Documentation CLIP

1002

Architecture de sécurité

Version	Date	Auteur	Commentaires
1.2	Vincent Strubel	24/10/2008	Ajout d'une fonction 3.8 décrivant le cloisonnement des rôles, et d'une section 5 décrivant les procédures de sécurité organisationnelles complémentaires aux mécanismes techniques, sur suggestion d'EADS D&S.
1.1.1	Vincent Strubel	15/10/2008	Ajout du contrôle des SP en fonction du contexte *vserver* au 3.7.
1.1	Vincent Strubel	01/10/2008	Ajout des traitements d'erreurs.
1.0.3	Vincent Strubel	08/08/2008	Prise en compte des remarques d'Olivier Levillain, mise à jour des remarques et des références.
1.0.2	Vincent Strubel	17/07/2008	Retrait des options GRKERNSEC_CHROOT_{MOUNT,MKNOD} de la configuration de *clip-kernel* (*2.6.22.24-r6*).
1.0.1	Vincent Strubel	18/06/2008	Correction des références.
1.0	Vincent Strubel	29/04/2008	Version initiale. A jour pour le *tag* CLIP_v03.00.01.

Table des matières

Introduction

Le présent document donne une description macroscopique des différentes fonctions de sécurité mises en oeuvre au sein d'un système CLIP, de leurs justifications et de leurs interdépendances. Cette description se veut générique et aussi indépendante que possible du type de système CLIP considéré (client CLIP mono- ou multi-niveaux, passerelle, etc...). Lorsque des différences significatives existent entre configurations CLIP, elles sont explicitement signalées dans le document. A défaut d'une telle mention, les fonctions décrites sont mises en oeuvre sur tous les systèmes CLIP.

Le document n'a pas vocation a décrire le détail de l'implémentation de chaque fonction de sécurité. Le lecteur est renvoyé à cette fin vers les différents documents de conception répertoriés en Annexe A. Par ailleurs, l'Annexe C liste un certains nombre de remarques de fond sur les fonctions de sécurité de CLIP, qui constituent autant de vulnérabilités résiduelles identifiées du système dans son implémentation actuelle.

Chaque fonction est décrite en quatre sections : une description du traitement réalisé par la fonction (hors gestion des erreurs), une description de l'objectif rempli par la fonction dans le cadre de la politique de sécurité globale de CLIP, une description des fonctions complémentaires assurant la mise en oeuvre correcte et systématique de la fonction (protection en intégrité et non contournement de la fonction), et enfin une section décrivant le traitement des erreurs, c'est-à-dire des opérations interdites par la fonction.

1 États du système

Les différents états dans lesquels un système CLIP (une fois installé) peut se trouver, et les transitions entre ces états, sont résumés dans la Figure 1.

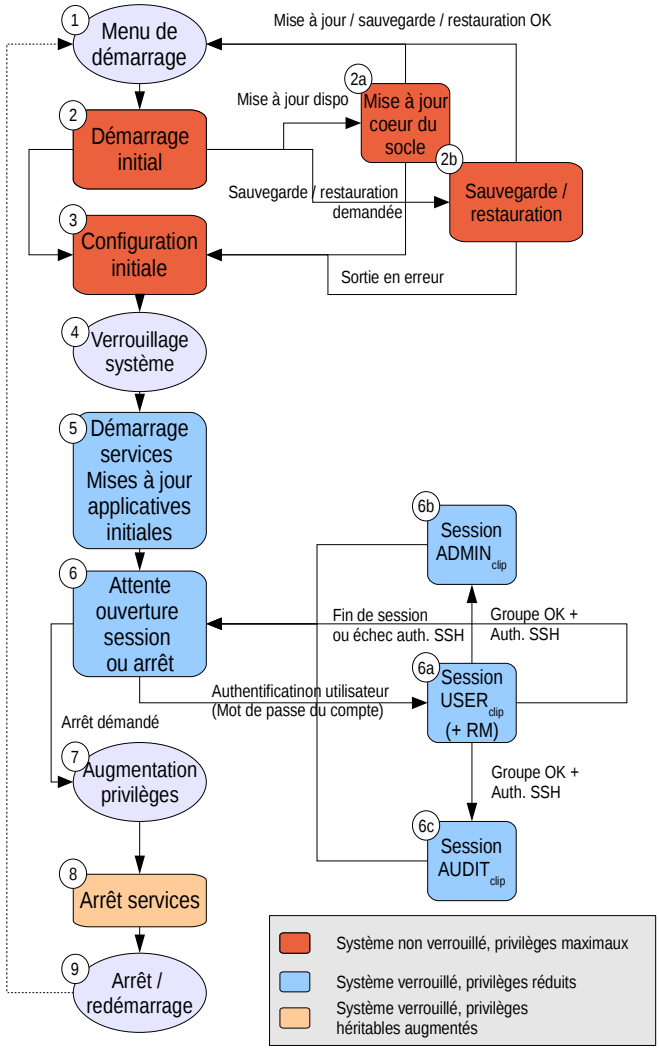

Figure 1: États d'un système CLIP et transitions entre ces états.

Ces différents états correspondent aux opérations suivantes (qui sont décrites avec plus de détail dans les document [CLIP_1001] et [CLIP_1301]):

1. Le menu de démarrage permet à l'utilisateur de sélectionner le système CLIP sur lequel il démarre : version à jour, ou version précédente. Après 10 secondes d'inactivité, le système à jour

est démarré par défaut.

2. Configuration initiale du système : horloge, vérification et montage des partitions de base, puis analyse des paquetages de mise à jour disponibles pour le coeur du socle, et des demandes de sauvegarde / réparation.

2a. Montage du système CLIP alternatif, et application de la mise à jour du coeur du socle, puis redémarrage en cas de succès uniquement (cf. [CLIP_DCS_13006]).

2b. Opérations de sauvegarde / restauration du système ou des données. Le déroulement normal de ces opérations donne normalement lieu à un redémarrage (avant sauvegarde ou après restauration), le cas échéant sur le système alternatif (cf. [CLIP_DCS_15094]).

3. Configuration initiale du système, sans lancer de processus pérenne autre que le démon *syslog-ng* de la cage $AUDIT_{clip}$. Cette configuration concerne notamment le positionnement de variables *sysctl*, le chargement des politiques de sécurité et de filtrage réseau, la configuration initiale de *devctl* et *veriexec*, et l'activation des interfaces réseau.

4. Verrouillage non révocable du système, par activation de *devctl* et des autres contrôles d'accès sur les montages, masquage global de certaines capacités, et réduction des capacités attribuées par défaut à *root* (cf. [CLIP_1201]). Ce verrouillage est réalisé en pratique en plusieurs étapes, entrecoupées d'autres étapes de configuration initiale (cf. [CLIP_1301]).

5. Mise à jour des paquetages secondaires CLIP et des éventuelles cages RM (paquetages primaires et secondaires), éventuellement (selon la configuration) précédée d'une recherche initiale de mises à jour disponibles sur le réseau. Les services du système (démons du socle, cages autres que $AUDIT_{clip}$) sont lancés à l'issue de ces mises à jour.

6. Attente d'une action de l'utilisateur sur l'écran d'ouverture de session XDM, soit pour ouvrir une session, soit pour demander l'arrêt ou le redémarrage du poste. Les services automatiques du socle, de $AUDIT_{clip}$, $UPDATE_{clip,}$ et des éventuelles vues AUDIT et UPDATE de cages RM, s'exécutent en tâche de fond pour :

- télécharger les paquetages de mise à jour
- appliquer immédiatement les mises à jour de paquetages secondaires
- éventuellement (en fonction de la configuration), maintenir la synchronisation horaire
- collecter les journaux et les répartir entre plusieurs fichiers de journaux
- négocier ou renégocier les associations de sécurité IPsec
- détecter et journaliser le branchement de support amovibles, bien qu'aucune autre action ne soit prise dans ce cas tant qu'aucune session utilisateur n'est ouverte

6a. Après authentification d'un utilisateur, les partitions (chiffrées, temporaires) de cet utilisateur sont montées dans les différents compartiments logiciels, puis une session graphique X11 est ouverte dans la cage $USER_{clip}$. Lorsque l'utilisateur ouvrant la session possède un profil administrateur ou auditeur (membre d'un groupe *core_admin* ou *core_audit*), cette session se limite à une ré-authentification auprès de la cage concernée pour ouvrir l'une des sessions décrites en 6b. ou 6c, et à exécuter les éventuels clients graphiques lancés depuis ces sessions. Sinon, la session offre un environnement graphique plus riche, permettant à l'utilisateur de gérer son compte (changement de mot de passe), et selon les cas d'utiliser des applications directement ou dans une session USER RM. Dès lors qu'une session $USER_{clip}$ est active (y compris dans le cas de sessions

d'administration ou d'audit, le branchement de supports amovibles de type clés USB déclenche un traitement spécifique, de vérification de signature, et en cas de succès de cette vérification, de déchiffrement du support. Quel que soit le type de session, les tâches de fond décrites en 6. poursuivent leur exécution. Les sessions $USER_{clip}$ accompagnées d'une session $ADMIN_{clip}$ ou $AUDIT_{clip}$ se terminent uniquement lors de la terminaison de cette session secondaire. Les autres sessions se terminent sur action de l'utilisateur au sein de $USER_{clip}$.

6b. Après ré-authentification d'un utilisateur de profil administrateur CLIP, une session interactive est lancée dans $ADMIN_{clip}$. La session se termine sur action de l'utilisateur, et entraîne la terminaison de la session $USER_{clip}$ associée.

