vue dispose d'une racine dédiée, montée depuis l'arborescence UPDATE (ce qui permet sa mise à jour comme un paquetage secondaire RM).
- La vue dispose de son répertoire */dev* dédié en lecture-seule, contenant un ensemble de *devices* adaptés à son fonctionnement, en particulier les pseudo-terminaux utiles dans son utilisation interactive. Plus précisément, ces périphériques sont :
  - *zero, null, full*
  - *urandom*, ainsi qu'un lien symbolique *random* pointant sur *urandom*
  - un lien symbolique *log* pointant vers la *socket log* créée à la racine de la vue (cf. 3), ce qui permet de ne pas créer de *socket log* directement dans */dev*, qui est en lecture-seule.
  - Les pseudo-terminaux *tty* et *ptmx*, complétés par un montage du système de fichiers *devpts* sur */dev/pts*.
- Le système de fichiers *proc* est monté dans la cage afin de permettre son utilisation interactive (il est en particulier indispensable au fonctionnement de *java*). Ce système de fichiers est monté en *bind* depuis le */proc* du socle CLIP, plutôt que directement, pour les raisons exposées en 2.1. Le cloisonnement *vserver* et *grsecurity* le limite pour l'essentiel aux */proc/<pid>* des processus de la vue.
- La vue dispose d'un accès en lecture seule aux exécutables et bibliothèques de la vue UPDATE, qui lui fournissent un environnement logiciel riche, adapté à son utilisation interactive.
- Les montages statiques de la vue, réalisés au lancement de la cage RM, sont complétés par des montages temporaires, qui peuvent être réalisés :
  - à chaque ouverture de session par un utilisateur : partition chiffrée de l'utilisateur, et système de fichiers */tmpfs* monté sur */tmp,* comme cela est détaillé dans le document [CLIP_1302].
  - lorsque l'utilisateur d'une session en cours connecte et monte un support USB sécurisé du niveau de la cage RM considérée, comme cela est détaillé dans le document [CLIP_DCS_15088].

  Ces montages sont réalisés par le socle CLIP dans le *namespace VFS* de la cage RM concernée. Les éventuels *devices* correspondants (dans le cas des systèmes de fichiers chiffrés) sont temporairement copiés dans le */dev* de la racine de la cage RM, le temps de réaliser l'opération de montage. Ils ne sont jamais directement exposés dans les vues de la cage.

Source	Point de montage	Type	Options
Montages de */etc/jails/rm_X/fstab.internal* Les sources sont relatives à la racine de la cage RM, les points de montages à la racine de la vue.			
/update_priv/user_root	*/*	*bind*	*ro,nodev,noatime*
/update/usr	*/user/usr*	*bind*	*ro,nosuid,nodev,noatime*
/update_priv/usr_local	*/user/usr/local*	*bind*	*ro,nosuid,nodev,noatime*
/update/lib	*/lib*	*bind*	*ro,nosuid,nodev,noatime*
/update/etc/shared	*/etc/shared*	*bind*	*ro,nosuid,nodev,noexec,noatime*
/user_priv/home	*/home*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/user_priv/var	*/var*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/update_priv/var/shared	*/var/shared*	*bind*	*rw,noexec,nodev,nosuid,noatime*
/admin_priv/etc.admin	*/etc/admin*	*bind*	*ro,nosuid,nodev,noexec,noatime*
Montages de */etc/jails/rm_X/fstab.external* Les sources sont relatives à la racine du socle CLIP, les points de montages à la racine de la vue.			
/etc/core	*/etc/core*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
/mounts/vsdev/rm_X/user_devs	*/dev*	*bind*	*ro,nosuid,noexec,noatime*
/proc	*/proc*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
<none>	*/tmp*	*tmpfs*	*rw,nosuid,nodev,noexec,noatime, size=16m,mode=1777*
<none>	*/dev/pts*	*devpts*	*rw,nosuid,noexec,noatime,nolock, gid=5,mode=0620*
Montages temporaires			
/dev/mapper/ _home_parts_rmX_<user>.part	*/home/user* (session)	*ext2*	*rw,nosuid,nodev,noexec,noatime*
<usertmp>	*/tmp* (session)	*tmpfs*	*rw,nosuid,nodev,noexec,noatime, mode=1777*
/dev/mapper/stockage_amovible _<user>_rmX	*/mnt/usb* (à la demande)	*vfat*	*rw,nosuid,nodev,noexec,uid=<user>, gid=<user>*

Tableau 5: Montages de la vue USER.

Paquetages

Un paquetage *busybox-user,* obtenu en compilant *sys-apps/busybox* avec la variable *DEB_NAME_SUFFIX* positionnée à *user,* fournit les utilitaires de base de la vue. Ceux-ci sont limités à un *shell* compatible *ash*, et à un ensemble d'utilitaires qui peuvent être invoqués par des scripts (comme par exemple le script *startkde*). Le *shell* de la vue n'a pas vocation à être utilisé interactivement, et ne comprend de ce fait pas de fonctions d'auto-complétion ou d'historique. Pour la même raison, la configuration *user* de *busybox* n'inclut aucun utilitaire utilisable uniquement en mode interactif, comme c'est le cas par exemple d'un éditeur de texte comme *vi*. Par ailleurs, le *shell busybox* utilise le drapeau *O_MAYEXEC*, spécifique à CLIP ([CLIP_1201]), lors de l'ouverture de fichiers, ce qui interdit l'exécution ou le "sourçage" de scripts depuis des montages portant l'option *noexec,* en particulier le répertoire *$HOME* de l'utilisateur.

