Documentation CLIP

1301

Séquences de démarrage et d'arrêt

Version	Date	Auteur	Commentaires
1.0.7	27/11/2008	Vincent Strubel	Correction de coquilles.
1.0.6	19/11/2008	Vincent Strubel	Ajout du niveau CLIP à la description de *usbadmin*.
1.0.5	30/07/2008	Vincent Strubel	Convention plus lisible pour les références. Mise à jour de *clip_user*.
1.0.4	23/07/2008	Vincent Strubel	Ajout de la séquence d'arrêt de *clock*. Correction de la taille de la *random seed* pour *urandom*.
1.0.3	16/07/2008	Vincent Strubel	Ajout des montages */home/admin.rmX* à *clip_viewers*.
1.0.2	07/07/2008	Vincent Strubel	Mise à jour des dépendances pour *backupsrv* et *databackupsrv*. Ajout de *mdamd*. Correction mineures de l'ordre de lancement.
1.0.1	12/03/2008	Vincent Strubel	Mise en forme.
1.0	12/12/2007	Vincent Strubel	Version initiale, à jour pour *baselayout-clip-1.7.8, core-services-2.1.15, clip-vserver-2.0.15, etc...*

Table des Matières

Introduction

La séquence de démarrage est un élément crucial de la sécurité d'un système CLIP. En effet, un tel système est à l'initialisation essentiellement équivalent à un système Linux standard. La spécificité majeure de la séquence de démarrage CLIP consiste en un verrouillage progressif du système au fur et à mesure du démarrage de ses services de base, par réduction ou révocation de privilèges et interdiction définitive de certaines opérations. Sur le plan fonctionnel, on peut identifier les principales étapes suivantes, qui correspondent à une augmentation progressive du nombre de fonctionnalités assurées par le poste CLIP :

- Démarrage des connexions réseau.
- Démarrage des cages CLIP (cf. [CLIP_1304]).
- Démarrage des cages RM, le cas échéant (cf. [CLIP_1401]).
- Démarrage de l'interface graphique, et attente de connexions utilisateur.

Parallèlement à cette progression fonctionnelle, on retiendra aussi les étapes suivantes, qui correspondent au verrouillage progressif du système :

- Activation des mécanismes de contrôle d'accès réseau de CLIP-LSM (cf. [CLIP_1201]).
- Chargement des politiques *IPsec* et configuration du pare-feu.
- Configuration de *devctl* et activation du contrôle d'accès correspondant (cf. [CLIP_1201]).
- Configuration et activation de *veriexec* (cf. [CLIP_1201])
- Réduction des masques de capacités maximal et par défaut

L'objet du présent document est de décrire plus précisément l'articulation entre ces deux progressions, ainsi que les spécificités de la séquence d'arrêt du système, qui doit surmonter le verrouillage de ce dernier pour terminer « proprement » ses différents services. La section 1 décrit l'initialisation du système jusqu'au démarrage des *runlevels*, tandis que les suivantes décrivent la séquence d'invocation de scripts associée à ces différents *runlevels*. Enfin, la section 6 est consacrée à l'arrêt du système.

1. Démarrage du noyau, lancement de *rc*

1.1. Organisation sur le disque et chargeur de démarrage

Deux systèmes CLIP sont normalement installés sur le disque dur d'un poste CLIP : une version à jour, qui est celle lancée par défaut au démarrage, et la version précédente, sur laquelle il reste possible de démarrer en cas de mauvais fonctionnement de la version à jour. Les deux versions sont installées sur des jeux de partitions distincts, mais conservent en commun :

- Une partition de démarrage, qui contient les noyaux des deux systèmes, ainsi que leurs éventuelles images de disques initiaux (*initrd*) et le chargeur de démarrage.
- Une partition de journaux, montée sous */var/log* dans le système CLIP actif. Cette partition commune permet de ne pas perdre les journaux lors du basculement d'un système CLIP à l'autre.
- Une partition de données utilisateur, montée sous */home* dans le système CLIP actif. Cette partition commune permet aux utilisateurs d'utiliser indifféremment l'un ou l'autre système sans perte de données.

Ces trois partitions communes occupent normalement les trois premières partitions primaires du disque dur. Du fait des limitations du schéma de partitionnement BIOS, les partitions propres à chacun des deux systèmes CLIP doivent ensuite être rassemblées dans la quatrième et dernière partition primaire, dont elles constituent donc des partitions logiques. On trouve aussi parmi ces partitions logiques la partition de *swap*, qui est commune aux deux systèmes[1]. La Figure 1 reprend schématiquement cette organisation du disque dur, dans le cas d'un système CLIP-RM.

CLIP utilise le chargeur de démarrage (*« bootloader »*) *extlinux*, choisi pour sa simplicité ainsi que pour la possibilité qu'il offre d'interdire toute interaction autre que le choix d'une entrée prédéfinie au démarrage[2]. Le code exécutable du chargeur de démarrage *extlinux* est réparti en trois blocs :

- Un bloc MBR (*Master Boot Record*), *mbr.bin*, qui est inscrit dans le premier secteur du disque dur (donc hors de toute partition).
- Un bloc de démarrage *extlinux.bss,* de 512 octets, inscrit dans le premier secteur de la partition primaire active du disque.
- Un fichier *extlinux.sys* du système de fichiers *ext2* ou *ext3* de la partition active.

Le bloc de démarrage *extlinux.bss* et le fichier *extlinux.sys* sont en fait issus d'un même fichier exécutable, dont ils constituent respectivement les 512 premiers octets et tout sauf les 512 premiers octets. Ces deux fichiers sont écrits sur le disque par la commande *extlinux* <*chemin*>, qui installe un chargeur de démarrage paramétré dans le répertoire <*chemin*> (dans lequel est copié *extlinux.sys*). La commande *extlinux* se charge aussi de *« patcher »* le bloc *extlinux.bss*, pour y inscrire le numéro

[1] Cette partition de *swap* n'est pas pour autant partagée par les deux systèmes : dans la mesure où le *swap* est initialisé à chaque démarrage avec un chiffrement utilisant une clé tirée aléatoirement, les données du *swap* précédent (utilisé avant redémarrage du système) ne sont jamais intelligibles pour le système CLIP actif.

[2] En particulier, la configuration de *extlinux* mise en oeuvre dans CLIP ne permet pas de modifier la ligne de commande passée au noyau chargé, ni de charger comme noyau un fichier arbitraire.

d'*inode* du répertoire <*chemin*>. Une fois cette commande exécutée, le fichier de configuration *extlinux.conf* et tous les autres fichiers qui doivent pouvoir être manipulés par le chargeur au démarrage – en particulier les noyaux et leurs éventuelles images *initrd* – doivent être copiés dans le répertoire <*chemin*>.

Au démarrage du système, la séquence de chargement est la suivante :

- Le BIOS charge et appelle le code *mbr.bin* présent dans le premier secteur du disque.
- Ce code parcourt la table de partitions jusqu'à trouver la partition active, dont il lit le premier secteur en mémoire. Ce premier secteur est supposé contenir *extlinux.bss*, qui est reconnu par un *« magic number »*. Si le *magic number* lu correspond bien à celui de *extlinux.bss*, *mbr.bin* passe la main à ce dernier. Dans le cas contraire, le démarrage est interrompu.
- *extlinux.bss* contient des fonctions élémentaires d'accès au système de fichier *ext2*, ainsi que le numéro d'*inode* du répertoire de configuration. Il peut ainsi ouvrir et charger en mémoire le fichier *extlinux.sys* contenant le reste du code du chargeur, ainsi que son fichier de configuration *extlinux.conf* et d'éventuelles extensions (typiquement, le fichier *menu.c32* contenant le code du menu « graphique » de choix de noyau).
- *extlinux.sys* crée l'interface de choix de système à partir de *extlinux.conf*, et lit le choix de l'utilisateur. Il charge ensuite le noyau, et l'éventuel initrd, en mémoire à partir de fichiers qui doivent se trouver dans le même répertoire que *extlinux.conf* et *extlinux.sys*. Il construit la ligne de commande noyau à partir des paramètres inscrits dans extlinux.conf, avant de passer la main au noyau chargé en mémoire.

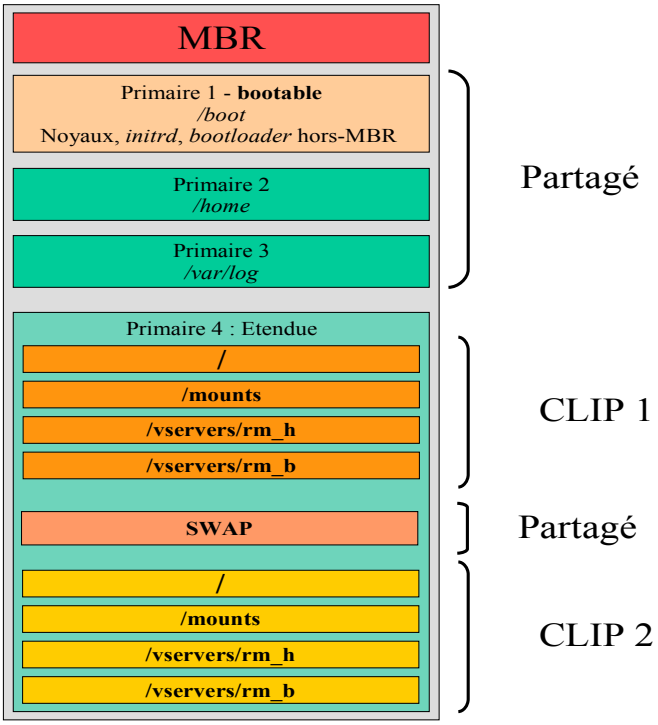

Figure 1: Organisation du disque dur d'un poste CLIP-RM.

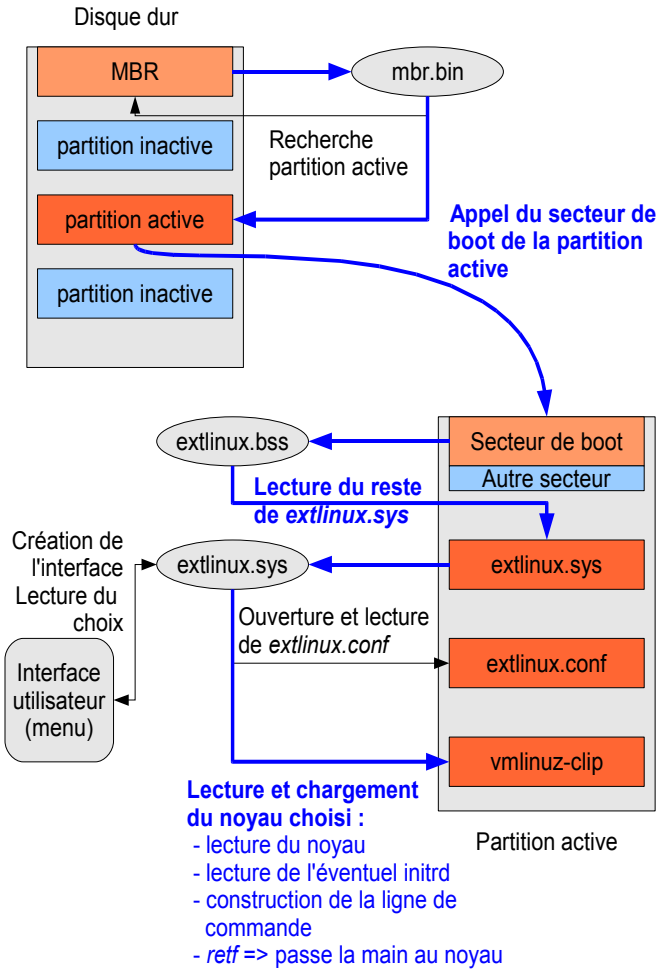

Figure 2: Fonctionnement de extlinux, le bootloader utilisé dans CLIP.