6c. Après ré-authentification d'un utilisateur de profil auditeur, une session interactive est lancée dans $AUDIT_{clip}$. La session se termine sur action de l'utilisateur, et entraîne la terminaison de la session $USER_{clip}$ associée.

7. Une demande d'arrêt ou de redémarrage sur l'écran d'ouverture de session XDM entraîne un changement de *runlevel*, qui déclenche une maximisation du masque de capacités héritables du démon *init*. Ce masque est transmis à tous les nouveaux processus lancés par *init*, en particulier les scripts d'arrêt du système, qui disposent ainsi d'un niveau de privilèges plus élevé que les processus lancés en 5. et 6., sans pour autant remettre en cause le verrouillage du système. Ces privilèges sont dans l'ensemble nécessaires à la réalisation des opérations d'arrêt du système, et cette étape garantit que ces opérations ne peuvent pas être réalisées en dehors de la séquence d'arrêt complète.

8. Les différents services et processus lancés en 5. et 6. sont arrêtés. Certaines (mais pas toutes) des opérations de configuration lancées en 4. sont aussi annulées à ce stade[1].

9. Le système est arrêté ou redémarré.

[1] Par exemple, les interfaces réseau sont désactivées et déconfigurées. Les politiques de sécurité IPsec et de filtrage IP ne sont en revanche pas supprimées ou modifiées avant l'arrêt du système.

2 Fonctions de défense en profondeur

2.1 Protection du noyau

Description de la fonction

Cette fonction interdit l'injection de code arbitraire dans le noyau, en fermant plusieurs vecteurs permettant la modification de l'espace mémoire noyau depuis la couche utilisateur du système d'une part, et en durcissant l'espace mémoire noyau contre l'exploitation de vulnérabilités dans le code noyau lui-même d'autre part.

La fonction repose en premier lieu sur la suppression des moyens légitimes de modification de l'espace noyau. En particulier, la fonction interdit le chargement de modules noyau (à l'exception éventuellement d'une phase initiale de démarrage), la modification directe de la mémoire noyau par accès à */dev/mem* et */dev/kmem*, ainsi que l'accès bas niveau aux interfaces de programmation matérielles (*PIO* en particulier), qui permettent d'exploiter les fonctionnalités matérielles pour contourner la protection mémoire du noyau.

Par ailleurs, les options *PAX_KERNEXEC* et *PAX_UDEREF* ([CLIP_1203]) offrent plusieurs mesures de protection contre l'injection de code arbitraire en mode noyau, principalement en rendant les pages de données du noyau non exécutables et les pages exécutables non inscriptibles, et en interdisant le déréférencement par le noyau de pointeurs vers des fonctions sous contrôle de la couche utilisateur.

Objectif de la fonction

Cette fonction vise à protéger l'intégrité du noyau au cours de son exécution. Cette intégrité conditionne celle de toutes les autres fonctions de sécurité du système.

Protection et non-contournement de la fonction

- Toutes les mesures de protection du noyau pendant son exécution peuvent être trivialement contournées par une modification de l'image noyau sur le disque. Cette menace est écartée par la fonction 2.4.
- La fonction est implémentée par du code noyau uniquement, elle assure donc par construction son auto-protection.
- Le chargement de modules d'un niveau d'intégrité inférieur au noyau est systématiquement interdit :
    - Dans une configuration CLIP statique (noyau sans modules), l'interface de chargement de modules n'est même pas incluse dans le noyau.
    - Dans une configuration CLIP modulaire, l'*initrd* interdit totalement le chargement de modules à l'aide de *GRKERNSEC_MODULE* ([CLIP_1203]) avant de monter la partition racine du système. Ainsi, sous réserve d'intégrité de l'*initrd*, les seuls modules qui peuvent être chargés sont ceux contenus dans l'*initrd*. Or cet *initrd* est généré lors de la mise à jour du socle ([CLIP_1301]), par du code intègre du fait de 2.6, et ensuite

stocké dans la même partition que l'image noyau, et donc protégé par 2.4. Il peut de ce fait être considéré comme de même niveau d'intégrité que le noyau.

- De manière redondante avec les mesures précédentes, la capacité *CAP_SYS_MODULE*, nécessaire à toute opération sur les modules, est masquée globalement lors du verrouillage du système (cf. section 1 et [CLIP_1301]).

- Les autres accès « légitimes » à la mémoire noyau sont systématiquement interdits :
  - Toute ouverture des périphériques *mem*, *kmem*, *ports*, *ioports* est systématiquement interdite dès le démarrage du système, par *GRKERNSEC_KMEM* (modifié) et *GRKERNSEC_IO* ([CLIP_1203])
  - Ces périphériques ne sont dans tous les cas pas exposés dans un autre compartiment logiciel que le socle CLIP, du fait de 2.8.
  - Les appels système *sys_iopl()* et *sys_ioperm()* rendent systématiquement une erreur dès le démarrage du système, du fait de *GRKERNSEC_IO*.
  - L'interface de debug noyau, */proc/kcore*, est entièrement supprimée par *GRKERNSEC_PROC_ADD*
  - De manière redondante vis-à-vis des mesures précédentes, la capacité *CAP_SYS_RAWIO*, nécessaire à tous ces types d'accès, est masquée globalement lors du verrouillage du système (cf. section 1et [CLIP_1301]).
- L'injection directe de code arbitraire dans le noyau suite à l'exploitation d'une vulnérabilité dans ce dernier est rendue impossible par l'exclusivité des droits en exécution et en écriture sur les zones de mémoire noyau, apportée par *PAX_KERNEXEC* ([CLIP_1203]) : les zones où du code arbitraire peut être injecté ne sont pas exécutables, et les zones exécutables ne sont pas modifiables.
- Un contournement partiel de cette protection est possible par l'exploitation d'une vulnérabilité pour détourner le flot de contrôle vers du code existant, mais choisi arbitrairement, du noyau (attaque de type *return-to-libc*, bien que le terme ne soit guère adapté dans un contexte noyau). Cependant, cette menace de contournement est mitigée par différentes mesures de *PAX_KERNEXEC* et par *PAX_UDEREF*, ainsi que par la configuration générale du système :
  - *PAX_UDEREF* interdit l'exploitation de certains types de vulnérabilités
  - La non-inscriptibilité des tables de pages et de descripteurs de segments, apportée par *PAX_KERNEXEC*, interdit d'utiliser une telle attaque pour désactiver la protection du code noyau ou la non-exécutabilité des données noyau, et ainsi ouvrir la voie à une seconde attaque qui injecterait du code arbitraire.
  - *PAX_KERNEXEC* interdit aussi l'exécution en mode noyau des zones mémoires exécutables de l'espace utilisateur, ce qui interdit de rediriger le flot d'exécution vers une telle zone (qui contiendrait effectivement du code arbitraire du point de vue du noyau).
  - La protection en confidentialité de certaines informations sur l'espace d'adressage noyau apportée par *GRKERNSEC_PROC_ADD*, l'absence de */proc/kallsyms*, et la non lisibilité de l'image noyau et du *System.map* (cf. 2.4), compliquent une telle exploitation.
  - La journalisation PaX, et la journalisation des envois de signaux par *grsecurity*, améliorent les possibilités de détection d'une telle tentative d'attaque.

- 3.1 et 2.8, ainsi que le masquage par *vserver* des informations globales du */proc* dans les cages ([CLIP_1202]), limitent dans tous les cas l'exposition des interfaces noyau hors du socle.

Traitement des erreurs

Les accès aux interfaces noyau interdits par la fonction (ouverture de périphériques, accès PIO et chargement de modules) sont simplement refusés au niveau de l'appel système qui tente de les réaliser : l'appel système retourne un code d'erreur *-EPERM* ou *-EACCESS* selon l'appel, et aucune journalisation n'est réalisée. En revanche, les violations des fonctions PaX *KERNEXEC* et *UDEREF* entraînent une terminaison immédiate du processus fautif (sous la forme d'un "*Oops*" noyau), et la journalisation de l'erreur dans le fichier de journaux *pax.log*.

2.2 Protection des processus contre l'injection de code

Description de la fonction

La fonction limite significativement les possibilités d'exécution de code arbitraire dans les processus du système suite à l'exploitation d'une vulnérabilité conduisant à une corruption de la mémoire de ces derniers.

La fonction repose principalement sur la non-exécutabilité des zones mémoire de données des processus, assurée par *PAX_PAGEEXEC* ([CLIP_1203]). Elle est complétée par l'interdiction par défaut des projections mémoire exécutables et inscriptibles (*PAX_MPROTECT*), et par une fonction de prévention et de détection de certains types de corruption mémoire (dépassement de tampon dans la pile) assurée par *Propolice* ([CLIP_1101]). Le contournement de la fonction est compliqué par la randomisation mémoire (*PAX_RANDMMAP* et *PAX_RANDUSTACK*).

Ces différents éléments de protection peuvent être désactivés de manière sélective pour certains exécutables, par modification des options PaX de ces exécutables, ou des options de compilation utilisées pour leur génération. Cette désactivation est réalisée lors de la génération du paquetage correspondant. Par défaut, toutes les protections tolérées fonctionnellement par un exécutable donné sont activées.

Objectif de la fonction

Cette fonction offre une première ligne de défense contre l'exploitation de vulnérabilités logicielles dans la couche utilisateur. Elle contribue implicitement à la protection de toutes les autres fonctions de sécurité.