Ces utilitaires de base sont complétés par l'environnement logiciel riche mis à disposition par la vue UPDATE, et accessible sous */lib, /usr* (paquetages primaires) et */usr/local* (paquetages secondaires). Dans la configuration standard, cette arborescence comprend principalement :

- Un serveur X11 VNC *tightvnc* ([TIGHTVNC]). Seul l'exécutable serveur est installé dans la cage, du fait d'une adaptation spécifique à CLIP de l'*ebuild net-misc/tightvnc*, qui permet de ne pas installer les exécutables client lorsque le drapeau USE *clip_vncserver* est défini.
- Un environnement de bureau KDE ([KDE]). Au sein de CLIP, cet environnement est modifié pour supprimer les modules qui ne peuvent pas fonctionner dans l'environnement sécurisé d'une cage CLIP, par exemple les modules d'administration de *kcontrol* qui nécessitent des privilèges *root*. De plus, la configuration KDE par défaut est adaptée par l'installation du paquetage *clip-data/kde-config-rm*, qui installe une arborescence partielle de configuration KDE dans le répertoire */usr/local/etc/kde* (qui est ajouté à KDEDIRS, afin d'être pris en compte par KDE, cf. [KDE_ADM]). Cette arborescence ajuste certains paramètres par défaut, notamment :
  - utilisation du double-clic plutôt que du simple-clic par défaut
  - ajout d'*applets* adaptés au panneau *kicker* et d'icônes sur le bureau
  - fond d'écran par défaut

  Par ailleurs, le script *startkde* est adapté de manière à ne pas chercher à exécuter de fichiers ou scripts du répertoire *$HOME* de l'utilisateur (ce qui serait de toutes manières interdit par l'option *noexec* du montage correspondant, et l'utilisation du *flag O_MAYEXEC* par le *shell* de la cage), et à ne jamais lancer l'application *kpersonnalizer* de personnalisation initiale de l'environnement.
- Un ensemble d'applicatifs utilisateur, notamment *openoffice, firefox, thunderbird*, ainsi qu'un *runtime java-1.6*.

5.2 Fonctionnement de la vue

5.2.1 Service USER

En dehors des sessions utilisateur, le seul processus de la vue est le démon *jailmaster*, qui s'y enferme après avoir lancé les services ADMIN et UPDATE. Ce démon est en attente d'une demande d'ouverture de session sur la *socket unix /var/run/start* de la vue. Cette *socket* est aussi exposée dans la vue visionneuse associée à la cage RM considérée au sein de la cage $USER_{clip}$ ([CLIP_1304]), ce qui permet le lancement de requêtes de session depuis cette cage.

Lorsqu'il reçoit une telle requête, le démon *jailmaster* accepte la demande de connexion puis lit les identifiants (*uid,gid*) de l'utilisateur à l'origine de la requête par un appel *getsockopt()*. Il crée ensuite un fils chargé de lancer la session utilisateur, sous cette identité. Ce fils détermine (en utilisant pour cela les fichiers *passwd* et *group* de */etc/core*) si l'utilisateur est membre du groupe *rm_admin* ou non, et adapte le type de session lancé en conséquence : session USER-ADMIN ou session USER simple respectivement. Dans les deux cas, la session incorpore un serveur X11 VNC, auquel se connectent les différents clients graphiques de la session. La nature de ces clients diffère en revanche en fonction du type de session :

- La session USER-ADMIN est limitée à un unique terminal *xterm*, en plein écran (pour l'écran virtuel du serveur VNC), qui exécute directement (et uniquement : *xterm* se termine en même temps que la commande) un client *ssh* pour se connecter en tant que *_admin* dans la vue ADMIN, à travers la boucle locale.
- La session USER normale lance un environnement de bureau KDE complet par une commande *startkde*, après avoir défini un certain nombre de variables d'environnement (*PATH, HOME, LC_ALL, KDEDIR*, etc.).

Le fils de *jailmaster* exécute le script de session adapté, après des appels *setgid()* et *setuid()* lui permettant de prendre l'identité de l'utilisateur. Le démon *jailmaster* attend son code de retour, qui se produit assez rapidement, car le script de session rend la main après avoir lancé la session en tâche de fond, comme décrit plus bas. Le démon *jailmaster* écrit alors un unique caractère sur la *socket /var/run/start* connectée par l'utilisateur. Un *'Y'* est écrit lorsque le traitement a abouti sans erreur (code retour nul pour le lancement de session), un *'N'* est écrit dans le cas contraire. Ce comportement est résumé dans la Figure 3.