L'ensemble de cette procédure est résumé dans la Figure 2. On notera bien qu'*extlinux* ne contient pas suffisamment de fonctions d'accès aux systèmes de fichiers *ext* pour lui permettre de manipuler plusieurs répertoires. Il est en revanche capable de suivre un lien symbolique au sein de son répertoire de configuration. Ainsi, la configuration utilisée dans CLIP repose sur deux fichiers de configuration *extlinux* distincts :

- Un fichier *extlinux_5.conf*, qui démarre par défaut sur la partition *5* (première partition logique de la partition étendue, qui se trouve être la quatrième partition BIOS du disque – cf. Figure 1), et de manière optionnelle sur la partition *10*. Ces deux partitions correspondent par convention aux systèmes de fichiers racines des deux systèmes CLIP installés sur le disque.
- Un fichier *extlinux_10.conf*, qui démarre par défaut sur la partition *10*, et optionnellement sur la

partition *5*.

Le fichier *extlinux.conf* réellement utilisé par le chargeur est dans ce cas un lien symbolique pointant vers celui des deux fichiers *extlinux_X.conf* qui démarre par défaut sur le système CLIP à jour. Le basculement de système actif lors d'une mise à jour du coeur du système CLIP peut donc être réalisé par simple mise à jour de ce lien symbolique. L'ensemble de ces fichiers, ainsi que les noyaux et *initrd* éventuels, est installé à la racine du système de fichiers *ext3* de la première partition primaire du disque, qui correspond au répertoire sur lequel pointe *extlinux.bss*.

1.2. Chargement du noyau, *initrd* éventuel

Le chargeur de démarrage passe obligatoirement un certain nombre de paramètres au noyau qu'il charge :

- La partition racine, sous la forme *root=/dev/XXX*,
- L'option *quiet*, qui supprime les messages *printk* normalement affichés au démarrage, et n'affiche que ceux qui correspondent à des avertissements ou à des erreurs (priorité *KERN_WARNING* ou supérieure),
- Les paramètres nécessaires au fonctionnement du *splash screen* décrit plus bas : *splash=silent,theme:clip CONSOLE=/dev/tty1*
- Éventuellement, la résolution et la profondeur du *framebuffer* VESA, sous la forme *vga=YYY*.

Deux alternatives sont possibles une fois le chargement du noyau achevé, selon le type de configuration matérielle. Dans le cas d'une configuration matérielle spécifiquement supportée par CLIP, c'est-à-dire à laquelle un paquetage noyau est dédié, le noyau est entièrement statique (tous les pilotes de périphériques sont « en dur »), et le démarrage est réalisé sans *initrd*. Une fois le noyau initialisé, celui-ci monte directement la partition racine à la racine de son *VFS*, et invoque */sbin/init*, qui lance la séquence de démarrage *userland.* Au contraire, dans le cas d'une configuration matérielle qui ne serait pas spécifiquement supportée par CLIP, le seul noyau utilisable est le noyau CLIP modulaire (paquetage *clip-kernel-modular*, issu de l'*ebuild sys-kernel/clip-kernel*), qui nécessite le chargement de modules noyaux avant de pouvoir accéder à la partition racine du système. Ce deuxième mode de fonctionnement nécessite le recours à une image disque initiale (*initrd*). Cette image est initialement contenue dans le fichier *initrd-clip.img*, situé dans le même répertoire */boot* que le noyau CLIP. Une fois décompressée[3], elle contient les fichiers et répertoires suivants :

- un interpréteur *nash* (copie de celui installé par le paquetage *sys-apps/mkinitrd*)
- une version statique de l'exécutable *insmod* (copie du *insmod.static* installé par *sys-apps/module-init-tools*)
- un script */init* interprétable par *nash*
- un répertoire */lib* contenant les modules noyaux à charger au démarrage

Le script *init* inclus dans cet *initrd* réalise les opérations suivantes :

- Montage de */proc* et de */sys*.

[3] Le fichier *initrd-clip.img* est au format classique pour un *initrd* pour noyau 2.6 : une archive *cpio*, compressée avec *gzip*.

- Chargement d'une liste de modules par des commandes *insmod,* éventuellement avec des paramètres spécifiques.
- Mise à un de */proc/sys/kernel/grsecurity/disable_modules* (cf. [CLIP_1203]), afin d'interdire tout chargement ultérieur de modules.
- Création des *devices* nécessaires (en particulier, la partition racine) dans */dev*
- Montage de */dev/root* (en lecture seule) sur */sysroot*
- Pivotement de la racine sur */sysroot*

On notera bien que la possibilité de charger des modules noyaux est révoquée de manière irréversible avant que le noyau n'accède d'une quelconque manière au disque dur. L'utilisation d'un *sysctl* spécifique à *grsecurity* est dans ce cas préférée à la suppression de *CAP_SYS_MODULES* du *cap-bound* du système (cf. [CLIP_1204]) car cette capacité est aussi nécessaire aux ajustements ultérieurs de *kernel.cap-bound, kernel.clip.rootcap* et *kernel.clip.reducecap* (cf. scripts de démarrage *sysctl* et *reducecap*) et ne peut donc pas être masquée avant le montage du disque racine. De ce fait, les seuls modules qui peuvent être chargés dans le noyau sont ceux inclus dans l'image *initrd* – qui sont tous effectivement chargés au démarrage.

L'image *initrd* des configurations CLIP modulaires est créée lors de l'installation du noyau par un script *postinst* du paquetage *kernel-modular*. Ce dernier paquetage installe non seulement le noyau et son *System.map* dans */boot*, mais aussi un ensemble de modules dans */lib/modules/<version noyau>*, qui correspondent à tous les pilotes inclus nativement dans Linux pour les catégories suivantes:

- cartes réseau 100Mbit/s et 1Gbit/s
- cartes graphiques (pilotes *framebuffer*)
- contrôleurs de disque (IDE, SATA)
- contrôleurs USB

Le script *postinst* du paquetage crée une arborescence type d'*initrd* dans un répertoire */tmp/initrd.XXXXXX* (où les *X* sont remplacés par des caractères aléatoires), en y copiant les exécutables nécessaires (*nash, insmod.static*) depuis l'arborescence du système CLIP courant. Puis le script parcourt le fichier */etc/modules* (s'il existe – le script sort en erreur dans le cas contraire), qui contient une liste de noms de modules, à raison d'un par ligne, sans l'extension *.ko* et éventuellement suivi sur la même ligne de ses paramètres de chargement. Pour chacun de ces modules, le script copie le module ainsi que les modules dont il dépend depuis */lib/modules/<version noyau>* dans le */lib* de l'arborescence d'*initrd*, et ajoute les commandes de chargement de ces modules au script */init* de cette même arborescence. Puis le script termine l'écriture de */init*, et compresse l'arborescence résultante pour générer un nouveau fichier *initrd-clip.img*. Ainsi, un *initrd* à jour et spécialisé pour la configuration matérielle est généré pour chaque nouvelle version du noyau. Le fichier */etc/modules* n'est géré par aucun paquetage : il est supposé avoir été créé lors de l'installation initiale du système, de manière adaptée à la configuration matérielle sur laquelle est réalisée l'installation.

De manière complémentaire à cette création d'un *initrd*, le script *postinst* du paquetage *sys-boot/syslinux* (qui installe le chargeur de démarrage *extlinux*) prend en compte de manière spécifique le cas d'une configuration modulaire (détecté à la présence d'un fichier *initrd-clip.img* dans */boot*). Dans ce cas, la ligne de commande de chaque noyau est adaptée de manière à, d'une part, ajouter l'argument *initrd=initrd-clip.img* nécessaire à la mise en oeuvre de l'*initrd*, et, d'autre part, supprimer l'argument *vga=YYY*, qui suppose l'utilisation d'un *framebuffer* VESA. A la place de ce dernier argument, la

première ligne du fichier */etc/bootargs* est incluse dans les arguments, lorsque ce fichier existe. Ce dernier fichier est supposé créé à l'installation, tout comme */etc/modules*, mais à la différence de ce dernier, son absence n'est pas considérée comme une erreur (le mode vidéo pouvant aussi éventuellement être spécifié par les paramètres de chargement du module associé au *framebuffer*).

La séquence de démarrage redevient commune aux configurations modulaires aussi bien que statiques après le montage de la partition racine. Dans tous les cas, le noyau lance comme processus de *pid 1* le démon */sbin/init*, qui invoque à son tour */sbin/rc* afin de lancer les différents scripts de démarrage.

1.3. Scripts de démarrage

Le système CLIP reprend le mécanisme de gestion des scripts de démarrage de la distribution *Gentoo* sur laquelle il est basé. Ce mécanisme, bien que semblable au mécanisme *System V* généralement utilisé par la plupart des distributions Linux, présente cependant un certain nombre de spécificités. En premier lieu, *Gentoo* n'utilise pas directement les *runlevels* numérotés, */etc/rcX.d* avec X=1,2,...,6 « classiques », mais plutôt des *runlevels* nommés de manière arbitraire, aussi appelés *softlevels*. De manière comparable à *System V,* chaque *softlevel* est défini par un sous-répertoire de */etc/runlevels*, contenant un ensemble de liens symboliques pointant vers des scripts de */etc/init.d*. Cependant, à la différence du système classique, ces liens portent chacun le même nom que le script vers lequel ils pointent, et non ce nom précédé d'un préfixe *Sxx* ou *Kyy* (avec *xx* et *yy* des numéros entre 1 et 99). Chaque lien symbolique d'un répertoire *softlevel* signifie que le script correspondant doit être démarré au lancement de ce *softlevel*. L'ordre de lancement des scripts n'est pas défini statiquement par numérotation, mais dynamiquement en fonction des dépendances que déclarent ces scripts. En l'absence de contraintes de dépendances, les scripts d'un *softlevel* sont lancés dans l'ordre alphabétique. Le changement de *softlevel* est réalisé par la commande */sbin/rc <nouveau softlevel>*. Un tel changement de *softlevel* consiste, dans un premier temps, à terminer (par appel à la fonction *stop()* du script correspondant de */etc/init.d*) tous les services actifs dans le niveau de départ qui ne sont pas définis comme actifs dans le nouveau niveau[4], puis dans un second temps à lancer (par appel à leur fonction *start()*) tous les services déclarés dans le nouveau niveau qui ne sont pas encore actifs.