Protection et non-contournement de la fonction

Les protections *Propolice* sont incluses dans le code exécutable. Elles sont protégées par 2.7 (dans les compartiments logiciels auxquels cette fonction s'applique) et 2.4 (pour les exécutables du socle). Leur contournement nécessite soit de ne pas modifier le canari lors du dépassement de tampon, soit de contourner la vérification du canari.

- Il est difficile de reproduire le canari, dans la mesure où cela nécessite de pouvoir écrire les caractères 255 (non-ASCII) et '\n', ce qui est impossible dans plusieurs types de dépassement de tampon.
- Il est très difficile, sauf dans le cadre d'une attaque locale vers un processus de même niveau de privilège que l'attaquant (sans grand intérêt en général), de reproduire la portion aléatoire du canari, qui doit :
    - soit être devinée, ce qui est quasi-impossible (16 bits d'entropie régénérée à chaque lancement du processus, terminaison du processus et journalisation au premier échec)
    - soit être lue par exploitation d'une autre vulnérabilité
    - soit être lue par attachement *ptrace*, ce qui nécessite l'exécution de code arbitraire local au même niveau de privilèges que le processus attaqué

- Le contournement des vérifications de canari n'est possible que dans des situations bien précises, rares dans la pratique :
  - Utilisation par la fonction vulnérable de pointeurs de fonctions stockés sur les étages supérieurs de la pile
  - Levée d'une exception (C++) par la fonction vulnérable

La non-exécutabilité des zones mémoire de données est implémentée par le noyau, elle est protégée en intégrité par 2.1. Elle n'est pas désactivable globalement. Son contournement nécessite soit de désactiver la protection *PAX_PAGEEXEC* pour l'exécutable attaqué, soit d'exploiter le code existant de l'exécutable attaqué plutôt qu'un code injecté (attaque de type *return-to-libc*), éventuellement afin de parvenir à créer une zone de mémoire accessible à la fois en écriture et en exécution.

- La désactivation des protections PAX est impossible dynamiquement, du fait de la protection en intégrité du noyau et de la modification de *grsecurity* interdisant la configuration d'ACL *grsecurity* (qui pourraient sinon désactiver ces protections) ([CLIP_1203])
- La désactivation des protections PAX est possible par modification de l'exécutable lui-même, mais cette approche est impossible :
  - pour les compartiments affectés par cette fonction, du fait de 2.7
  - pour les exécutables du socle, du fait de 2.4
- Les attaques de type *return-to-libc* sont fortement compliquées par la randomisation mémoire (*PAX_ASLR* - [CLIP_1203] - complété par l'utilisation d'exécutables *PIE* - [CLIP_1101] - et la protection *GRKERNSEC_BRUTE* contre les attaques en force brute - [CLIP_1203])
- Les attaques de type *return-to-libc* ne peuvent en général pas créer de zones mémoire accessibles en écriture et exécution pour y injecter du code, du fait de *PAX_MPROTECT* ([CLIP_1203]).
- Les attaques de type *return-to-libc* sont mitigées par la perte de certains privilèges lors d'un appel *exec()*, notamment :
  - les privilèges spécifiquement attribués par CLIP-LSM ([CLIP_1201])
  - les privilèges d'accès au réseau ([CLIP_1201]) (y compris pour les *sockets* existantes, ce qui interdit la réutilisation de celles du processus exploité).
- La détection des tentatives de contournement est assurée par la journalisation *PaX* et *Propolice*.

Traitement des erreurs

Les erreurs *Propolice* (échec de vérification d'un "canari" lors du retour d'une fonction) entraînent la terminaison du *thread* fautif par la couche utilisateur, et la journalisation de l'erreur dans le fichier de journaux *ssp.log*. Les erreurs *PAX_PAGEEXEC* entraînent la terminaison du processus fautif (tous *threads* confondus) par le noyau, et la journalisation de l'erreur dans le fichier *pax.log*. Les erreurs *PAX_MPROTECT* entraînent simplement l'échec de l'appel fautif (*mmap()* ou *mprotect()*), qui retourne une erreur *-EPERM*, sans journalisation particulière. Enfin, les fonctions *PAX_RANDMMAP* et *PAX_RANDUSTACK* ne déclenchent pas directement d'erreurs, mais font généralement échouer une tentative d'exploitation de vulnérabilité de type *return-to-libc*. Dans la plupart des cas, ces échecs se

traduisent par des tentatives d'accès mémoire interdits, entraînant l'envoi d'un signal *SIGSEGV* au *thread* fautif. Ce signal entraîne normalement la terminaison du *thread* (sauf traitement spécifique du signal), et est dans tous les cas journalisé dans le fichier *grsec.log*.

Remarque 1 : Exécutables à protection réduite

Les protections PaX *et* Propolice *peuvent être désactivées sélectivement pour certains exécutables par les développeurs de paquetages. A ce stade, aucun exécutable déployé dans un système CLIP ne désactive* PAX_PAGEXEC. *En revanche, plusieurs exécutables sont déployés avec* Propolice *et/ou* PAX_MPROTECT *désactivés, ce qui réduit la résistance de la fonction pour ces exécutables.*

2.3 Réduction des privilèges

Description de la fonction

Cette fonction réduit les privilèges accordés aux processus du système au minimum nécessaire à leur fonctionnement. Les privilèges accordés automatiquement aux processus *root* sont réduits, lors du verrouillage du système (cf. section 1) à un sous ensemble « non dangereux » des capacités POSIX, composés de privilèges dont la portée est typiquement limitée à la couche utilisateur, ne permettant pas de configuration globale du système. Les bits 's' sur les exécutables ne permettent pas l'attribution des privilèges *root* à un utilisateur non privilégié. Certains privilèges supplémentaires peuvent être accordés lors de l'exécution (par *root* uniquement, ou par tout utilisateur) de certains exécutables, après vérification de leur empreinte cryptographique (et éventuellement de celles de leurs bibliothèques). Cette attribution peut éventuellement être limitée aux cas où le processus dispose déjà des privilèges supplémentaires dans son masque héritable, ce qui permet de créer des chemins d'exécution privilégiés. Les privilèges supplémentaires accordés ne permettent en aucun d'obtenir certaines capacités masquées globalement (*CAP_SYS_ADMIN*, *CAP_SYS_MODULE*, *CAP_LINUX_IMMUTABLE*), et ne permettent pas, au sein d'une cage *vserver*, d'accéder à des privilèges interdits par la cage.

La réduction des privilèges par défaut de *root*, et la non prise en compte des bits *setuid root* sont apportés par le LSM CLIP ([CLIP_1201]), de même que l'attribution de privilèges supplémentaires, par le sous-système *veriexec*. Le masquage global de capacités est obtenu par réduction du *sysctl kernel.cap-bound* lors du verrouillage du système ([CLIP_1301]). Le masquage de capacités par cage *vserver* relève de 3.1. *veriexec* ne permet pas de dépasser ces différentes limites, dans la mesure où elles sont évaluées après l'ajout des capacités apportées par *veriexec*, et dans la mesure où le LSM CLIP ne permet pas le dépassement du *cap-bound* par les capacités héritables.

Objectif de la fonction

La réduction de privilèges vise à limiter la portée d'une attaque réussie par exploitation d'une vulnérabilité dans un exécutable du système. A ce titre, elle contribue à la protection et au non-contournement de la plupart des autres fonctions de sécurité du système.

Protection et non-contournement de la fonction

- L'attribution des privilèges est réalisée par le noyau, dont l'intégrité est assurée par 2.1.
- La réduction globale des privilèges *root* ne peut pas être désactivée une fois le système verrouillé : le masque de capacités attribuées par défaut à *root* ne peut dans tous les cas qu'être réduit, et même cette opération devient impossible après la suppression de *CAP_SYS_MODULE* du *cap_bound*.

Deux approches sont possibles pour contourner la fonction : créer une entrée *veriexec* donnant plus de privilèges à un exécutable, ou prendre le contrôle d'un exécutable privilégié

- la première approche est interdite par le contrôle d'accès *veriexec* depuis tout compartiment autre que le socle ou $\text{UPDATE}_{\text{clip}}$ (ou la vue UPDATE d'une cage RM, avec une portée limitée à cette cage).
- la première approche est interdite par le contrôle d'accès *veriexec* pour tout exécutable du coeur

de chaque compartiment (plus généralement, tout exécutable accessible en lecture seule du fait des options de montage). La fonction 2.7 vient compléter cette mesure, pour les compartiments qu'elle concerne, en interdisant à l'attaquant d'utiliser une copie de l'exécutable souhaité installée par ses soins hors du coeur du compartiment afin d'échapper à ce contrôle d'accès

- la désactivation de ce contrôle d'accès est interdite dans le socle par 2.4
- la désactivation de ce contrôle d'accès est interdite dans les autres compartiments, de manière redondante, par :
  - le contrôle d'accès sur les montages du LSM CLIP ([CLIP_1201])
  - le masquage de *CAP_SYS_ADMIN* par les cages *vserver* ([CLIP_1202])

- La deuxième approche est contrée par différentes protections des processus privilégiés apportées par le LSM CLIP, en particulier :
  - l'interdiction de toute opération *ptrace* ou de l'accès aux fichiers privilégiés du répertoire */proc/<pid>* d'un fichier plus privilégié, même exécuté sous la même identité
  - l'interdiction d'envoyer un signal à un tel processus, y compris par des moyens détournés
  - la protection accrue du chargement de bibliothèques pour un tel processus
  - la protection contre les écritures de tous les fichiers projetés en mémoire avec des permissions en exécutions (ces fichiers servant de *swap* pour leurs projections mémoire, il serait sinon possible de modifier la mémoire d'un processus plus privilégié en modifiant par exemple une de ses bibliothèques sur le disque)
  - la protection systématique contre les écritures du périphérique utilisé pour le *swap* (réalisée par la configuration *devctl*, et complétée par l'interdiction de tout *swapon* après l'activation de ce dernier).
- La deuxième approche est aussi contrée par la protection en intégrité des exécutables et bibliothèques sur le disque, assurée par les fonctions 2.4 et 2.7, qui complète la vérification d'empreintes cryptographiques par *veriexec*.
- Les possibilités d'exploiter une vulnérabilité dans un processus plus privilégié sont largement réduites par 2.2.
- La journalisation *PaX*, *grsecurity* (envoi de signaux en particulier) et CLIP-LSM (échecs de vérifications *veriexec*, opérations interdites entre processus de niveaux de privilèges différents), couplée à 2.9, facilite la détection des tentatives de contournement de la fonction.