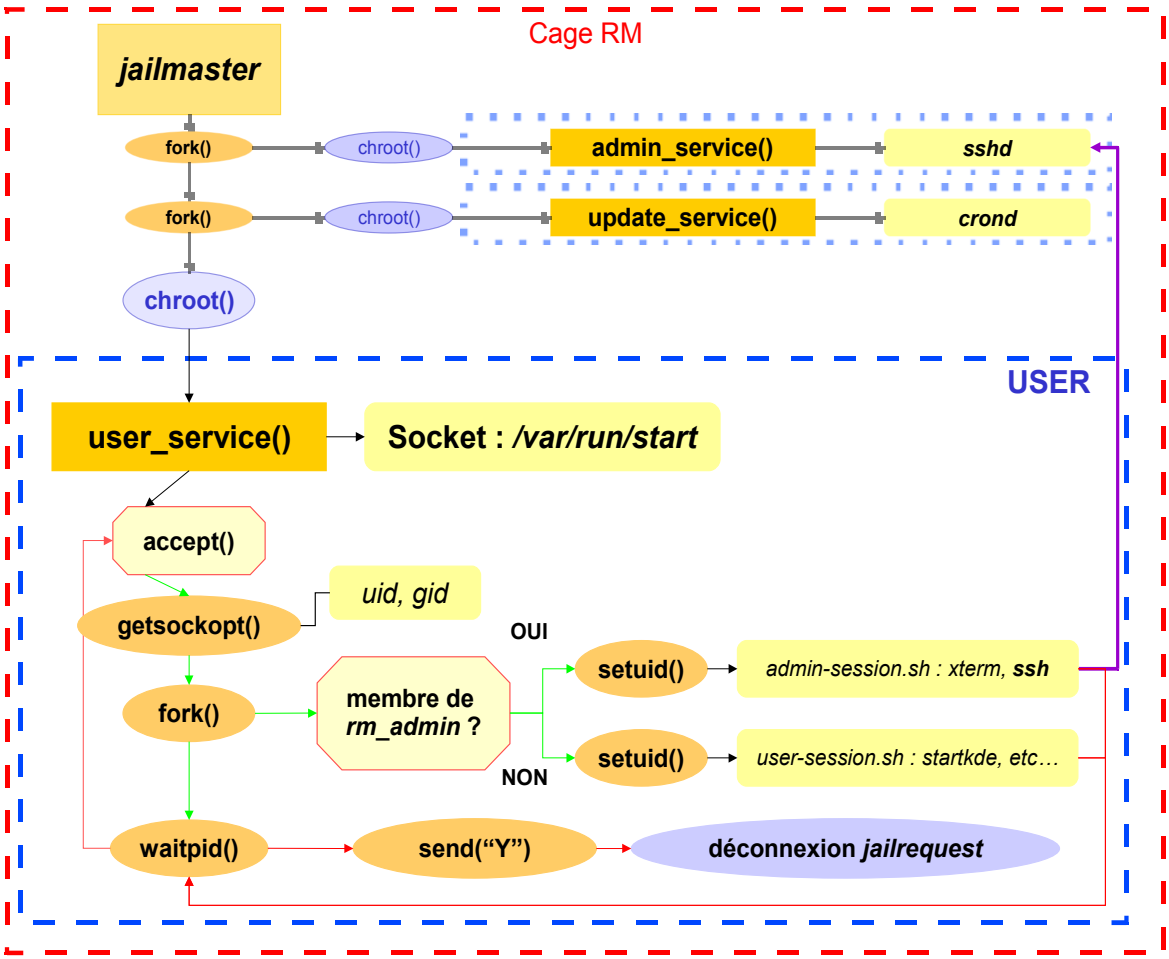

Figure 3: Principe du service USER.

5.2.2 Ouverture de session par $USER_{clip}$

L'ouverture d'une session (USER ou USER-ADMIN) dans une cage RM est réalisée à l'initiative de la cage $USER_{clip}$. Elle se déroule en deux étapes, pilotées dans $USER_{clip}$ par le script */usr/local/bin/rm_X_session.sh*. Dans une premier temps, une connexion auprès de *jailmaster* permet d'initialiser une session au sein de la cage RM, en lançant en particulier le serveur X11 VNC de la vue USER de cette cage. Dans un second temps, un client VNC vient, depuis $USER_{clip}$, se connecter à ce serveur X11, afin de donner accès à la session RM. Plus précisément, le déroulement de cette ouverture de session est le suivant :

- ***(USER$_{clip}$)*** Le script *rm_X_session.sh* est lancé dans $USER_{clip}$, typiquement par *fbpanel* lors d'un clic sur un bouton lanceur RM.
- ***(USER$_{clip}$)*** Ce script invoque l'utilitaire *jailrequest* en lui passant en argument le chemin de la *socket /viewers/rm_X/vserver/start*, correspondant à la *socket /user/var/run/start* créée par *jailmaster* dans la cage RM. *jailrequest* se connecte à cette *socket* et attend d'y lire un caractère en retour.
- ***(RM - vue USER)*** Sur réception de la connexion, le démon *jailmaster* "*forke*" un processus fils, qui lance la session RM appropriée, en invoquant *xinit* pour lancer d'une part le serveur X11 VNC et d'autre part la session cliente. Le serveur VNC est adapté pour écouter uniquement sur une *socket* de type *unix* (cf. [CLIP_1303]), et paramétré pour écouter sur la *socket /var/run/vnc/vnc1* (visible sous */viewers/rm_X/vserver/run/vnc/vnc1* dans $USER_{clip}$), et accepter exactement une et une seule connexion sur cette *socket,* authentifiée par un mot de passe trivial placé dans */var/run/vnc/vncpasswd*[9].
- ***(RM - vue USER)*** Le fils de lancement de session de *jailmaster* se termine dès que la session est lancée en tâche de fond. *jailmaster* écrit alors sur la *socket* connectée un caractère unique, 'Y' ou 'N'.
- ***(USER$_{clip}$)*** *jailrequest* se termine dès qu'il reçoit ce caractère. Le code de retour est fonction du caractère reçu : un code de succès (0) est rendu pour 'Y', et un code d'erreur pour 'N'. Dans ce dernier cas, le traitement s'interrompt immédiatement dans $USER_{clip}$.
- ***(USER$_{clip}$)*** En cas de succès de *jailrequest*, un *cookie Xauthority* de niveau approprié pour RM_X (cf. [CLIP_1303]) est généré dans */viewers/rm_X/xauth* (plus précisément, *dans /xauth/rm_X*, qui est une vue en lecture-écriture de ce répertoire au sein de $USER_{clip}$*),* puis *viewer-launch* est invoqué pour lancer */bin/viewer.sh* dans la vue visionneuse RM_X, c'est-à-dire enfermé par *chroot* dans */viewers/rm_X* (cf. [CLIP_1304]).
- ***(USER$_{clip}$ - vue visionneuse)*** Le script */bin/viewer.sh* attend l'apparition d'une *socket /vserver/vnc/vnc1* pendant un maximum de dix secondes. Si aucune *socket* n'apparaît, le script se termine immédiatement et le traitement est interrompu dans $USER_{clip}$. Sinon, un client VNC *vncviewer* est lancé pour se connecter à */vserver/run/vnc/vnc1*, avec le mot de passe contenu dans */vserver/run/vnc/vncpasswd*.