Par défaut, la distribution *Gentoo* met normalement en oeuvre deux tels *softlevels*, *boot* et *default*. Ces *softlevels* sont utilisés automatiquement au démarrage du fait de la configuration du démon *init* par le fichier */etc/inittab*. La commande */sbin/rc boot* est lancée au démarrage (*id 'rc'* dans *inittab*), tandis que */sbin/rc default* est lancée à l'entrée dans les *runlevels* « traditionnels » 3, 4 et 5 (le *runlevel* par défaut étant 3). Deux autres *softlevels* sont prévus par *Gentoo*, mais pas utilisés en pratique (les répertoires correspondants sont vides) : *single* pour le *runlevel* 1 et *nonetwork* pour le *runlevel* 2. Enfin, */sbin/rc* reconnaît directement les mots-clés *sysinit*, *shutdown* et *reboot*, qui sont traités comme des *runlevels* spécifiques auxquels aucun répertoire de configuration n'est associé. Ces « pseudo-*softlevels* » sont invoqués respectivement avant *boot* au *démarrage* (*sysinit, id 'si'* dans *inittab*) et à l'entrée dans les *runlevels* 0 (*shutdown*) et 6 (*reboot*).

CLIP reprend cette configuration, mais ajoute – par une ligne supplémentaire dans */etc/inittab* – un passage automatique par le *softlevel nonetwork* entre *boot* et *default*. Ce *softlevel* supplémentaire est utilisé principalement pour la mise à jour du coeur du socle CLIP. Par ailleurs, une ligne *'ca'* dans *inittab* configure le comportement sur réception de la combinaison de touches *Control+Alt+Suppr* (non

[4]Ou dans le niveau *boot*, qui est traité de manière spécifique : les services du niveau *boot* ne sont pas terminés lors d'une commutation entre *softlevels* « normaux » - c'est à dire autres que les *softlevels* d'arrêt.

interceptée, c'est-à-dire en dehors de la session X11). Dans ce cas, un système CLIP redémarre par la commande *shutdown -r now*, qui lance le runlevel 6 et donc la commande */sbin/rc reboot*.

Par ailleurs, le script */sbin/rc* réalise plusieurs appels à la fonction *« add-on » splash()*, fournie par le paquetage *media-gfx/splashutils*, de manière à charger et mettre à jour le *splash screen* (fond d'écran et barre de progression affichés pendant les séquences de démarrage et d'arrêt), qui repose sur les outils du projet *fbsplash / gensplash* (cf. [FBSPLASH]). On notera bien que seule la partie *« userland »* de ces outils est mise en oeuvre : le *patch* noyau permettant la décoration des consoles texte n'est pas appliqué au noyau CLIP.

Les scripts *init.d gentoo* sont interprétés par */sbin/runscript.sh*, qui est lui-même un script *bash*. Les dépendances de ces scripts sont exprimées dans la fonction *depend()* définie au sein du script. Cette fonction peut contenir plusieurs lignes de définition de dépendances, composées chacune d'un mot clé de condition, suivi d'un ou plusieurs noms de scripts. Les conditions supportées sont les suivantes (en désignant par A le script courant) :

- **need B :** le script B doit être démarré avant A. Si A doit être lancé dans un *softlevel* donné, B sera préalablement lancé au sein de ce *softlevel* s'il n'est pas encore démarré, et ce même s'il n'est pas défini comme actif dans ce niveau. Un arrêt de B entraînera un arrêt préalable de A. Si le lancement de B échoue, ou si B n'est pas trouvé dans */etc/init.d*, A ne sera pas lancé même s'il est actif dans un niveau. Réciproquement, B ne pourra pas être arrêté tant que A ne le sera pas.
- **use B :** le script B est utilisé par A lorsqu'il est actif. Si les deux scripts sont actifs dans un même *softlevel,* B sera lancé avant A. B ne sera en revanche pas lancé automatiquement au démarrage de A, s'il n'est pas déjà actif, et le lancement de A ne sera pas perturbé par l'échec de celui de B. L'arrêt de B n'aura pas d'influence sur le fonctionnement de A. On notera que la seule différence entre cette condition et la condition *« after »* décrite ci-dessous réside dans l'affichage des dépendances du script par la commande */etc/init.d/A iuse*. Cette commande n'étant pas accessible à l'administrateur sous CLIP, les deux conditions sont équivalentes dans ce cas, et *after* sera utilisée de manière préférentielle.
- **after B :** le script A est lancé après B, lorsque les deux sont actifs dans un même *softlevel*. Symétriquement, lors d'un changement de *softlevel*, A sera arrêté avant B. Les autres événements concernant B n'ont aucune influence sur A.
- **before B :** contraposée de *« after B »* : le script A est lancé avant B lorsque les deux sont actifs dans un même *softlevel*.
- **provide C :** le script A satisfait les conditions de dépendances *need* vis-à-vis de C (qui est typiquement une dépendance virtuelle : le fichier */etc/init.d/C* n'existe pas). Permet de gérer des alternatives dans les dépendances. On notera que si plusieurs scripts présents dans */etc/init.d* peuvent satisfaire une dépendance virtuelle *need C*, le premier dans l'ordre alphabétique sera utilisé.

Indépendamment de cette gestion de dépendances, un script *init.d* peut être défini comme critique, auquel cas il sera démarré automatiquement – s'il n'est pas déjà actif – au début de chaque *softlevel*, avant tout script non-critique du niveau. Les différents scripts critiques sont démarrés dans l'ordre dans lequel ils sont déclarés dans la variable *CRITICAL_SERVICES*, sans aucune gestion de dépendances. La variable *CRITICAL_SERVICES* est initialisée par une liste de services critiques inscrite en dur dans */sbin/rc*. Ces services peuvent être considérés comme les services critiques du *softlevel boot.* Chaque *softlevel* peut ajouter(mais naturellement pas retirer) lors de son traitement des services à la liste des

services critiques, en les listant dans un fichier */etc/runlevels/<nom du niveau>/.critical*. En cas d'échec du démarrage d'un service critique (la fonction *start()* de ce service rend un code de retour non nul), le système est immédiatement redémarré par *reboot -f* (arrêt direct de la machine, sans passer par la séquence d'arrêt normale), après un simple remontage en lecture-seule de tous les montages actifs. Aucun service critique autre que ceux définis par défaut n'est utilisé dans CLIP à ce stade.

L'interpréteur */sbin/runscript.sh* fonctionne en « sourçant » le script *init.d* qu'il interprète, après avoir défini des valeurs par défaut pour les différentes fonctions que ce script peut ensuite surcharger, ainsi que pour la fonction *exit()*, qui est systématiquement interceptée. Les fichiers suivants sont automatiquement sourcés avant le script, s'ils sont présents :

- */etc/conf.d/<nom du script>*
- */etc/conf.d/basic*
- */etc/rc.conf*

Une fois le code du script importé, *runscript.sh* invoque la fonction appropriée : *start(), stop(), restart(), etc...* L'état courant du système (scripts dans l'état démarré, cache de dépendances, etc.) est stocké dans un système de fichiers *tmpfs* monté sur */var/lib/init.d* (avec les options *noexec,nosuid,nodev* dans le cas de CLIP). On notera que le sous-système *gensplash* qui assure l'affichage d'une barre de progression lors du démarrage conserve son état dans un autre système de fichiers *tmpfs*, monté sur */lib/splash/cache* (même options de montage *noexec,nosuid,nodev* au sein de CLIP).

1.4. Import sécurisé de variables

Le mécanisme traditionnel d'import de variables de configuration dans les scripts de démarrage, par appel à *source <fichier de configuration>*, n'est pas acceptable au sein de CLIP pour importer des variables définies par l'administrateur local (c'est-à-dire modifiable par *admin* dans la cage $ADMIN_{clip}$). En effet, l'appel à *source* exécute automatiquement tout code inclus dans le fichier importé, ce qui permettrait à l'administrateur d'escalader ses privilèges en exécutant du code sous l'identité *root*, hors de toute cage. Il est donc nécessaire d'importer ces variables sans exécuter les commandes éventuellement définies dans le fichier de configuration. Cet import peut typiquement être réalisé par un *grep* du nom de variable dans le fichier de configuration, suivi d'un appel à *eval* pour réaliser l'affectation de la variable dans le script courant. Il est cependant nécessaire dans ce cas de valider au préalable le contenu de la variable importée, afin d'éviter l'exécution de code arbitraire lors de l'appel *eval* (ou de commandes ultérieures prenant la variable pour argument), dans le cas d'une variable mal formée qui contiendrait par exemple un ';'. A cette fin, le paquetage *clip-libs/clip-sub* fournit, dans le fichier */lib/clip/import.sub,* une fonction *import_conf* permettant l'import sécurisé de variables. Son prototype est le suivant :

```
import_conf <fichier> <filtre> <var1> [<var2> ... <varn>]
```

avec :

- *<fichier>* le fichier de configuration depuis lequel l'import doit être réalisé.
- *<filtre>* une expression régulière étendue (cf. *man grep*) définissant le format acceptable pour la ou les variables à importer.
- *<var1>* (*<var2>* ... *<varn>*) un ou plusieurs noms de variables à importer

Pour un tel appel, la fonction réalise les traitements suivants :

- lecture de <*fichier*>
- suppression dans le fichier lu des lignes commençant par '#'
- suppression des espaces et tabulations en début de ligne
- pour chaque nom de variable <*var*> dans <*var1*> ... <*varn*> :
  - recherche de la dernière ligne commençant par *« <var>= »* dans le fichier lu
  - suppression du motif <*var*>= dans cette ligne
  - recherche du premier motif <*motif*> conforme à <*filtre*> dans cette ligne (appel à *grep -oEe | head -n 1*)
  - évaluation de <*var*>='<*motif*>' dans le script courant

Il est ainsi possible d'importer une ou plusieurs variables, sans risque d'exécution de commandes arbitraires lors de l'import, et avec un format valide ne risquant pas de laisser des possibilités d'exécution de commandes arbitraires dans les manipulations ultérieures de la variable (sous réserve d'une bonne définition de l'expression régulière filtre). Le fichier */lib/clip/import.sub* définit à titre d'exemple une expression régulière adaptée à l'import d'adresses IPv4 :

```
_IMPORT_FILTER_ADDR="[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}"
```

Par ailleurs, *import.sub* fournit une fonction complémentaire *unimport_conf*, qui prend en argument la même liste de variables que celle passée à un appel *import_conf*, et réalise un *unset* de chacune de ces variables afin de nettoyer l'environnement après utilisation des variables importées.