Traitement des erreurs

La fonction ne génère aucune erreur directement : les privilèges non nécessaires ne sont simplement pas attribués. Indirectement, ces privilèges manquants peuvent entraîner l'échec d'appels systèmes privilégiés, qui rendent dans ce cas un code de retour *-EPERM* ou *-EACCESS* selon les appels, sans journalisation spécifique.

2.4 Protection en écriture des fichiers du coeur du socle

Description de la fonction

Cette fonction interdit, en dehors de la phase de démarrage, toute écriture sur les deux partitions du disque qui contiennent le coeur du socle CLIP (paquetages primaires) : la partition montée à la racine du système, et la partition contenant le noyau et le chargeur de démarrage (ci-après dénommée « partition d'amorçage »). Elle est implémentée par le montage en lecture seule de la racine, et l'absence de tout montage (en dehors des opérations de mise à jour du socle ou de sauvegarde / restauration) de la partition d'amorçage, complétés de mesures spécifiques interdisant toute modification de ce plan de montage, et toute écriture par accès direct aux périphériques de type bloc associés à ces montages. Cette interdiction d'accès est globale, applicable à tous les utilisateurs (quel que soit leur niveau de privilèges), quel que soit le compartiment logiciel (socle CLIP y compris).

De manière complémentaire à ces mesures de sécurité, les paquetages primaires CLIP sont construit de manière à ne pas installer leurs fichiers sensibles (exécutables, fichiers de configuration) en dehors de ces deux partitions. Certains paquetages sont de plus adaptés de manière à s'accommoder d'une racine en lecture seule, notamment en remplaçant les fichiers normalement modifiables par des liens symboliques – non modifiables – vers des fichiers modifiables stockés sur d'autres partitions (par exemple */etc/mtab*, remplacé par un lien symbolique vers */proc/mounts*, ou */etc/passwd* par un lien vers */etc/core/passwd*).

Objectif de la fonction

Cette fonction vise à garantir l'intégrité sur le disque des fichiers fondamentaux du système, dont découle l'intégrité de la plupart des autres fonctions de sécurité.

Protection et non-contournement de la fonction

- La fonction est réalisée par le noyau, sur la base d'une configuration initiale réalisée au cours de la séquence de démarrage. L'intégrité de ces deux composants est assurée par les fonctions 2.1 et 2.6.
- Le contrôle du droit d'accès en écriture est systématiquement réalisé pour toutes les opérations sur les fichiers d'un système de fichiers. Ces opérations sont naturellement impossibles pour un système de fichiers non monté. Par ailleurs, *devctl* ([CLIP_1201]) interdit les accès en écriture à bas niveau (sans passer par le système de fichiers des partitions protégées) à travers les périphériques blocs */dev/sdaX* (ou */dev/hdaX*), notamment :
    - les écritures directes sur les périphériques
    - les projections mémoire en écriture (*MAP_SHARED*) des périphériques
    - les projections *loop* ou *device-mapper* des périphériques, avec des droits en écriture

La fonction n'est par conséquent pas jugée contournable tant qu'elle est active et correctement configurée. Sa désactivation ou sa reconfiguration nécessitent soit de désactiver ou reconfigurer *devctl*, soit de modifier les options de montage de ces deux partitions.

- Les fichiers périphériques correspondant à ces deux partitions, et au disque entier, ainsi que

leurs éventuels points de montages, ne sont présents, par construction, que dans le socle CLIP. Un contournement est donc a priori impossible depuis tout autre compartiment logiciel. La création d'un nouveau périphérique dans ces compartiments est interdite par 2.8.

- *devctl* est activé par le positionnement à 0 de la variable *sysctl kernel.clip.mount*. Par ailleurs, la création ou la suppression d'entrées *devctl* est interdite tant que *kernel.clip.mount* est nulle. L'interface *sysctl* associée à cette variable ne permet que la mise à zéro de bits de la variable, et en aucun cas leur mise à un. Il est donc impossible de désactiver ou reconfigurer *devctl* une fois la fonction activée.
- La capacité *CAP_SYS_ADMIN*, nécessaire à la modification des montages, n'est attribuée qu'à un nombre très restreint de processus *root* du socle uniquement, du fait de la mise en oeuvre de *veriexec* ([CLIP_1201]). Les contrôles supplémentaires sur les montages apportés par CLIP-LSM interdisent le remontage en lecture écriture de tout système de fichiers monté en lecture seule. De plus, le sous-système *devctl* est aussi interrogé lors de chaque opération de montage, et sa configuration interdit tout montage en lecture/écriture de la partition racine, et tout montage de la partition d'amorçage.
- La fonction n'est pas active au démarrage du système. Cependant, la menace correspondante est mitigée par 2.6, qui permet d'écarter les possibilités d'opérations malveillantes lors de cette phase de vulnérabilité. On notera cependant que 2.6 dépend elle-même de la fonction courante. La non-vulnérabilité de la séquence de démarrage fait en effet intervenir la transitivité de la fonction de protection du socle contre les écritures :
  - Le socle est supposé intègre au début du premier démarrage du système (suite à une installation depuis un support de confiance).
  - Si le socle est supposé intègre au début du démarrage *n*, *2.6* sera assurée lors de ce démarrage, ce qui protégera l'intégrité du socle jusqu'à l'activation de la présente fonction, qui assurera à son tour l'intégrité jusqu'au prochain démarrage *n+1* (la modification du disque alors que le système est éteint étant écartée par hypothèse).
  - Par récursivité, le socle reste intègre et 2.6 est valable lors de chaque démarrage.

Traitement des erreurs

Le montage en lecture-seule entraîne un échec des appels système (typiquement *open()* avec l'option *O_WRONLY* ou *O_RDWR*) qui octroieraient un accès en écriture à un fichier des partitions concernées, avec un code de retour *-EROFS* et sans journalisation. Les accès directs aux périphériques de type bloc sous-jacents (par *open()*, *mount()* ou un *ioctl()* de projection *loop* ou *device-mapper*) sont refusés avec une erreur *-EACCESS*, et sont journalisés dans le fichier *clsm.log*.

Remarque 2 : Accès en écriture au socle sur les configurations utilisant un disque RAID

Sur les configurations CLIP utilisant un disque RAID (configurations passerelles), devctl *n'est pas configuré de manière à interdire tout accès en écriture à bas niveau aux partitions racine et d'amorçage, dans la mesure où un tel accès est nécessaire à la gestion de la redondance RAID sur demande de l'administrateur local (par exemple, remplacement d'un disque), selon les modalités décrites en [CLIP_DCS_15093]. La fonction de protection en écriture*

du socle est donc toujours contournable sur une telle configuration par un attaquant de bon niveau technique (suffisant à interpréter à bas niveau la structure du système de fichiers stocké sur le périphérique), qui disposerait de la capacité à exécuter du code sous l'identité root *dans le socle.*

2.5 Protection en écriture du coeur des compartiments UPDATE

Description de la fonction

Cette fonction assure la protection contre les écritures des coeurs (paquetages primaires) des compartiments de mise à jour, cage $UPDATE_{clip}$, et éventuellement vues UPDATE des cages RM. Cette protection est assurée par le montage en lecture seule de cette racine, dans l'arborescence de la cage, complété par l'aménagement des paquetages primaires afin de s'adapter à une telle restriction.

Objectif de la fonction

L'objectif de cette fonction est de protéger le coeur de ces compartiments, en particulier les utilitaires de mise à jour, contre toute modification, et ainsi de limiter les possibilités de pérennisation d'une attaque qui permettrait l'exécution de code arbitraire dans ces compartiments. A la différence de la protection offerte par 2.7, une telle attaque serait en mesure de modifier de manière pérenne des exécutables utilisés par les autres compartiments (mais normalement pas par le compartiment UPDATE concerné), mais pas d'interdire ultérieurement la mise à jour intègre de ces exécutables. Cette fonction contribue ainsi fortement à la protection de 4.4 et 4.5.