[9] Ce mot de passe, qui est requis pour le fonctionnement de *Xvnc*, ne joue aucun rôle dans la sécurité du poste, dans la mesure où la connexion *vnc* est purement locale, et contrôlée par les accès au système de fichiers.

- ***(RM - vue USER)*** Le serveur *Xvnc* reçoit et accepte la connexion sur */var/run/vnc/vnc1*. Il n'accepte aucune autre connexion après celle-ci, et se termine automatiquement si le client se déconnecte (cas de la terminaison de session depuis $USER_{clip}$). Le serveur est par ailleurs aussi terminé par *xinit* lorsque la session cliente X11 dans USER se termine (cas de la terminaison depuis RM/USER).

Cette procédure est reprise dans la Figure 4.

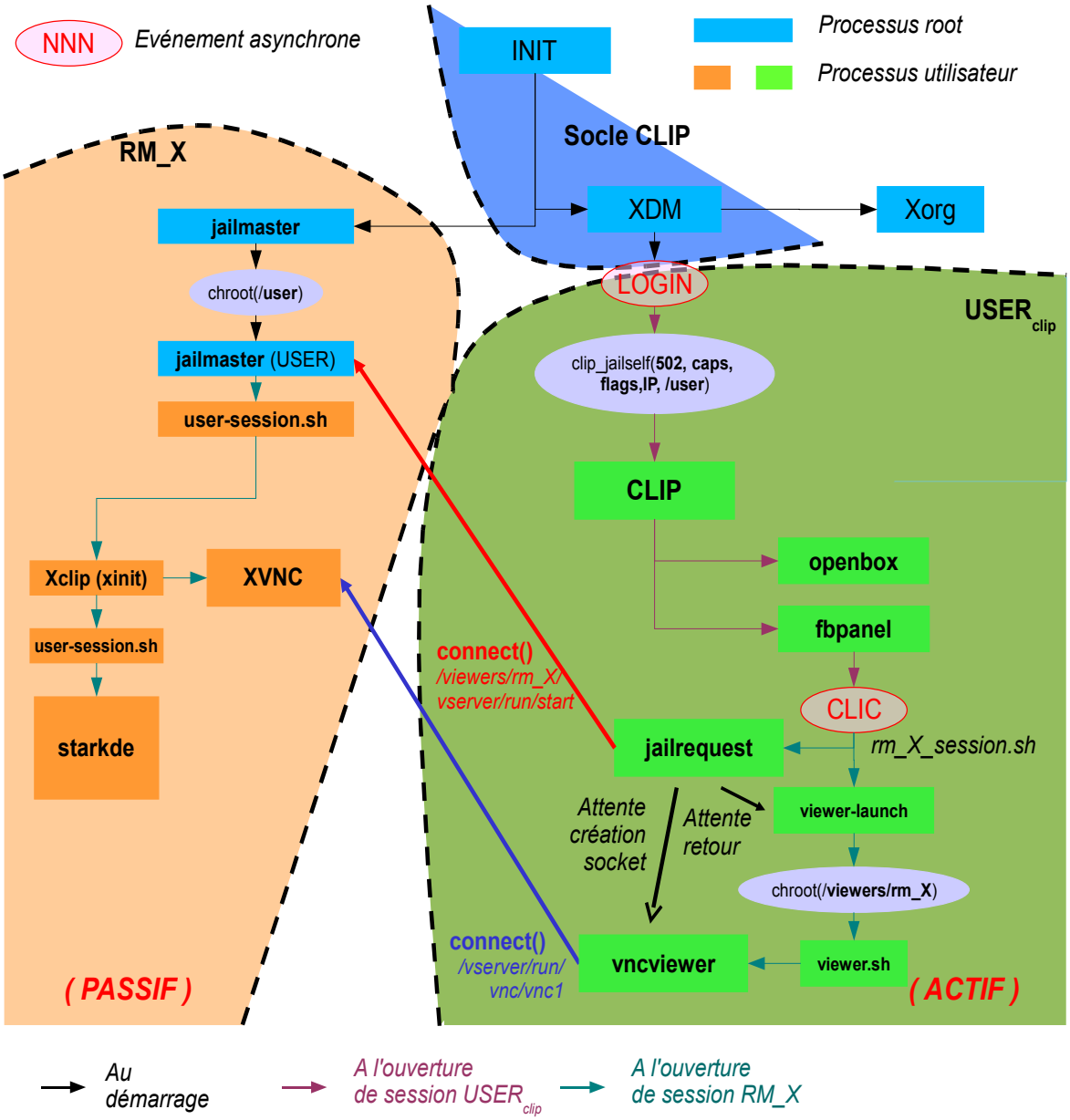

Figure 4: Lancement de session utilisateur dans une cage RM

5.2.3 Session USER

Ce dernier paragraphe décrit la session lancée au sein de la vue USER par un fils de *jailmaster*, suite à la réception d'une requête issue de $USER_{clip}$. Cette session est lancée en plusieurs étapes, afin de répondre à plusieurs contraintes potentiellement contradictoires :