Les sections qui suivent décrivent les différents *softlevels* utilisés dans CLIP, et en particulier les scripts *init* qui y sont actifs. Pour chacun de ces scripts, les dépendances sont décrites selon les conventions *gentoo* (cf. [GENTOO_INIT]). Les fichiers de configuration utilisés par le script sont aussi listés. Lorsqu'un fichier est décrit entre crochets, il n'est utilisé qu'à l'aide des fonctions d'import sécurisé décrites en 1.4, et peut donc être exposé en lecture-écriture à l'administrateur. De même, les fichiers décrits entre parenthèses sont utilisés uniquement par des exécutables invoqués par le script, dont les fonctions de lecture de fichiers de configuration sont supposées dépourvues de failles. Dans les autres cas, le fichier est soit sourcé directement dans le script, soit lu par le script de démarrage lui-même, par exemple par *cat*. Sa modification équivaut dans ce cas à l'exécution de commandes arbitraires, sous l'identité de *root*, au stade du démarrage où il est lu. Il est rappelé que le fichier */etc/conf.d/rc* est sourcé implicitement par tous les scripts de démarrage, et qu'un script nommé */etc/init.d/<script>* source automatiquement */etc/conf.d/<script>* lorsque ce dernier est présent. Le fichier *rc* n'est jamais listé parmi les fichiers de configuration des différents scripts, tandis que le fichier homonyme de chaque script est cité lorsqu'il est effectivement présent dans une installation CLIP.

Les scripts sont décrits dans l'ordre de leur lancement au démarrage d'un système CLIP. Sauf mention explicite du contraire, les actions décrites sont celles réalisées par la fonction *start()* de chaque script. L'arrêt d'un service n'entraîne aucune opération (autre que la gestion des dépendances) lorsqu'aucune telle opération n'est décrite explicitement.

2. Niveau *sysinit*

Le *softlevel sysinit* constitue un niveau particulier, au même titre que *shutdown* et *reboot*, qui n'est associé à aucun répertoire de liens symboliques dans */etc/runlevels*. Il est lancé par *init* avant tout *runlevel System V* (mot clé « si » dans */etc/inittab*), et réalise les opérations les plus critiques au démarrage du système. Les premières de ces opérations sont écrites directement dans */sbin/rc*, plutôt que dans des scripts séparés dans */etc/init.d*. Elles consistent à :

- Monter */proc*. Dans un système CLIP, la variable *RC_USE_FSTAB* étant positionnée à *yes* dans */etc/conf.d/rc*, ce montage est réalisé en utilisant la ligne */proc* de */etc/fstab*, ce qui permet de passer les options de montage *nodiratime,nosuid,nodev,noexec*.
- Monter */sys* (uniquement pour un noyau 2.6). De la même manière que pour */proc*, ce montage utilise dans CLIP les options définies dans */etc/fstab*, soit *nosuid,nodev,noexec*.
- Configurer */dev*. Sur un système CLIP, la variable *RC_DEVICES* étant positionnée à *static* dans */etc/conf.d/rc*, aucune opération de configuration n'est réalisée (pas de montage spécifique, pas de lancement de *udev* ou *devfsd*). On notera que le montage du */dev* complet en lecture-écriture depuis */mounts/dev* n'est réalisé qu'ultérieurement, par le script de montage *localmount*. Avant cela, le */dev* utilisé est celui de la partition racine, qui ne contient que les *devices* essentiels, et n'est accessible qu'en lecture seule.
- Activation de la journalisation des messages affichés sur la console de démarrage (*bootlog*). Cette fonctionnalité n'est pas activée à ce stade dans CLIP du fait de conflits avec *gensplash*. Il serait néanmoins souhaitable à terme de disposer de ce type de service. La principale difficulté à prendre en compte est liée au fait que */var/log* n'est pas inscriptible avant le lancement de *localmount*.
- Configuration du niveau de verbosité de la console (*dmesg -n 1)*
- Activation du *splash screen gensplash*.
- Lancement des services critiques initiaux, décrits ci-dessous. En cas d'échec au lancement, le système est redémarré immédiatement.
- Montage du répertoire d'état de *rc* sur */var/lib/init.d*. Dans un système CLIP, un système de fichiers de type *tmpfs* et de taille 2 Mo est utilisé à cette fin (*svcfstype=tmpfs* et *svcsize=2048*) dans */etc/conf.d/rc*.
- Création du cache de dépendances des scripts *init.d* par appel à *depscan.sh*.
- Enregistrement des services critiques comme démarrés (création de liens symboliques dans */var/lib/init.d/started*)
- Création des enregistrements de *login /var/run/utmp* et */var/log/wtmp*.
- Désactivation de *bootlog* (non utilisé sur CLIP à ce stade).

Les services critiques lancés au cours du traitement *sysinit* sont énumérés ci-dessous, dans l'ordre de leur lancement (qui est réalisé sans gestion de dépendances). On notera que ces services sont officiellement attribués au *softlevel boot*, et qu'à ce titre il ne sont jamais arrêtés lors d'une commutation de *softlevel*, hors arrêt du système. La variable d'environnement *BOOT* est positionnée à *yes* pour tous

les scripts lancés au cours du traitement de ce niveau.

checkroot ***[critique]***

Dépendances : *before **
Fichier de configuration : *aucun*

Réalise un *fsck* du système de fichier racine, si ce *fsck* est requis par le champ *fs_passno* de la ligne correspondante dans */etc/fstab* (ce qui est le cas à ce stade sous CLIP), ou si le paramètre *forcefsck* a été passé sur la ligne de commande du noyau (ce qui ne peut être fait interactivement sur un système CLIP, mais qui pourrait à terme être spécifié par une entrée dédiée dans le menu de démarrage). Le *fsck* est réalisé en mode entièrement automatique. Selon le code d'erreur renvoyé par la commande *fsck*, le script décide de continuer la séquence de démarrage (code d'erreur nul ou 1), de redémarrer (code 2 ou 3), ou de relancer *fsck* avec l'option *–y* avant de redémarrer (autres cas).

checkfs ***[critique]***

Dépendances : *need checkroot*
Fichiers de configuration : */etc/fstab*

Réalise un *fsck* des systèmes de fichiers autres que la racine spécifiés dans */etc/fstab*. Ce *fsck* est réalisé de manière entièrement automatique, et en respectant le champ *fs_passno* de */etc/fstab*. Le démarrage est poursuivi pour tout code de retour compris entre 0 et 3. Lorsqu'un autre code de retour est rencontré, *fsck* est invoqué à nouveau avec l'option *–y*, puis le système est redémarré.

Lorsque des volumes RAID ou LVM sont spécifiés, ceux-ci sont configurés par ce script avant l'appel à *fsck*.

localmount ***[critique]***

Dépendances : *need checkfs*
Fichiers de configuration : */etc/fstab*

Réalise le montage de tous les systèmes de fichiers spécifiés dans */etc/fstab*, autres que ceux de type *proc* ou portant l'option *noauto*. Le *swap* est ensuite configuré de manière spécifique : la première ligne de */etc/fstab* comportant *swap* comme champ *fs_file* définit par son premier champ *(fs_spec)* le fichier spécial de type bloc à utiliser comme support du *swap*. Ce fichier est alors projeté par *dm_crypt* sur */dev/mapper/swap0*, avec un chiffrement de type *AES-256-LRW-BENBI:SHA256* et une clé aléatoire tirée sur */dev/urandom*. Le fichier projeté est alors configuré comme *swap*. Il est important de s'assurer que ce fichier demeure la première projection réalisée à l'aide du *device mapper* au démarrage du système, ou plus précisément que cette projection conserve le numéro de mineur *0*, afin d'assurer la cohérence avec le script *devctl* décrit plus bas.

On notera que l'import depuis des fichiers de */etc/admin* n'est valide qu'au delà de ce point, car le script *localmount* assure notamment le montage de */etc/admin* depuis */mounts/admin_priv/etc.admin*.

hostname ***[critique]***

Dépendances : *need localmount*
Fichiers de configuration : *[/etc/admin/conf.d/hostname]*

Configure le *hostname* du poste en fonction de la variable *HOSTNAME* définie dans le fichier */etc/admin/conf.d/hostname*. Cette variable est importée de manière sécurisée, et doit être une suite de caractères alphanumériques. Le nom de machine ainsi configuré est aussi enregistré dans la variable *HOSTNAME* inscrite dans */var/env.d/01hostname*.

clock ***[critique]***

Dépendances : *need localmount*
Fichiers de configuration : */etc/conf.d/clock*

Réalise la configuration de l'horloge système à partir de l'horloge matérielle.

A l'arrêt du système, le script réalise l'opération inverse, en inscrivant l'heure système (qui a pu être modifiée par l'administrateur local, ou suite à une synchronisation NTP) dans l'horloge matérielle.

bootmisc ***[critique]***

Dépendances : *need localmount*
before logger
Fichiers de configuration : *aucun*

Réalise diverses opérations de maintenance du système :

- suppression de tous les fichiers de */var/lock*
- suppression de tous les fichiers de */var/run* autres que *random-seed* et *utmp*

3. Niveau *boot*

Ce *softlevel* est quant à lui bien associé à un répertoire de */etc/runlevels*, mais reste traité de manière spécifique par *rc* aussi bien que par *init*. Il est lui aussi lancé par *init* avant tout *runlevel System V* (mot clé « rc » dans *inittab*). Par ailleurs, les scripts inscrits dans le *softlevel boot* sont automatiquement ajoutés à tout autre *softlevel* « normal » par */sbin/rc*, ce qui leur évite d'être terminés avant la séquence d'arrêt du système. La variable d'environnement *BOOT* est positionnée à *yes* pour tous les scripts lancés au cours du traitement de ce niveau. Par ailleurs, les scripts critiques lancés au cours du traitement *sysinit* sont rattachés du point de vue de la gestion de dépendances à ce niveau.

clip_data_backup_restore

Dépendances : *need localmount*
Fichiers de configuration : *aucun*

Réalise une éventuelle restauration des données utilisateur sauvegardées en invoquant */sbin/clip_data_backup_restore_boot* (cf. [CLIP_DCS_15094]).

clip_update_unlock

Dépendances : *need localmount*
before clip_backup_restore
Fichiers de configuration : *aucun*

Supprime les verrous globaux du système de mise à jour (*[...]/update_priv/pkgs/clip_update.lock* dans chaque cage affectée, UPDATEclip dans tous les cas et éventuellement RM_H et RM_B).

clip backup_restore

Dépendances : *after clip_data_backup_restore*
Fichiers de configuration : *aucun*

Réalise une éventuelle restauration d'un système CLIP sauvegardé en invoquant */sbin/clip_backup_restore_boot* (cf. [CLIP_DCS_15094]).

clip_data_backup_save

Dépendances : *after clip_backup_restore*
Fichiers de configuration : *aucun*

Réalise une éventuelle sauvegarde des données utilisateur en invoquant */sbin/clip_data_backup_save_boot* (cf. [CLIP_DCS_15094]).