Protection et non-contournement de la fonction

- Le contrôle des options de montage est assuré entièrement par le noyau, dont l'intégrité est assurée par 2.1.
- La configuration initiale des options de montage est réalisée au cours de la séquence de démarrage, de manière intègre du fait de 2.6.
- La reconfiguration des options de montage est impossible depuis tout compartiment autre que le socle, du fait du masquage de *CAP_SYS_ADMIN* dans toutes les cages *vserver* ([CLIP_1202]) et du contrôle supplémentaire des options de montage réalisé par CLIP-LSM ([CLIP_1101]).
- Le contournement des options de montage par écriture directe sur le périphérique de type bloc associé à ces montages est impossible depuis tout compartiment autre que le socle, car ce périphérique n'est pas exposé dans ces compartiments (cette propriété étant elle-même assurée par 2.8)
- La reconfiguration des options de montage du coeur de $UPDATE_{clip}$, ou l'accès direct au périphérique bloc correspondant, sont impossibles dans le socle CLIP une fois le système verrouillé. En effet, le coeur de $UPDATE_{clip}$ est installé sur la partition racine du système, qui est protégée par 2.4.

Traitement des erreurs

Le montage en lecture-seule entraîne un échec des appels système (typiquement *open()* avec l'option *O_WRONLY* ou *O_RDWR*) qui octroieraient un accès en écriture à un fichier des partitions concernées, avec un code de retour *-EROFS* et sans journalisation.

2.6 Protection en intégrité de la séquence de démarrage

Description de la fonction

La fonction assure la protection en intégrité des scripts exécutés lors de la séquence de démarrage jusqu'au verrouillage du système. Cette intégrité concerne aussi bien les scripts et exécutables invoqués au cours de cette séquence que les données qu'ils manipulent. Elle repose sur deux propriétés. D'une part, l'ensemble des scripts et exécutables invoqués au cours de la séquence de démarrage est protégé en intégrité (vis-à-vis de tout utilisateur interne au système après verrouillage) sur le disque, de même que la plupart des fichiers de configuration de ces scripts et exécutables. D'autre part, tous les fichiers de configuration utilisés par ceux-ci qui ne sont pas ainsi protégés en intégrité sont, soit manipulés à l'aide de fonctions d'import sécurisé de paramètres de configuration, qui permettent de n'importer que les paramètres nécessaires après validation de leur format, soit font l'objet d'une vérification de signature.

Objectif de la fonction

Cette fonction assure l'intégrité de l'ensemble du code exécuté avant le verrouillage du système, dans une configuration temporaire dans laquelle plusieurs autres fonctions de sécurité ne sont pas valables. Elle assure ainsi en particulier le non-contournement de 2.4 et de 4.5. De plus, elle garantit que seul les services légitimes et nécessaires au fonctionnement du système sont effectivement lancés.

Protection et non-contournement de la fonction

Les scripts de démarrage, de même que l'essentiel de leur fichiers de configuration, sont stockés sur la partition racine du système, et sont donc protégés en intégrité sur le disque, dès lors que le système est dans l'état verrouillé, par 2.4. Ces scripts sont de plus écrits de manière à n'invoquer que des exécutables situés sur la même partition, qui ne mettent eux-mêmes en oeuvre que des bibliothèques et des fichiers de configuration installés sur cette même partition. Cette dernière propriété est assurée par la répartition des paquetages CLIP entre paquetages primaires (installés sur la partition racine) et paquetage secondaires, les paquetages primaires formant un tout cohérent, sans dépendance vis-à-vis de paquetages secondaires.

Les seules exceptions à ce principe sont :

- Les fichiers de configuration modifiables par l'administrateur local (dans la cage $ADMIN_{clip}$), qui sont traités par des fonctions d'import spécifiques (*/lib/clip/import.sub*), garantissant l'import sélectif des variables utiles, avec vérification de leur format par une expression régulière. Ces fonctions, décrites plus en détail dans [CLIP_1301], interdisent une corruption de la séquence de démarrage par l'administrateur local. La modification des fichiers de configuration concernés est par ailleurs réservée au rôle d'administrateur local, du fait de 3.8. On notera de plus qu'en dehors de ces imports sécurisés, la lecture des fichiers de configuration

supposés intègres est réalisée par une commande *source* en langage *bash*. Dans la mesure où l'interpréteur *bash* est modifié de manière à n'accepter pour *source* que des fichiers accessibles par un montage permettant l'exécution de programmes (c'est-à-dire sans option *noexec*), la fonction 2.7 contribue à bloquer d'éventuels détournement des fonctions d'import non sécurisés vers des fichiers sous le contrôle des utilisateurs de compartiment logiciels autres que le socle CLIP et les cages et vues UPDATE.

- Les fichiers de paquetages manipulés lors de la mise à jour du coeur CLIP. Les signatures des paquetages sont vérifiées avant tout autre manipulation par la séquence de démarrage (cf. [CLIP_DCS_13006]), ce qui protège l'intégrité de cette portion de la séquence de démarrage vis à vis de paquetages malicieux créés par la cage $UPDATE_{clip}$, par ailleurs seule cage capable d'injecter de tels paquetages malicieux du fait de 3.1.
- Les fichiers de sauvegarde manipulés par les scripts de sauvegarde / restauration (cf. [CLIP_DCS_15094]). Ces fichiers ne sont pas authentifiés, à la différence des paquetages. En revanche, ils ne sont accessibles en écriture que lors de la séquence de démarrage, avant verrouillage. En effet, après verrouillage du système, les partitions de stockage des fichiers de sauvegarde sont montées en lecture seule, et leur montage en lecture-écriture est interdit par les permissions *devctl* (cf. [CLIP_1201]). Ainsi, ces fichiers sont protégés en intégrité vis-à-vis de tout attaquant interne au système, une fois celui-ci verrouillé. Les éventuels modifications de ces fichiers ne sont réalisées que par le code de la séquence de démarrage elle-même, qui est intègre par transitivité de la présente fonction.

Traitement des erreurs

Les accès en écriture (appels système *open()*) aux fichiers protégés en intégrité sont refusés, avec le code d'erreur *-EROFS*, et sans journalisation. Les erreurs d'importation depuis les fichiers non protégés en intégrité (absence du fichier, valeur importée non conforme à l'expression régulière attendue) entraînent l'échec de l'importation (la variable importée n'est pas définie), qui est traitée au cas par cas par le script de démarrage concerné (entraînant le basculement en mode réseau sans échec dans le cas des scripts réseau, cf. [CLIP_1501], et le non démarrage du script dans les autres cas). Ces erreurs d'importations sont par ailleurs signalées par un message sur la console (le *"splash screen"* de démarrage disparaît, le message d'erreur est affiché sur la console texte précédé d'un caractère *'*'* jaune ou rouge), et, sous réserve que l'erreur se produise après le démarrage du service de journalisation, journalisées dans le fichier *messages*.

Remarque 3 :* Intégrité des méta-données *dpkg* et *apt

Les méta-données des outils de gestion de paquetages dpkg *et* apt *sont stockées, pour ce qui concerne CLIP, dans /var/pkg, répertoire qui reste accessible en écriture à root du socle CLIP et de la cage* $UPDATE_{clip}$*. Ces fichiers ne sont pas spécifiquement authentifiés. Il reste donc possible, pour root du socle ou de* $UPDATE_{clip}$*, de porter atteinte à l'intégrité de la séquence de démarrage en modifiant de manière malicieuse ces fichiers, selon l'une ou l'autre des méthodes suivantes :*

- *soit, en exploitant une vulnérabilité de* dpkg *ou* apt*, afin de détourner de manière malicieuse leur exécution. Cette approche est cependant*

partiellement bloquée par 2.2.

- *soit, en modifiant directement les scripts de désinstallation de certains paquetages installés dans le coeur CLIP (ces scripts sont en effet stockés parmi ces méta-données), de manière à exécuter des commandes arbitraires lors de la prochaine mise à jour de ces paquetages.*

2.7 W^X sur les fichiers

Description de la fonction

La fonction assure l'exclusivité des droits en écriture et en exécution sur les montages VFS, dans les compartiments logiciels suivants :

- $ADMIN_{clip}$
- $AUDIT_{clip}$
- $USER_{clip}$
- vues ADMIN, AUDIT et USER des cages RM_H et RM_B

La fonction est principalement implémentée à l'aide des options de montage *noexec* et *ro/rw*. La configuration des montages visibles au sein des arborescences des différents compartiments concernés fait que les montages accessibles en écriture (c'est-à-dire sans option *ro*) se voient tous attribuer l'option *noexec*. Cet ajustement des options de montage repose largement sur le support de montages *bind* en lecture-seule, apporté par *vserver* ([CLIP_1202]).

La configuration des montages interdit l'exécution par le noyau d'exécutables ou de bibliothèques dans les montages *noexec*. Elle est complétée par la mise en oeuvre d'interpréteurs modifiés (*shell* des différents compartiments), qui s'assurent de l'absence de l'option de montage *noexec* avant d'exécuter (ou de « sourcer ») un fichier de commandes.

On notera que la fonction n'est pas valable dans son intégralité pour la cage $UPDATE_{clip}$ et pour les vues UPDATE d'éventuelles cages RM, mais qu'une protection partielle est cependant apportée dans ces cas par 2.5.

Objectif de la fonction

Le W^X sur les fichiers interdit la pérennisation, par l'installation d'exécutables ou de bibliothèques malicieux, d'une attaque réussie qui permettrait à l'attaquant d'exécuter du code ou des commandes arbitraires dans un ou plusieurs des compartiments logiciels concernés.