- Le fils de *jailmaster* qui exécute le lancement de session doit rendre la main assez rapidement, afin de permettre à *jailmaster* de répondre à *jailrequest.*
- Un script doit cependant attendre la fin de la session USER (et non de sa seule initialisation), de manière à être en mesure d'exécuter d'éventuelles opérations de nettoyage en fin de session (par exemple, suppression de la *socket /var/run/vnc/vnc1,* si elle n'a pas été supprimée par le serveur *Xvnc*). De plus, ce script ne doit pas utiliser le *$HOME* (*/home/user*) ni le *$TMP* (*/tmp*) de l'utilisateur, afin de ne pas être tué par le socle lors du démontage de ces montages de session (cf. [CLIP_1302]).
- Enfin, les autres programmes de la session USER ont intérêt à utiliser le *$HOME* et / ou le *$TMP* de l'utilisateur, pour être automatiquement terminés lors de la fin de session utilisateur (si la terminaison de leur serveur X11 ne suffit pas à entraîner la leur).

Afin de concilier ces différents impératifs, le lancement d'une session USER est réparti entre trois scripts, qui s'appellent successivement, dans cet ordre :

- Un script à retour immédiat, qui est celui lancé par *jailmaster.* Ce script réalise quelques vérifications sur le type de compte utilisateur pour lequel la session doit être ouverte. Il rejette en particulier le compte *root*, et les comptes non définis (dans */etc/core/passwd*). Si l'utilisateur est accepté, la variable SESSION est définie avec le chemin du script de session cliente à invoquer en troisième étape. Puis le script */usr/local/bin/clip-update-user-data* est invoqué. Ce script va simplement exécuter dans l'ordre alphabétique tous les scripts trouvés dans le répertoire */usr/local/bin/clip-user-data-update-scripts/,* répertoire qui peut être utilisé par tout paquetage secondaire qui, suite à une mise à jour, devrait modifier les données utilisateurs (typiquement des fichiers de configuration personnalisés) stockées dans les différents répertoires *$HOME* chiffrés, qui ne sont pas accessibles avant l'ouverture de session. Enfin, le script lance en tâche de fond le script */usr/local/bin/XClip* décrit ci-dessous, puis se termine immédiatement afin de permettre à *jailmaster* de répondre à *jailrequest*. Deux scripts à retour immédiat peuvent être invoqués par *jailmaster*, selon que l'utilisateur appartient ou non au groupe *rm_admin*. Dans le premier cas (session USER-ADMIN), le script */usr/local/bin/admin-session.sh* est lancé, et positionne *SESSION* à */usr/local/bin/xterm-session.sh*. Dans le second cas (session USER normale), */usr/local/bin/user-session.sh* est lancé et positionne *SESSION* à */usr/local/kde/3.5/bin/startkde*. La définition de *SESSION* constitue la seule différence entre les deux scripts, qui lancent ensuite le même script *XClip.* Ce premier script assure aussi le positionnement d'un certain nombre de variables d'environnement utiles par la suite, en particulier :
  - *SHELL*, définie à */bin/sh*
  - *LANG* et *LC_ALL*, définies à *fr_FR* afin d'assurer une localisation en français de la session utilisateur.
  - *HOME* définie à */home/user*, quel que soit l'utilisateur. Ce répertoire est le point de montage de la partition chiffrée de chaque utilisateur.

- *PATH* est défini de manière à contenir */usr/local/bin*, répertoire dans lequel sont installés l'essentiel des applicatifs utilisateur.
- Dans le cas d'une session utilisateur normale, *KDEDIRS* est positionné de manière à prendre en compte l'arborescence de configuration située sous */usr/local/etc/kde,* dans laquelle *kde-config-rm* installe ses fichiers.

- Un script *XClip*, exécuté en tâche de fond et qui attend la terminaison de la session. Ce script, comme le précédent, est lancé à la racine de la vue USER, et n'utilise de manière durable ni le *$HOME* ni le *$TMP* de l'utilisateur, ce qui fait qu'il n'est pas tué en fin de session. Il génère un *cookie* et l'autorisation d'accès *Xauthority* associée pour la cage[10], puis appelle *xinit* pour lancer simultanément le serveur *Xvnc* et la session cliente, définie par la variable d'environnement *SESSION*, telle qu'elle a été positionnée par le script précédent. Ce lancement par *xinit* (couplé avec le fait que *xinit* ne soit pas tué automatiquement en fin de session) assure la terminaison du serveur *Xvnc* lorsque la session cliente se termine. Le script *XClip* attend la terminaison de *xinit*, puis supprime un certain nombre de fichiers et répertoires temporaires[11] avant de se terminer lui même.
- Le script de session cliente est celui qui est exécuté par *xinit* en même temps que *Xvnc*. Sa terminaison entraîne celle de l'ensemble de la session. Le chemin de ce script est celui passé à *XClip* par la variable *SESSION*. Il s'agit donc du script *startkde* adapté à CLIP dans le cas d'une session USER normale, et d'un script spécifique *xterm-session.sh* dans le cas d'une session USER-ADMIN. Ce dernier se contente de lancer un terminal *xterm* en lui faisant exécuter directement la commande *ssh _admin@127.0.01* permettant de se connecter dans la vue ADMIN, et en le faisant se terminer immédiatement au retour de celle-ci. Dans les deux cas, le script de session cliente commence son traitement en se positionnant dans *$HOME*, et lance des clients graphiques qui utilisent *$TMP* en particulier pour se connecter à la *socket /tmp/.X11-unix/X0.* Tous ces processus sont donc tués par le socle CLIP (*SIGTERM*, puis *SIGKILL*) en fin de session, afin de permettre le démontage de ces montages de session.