clip_backup_save

Dépendances : *after clip_data_backup_save*
Fichiers de configuration : *aucun*

Réalise une éventuelle sauvegarde du système en invoquant */sbin/clip_backup_save_boot* (cf. [CLIP_DCS_15094]).

keymaps

Dépendances : *need localmount*
Fichiers de configuration : */etc/conf.d/keymaps*

Configure le *mapping* clavier et l'encodage de caractères utilisés par la console. Le clavier est configuré par défaut en *azerty* français, mais on notera bien que la console n'est pas utilisée en dehors d'une configuration de développement.

consolefont

Dépendances : *need localmount keymaps*
Fichiers de configuration : */etc/conf.d/consolefont*

Configure la police de caractères utilisée par la console.

devctl

Dépendances : *need localmount*
before reducecaps
Fichiers de configuration : */proc/mounts*

Configure et active *devctl* (cf. [CLIP_1201]) en fonction de la partition racine obtenue par lecture de */proc/mounts*. A ce stade, les permissions définies par ce script sont les suivantes :

- Aucune permission sur la partition contenant le noyau et le chargeur de *boot*, ni sur le numéro de mineur *device-mapper* associé au *swap* chiffré.
- Permissions *RO, EXEC, DEV* et *SUID* sur tout le reste du disque de démarrage (pas d'accès en écriture), sauf dans le cas d'un disque racine en RAID, pour lequel la permission *RW* doit être laissée pour l'heure afin de permettre la reconstruction du disque en cas d'incident.
- Permission *RW* sur les autres mineurs du *device-mapper* (pas d'accès *EXEC, DEV* ou *SUID*).

logfiles

Dépendances : *before sysctl*
Fichiers de configuration : *[/etc/admin/conf.d/logfiles]*

Assure la sauvegarde des anciens fichiers de journaux et crée et configure les nouveaux fichiers, en leur attribuant les permissions et attributs appropriés.

Les fichiers de journaux pris en compte par ce script sont ceux de la liste *SYSLOG_FILES*, définie en tête du script. Ces fichiers correspondent à ceux dans lesquels le démon *syslog-ng* exécuté dans la cage $AUDIT_{clip}$ est amené à écrire. Ces fichiers sont créés dans */var/log*, et sauvegardés dans */var/log/keep*. Les sauvegardes sont compressées par *gzip* et nommées *<fichier>.gz.<num>*, avec *<num>* un entier. Une nouvelle sauvegarde est créée à chaque démarrage pour tout fichier dont la taille dépasse une limite inférieure configurable à l'aide de la variable *KEEP_SIZE* définie dans */etc/admin/conf.d/logfiles*. Dans ce cas, les fichiers de sauvegarde existants sont décalés

(<*fichier*>.*gz.1* vers <*fichier*>.*gz.2*, etc.), et le fichier de journal trouvé au démarrage est sauvegardé en <*fichier*>.*gz.1*, puis un nouveau fichier <*fichier*> est créé. Le nombre maximum de sauvegardes conservées (valeur maximum de <*num*> ci-dessus) peut être défini entre 5 et 99 dans la variable *KEEP_FILES_NUMBER* importée de manière sécurisée depuis le fichier */etc/admin/conf.d/logfiles*. Si *KEEP_FILES_NUMBER* est défini à une valeur inférieure à 5, cette valeur est ignorée et 5 fichiers de sauvegarde sont conservés. De même, le script impose une taille minimale de 4096 octets pour *KEEP_SIZE*.

Le script attribue des permissions et attributs appropriés pour ces différents fichiers, plus précisément :

- Propriétaire *syslog:syslog*, droits en lecture-écriture pour le propriétaire et en lecture seule pour le groupe, attribut *APPEND_ONLY* pour les fichiers de journaux actifs.
- Propriétaire *root:syslog*, droits en lecture seule pour le propriétaire et le groupe, attribut *IMMUTABLE* pour les fichiers sauvegardés.

Le script crée enfin une sauvegarde des journaux noyau à l'instant de son invocation, dans */var/log/dmesg*, avec pour propriétaire *root:syslog*, des droits en lecture seule pour le propriétaire et le groupe, et l'attribut *IMMUTABLE*. Cette sauvegarde permet de garder trace du démarrage du noyau, ces journaux étant par ailleurs rapidement écrasés dans le tampon de messages du noyau.

urandom

Dépendances : *need localmount*
Fichiers de configuration : *aucun*

Initialise le générateur d'aléa */dev/urandom* en y injectant le contenu de */var/run/random-seed*.

Lors de l'arrêt du système, sauvegarde l'état de */dev/urandom* (2048 octets) dans ce même fichier.

vprocunhide

Dépendances : *before sysctl*
Fichiers de configuration : */etc/vprocunhide/all*
/etc/vprocunhide/none
/etc/vprocunhide/watch

Configure la visibilité des fichiers du */proc* dans les différents contextes *vserver*, à l'aide de l'utilitaire *vsattr* (cf. [CLIP_1202]). Chacun des fichiers de configuration peut contenir une liste de fichiers (fichiers réguliers ou répertoires), un par ligne sous la forme d'un chemin absolu. Ces listes sont traitées comme suit[5] :

- Les fichiers listés dans *all* (aucun dans CLIP à ce stade) perdent l'attribut *IATTR_HIDE* et deviennent ainsi visibles dans tous les contextes.
- Les fichiers listés dans *none* (*/proc/acpi* dans CLIP à ce stade) perdent l'attribut *IATTR_ADMIN* et deviennent ainsi invisibles, tous contextes confondus.
- Les fichiers listés dans *watch* (*/proc/veriexec* dans CLIP à ce stade) gagnent l'attribut *IATTR_WATCH* et deviennent ainsi visibles dans le contexte WATCH, en plus du contexte

[5] Il est rappelé que chaque fichier ou répertoire du */proc* reçoit au démarrage les attributs par défaut *IATTR_HIDE* et *IATTR_ADMIN*, qui le rendent visible dans le contexte ADMIN, et celui-ci uniquement.

ADMIN.

4. Niveau *nonetwork*

Ce *softlevel* ne fait l'objet d'aucun traitement spécifique par */sbin/rc*, mais est lancé de manière spécifique par *init* au sein d'un système CLIP, avant le traitement des *runlevels System V* (mot clé « rc1 » dans *inittab*). Il est important de noter que contrairement à *boot*, tout script activé dans *nonetwork* qui ne serait pas repris dans *default,* niveau suivant dans CLIP, serait arrêté lors du passage de *nonetwork* à *default*, et éventuellement relancé ultérieurement au cours du traitement de *default* afin de satisfaire une dépendance *need*.

veriexec

Dépendances : *before reducecap clip_servers*
need localmount

Fichiers de configuration : */etc/veriexec/ctx*
/etc/veriexec/ctxlevels
/etc/veriexec/update
(/etc/verictl.d/)*
(/usr/local/etc/verictl.d/)*

Réalise la configuration de *veriexec* (cf. [CLIP_1201]) pour le socle et les cages CLIP. L'utilitaire *verictl* est invoqué successivement afin de réaliser les opérations suivantes, dans l'ordre :

- Création des contextes *veriexec* associés aux cages CLIP et RM (le contexte associé au socle est créé par défaut par le noyau), tels que définis dans */etc/veriexec/ctx*, avec les options et niveaux initiaux définis dans ce même fichier. Le format de chaque ligne de ce fichier est le même que celui passé par *–c <argument>* dans une commande *verictl* de création de contexte.
- Chargement des entrées *veriexec* associées aux fichiers (exécutables et bibliothèques) de la partition racine, telles que définies par les fichiers de */etc/verictl.d*. Ces fichiers sont au format des fichiers passés par *–f <fichier>* sur la ligne de commande de *verictl*. On notera que ces entrées peuvent être créées aussi bien dans le contexte du socle que dans l'un de ceux créés juste auparavant, selon la valeur de leur champ contexte. Fonctionnellement, il est attendu de ces fichiers qu'ils ne créent pas d'entrées dans les contextes des cages RM, cette tâche étant attribuée par ailleurs au script *clip_servers*.
- Activation de *veriexec* dans le contexte du socle, en lui attribuant un niveau inscrit en dur dans le fichier. Cette activation positionne aussi les drapeaux *enforce_mntro* et *lvl_immutable* pour ce contexte, avec pour conséquence qu'il devient impossible de créer ou supprimer des entrées pour des fichiers de la partition racine (montée en lecture seule).
- Configuration du numéro de contexte *veriexec* de mise à jour, écrit dans le fichier */etc/veriexec/update*, si ce fichier existe.
- Chargement des entrées *veriexec* associées aux fichiers des autres partitions (essentiellement */usr/local*), telles que définies par les fichiers de */usr/local/etc/verictl.d*. Ces fichiers sont au même format que ceux de */etc/verictl.d*, et sont aussi susceptibles de définir des entrées dans

l'un quelconque des contextes créés à ce stade.

- Activation et ajustement des niveaux des contextes associés aux cages CLIP (mais non RM), en fonction du contenu du fichier */etc/veriexec/ctxlevels*. Chaque ligne de ce fichier est au format suivant :

```
<contexte>  <niveau>
```

avec :

- <*contexte*> le numéro de contexte, selon les conventions du C.
- <*niveau*> le niveau final du contexte, sous la forme d'une liste de mots clés de contexte séparés par des ':'.

On notera que les connexions au réseau ne sont dans tous les cas possibles qu'après l'exécution de ce script, dans la mesure où le privilège CLSM d'accès au réseau ne peut être attribué que par une entrée *veriexec*.

clip_audit

Dépendances : *need clock hostname*
provide logger
Fichiers de configuration : */etc/jails/audit/**
[/etc/admin/conf.d/net]
(/etc/syslog-ng/syslog-ng.conf)

Crée et configure la cage $AUDIT_{clip}$, et lance un démon *syslog-ng* qui s'enferme dans cette cage. L'adresse de la cage est importée de */etc/admin/conf.d/net*.

A l'arrêt du service, tous les processus du contexte associé à la cage $AUDIT_{clip}$ sont terminés par *vsctl stop*.

clip_core_install

Dépendances : *need localmount*
Fichiers de configuration : ***/etc/admin/clip_install/****
*/mounts/update_root/etc/clip_update/**

Réalise la mise à jour du cœur du système CLIP installé sur la partition CLIP alternative, lorsqu'une telle mise à jour est disponible. Dans ce cas, le système redémarre automatiquement au terme de la mise à jour, avec la partition la plus à jour comme partition de démarrage par défaut.

sysctl

Dépendances : *need logfiles vprocunhide*
after clip_core_install
Fichiers de configuration : */etc/sysctl.conf*

Écrit les variables *sysctl* définies dans le fichier de configuration. A ce stade, ces affectations servent à :

- Désactiver l'*Explicit Congestion Notification* (ECN) dans la pile IP.
- Activer les *syncookies* TCP.
- Activer le *reverse path filter* IP.
- Désactiver le *source routing* et la prise en compte de toutes les redirections ICMP.
- Désactiver la réponse à tous les paquets ICMP *echo-request*.
- Configurer un redémarrage automatique en 3 secondes en cas de *kernel panic*.
- Réduire le *cap-bound* du système (cf. [CLIP_1204]), en retirant les capacités *CAP_SYS_RAWIO* (dont les effets sont dans tous les cas limités par *grsecurity,* cf. [CLIP_1203]) et *CAP_LINUX_IMMUTABLE*. *CAP_SYS_MODULE* est par ailleurs retirée ultérieurement par le script *reducecap*.
- Sur les passerelles uniquement, activer le routage des paquets (*ip_forward=1*).