Protection et non-contournement de la fonction

- La fonction principale est implémentée au sein du noyau et protégée par 2.1. Sa configuration initiale est réalisée par des exécutables du socle à partir de fichiers de configuration du socle, l'ensemble étant protégé en intégrité par 2.4. La fonction ne peut pas être désactivée, mais peut éventuellement être reconfigurée par modification des options de montage, ce qui nécessite la capacité effective *CAP_SYS_ADMIN*.
- La fonction ne peut pas être reconfigurée depuis les compartiments qu'elle concerne car les options de montage ne peuvent pas être modifiées. Cette propriété est garantie par deux éléments redondants :
  - le masquage systématique de la capacité *CAP_SYS_ADMIN* dans les cages *vserver* associées ([CLIP_1202])
  - l'interdiction par CLIP-LSM d'augmenter la permissivité des options de montage par

remontage ou montage *bind* ([CLIP_1201])

- La fonction ne peut pas être reconfigurée par un utilisateur non-*root* du système (tous compartiments confondus) dans la mesure où la capacité *CAP_SYS_ADMIN* n'est donnée à aucun processus non-*root* (du fait du mode d'attribution des capacités par défaut, de l'absence de prise en compte des bits *setuid root*, et de la restriction à *root* des entrées *veriexec* conférant cette capacité - [CLIP_1201])
- La fonction ne peut que difficilement être reconfigurée par l'utilisateur *root* du socle en dehors des phases de démarrage et d'arrêt du système dans la mesure où la capacité *CAP_SYS_ADMIN* n'est pas attribuée automatiquement à cet utilisateur ([CLIP_1201]).
    - cette propriété n'est pas valable lors du démarrage du système, avant la réduction du *sysctl kernel.clip.rootcap*.
    - Cette propriété n'est que partiellement valable lors de l'arrêt du système, car les nouveaux processus créés par *init* disposent dans ce cas de la capacité *CAP_SYS_ADMIN* héritable ([CLIP_1301]), suffisante à reconfigurer les montages avec */bin/mount* du socle.
- La fonction ne peut que difficilement être reconfigurée par l'utilisateur *root* du socle en dehors de la phase de démarrage du système, car CLIP-LSM interdit l'augmentation de la permissivité d'un montage par remontage ou montage *bind*.
    - Cette propriété n'est pas valable lors du démarrage du système, avant l'activation du *sysctl kernel.clip.mount*
    - Cette propriété n'interdit pas le cas échéant la création depuis le socle au sein d'un compartiment affecté par la fonction d'un nouveau montage accessible en écriture et exécution, à partir d'un montage rompant le principe *W^X* au sein du socle.
- La configuration ou reconfiguration de la fonction est dans tous les cas journalisée par *grsecurity* ([CLIP_1203]).
- La fonction ne peut pas être contournée par le montage d'un support amovible, dans la mesure où le système impose automatiquement l'option *noexec* sur de tels montages (cf. [CLIP_DCS_15088]). Par ailleurs, cette option est imposée par le LSM CLIP pour tout montage non spécifiquement référencé dans la base *devctl* (cf. [CLIP_1201]) ce qui interdit, au terme de la séquence de démarrage, tout montage rompant le principe W^X même en cas de défaillance du *userland* CLIP.

Traitement des erreurs

Les exécutions sont refusées sur les montages accessibles en écriture au moment des appels système *execve()* (code de retour *-EACCESS*), *mmap()* ou *mprotect()* avec le drapeau *PROT_EXEC* (code de retour *-EPERM*) ou *open()* avec le drapeau *O_MAYEXEC*, spécifique à CLIP[2] (code de retour *-EACCESS*). Ces erreurs ne sont pas automatiquement journalisées, sauf traitement spécifique par le processus qui a provoqué l'erreur. Par exemple, les *shells* CLIP journalisent les erreurs d'ouverture *O_MAYEXEC*, dans le fichier *messages* ou *rm_X_messages* (selon le compartiment d'origine : socle ou cage CLIP, ou cage RM, respectivement).

[2] Il est rappelé que ce drapeau est utilisé en particulier par les *shells* - *bash* ou *busybox* - déployés au sein de CLIP avant de "sourcer" un script.

Symétriquement, les appels système *open()* qui conduiraient à un accès en écriture à un fichier d'un montage accessible en exécution sont rejetés du fait du montage en lecture-seule, avec un code de retour -*EROFS* et sans journalisation.

2.8 Contrôle des périphériques exposés

Description de la fonction

La fonction limite les périphériques matériels directement accessibles depuis les compartiments autres que le socle au strict minimum nécessaire à chaque compartiment. En particulier, à l'exception de la cage X11 qui est en mesure d'accéder au *framebuffer* vidéo du système, seuls des périphériques « virtuels » en mode caractère (*null*, *full*, *urandom*, pseudo-terminaux), ainsi éventuellement que – de manière temporaire – les périphériques *device-mapper* associés aux partitions chiffrées à monter dans un compartiment, sont exposés aux différentes cages du système (cf. [CLIP_1202]). Cette restriction est implémentée en exposant dans chaque compartiment un */dev* spécifique, monté par *bind* depuis un répertoire du socle dédié à ce compartiment. Les paquetages d'installation du socle sont produits de manière à ne créer dans ces répertoires que les périphériques strictement nécessaires.

Objectif de la fonction

Cette fonction vise à limiter autant que possible l'exposition d'interfaces noyau aux compartiments autres que le socle (complétant ainsi 2.1), et à éviter d'exposer des périphériques qui permettraient d'échapper aux fonctions de cloisonnement (3.1 et 3.2).

Protection et non-contournement de la fonction

- Le montage des répertoires monté sur */dev* dans les différents compartiments, et son peuplement par les fichiers appropriés, est assuré par le socle, par des applicatifs dont l'intégrité est assurée par 2.4 et 2.6.
- Il est impossible depuis les compartiments concernés d'utiliser d'autres périphériques que ceux montés dans */dev*, dans la mesure où tous les autres montages composant l'arborescence de fichiers de ces compartiments portent l'option *nodev*.
- Il est impossible d'ajouter de nouveaux périphériques dans les répertoires montés sur */dev* depuis les compartiments concernés, pour deux raisons redondantes :
  - le montage sur */dev* est réalisé en lecture seule
  - la capacité *CAP_MKNOD* nécessaire à la création de *devices* est systématiquement masquée en dehors du socle.
- Il est difficile de modifier les *devices* exposés dans ces compartiments depuis les socles, du fait de 2.3, qui limite à quelques exécutables invoqués par *root* l'accès aux capacités *CAP_MKNOD* (nécessaire pour ajouter des *devices* dans les répertoires existants) ou *CAP_SYS_ADMIN* et *CAP_CONTEXT* (nécessaires pour monter d'autres répertoires contenant des *devices* existants dans les compartiments concernés).
- Il est impossible d'importer des périphériques supplémentaires à l'aide d'un périphérique amovible, car CLIP-LSM n'autorise le montage de tels périphériques (et, de manière générale,

de tout périphérique non identifié a priori) qu'avec l'option *nodev* (cf. [CLIP_1201]).

Traitement des erreurs

La fonction ne réalise pas à proprement parler de contrôle d'accès, mais uniquement un paramétrage des périphériques exposés, donc elle ne génère pas directement d'erreurs ou de journaux.

2.9 Journalisation

Description de la fonction

Cette fonction assure la collecte centralisée, au sein de la cage $AUDIT_{clip}$, des journaux produits par les différents composants du système (noyau et processus). La collecte est effectué par un démon *syslog-ng* enfermé dans $AUDIT_{clip}$, sur un ensemble de *sockets* UNIX ouvertes dans les autres compartiments du système (avant l'enfermement du démon *syslog-ng* dans sa cage), ainsi que sur l'interface de lecture des journaux noyau. Une fois reçus par le démon de collecte, les messages sont copiés, en fonction de leur nature et / ou de leur origine, dans un ou plusieurs fichiers de journaux protégés en intégrité par un drapeau *append-only*.

Lorsque des cages RM sont présentes sur le système, un autre démon *syslog-ng* exécuté dans la vue AUDIT de chaque cage RM assure un premier niveau de collecte en centralisant les messages générés au sein de toutes les vues de la cage, avant de les recopier sur l'unique *socket* de collecte ouverte dans la cage par le démon *syslog-ng* d'$AUDIT_{clip}$.

Objectif de la fonction

Cette fonction permet la détection d'attaques ou de tentatives d'attaques, et leur compréhension après analyse. Elle contribue à ce titre à la protection de toutes les autres fonctions.

Protection et non-contournement de la fonction

Plusieurs niveaux de contournement possibles de la fonction de journalisation doivent être envisagés. Un attaquant peut en effet tenter d'interrompre ou de falsifier la génération de journaux ou leur collecte, ou encore de détruire ou modifier les fichiers où sont ensuite stockés les journaux collectés.

La fonction n'offre que des garanties partielles quant à la génération des journaux

- La génération des journaux noyau est assurée par le noyau, et est donc protégée par 2.1.
- En revanche, la génération des journaux de la couche utilisateur est laissée à la charge de chaque processus. Ainsi, l'interruption ou la corruption d'un processus entraînent celles de la génération de journaux par ce processus. Cette menace ne peut être que mitigée par les différents mécanismes de cloisonnement (3.1 et 3.2) et par la protection générique des processus (2.2).
- Il est aussi possible pour un processus d'introduire de faux messages de journalisation sur l'une des *sockets* de collecte de *syslog-ng*. Seule l'interface de journalisation noyau est protégée de cette menace, dans la mesure où elle est en lecture seule.
    - Cette menace est cependant mitigée par l'identification de la provenance des messages,

réalisée par *syslog-ng*. Pour un message donné, la fonction garantit la provenance du message en le classant (par le choix des fichiers dans lesquels il est stocké) dans l'une des catégories suivantes : noyau (non falsifiable), cages CLIP (toutes confondues), cages RM éventuellement (traitées chacune individuellement).