Ces trois étapes sont résumées dans la Figure 5.

[10] *Cookie* défini dans le domaine de sécurité *Trusted* de l'extension *Xsecurity*, qui est le seul domaine utilisé au sein des cages RM, contrairement aux cages CLIP (cf. [CLIP_1303]).

[11] En particulier la *socket /var/run/vnc/vnc1*, qui est créée sous l'identité de l'utilisateur de la session dans un répertoire en mode 1777. Si cette *socket* n'est pas supprimée en fin de session, il sera impossible de lancer ensuite une session sous un autre compte, faute de pouvoir la supprimer alors.

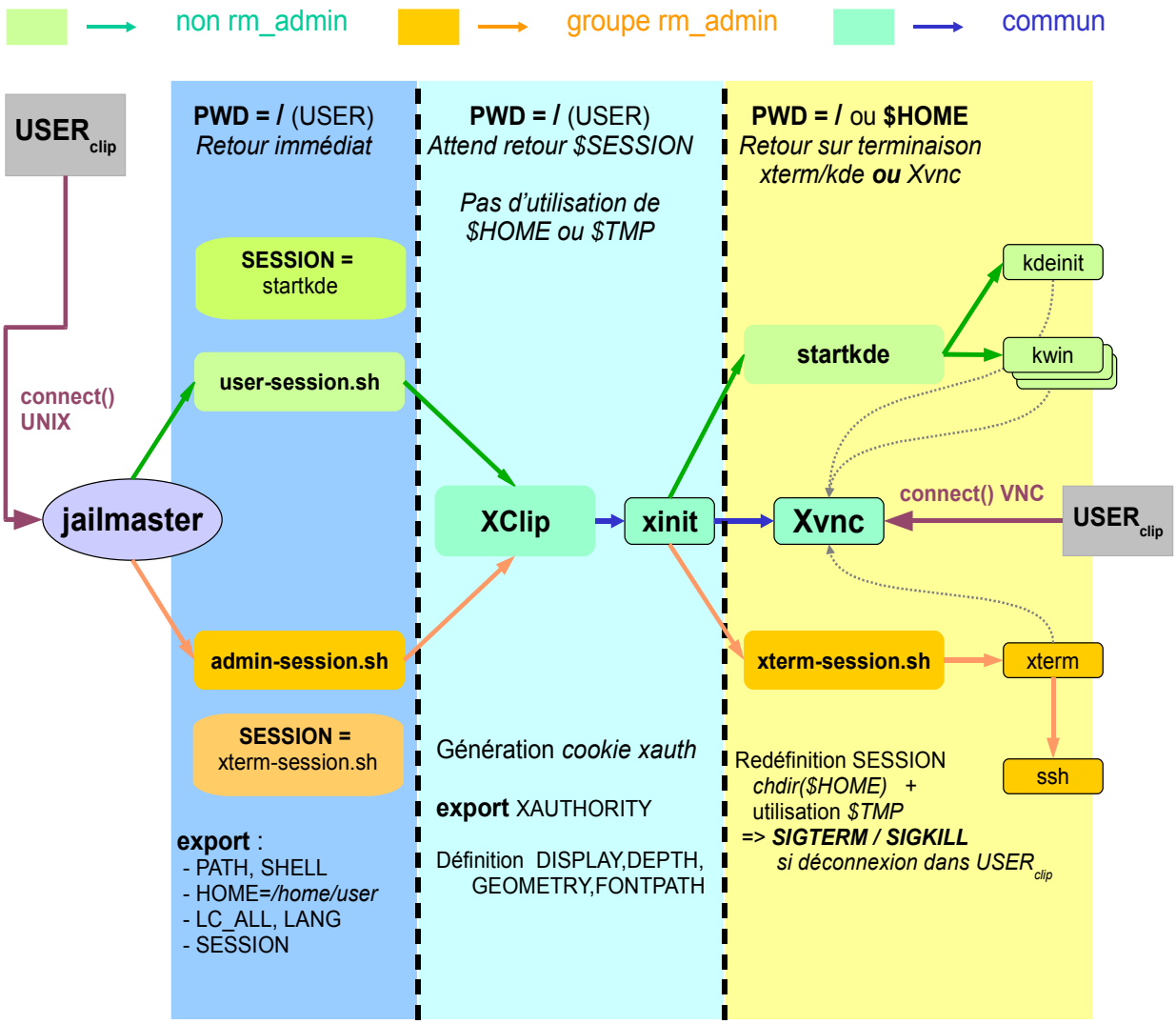

Figure 5: Séquence de lancement d'une session dans la vue USER.

Deux enchaînements d'appels sont possibles, selon que l'utilisateur est membre du groupe *rm_admin* ou non. Les trois sections représentées selon l'axe horizontal sont à interpréter comme suit :

- **section de gauche : exécutables à retour immédiat**
- **section centrale : exécutables qui rendent la main et attendent la terminaison de leurs fils (peuvent définir des *traps*)**
- **section de droite : exécutables qui sont tués par le socle en cas de fin de session dans celui-ci**

6 Cages SECURE_UPDATE_RM_X

Une cage SECURE_UPDATE_RM_X est associée à chaque cage RM_X, afin de réaliser les mises à jours de paquetages primaires RM de cette dernière. Les deux cages partagent le même numéro de contexte *vserver*, et sont par conséquent exclusives l'une de l'autre : la cage SECURE_UPDATE_RM_X ne peut être active que lorsque la cage RM_X est inactive, et réciproquement.