On notera qu'après l'invocation de ce script, il devient impossible de modifier les fichiers de journaux (sauf en ajout), les fichiers de sauvegarde de journaux, et les attributs de visibilité dans */proc*. Ce script n'est invoqué qu'après *clip_core_install* de manière à permettre à ce dernier de positionner ou de modifier des attributs sur les fichiers installés sur le jeu de partition alternatif.

5. Niveau *default*

Ce niveau constitue le *softlevel* par défaut sous CLIP, activé au titre du *runlevel System V* numéro 3. Outre les scripts énumérés ci-dessous, les scripts activés dans *nonetwork* (sauf *clip_core_install*) sont repris dans ce niveau afin d'éviter qu'ils ne soient arrêtés et relancés lors du changement de niveau.

clip_user

Dépendances : *need localmount veriexec*
before xdm reducecap

Fichiers de configuration : */etc/fstab.user*
/etc/conf.d/clip
*/etc/jails/user/**

Crée l'arborescence de la cage $USER_{clip}$ en fonction de */etc/fstab.user*. On notera que ces montages sont réalisés dans le *namespace VFS* du socle et ne s'accompagnent pas de la création de la cage elle-même. La cage est créée par le démon *xdm* lors de chaque ouverture de session utilisateur, et hérite alors de l'arborescence créée par ce script.

Ce script assure aussi la suppression d'éventuels fichiers *Xauthority* propres aux sous-vues visionneuses (laissés en place suite à l'interruption brutale d'un service) avant l'exposition de ces fichiers dans la cage.

Enfin, le script assure au démarrage le lancement du démon *pwcheckd* (dans le socle), qui permet une ré-authentification des utilisateurs de $USER_{clip}$, notamment à fin de déverrouiller une session graphique (cf. [CLIP_1302]).

A l'arrêt du service, la cage $USER_{clip}$ est terminée, si elle est encore active, par appel à *vsctl stop*, puis */user/home/user* est démonté (s'il est encore monté – typiquement lorsque *stop()* est appelée alors qu'une session utilisateur est ouverte), et l'arborescence définie dans */etc/fstab.user* est démontée dans le *namespace* du socle, dans l'ordre inverse de son montage. Enfin, le démon *pwcheckd* est terminé dans le socle par un appel *killall pwcheckd*.

clip_viewers [CLIP-RM]

Dépendances : *need clip_user*
before reducecap

Fichiers de configuration : */etc/conf.d/clip*
/etc/viewers//**

Crée les arborescences des vues visionneuses au sein de la cage $USER_{clip}$, pour chaque cage RM déclarée dans */etc/conf.d/clip*. Un répertoire */tmp/.X11-unix* est créé au préalable dans le socle avec les droits *1777*, dans la mesure où ce répertoire doit être monté dans les arborescences de visionneuses.

Par ailleurs, ce script crée aussi les montages *loop* des répertoires */home/adminrmX* depuis les fichiers images */home/admin.rmX*, montages qui sont ensuite re-projetés dans les cages RM_X (cf. [CLIP_1304] et [CLIP_1401]). Ce script est utilisé pour ces montages préférentiellement à *clip_servers*, qui intervient après le verrouillage du système et ne peut donc plus conférer les capacités

nécessaires à l'utilitaire *mount*.

A l'arrêt du service, les arborescences sont démontées dans l'ordre inverse de leur montage.

Remarque 1 : Arrêt des vues visionneuses.

Contrairement à l'arrêt de la cage USER$_{clip}$, l'arrêt des vues visionneuses n'est à ce stade pas précédé de la terminaison des processus s'y exécutant. De ce fait, le démontage des arborescences correspondantes peut échouer, et entraîner l'échec du démontage de l'arborescence USER$_{clip}$. Une solution pourrait reposer sur vsctl*, appelé spécifiquement au sein des* chroots *internes à la cage USER$_{clip}$. On notera que le cloisonnement des* chroots *en termes d'envoi de signaux (réalisé par* grsecurity *et* CLIP-LSM, cf. [CLIP_1201]*) permettrait de réaliser l'arrêt souhaité par une séquence* kill –SIG{TERM|KILL} –1.

clip_x11

Dépendances : *need localmount veriexec*
before reducecap xdm
Fichiers de configuration : */etc/fstab.x11*

Crée l'arborescence de la cage X11 à partir de */etc/fstab.x11*. Tout comme pour la cage USER$_{clip}$, cette arborescence est créée dans le *namespace* du socle, tandis que la cage n'est créée qu'ultérieurement, au démarrage de *xdm*.

A l'arrêt du service, l'arborescence est simplement démontée dans le *namespace* du socle, dans l'ordre inverse de son montage[6].

netfilter

Dépendances : *before networking reducecap*
need veriexec
Fichiers de configuration : *[/etc/admin/conf.d/net]*
[/etc/admin/conf.d/netfilter]

Configure le pare-feu *netfilter* en fonction des adresses paramétrées dans */etc/admin/conf.d/net* et des ports autorisés dans */etc/admin/conf.d/netfilter*.

Voir aussi [CLIP_1501].

spm *(setkey ou spmd)*

Dépendances : *before networking reducecap clip_audit*
need veriexec
provide spm
Fichiers de configuration : *[/etc/admin/conf.d/net]*
/etc/ipsec.conf ou */etc/ike2/racoon2.conf.skel*

(*spm* constitue uniquement une dépendance virtuelle, satisfaite par l'un des deux scripts *setkey* ou *spmd*).

[6] Lors de l'appel à la fonction *stop()* de *clip_x11*, la cage X11 est déjà supposée terminée par l'arrêt du service *xdm*.

Configure les politiques de sécurité IPsec en fonction des adresses paramétrées dans */etc/admin/conf.d/net*. Cette configuration peut être réalisée de deux manières différentes, selon la configuration installée sur le poste :

- Soit par des commandes *setkey* définies dans */etc/ipsec.conf* (modifié par les adresses propres au poste), dans le cas où *racoon* (IKEv1) est utilisé (script *setkey*).
- Soit en lançant le démon *spmd* avec un fichier de configuration créé temporairement dans */var/run* à partir du squelette */etc/ike2/racoon2.conf.skel* et des adresses propres au poste, dans le cas où *racoon2* (IKEv2) est utilisé (script *spmd*).

A l'arrêt du service, aucune opération n'est réalisée dans le cas où *racoon* est utilisé, tandis que dans le cas où *racoon2* est utilisé, le démon *spmd* est terminé par la commande *spmd –k.*

Voir aussi [CLIP_1501].

networking

Dépendances : *need netfilter spm veriexec clip_audit*
Fichiers de configuration : *[/etc/admin/conf.d/net]*

Configure les interfaces réseaux (*lo* puis interfaces *ethX*) et la table de routage, en fonction des adresses paramétrées dans */etc/admin/conf.d/net*.

A l'arrêt du service, les interfaces réseaux (*ethX* puis *lo*) sont désactivées, et la table de routage est vidée.

Voir aussi [CLIP_1501].

reducecap

Dépendances : *before xdm raccoon iked*
after networking netfilter setkey spmd
need veriexec vprocunhide devctl
Fichiers de configuration : *aucun*

Réalise la dernière étape de verrouillage du système, en réduisant le masque de capacités par défaut de *root* par l'intermédiaire du *sysctl kernel.clip.rootcap* (cf. [CLIP_1201]), puis en activant les contrôles d'accès aux montages CLIP-LSM en mettant *kernel.clip.mount* à 0, puis enfin en supprimant la capacité *CAP_SYS_MODULE* de *kernel.cap-bound*.

backupsrv

Dépendances : *before xdm*
need reducecap
Fichiers de configuration : *aucun*

Lance le démon */sbin/backupsrv* de requête de sauvegarde/restauration du système CLIP (cf. [CLIP_DCS_15094]).

A l'arrêt du système, le démon est arrêté.

clip_admin

Dépendances : *need networking reducecap clip_audit*
Fichiers de configuration : */etc/jails/admin/**
[/etc/admin/conf.d/net]

Crée et configure la cage $ADMIN_{clip}$, en y lançant un démon *sshd*. L'adresse de la cage est importée de */etc/admin/conf.d/net*.

A l'arrêt du service, tous les processus du contexte associé à la cage $ADMIN_{clip}$ sont terminés par *vsctl stop.*

clip_update

Dépendances : *need networking reducecap clip_audit*
Fichiers de configuration : */etc/jails/update/**
[/etc/admin/conf.d/net]

Crée et configure la cage $UPDATE_{clip}$, en y lançant un démon *crond*. L'adresse de la cage est importée de */etc/admin/conf.d/net*. Avant le lancement du démon *crond*, le script */bin/cleanup_log.sh* (paquetage *sys-apps/busybox*) est lancé dans la cage de manière à supprimer les fichiers temporaires laissés dans le */var/log* de l'arborescence de la cage par *clip_install* et *clip_download*.

A l'arrêt du service, tous les processus du contexte associé à la cage $UPDATE_{clip}$ sont terminés par *vsctl stop.* Voir aussi [CLIP_1304].

kmpd *(racoon ou iked)*

Dépendances : *need spm reducecap networking*
provide kmpd
Fichiers de configuration : *[/etc/admin/conf.d/net]*
/etc/init.d/racoon ou */etc/ike2/racoon2.conf.skel*

(kmpd constitue uniquement une dépendance virtuelle, satisfaite par l'un des deux scripts *racoon* ou *iked*).

Lance le démon de négociation d'associations de sécurité IPsec. Deux démons peuvent être lancés selon la configuration installée sur le poste :

- Soit le démon *racoon* (IKEv1) est lancé (script *racoon*), avec un fichier de configuration temporaire généré à partir d'un squelette contenu dans le script lui-même et des adresses de */etc/admin/conf.d/net*
- Soit le démon *iked* (IKEv2) est lancé (script *iked*), avec un fichier de configuration temporaire généré à partir d'un squelette contenu dans */etc/ike2/racoon2.conf.skel* et des adresses de */etc/admin/conf.d/net*

A l'arrêt du service, le démon lancé au démarrage est arrêté.