La fonction est en revanche bien protégée contre les interférences vis-à-vis de la collecte des journaux :

- L'exécutable et les bibliothèques du démon de collecte centralisée des journaux sont stockés dans le coeur du socle CLIP, et donc protégés en intégrité par 2.4.
- Il est impossible de tuer le démon de collecte centralisée des journaux depuis un compartiment logiciel autre que le socle, du fait de 3.1.
- Il est impossible de tuer le démon de collecte centralisée des journaux après le verrouillage du système et en dehors de la séquence d'arrêt, car ce démon dispose du privilège CLIP LSM *CLSM_PRIV_IMMORTAL* (cf. [CLIP_1201])
- Ce même privilège protège aussi le démon contre la terminaison en cas de manque de mémoire. Il est donc impossible de forcer la terminaison du démon par le noyau en consommant de la mémoire : le ou les processus consommateurs seront terminés bien avant *syslog-ng*.
- Les éventuels démons *syslog-ng* intermédiaires de collecte au sein d'une cage RM ne peuvent pas être tués par les processus des vues autres que AUDIT de la cage, du fait de 3.2. Ces démons étant seuls à s'exécuter dans chaque vue AUDIT, 3.2 interdit globalement leur terminaison par un quelconque autre processus de la même cage.
- La suppression des *sockets* de collecte des journaux créées par le démon principal de collecte des journaux au sein des cages CLIP est impossible depuis les cages concernées, ces *sockets* n'étant exposées qu'à travers un montage en lecture seule (qui est lui même protégé par le masquage de *CAP_SYS_ADMIN* dans les cages et le contrôle des opérations de montage par CLIP-LSM – cf. [CLIP_1202] et [CLIP_1201]). Cette opération reste en revanche possible depuis le socle.
- Pour les mêmes raisons, la suppression des *sockets* de collecte des journaux créées dans les vues des cages RM par les démons de collecte intermédiaire n'est pas possible depuis les vues USER, ADMIN et AUDIT. La vue UPDATE est en mesure de supprimer les *sockets* créées dans les trois autres vues, mais pas sa propre *socket*.

Enfin, la protection des fichiers de stockage des journaux est bien assurée :

- Il est rigoureusement impossible de modifier ces fichiers autrement qu'en ajout, ou de les supprimer, une fois le système verrouillé, dans la mesure où les attributs *APPEND_ONLY* sont placés sur tous ces fichiers, et incontournables du fait du masquage global de la capacité *CAP_LINUX_IMMUTABLE*.
- L'ajout direct de faux messages en fin de fichier reste possible pour *root* dans le socle ou $AUDIT_{clip.}$ Il est en revanche impossible depuis les autres compartiments, où ces fichiers ne sont pas exposés.

Traitement des erreurs

La fonction ne génère pas d'erreurs.

Remarque 4 : Saturation de la partition de journaux

Aucun mécanisme ne permet à ce stade de prendre en compte les possibilités de saturation de la partition de journaux dans CLIP. Ainsi, un attaquant pourrait forcer une saturation de cette partition par des actions répétées déclenchant la génération de journaux, et profiter ensuite de l'impossibilité pour le démon syslog-ng *de stocker les journaux collectés sur une partition pleine, pour lancer des actions non journalisées.*

2.10 Chiffrement du swap

Description de la fonction

La fonction réalise le chiffrement du swap du système, par une projection *dm-crypt* chiffrée en AES-256-LRW-BENBI, avec une clé tirée aléatoirement à chaque démarrage. Le swap chiffré est créé par un script de démarrage, dans le *softlevel boot* (cf. [CLIP_1301]).

Objectif de la fonction

La fonction permet d'interdire le stockage en clair sur le disque de données normalement chiffrées, mais manipulées en clair dans la mémoire de certains processus. De ce fait, elle contribue au non contournement de toutes les fonctions de sécurité du système qui reposent sur la manipulation d'éléments secrets, en particulier 3.4, 3.5, 3.7 et l'ensemble des fonctions d'authentification (4).

Protection et non-contournement de la fonction

- Le swap chiffré est configuré par des utilitaires du coeur du socle, intègres grâce à 2.4, exécutés au cours de la séquence de démarrage, dans un environnement intègre du fait de 2.6. Ces propriétés d'intégrité garantissent la bonne génération aléatoire de la clé (selon les principes décrits en [CLIP_1206]), la bonne configuration de la projection *dm-crypt*, et l'absence de rémanence de la clé de chiffrement.
- Le LSM-CLIP interdit, une fois le système verrouillé, la création de nouveaux swaps. Il est de ce fait impossible de créer un swap non chiffré en plus du swap chiffré, dans l'espoir qu'une partie des pages « swappées » y soient écrites ([CLIP_1201]).
- Les *devices* correspondant au *swap* (*device* physique correspondant à la partition chiffrée, et *device dm-crypt* correspondant à son image déchiffrée) ne sont exposés que dans le socle CLIP. Les fonctions 3.1 et 2.8 garantissent l'échec de toute tentative d'accès direct à ces périphériques depuis une cage.
- Le sous-système *devctl* du LSM CLIP ([CLIP_1201]) est configuré, lors du verrouillage du système, de manière à interdire tout accès, y compris depuis le socle CLIP, au *device dm-crypt* associé au swap, qui donnerait sinon accès – tant que le système est allumé – aux informations claires stockées dans le swap.
- Le bridage de *CAP_SYS_ADMIN* dans les cages (3.1) interdit tout démontage du swap chiffré depuis celles-ci. La réduction des privilèges réalisée par 2.3 limite significativement les chemins d'attaques permettant d'obtenir un tel démontage depuis le socle CLIP. Dans tous les cas, le démontage du swap chiffré n'entraîne au pire qu'une menace en disponibilité, dans la

mesure où il reste rigoureusement impossible de créer un nouveau swap une fois le système verrouillé.

- Le chiffrement utilisé pour le swap est conforme au référentiel [CRYPTO].

Traitement des erreurs

Les tentatives de lecture du périphérique clair correspondant au *swap*, sur un système allumé, échouent du fait de *devctl*, l'appel système qui tente de réaliser l'accès rendant dans ce cas un code d'erreur *-EPERM* ou *-EACCESS*. Ces erreurs sont journalisées dans le fichier de journaux *clsm.log*.

Les accès directs au périphérique chiffré ne déclenchent pas d'erreur ni de journalisation, mais ne permettent pas la lecture de données intelligibles, du fait du chiffrement.

2.11 Contrôle des exécutables accédant au réseau

Description de la fonction

La fonction réalise un contrôle des processus qui tentent d'accéder au réseau, et n'autorise cet accès que lorsqu'un processus correspond à un exécutable possédant le privilège CLIP-LSM associé au type d'accès demandé. Le contrôle est réalisé par le noyau (LSM CLIP), à partir des privilèges associés aux exécutables par la base *veriexec*, elle-même configurée par la séquence de démarrage et les fonctions de mise à jour.

Objectif de la fonction

La fonction limite le nombre d'applications exposées aux flux réseau (et donc a priori à une menace accrue), et limite les possibilités d'exploitation pilotée à distance d'une vulnérabilité dans une application. Il est ainsi impossible par exemple pour un *shellcode* exécuté par une application bureautique vulnérable de se connecter au réseau, tout comme il est impossible pour un navigateur réseau d'exécuter un *shell* en lui transmettant ses *sockets INET* ouvertes.

Protection et non-contournement de la fonction

- La vérification des droits d'accès est réalisée par le noyau, et est donc intègre et non contournable, du fait de 2.1.
- Le reparamétrage des autorisations (typiquement, pour autoriser des exécutables supplémentaires à accéder au réseau) n'est permis que depuis le socle ou la cage $\text{UPDATE}_{\text{clip}}$ (pour l'ensemble du système), ou de manière partielle depuis les cages RM (pour les processus de la cage uniquement). Par ailleurs, le reparamétrage nécessite un exécutable disposant du privilège *CLSM_PRIV_VERICTL*, exécutable qui n'est exposé dans les cages RM qu'au sein de la vue UPDATE. Il est donc impossible de reparamétrer la fonction depuis une cage CLIP autre que $\text{UPDATE}_{\text{clip}}$, et depuis une vue autre que UPDATE d'une cage RM, du fait de 3.1 et 3.2.
- Le reparamétrage des autorisations n'est pas permis pour un fichier accessible uniquement en lecture seule du fait des options de montage. Tout reparamétrage est donc impossible pour les utilitaires du coeur CLIP, du fait de 2.4. On notera en revanche que le reparamétrage des autorisations pour les utilitaires du coeur des cages RM reste possible, non pas depuis la vue UPDATE de ces cages (qui voit le coeur en lecture seule, du fait de 2.5), mais depuis la racine de la cage (cf. [CLIP_1401]).

Traitement des erreurs

Les accès réseau refusés prennent la forme d'erreurs sur les appels systèmes de la couche *socket* réalisant ces accès : création de *sockets* non locales (*socket()*), connexion ou mise en écoute sur de telles *sockets* (*connect(), bind(), accept(), listen()*), ou échanges sur des *sockets* déjà connectées (*send(), recv()* et appels équivalents). Les accès refusés entraînent le retour d'un code d'erreur *-EPERM* sur l'appel système concerné, et la journalisation du refus dans le fichier de journaux *clsm.log*.