6.1 Configuration de la cage

6.1.1 Configuration *vserver*

La configuration *vserver* d'une cage SECURE_UPDATE_RM_X est dans une large mesure identique à celle de la cage RM_X à laquelle elle est associée, mais avec une racine ramenée à la racine de la vue UPDATE de cette dernière, soit :

- numéro de contexte *vserver* : 1001 pour SECURE_UPDATE_RM_H, 1002 pour SECURE_UPDATE_RM_B. L'identité des numéros de contexte garantit l'exclusivité des cages SECURE_UPDATE_RM_X et RM_X : l'une ne peut pas être démarrée par erreur par une commande *vsctl start* ou *vsctl setup* pendant que l'autre est active.
- racine égale à la racine de la vue UPDATE de la cage RM_X : */vservers/rm_X/update*.
- commande initiale non définie (c'est-à-dire définie à <*invalid*>), la cage étant démarrée par une séquence *vsctl setup* plutôt que directement par *vsctl start*.
- pas d'adresse IP fixe, l'adresse (unique) de la cage est définie à partir d'un paramètre de configuration dynamique du poste lors de son démarrage. Ces paramètres sont importés depuis le fichier de configuration */etc/admin/conf.d/net.* En particulier, les adresses de RM_H et RM_B sont calculées à partir de RMH_ADDR / RMH_MASK et RMB_ADDR / RMB_MASK, respectivement. Ainsi, les cages SECURE_UPDATE_RM_X ont les mêmes adresses que les cages RM_X associées.
- Les capacités maximales autorisées dans la cage sont les mêmes que celles de la cage RM_X, à l'exception de *CAP_NET_BIND_SERVICE*, qui n'est pas nécessaire en l'absence de démon *ssh,* soit en résumé :
  - *CAP_CHOWN*
  - *CAP_DAC_OVERRIDE*
  - *CAP_DAC_READ_SEARCH*
  - *CAP_FOWNER*
  - *CAP_FSETID*
  - *CAP_KILL*
  - *CAP_SETGID*

- *CAP_SETUID*
- *CAP_SYS_CHROOT*

- Les drapeaux de contexte *vserver* sont, comme pour une cage RM, les drapeaux par défaut listés dans [CLIP_1202].

6.1.2 Montages

L'arborescence d'une cage SECURE_UPDATE_RM_X est la même que celle de la vue UPDATE de la cage RM_X associée, mais avec une racine (/ et */etc/admin*) exposée en lecture-écriture. Les fonctions de mise à jour de paquetages secondaires RM peuvent ainsi être réutilisées (mêmes chemins, mêmes méta-données), en disposant des droits en écriture nécessaires à la mise à jour des paquetages primaires. La cage réutilise aussi les *devices* de la vue UPDATE. En revanche, les répertoires d'installation de paquetages secondaires RM (*/usr/local* et les racines des autres vues) ne sont pas exposés dans la cage, ce qui interdit notamment l'appel d'un exécutable installé par un paquetage secondaire dans l'un des *maintainer-scripts* d'un paquetage primaire. Ces différents montages sont résumés dans le Tableau 6.

Source	Point de montage	Type	Options
Montages de */etc/jails/secure_update_rm_X/fstab.external* Les sources sont relatives à la racine du socle CLIP, les points de montages à la racine de la vue.			
/vservers/rm_X/update_root	*/*	*bind*	*rw,nodev,noatime*
/vservers/rm_X/update_priv/var	*/var*	*bind*	*rw,nosuid,nodev,noatime*
/vservers/rm_X/update_priv/tmp	*/tmp*	*bind*	*rw,nosuid,nodev,noexec,noatime, mode=1777*
/vservers/rm_X/admin_priv/etc.admin	*/etc/admin*	*bind*	*rw,nosuid,nodev,noexec,noatime*
/etc/core	*/etc/core*	*bind*	*ro,nosuid,nodev,noexec,noatime,nolock*
/mounts/vsdev/rm_X/update_devs	*/dev*	*bind*	*ro,nosuid,noexec,noatime*
/vservers/rm_X/var/run/logger	*/dev/log*	*bind*	*rw,nosuid,nodev,noexec*

Tableau 6: Montages d'une cage SECURE_UPDATE_RM_X.

Un point remarquable est le montage en *bind* du */var/run/logger* de la racine de la cage RM_X sur le */dev/log* (c'est-à-dire sur le */log*, vers lequel */dev/log* est un lien symbolique) de la cage SECURE_UPDATE_RM_X. En effet, la mise à jour des paquetages primaires est lancée sans service AUDIT dans la cage SECURE_UPDATE_RM. Afin de permettre malgré cela la collecte des journaux de mise à jour, la *socket* de collecte crée dans l'arborescence RM_X par le démon de collecte principale du système (*syslog-ng* de $AUDIT_{clip}$) est exposée directement aux clients de la cage SECURE_UPDATE_RM_X.

6.1.3 Configuration *veriexec*

Une cage SECURE_UPDATE_RM_X dispose par défaut du contexte *veriexec* créé par */etc/init.d/veriexec* pour la future cage RM_X. Cependant, ce contexte n'est pas actif durant le fonctionnement de SECURE_UPDATE_RM_X, puisqu'il n'est activé qu'au lancement de RM_X (et désactivé lors de son éventuel arrêt). En pratique, aucun des exécutables lancés dans la cage ne nécessite à ce stade de privilèges supplémentaires attribués par *veriexec*.