Voir aussi [CLIP_1501].

clip_download

Dépendances : *need localmount kmpd clip_update*
Fichiers de configuration : */etc/jails/update/context*
[/etc/admin/clip_download/clip_download.conf]

Réalise éventuellement un téléchargement initial des mises à jour de CLIP (et, le cas échéant, des cages RM), puis lance le démon de contrôle des téléchargements. Le script lit en premier lieu le fichier de configuration */etc/admin/clip_download/clip_download.conf*. Si ce fichier ne contient pas une affectation *run_at_boot=yes*, aucun téléchargement n'est effectué. Dans le cas contraire, le ou les téléchargements sont lancés en appelant le script *clip_download* dans la cage $UPDATE_{clip}$. Le premier téléchargement (mises à jour de CLIP) est relancé cinq fois en cas d'erreur, avec des délais de 0.5, 1 puis 2 secondes entre deux appels successifs, afin de laisser le temps au démon IKE d'établir l'association de sécurité nécessaire au téléchargement.

Après cette phase de téléchargement, le démon de contrôle des téléchargements (qui permet à l'administrateur de demander explicitement un téléchargement, ou de verrouiller ou déverrouiller les mises à jour), */usr/sbin/downloadmaster*, est systématiquement lancé.

A l'arrêt du système, le démon *downloadmaster* est arrêté.

La variable *run_at_boot* est par défaut positionnée à « *no* », afin de ne pas réaliser de téléchargement pendant la séquence de démarrage, dans la mesure où ces opérations peuvent être coûteuses en temps (même lorsqu'aucune nouvelle mise à jour n'est disponible). Dans ce cas, le téléchargement des mises à jour est réalisé automatiquement toutes les heures par le démon *crond* de la cage $UPDATE_{clip}$, sans perturber de manière sensible le fonctionnement du reste du système. La possibilité est laissée à l'administrateur local d'activer un téléchargement initial au démarrage pour le cas de systèmes qui seraient laissés éteints pendant de longues périodes, pour lesquels il est important de télécharger et d'appliquer au plus tôt les mises à jour de sécurité.

clip_apps_install

Dépendances : *need localmount clip_update*
after clip_core_install clip_download
Fichiers de configuration : */etc/jails/update/context*[7]

Réalise ponctuellement au démarrage une mise à jour des paquetages secondaires de CLIP, en lançant le script *clip_install* au sein de la cage $UPDATE_{clip}$. Il est rappelé que, après cette invocation initiale, *clip_install* est relancé toutes les heures dans $UPDATE_{clip}$ par le démon *crond*.

[7] *clip_install* utilise ses propres fichiers de configuration, stockés dans */etc/admin/clip_install* et */etc/clip_update* (dans l'arborescence de la cage $UPDATE_{clip}$), mais ces fichiers ne sont lus qu'au sein de la cage.

rm_core_install [CLIP-RM]

Dépendances : *need localmount*
after clip_download
before clip_servers

Fichiers de configuration : */etc/jails/secure_update_*/**
/etc/conf.d/clip
[/etc/admin/conf.d/net]

Réalise la mise à jour des cœurs de cages RM, en créant les cages *secure_update_rm_X* correspondant aux cages RM du système (définies dans */etc/conf.d/clip*), et en y lançant le script *clip_install*, après avoir lancé le script */bin/cleanup_log.sh* pour supprimer les fichiers temporaires créés dans le répertoire */var/log* de la cage (qui correspond au répertoire */var/log* de la vue UPDATE de la cage RM correspondante) par les invocations précédentes de *clip_install*. L'adresse IP de chaque cage est importée depuis le fichier */etc/admin/conf.d/net*.

La journalisation des actions de la cage est assurée par un montage *bind* de la socket *syslog* */vservers/rm_X/var/run/logger*[8] sur le */dev/log* de la cage. Dans la mesure où ce */dev/log* est un lien symbolique vers *../var/run/log*, fichier qui n'existe pas nécessairement à ce stade (il n'est créé qu'au démarrage de la vue AUDIT de *rm_X*, au lancement de *clip_servers*, et n'existe en particulier pas au premier démarrage du poste, lorsque *rm_core_install* est lancé pour la première fois), le script *rm_core_install* crée ce fichier avant de lancer la mise à jour, et le supprime à la fin de celle-ci.

clip_servers [CLIP-RM]

Dépendances : *need networking vprocunhide reducecap clip_audit*

Fichiers de configuration : */etc/jails/rm_*/**
/etc/conf.d/clip
[/etc/admin/conf.d/net]

Configure et démarre les cages RM configurées dans */etc/conf.d/clip*, avec les adresses importées depuis */etc/admin/conf.d/net*. La séquence de démarrage d'une cage RM est la suivante :

- Configuration de la cage, et maintien de celle-ci, par *vsctl setup* (cf. [CLIP_1202])
- Lancement par *vsctl enter* du script *verictl.sh*, enfermé dans les sous-arborescences *chroot /update_root* puis */update* de la cage. Ce script va à son tour charger, dans le contexte *veriexec* associé à la cage[9] les entrées *veriexec* définies par les fichiers de */etc/verictl.d/* (dans l'arborescence de */update_root*) puis */usr/local/etc/verictl.d/* (dans l'arborescence de */update*).
- Activation de *veriexec* dans la cage par attribution au contexte correspondant du niveau défini dans le fichier *veriexec.lvl-up* de l'arborescence de configuration de la cage.
- Lancement, par *vsctl enter*, d'un démon *syslog-ng* dans la cage. L'exécutable et ses fichiers de configuration sont recherchés dans l'arborescence de la vue UPDATE de la cage, mais le démon est lancé à la racine de la cage par *vsctl*. Ce démon va ensuite (après ouverture de ses *sockets*

[8] Cette socket est créée par le démon *syslog-ng* du socle (cage $AUDIT_{clip}$), au lancement de *clip_audit*. Elle sert ultérieurement à recevoir les messages transmis par le démon *syslog-ng* de la vue AUDIT de la cage *rm_X*.

[9] Contexte dont il est rappelé qu'il a été créé auparavant par le script *veriexec*. La création de contextes *veriexec* n'est par ailleurs plus possible à ce stade du démarrage, du fait du positionnement, par ce même script *veriexec*, du drapeau *ctx_immutable* dans le niveau du contexte du socle.

d'écoute dans les différentes vues) s'enfermer lui-même par *chroot* dans la vue AUDIT de la cage.

- Lancement, par *vsctl enter*, de *jailmaster* dans la cage. Là encore, l'exécutable est recherché dans la vue UPDATE, mais lancé à la racine de la cage, à partir de laquelle il s'enfermera lui-même dans la vue USER, après avoir lancé les vues ADMIN et UPDATE.
- Terminaison de l'attente *vsctl setup* par un *vsctl endsetup*.

A l'arrêt du service, chaque cage est successivement terminée par la séquence d'opérations suivante :

- Désactivation de *veriexec* dans la cage par attribution au contexte correspondant du niveau défini dans le fichier *veriexec.lvl-down* de l'arborescence de configuration de la cage.
- Lancement par *vsctl enter* du script *verictl.sh*, enfermé dans les sous-arborescences *chroot /update* puis */update_root* de la cage. Ce script est dans ce cas lancé avec l'option *–u*, et va de ce fait supprimer, dans le contexte *veriexec* associé à la cage les entrées *veriexec* définies par les fichiers de */usr/local/etc/verictl.d/* (dans l'arborescence de */update*) puis */etc/verictl.d/* (dans l'arborescence de */update_root*).
- Terminaison de la cage par appel à *vsctl stop*.

On notera que l'arrêt de la cage ne supprime pas le contexte *veriexec* associé à celle-ci (ce qui serait de toute manière impossible du fait du niveau du contexte du socle), mais le ramène dans un état compatible avec un redémarrage de la cage par le même script. Il est en particulier possible d'invoquer *clip_servers restart* pour redémarrer les cages RM.

Voir aussi [CLIP_1401] .

clip_sshd

Dépendances : *need networking reducecap clip_audit*
Fichiers de configuration : */etc/jails/audit/**

Lance un démon *sshd* dans la cage $AUDIT_{clip}$. Ce lancement séparé du démon *ssh,* à la différence de la cage $ADMIN_{clip}$ pour laquelle le démon *ssh* est lancé directement par le script *clip_admin* qui crée la cage, est rendu nécessaire par le fait que *clip_audit* est invoqué avant la configuration des interfaces réseau, ce qui ne permet pas de lancer un démon *ssh* qui ne pourrait pas faire un *bind()* sur la boucle locale.

Aucune action spécifique n'est réalisée à l'arrêt du système, le démon *sshd* étant arrêté par l'arrêt du service *clip_audit*.

databackupsrv

Dépendances : *before xdm*
need reducecap
Fichiers de configuration : *aucun*

Lance le démon */sbin/databackupsrv* de requête de sauvegarde/restauration des données utilisateur (cf. [CLIP_DCS_15094]). A l'arrêt du système, le démon est arrêté.

rm_apps_install [CLIP-RM]

Dépendances : *need localmount clip_servers*
after rm_core_install
before xdm

Fichiers de configuration : */etc/jails/rm_*/context*
/etc/conf.d/clip

Lance une mise à jour initiale des paquetages secondaires des cages RM du système (telles que définies dans la variable *CLIP_JAILS* de */etc/conf.d/clip*) en lançant le script *clip_install* dans la vue UPDATE de chacune de ces cages.

usbadmin [CLIP-RM]

Dépendances : *before xdm*
Fichiers de configuration : *aucun*

Lance les démons */sbin/usbadmin_clip, /sbin/usbadmin_rmh* et */sbin/usbadmin_rmb* de gestion des clés USB pour les niveaux CLIP, RM_H et RM_B. A l'arrêt du système, ces démons sont terminés par *killall <nom du démon>* (plutôt que par *start-stop-daemon*, qui n'est pas compatible de l'utilisation d'un unique exécutable pour ces trois démons, qui sont lancés par des liens symboliques pointant vers */sbin/usbadmin*).

mdadmd [CLIP-GTW]

Dépendances : *need reducecap*
Fichiers de configuration : *aucun*

Lance le démon */sbin/mdadmd* de gestion des synchronisations RAID, sur une configuration de type passerelle. A l'arrêt du système, ce démon est terminé par *start-stop-daemon --stop*.

useradmin

Dépendances : *before xdm*
Fichiers de configuration : *aucun*

Lance les démons */sbin/usersrv*, */sbin/usersrvaudit* et */sbin/usersrvadmin* de gestion des comptes utilisateurs. A l'arrêt du système, ces démons sont terminés par *start-stop-daemon –stop*.

xdm

Dépendances : *need localmount reducecap clip_x11 clip_user clip_audit*
Fichiers de configuration : *[/etc/admin/conf.d/net]*
*/usr/local/etc/X11/**
/etc/inittab

Lance le démon XDM pour attendre les connexions utilisateur. Ce démon est lancé dans le socle, mais crée immédiatement la cage X11 en utilisant les montages réalisés par le script *clip_x11*, et en y lançant le serveur X11. Lors de chaque connexion utilisateur, le démon XDM assure aussi la création de la cage $USER_{clip}$, en utilisant pour cela les montages réalisés par le script *clip_user*.