Par ailleurs, la procédure d'installation des paquetages primaires est lancée avec une variable d'environnement *BOOTSTRAP_NOVERIEXEC* exportée à une valeur non nulle, tout comme pour l'installation initiale du système ou la mise à jour des paquetages primaires CLIP (cf. [CLIP_DCS_13006]). Cette variable a pour effet de désactiver la mise à jour de la base *veriexec* par les *maintainer-scripts* des paquetages installés (cf. [CLIP_1101]). En effet, les entrées *veriexec* associées aux paquetages primaires RM ne sont insérées dans la base *veriexec* que lors du lancement de la cage RM_X (cf. 1.2.1).

6.2 Fonctionnement de la cage

Toutes les cages SECURE_UPDATE_RM du système sont démarrées par un unique script, */etc/init.d/rm_core_install* (*app-clip/clip-install-rm*). La liste des cages à lancer est déduite de la liste des cages RM du système, elle-même lue dans la variable CLIP_JAILS du fichier */etc/conf.d/clip*. Cette variable constitue un paramètre système, qui n'est en aucun cas modifiable par l'administrateur du poste. Le script *rm_core_install* est lancé avant le script *clip_servers* dans la séquence de démarrage, mais son échec n'interdit pas le lancement de ce dernier (pas de dépendance *"need"*, cf. [CLIP_1301]).

Ce script importe de manière sécurisée depuis */etc/admin/conf.d/net* les adresses et longueurs de préfixes réseau attribuées aux cages RM, puis lance l'une après l'autre les cages SECURE_UPDATE_RM en leur attribuant les adresses de sous-réseaux ainsi importées. Chaque cage est lancée selon une procédure en plusieurs étapes, comme cela est le cas pour les cages RM :

- Création, s'il n'existe pas, du fichier */vservers/rm_X/update_root/log*, qui sert ensuite de point de montage de la *socket* de collecte de journaux de la cage.
- Génération d'un *cookie* d'authentification *vsctl*, stocké dans l'environnement du script.
- Création de la cage par une commande *vsctl setup* utilisant ce même *cookie*, en passant comme argument supplémentaire l'adresse à attribuer au contexte réseau de la cage.
- Lancement, par une commande *vericTl enter,* d'un script */bin/cleanup_log.sh* dans la cage. Ce script assure la suppression des fichiers de journaux créés dans le */var* privé de la cage par les procédures d'installation de paquetages RM précédentes. Ces fichiers ne sont pas consultables par les interfaces nominales d'audit du système, et ne sont utilisés que par les développeurs CLIP pour l'analyse d'un éventuel problème de mise à jour.
- Lancement, par une commande *vericTl enter*, du script */usr/bin/clip_install_rm_core*, pour procéder à la mise à jour des paquetages primaires RM.
- Terminaison de la commande *vsctl setup* par une commande *vsctl endsetup* utilisant le même *cookie*. Cette commande n'est lancée qu'au retour de *clip_install_rm_core*, à un moment où *vsctl setup* constitue le seul processus de la cage. De ce fait, elle entraîne directement la terminaison de la cage.

Ainsi, la cage est à la fois démarrée et terminée par la fonction *start()* du script de démarrage */etc/init.d/rm_core_install*. Ce script ne comporte aucune fonction *stop()*, en l'absence d'actions à réaliser à l'arrêt du système.

Le script *clip_install_rm_core* invoque quant à lui le script *clip_install* standard ([CLIP_DCS_13006]), avec des arguments adaptés pour une mise à jour des paquetages primaires RM, et des sorties standard et d'erreur redirigées vers un fichier du */var/log* privé de la cage.

Annexe A Références

[CLIP_1001] *Documentation CLIP – 1001 - Périmètre fonctionnel CLIP*

[CLIP_1002] *Documentation CLIP – 1002 – Architecture de sécurité*

[CLIP_1003] *Documentation CLIP – 1003 – Paquetages CLIP*

[CLIP_1101] *Documentation CLIP – 1101 – Génération de paquetages CLIP*

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

[CLIP_1202] *Documentation CLIP – 1202 – Patch Vserver*

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

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

[CLIP_1205] *Documentation CLIP – 1205 – Implémentation CCSD en couche noyau*

[CLIP_1301] *Documentation CLIP – 1301 – Séquences de démarrage et d'arrêt*

[CLIP_1302] *Documentation CLIP – 1302 – Fonctions d'authentification locale*

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

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

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

[CLIP_DCS_13006] *Spécification fonctionnelle des outils de gestion des mises à jour, CLIP-ST-13000-006-DCS*

[CLIP_DCS_14009] *Spécification fonctionnelle des outils de mise à disposition des mises à jour, CLIP-ST-14000-009-DCS*

[CLIP_DCS_15088] *Conception de l'étude de la problématique de stockage sur support amovible, CLIP-DC-15000-088-DCS*

[BUSYBOX] *Busybox, http://busybox.net/*

[PRIVSEP] *Privilege Separated OpenSSH, http://www.citi.umich.edu/u/provos/ssh/privsep.html*

[PRIV] *Preventing Privilege Escalation,* Niels Provos, Markus Friedl et Peter

Honeyman, *12thUSENIX Security Symposium*.
http://www.citi.umich.edu/u/provos/papers/privsep.pdf

[KDE] *K Desktop Environnement, http://www.kde.org*

[KDE_ADM] *KDE System Administration,*
http://techbase.kde.org/KDE_System_Administration

[TIGHTVNC] *TightVNC, http://www.tightvnc.com*

Annexe B Liste des figures

Annexe C Liste des tableaux

Annexe D Liste des remarques