Le lancement du démon n'est pas fait directement par le script, mais par *init* lui-même, depuis un *runlevel* spécifique, 'a' (*runlevel 'ondemand'*, qui exécute simplement des commandes supplémentaires, sans quitter le *runlevel* de départ, cf. *man inittab(5)*), qui lance uniquement la commande */usr/local/etc/X11/startDM.sh*. Cette procédure particulière permet d'éviter une *race-condition* potentielle entre *xdm* et les démons *agetty* (lancés uniquement dans une configuration de développement), pour l'accès aux *ttyN*. Le passage par un *runlevel* dédié assure que *xdm* n'est lancé qu'après le lancement de toutes les commandes spécifiées dans le *inittab* pour le *runlevel* par défaut, en particulier le lancement des démons *agetty*.

Lors de l'arrêt du service, le démon *xdm* est terminé par un *kill*, puis la cage X11 est terminée par un *vsctl stop*.

6. Arrêt du système : niveaux *reboot* et *shutdown*

Les deux niveaux *reboot* et *shutdown* font l'objet d'un traitement spécifique par */sbin/rc*, et ne sont associés à aucun sous-répertoire de */etc/runlevels*. Ils ne diffèrent que par la dernière commande invoquée en fin de traitement : */etc/init.d/reboot.sh* (qui appelle */sbin/reboot*) dans un cas, et */etc/init.d/shutdown.sh* (qui appelle */sbin/halt*) dans l'autre.

La commutation depuis un *softlevel* standard (typiquement *default* sous CLIP) vers l'un de ces deux niveaux s'accompagne d'une gestion particulière des dépendances lors de l'arrêt des services actifs. D'une part, ce n'est que dans ce cas (ou lors d'une commutation vers le *softlevel single*) que les scripts lancés au niveau *boot* (y compris ceux réellement lancés dès le niveau *sysinit*) sont arrêtés automatiquement. D'autre part, l'ordre d'arrêt est artificiellement modifié de telle sorte que ne soient arrêtés dans un premier temps que les services actifs non-critiques qui ne fournissent pas (par la directive *provide* dans leur déclaration de dépendances) le service *virtuel « logger »,* c'est à dire le service de journalisation système typiquement associé à un démon *syslog*. Au sein d'un système CLIP, ce service correspond au script *clip_audit*, qui n'est donc pas arrêté au cours de cette première passe. De plus, les services nécessaires au fonctionnement d'un service *logger*, au titre d'une dépendance *need*, ne sont pas non plus arrêtés à ce stade du traitement. Après cette première passe, les services de type *logger* sont arrêtés. Enfin, tous les services restants – c'est-à-dire normalement les services critiques ainsi que ceux nécessaires au fonctionnement de services *logger* – sont terminés. On notera que dans le cas de CLIP, le seul service non-critique dont dépende *clip_audit* est *veriexec,* qui ne réalise aucune opération lors de son arrêt, pas plus que les scripts critiques du système. Ainsi, les seules opérations réalisées après arrêt de la journalisation sont celles, décrites ci-dessous, qui sont inscrites dans *halt.sh.*

L'ordre d'arrêt des services sur un poste CLIP lors d'un arrêt ou d'un redémarrage depuis le niveau *default* est le suivant (les services notés entre parenthèses sont ceux qui ne réalisent aucune opération lors de leur terminaison) :

- ***xdm***
- ***backupsrv***
- ***clip_admin***
- ***(clip_apps_install)***
- ***(sysctl)***
- ***(logfiles)***
- ***(clip_backup_save)***
- ***(clip_data_backup_save)***
- ***(clip_backup_restore)***
- ***(clip_core_install)***
- ***(clip_data_backup_restore)***
- ***(rm_apps_install)***
- ***clip_servers***
- ***(rm_core_install)***

- *clip_download*
- *(clip_sshd)*
- *clip_update*
- *(clip_update_unlock)*
- *databackupsrv*
- *mdadmd*
- *racoon / iked*
- *(reducecap)*
- *clip_viewers*
- *clip_user*
- *clip_x11*
- *(consolefont)*
- *(devctl)*
- *(keymaps)*
- *networking*
- *(netfilter)*
- *(setkey) / spmd*
- *urandom*
- *usbadmin*
- *useradmin*
- *clip_audit*
- *(veriexec)*
- *(vprocunhide)*
- *(bootmisc)*
- *clock*
- *(hostname)*
- *(localmount)*
- *(checkfs)*
- *(checkroot)*

Une fois tous les services actifs arrêtés, les deux niveaux appellent (en le « sourçant ») le script */etc/init.d/halt.sh*, afin de réaliser les opérations spécifiques à l'arrêt du système. Sous CLIP, ce script réalise les opérations suivantes :

- Démontage de tous les systèmes de fichiers *tmpfs* non utilisés.
- Désactivation de tous les *swap*, et de la projection *dm-crypt /dev/mapper/swap0*.
- Écriture d'un enregistrement d'arrêt dans */var/log/wtmp*
- Arrêt de toutes les cages *vserver* encore actives par *vsctl stop.* Cette opération n'est réalisée qu'à des fins de robustesse de la séquence d'arrêt, les cages ayant normalement été terminées par les fonctions *stop()* des scripts qui les ont créées à l'origine.
- Désactivation de toutes les projections *dm-crypt* restantes, puis de toutes les projections *loopback* restantes. Là encore, il s'agit plus d'assurer la robustesse de la séquence d'arrêt, ces projections ayant normalement été désactivées par les scripts de fermeture de session. Ce passage est en particulier utile si la commande d'arrêt est lancée alors qu'une session utilisateur est en cours (ce qui n'est pas le cas normalement, le bouton de redémarrage n'étant accessible qu'en dehors de la session utilisateur).
- Démontage de */dev/pts*.
- Tentative de démontage de tous les montages de type autre que *tmpfs, devpts, sysfs, proc, usbfs*.
- Tentative de remontage en lecture seule de tous les montages restants. Cette tentative est répétée trois fois en cas d'échec, en réalisant un appel *killall5 -9* après chaque tentative pour essayer de forcer la terminaison des processus encore actifs.

Le script rend un code d'erreur correspondant à la dernière tentative de remontage en lecture seule. Enfin, le script d'arrêt, *shutdown.sh* ou *halt.sh* selon le niveau, est appelé afin de réaliser l'arrêt matériel.

Une particularité du système CLIP est que le masque de capacités POSIX attribué à *root* est réduit à l'issue de la séquence initiale du démarrage, ce qui rend certaines opérations de configuration réalisées au cours de cette séquence de démarrage impossibles à réaliser ultérieurement. Ainsi, la configuration des interfaces réseau nécessite la capacité *CAP_NET_ADMIN* lors de l'invocation de */sbin/ip*. La configuration initiale des interfaces réseau est réalisée par le script */etc/init.d/networking*, qui est lancé avant *reducecap*, à un moment où *root* dispose encore automatiquement de *CAP_NET_ADMIN,* ce qui permet de réaliser les opérations de configuration. En revanche, après le lancement du script *reducecap*, *CAP_NET_ADMIN* n'est plus attribuée à *root* lorsqu'il invoque */sbin/ip.* Il est ainsi impossible de reconfigurer à la volée les interfaces réseau, mais aussi de les désactiver, et ce y compris durant la séquence d'arrêt du système. De manière similaire, il n'est plus possible par défaut de démonter des montages VFS par appel à *umount*, ou de désactiver des *swaps* par *swapoff,* ce qui interdit un arrêt « propre » du système (par manque de la capacité *CAP_SYS_ADMIN*).

Une solution naïve au problème pourrait consister à attribuer, à l'aide de *veriexec* (cf. [CLIP_1201]), les capacités nécessaires aux exécutables (sous la forme de capacités « forcées ») idoines lorsqu'ils sont appelés par *root*. Cependant, cette attribution automatique réduirait fortement l'intérêt de la réduction des capacités attribuées par défaut à *root*, puisqu'elle permettrait à ce dernier de reconfigurer arbitrairement les interfaces réseau ou de réaliser des montages arbitraires lors du fonctionnement normal du système. Une solution plus adaptée est donc mise en oeuvre dans CLIP, qui s'appuie sur le mécanisme d'héritage de capacités, lequel permet de n'attribuer à un fichier exécutable qu'une capacité potentielle, activable par le masque de capacités héritables de l'appelant. Il est ainsi possible d'associer à */sbin/ip* une entrée *veriexec* telle que le processus résultant de l'exécution du fichier ne dispose de

CAP_NET_ADMIN (capacité effective) que si l'appelant est *root* et disposait déjà de *CAP_NET_ADMIN* dans son masque de capacités héritables. De telles capacités potentielles sont attribuées à tous les exécutables qui en ont besoin pour compléter la séquence d'arrêt. Ce travail est complété par un patch spécifique appliqué au démon *init* (paquetage *sys-apps/sysvinit*), avec pour effet que le démon maximise son masque de capacités héritables avant de passer dans les *runlevels* 1 ou 6 (correspondant respectivement aux *softlevels shutdown* et *reboot*). Une telle opération est possible dans la mesure où *init*, qui a été lancé bien avant *reducecap*, dispose de toutes les capacités (sauf *CAP_SETPCAP*) dans son masque permis. Son masque héritable est par défaut laissé à 0, mais peut à tout moment être monté jusqu'à la valeur de son masque permis. Lors du passage dans un *runlevel* d'arrêt, *init* maximise son masque héritable avant d'invoquer */sbin/rc shutdown|reboot*, qui hérite de ce masque héritable et le transmet au script *init.d* dont il appelle la fonction *stop()* et aux commandes qu'il exécute directement. Tous les utilitaires privilégiés invoqués dans ce cadre disposent ainsi des capacités nécessaires à leur fonctionnement, qui restent par ailleurs interdites en dehors de ces phases spécifiques d'arrêt.

Annexe A : Références

[CLIP_1001] *Documentation CLIP – 1001 - Périmètre fonctionnel du système CLIP-RM*

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

[CLIP_1202] *Documentation CLIP – 1202 – Vserver : description et mise en œuvre*

[CLIP_1203] *Documentation CLIP – 1203 – Grsecurity : description et mise en œuvre*

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

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

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

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

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

[CLIP_DCS_15094] *Document de conception de l'étude sur le retour à une configuration antérieure, CLIP-DC-15000-094-DCS Ed0 Rev 3*

[GENTOO_INIT] *Gentoo Linux Documentation : Initscripts*
http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=4

[FBSPLASH] *Projet fbsplash : http://dev.gentoo.org/~spock/projects/gensplash*

Annexe B : Liste des remarques

Annexe C : Liste des figures