3 Fonctions de cloisonnement

3.1 Cloisonnement système lourd (cages)

Description de la fonction

Cette fonction assure le cloisonnement des cages. Elle associe un identifiant unique, *xid*, à un ensemble de processus (et, transitivement, à tous leurs descendants), de telle sorte que ces processus ne puissent interagir qu'avec des processus de même *xid* (par signaux, attachement *ptrace* ou interactions *par /proc...)*, et avec des ressources système (*IPC System V*, *sockets* abstraites, ...) créées par des processus de même *xid*. Elle associe de plus à chaque *xid* :

- Un espace de nommage *VFS* spécifique et une vue du système de fichiers confinée à une sous-arborescence propre de l'arborescence de fichiers principale du système.
- Une ou plusieurs adresses IP autorisées, telles que les processus de cet *xid* ne puissent émettre que des paquets réseau ayant pour adresse source une adresse autorisée, et ne puissent recevoir que des paquets réseau ayant pour adresse destination une adresse autorisée, et ce y compris sur la boucle locale réseau.
- Un masque de capacités POSIX autorisées, tel que les processus de cet *xid* ne puissent mettre en oeuvre que les capacités autorisées.

La fonction est assurée par la mise en oeuvre de contextes *vserver*, configurés à l'aide d'utilitaires spécifiques à CLIP. Le détail de cette mise en oeuvre est donné dans [CLIP_1202].

Objectif de la fonction

Cette fonction est la fonction de cloisonnement principale sous CLIP, permettant de construire les différentes cages du système. Elle assure la séparation des rôles entres les différentes cages CLIP, et la séparation des niveaux de sensibilité entre les différentes cages RM. De plus, elle réduit de manière systématique les privilèges (au sens des capacités POSIX, mais aussi de la limitation des ressources système accessibles, indépendamment des capacités) de la plupart des processus du système, et complète de ce fait la fonction 2.3.

Protection et non-contournement de la fonction

- La fonction de cloisonnement proprement dite est implantée dans le noyau, et protégée en intégrité du fait de 2.1.
- La configuration initiale de la fonction est réalisée au démarrage du système par des utilitaires et fichiers de configuration du coeur du socle CLIP. Cette séquence de configuration initiale est donc protégée en intégrité du fait de 2.4 et 2.6.
- La reconfiguration du cloisonnement lourd est impossible depuis une cage du système, car :
  - les capacités nécessaires à une telle reconfiguration ne sont autorisées dans aucune cage
  - la reconfiguration n'est de toutes manières jamais autorisée pour un processus enfermé dans

une cage (*xid* non nul).

- La reconfiguration du cloisonnement lourd n'est autorisée au sein du socle que pour quelques exécutables privilégiés du fait de 2.3. Ces exécutables ne permettent, sauf vulnérabilité dans les exécutables eux-mêmes, qu'une reconfiguration partielle des cages :
  - soit parce qu'ils ne réalisent que des opérations prédéfinies sur les cages, et ne permettent pas la création d'une cage arbitraire (*syslog-ng*, *xdm*)
  - soit parce qu'ils permettent la création de cages arbitraires, mais uniquement à partir de fichiers de configuration stockés dans le coeur du socle CLIP (*vsctl*), donc non modifiables au sein d'un système CLIP après verrouillage, du fait de 2.4. Le remplacement de ces fichiers de configuration reste possible par création d'un nouveau montage masquant les fichiers de configuration légitimes, mais une telle opération nécessite des privilèges élevés, ce qui limite significativement les chemins d'attaque possibles du fait de 2.3.

  Par ailleurs, les processus possédant les privilèges nécessaires à la reconfiguration des cages sont protégés par 2.3 vis-à-vis de tout processus ne possédant pas des privilèges au moins égaux. Les exécutables correspondants sont protégés en intégrité par les restrictions d'accès en écriture aux systèmes de fichiers qui les hébergent (vis-à-vis de tout le système pour *syslog-ng* et *vsctl*, du fait de 2.4 ; uniquement vis-à-vis des cages autres que $\text{UPDATE}_{\text{clip}}$ pour *xdm*, du fait de 2.7), mais aussi par la vérification d'empreinte cryptographique dont ils font l'objet du fait de 2.3.
- Les processus des différentes cages CLIP peuvent échanger entre eux des informations, du fait de canaux de communication créés par l'intermédiaire de systèmes de fichiers partagés. Le cloisonnement réalisé entre ces cages se limite donc à l'intégrité (des systèmes de fichiers propres à une cage, et des environnements d'exécution des processus de cette cage, vis-à-vis des autres cages). En revanche, des mesures spécifiques (systèmes de fichiers de base distincts, montages partagés uniquement avec des options très restrictives) interdisent toute communication entre les processus des différentes cages RM, assurant ainsi un cloisonnement à la fois en intégrité et en confidentialité (cf. [CLIP_1202]).
- Le contournement du cloisonnement lourd par des flux réseau sur la boucle locale est interdit par le filtrage réseau 3.6, qui n'autorise que les flux légitimes, et interdit en particulier les flux inter-niveaux.
- Le contournement du cloisonnement lourd au niveau des flux réseau vers ou depuis des systèmes distants (par exemple par écoute passive, ou contrefaçon de paquets) est interdit par les politiques de chiffrement IPsec des flux extérieurs (3.7).
- Le contournement du cloisonnement lourd au niveau de l'affichage est interdit par la labellisation des clients graphiques en fonction de leur niveau (3.3).
- Le contournement du cloisonnement lourd par l'intermédiaire de périphériques matériels (périphériques partagés créant des canaux de communication, ou périphériques accordant un niveau d'accès au matériel qui permettrait de contourner les restrictions de privilèges des cages) est interdit par le contrôle strict des périphériques exposés dans les cages (2.8).

Traitement des erreurs

Le cloisonnement est réalisé au niveau du noyau vis-à-vis de la couche utilisateur. Les accès interdits

sont donc refusés au niveau des appels système correspondants. Le contrôle d'accès réalisé par la fonction prend pour l'essentiel la forme d'un contrôle de la visibilité des objets du système. Ainsi, une tentative d'accéder à un objet non autorisé (par exemple un montage d'un autre *namespace VFS*, un processus hors cage, ou un objet IPC hors cage) se traduira par un message d'erreur équivalent à ce qu'il serait si l'objet concerné n'existait pas : retour d'un code d'erreur *-ENOENT* (accès à un objet local), ou le cas échéant *-EADDRNOTAVAIL* (tentative d'utiliser une adresse IP non attribuée à la cage courante). Les accès interdits par la réduction des privilèges au sein de la cage sont en revanche traités comme des erreurs classiques liées à des privilèges insuffisants, les appels système correspondants renvoyant un code d'erreur *-EPERM* ou *-EACCESS*.

Ces accès rejetés ne sont en général pas journalisés, à l'exception de quelques cas spécifiques (par exemple les tentatives d'accès à des fichiers masqués du */proc*), qui sont détaillés dans le document de référence [CLIP_1202], et qui sont journalisés dans le fichier *vserver.log*.

3.2 Cloisonnement système léger (vues)

Description de la fonction

Cette fonction assure le cloisonnement des vues, sous-compartiments logiques des cages CLIP. Elle permet de confiner un ensemble de processus (et leurs descendants par transitivité) à une sous-arborescence du système de fichiers (ou « vue » du système de fichiers), et d'interdire toute interaction directe (envoi de signaux, attachement *ptrace* ou accès aux fichiers */proc/<pid>*), à l'initiative des processus de la vue, entre les processus de la vue et les autres processus du système.

La fonction est implémentée à l'aide de prisons *chroot*, complétées par les fonctionnalités de protection des processus hors *chroot* de l'option *GRKERNSEC_CHROOT_FINDTASK* ([CLIP_1203]), et durcies par d'autres options *grsecurity* et CLIP-LSM. Contrairement au mode de fonctionnement UNIX standard, le fait pour un processus d'être enfermé dans une prison *chroot* est tracé dans l'état du processus (à travers les étiquettes de sécurité CLIP-LSM, cf. [CLIP_1201]).

Objectif de la fonction

Cette fonction permet un sous-cloisonnement de certaines des cages du système, dans la mesure où la fonction de cloisonnement lourd (3.1) ne permet pas de définir plusieurs niveaux de cloisonnement imbriqués. Elle permet notamment la séparation des rôles au sein des cages RM, et est aussi mise à profit pour rendre plus robuste le cloisonnement graphique (3.3). On notera que cette fonction, à la différence du cloisonnement lourd, ne permet pas d'assurer un réel cloisonnement en confidentialité, dans la mesure où elle ne contrôle pas certains canaux de communication coopératifs entre deux vues (par exemple, queues de messages *System V*, ou *"morse"* par l'intermédiaire d'un sémaphore *System V*).

Protection et non-contournement de la fonction

- La fonction proprement dite est assurée par le noyau, et est donc protégée en intégrité par 2.1.
- La configuration de la fonction est limitée à des exécutables spécifiques (auxquels est attribuée la capacité *CAP_SYS_CHROOT* : *chroot-launch* dans CLIP, et *jailmaster* dans les éventuelles cages RM) par 2.3. Ces exécutables ne créent des vues que selon des modalités (racine, commande initialement lancée dans la vue) prédéfinies et non modifiables, sauf à modifier l'exécutable lui-même, cette dernière approche étant contrée à la fois par la protection en écriture des systèmes de fichiers sur lesquels ces exécutables sont installés (2.7) et par la vérification d'empreinte cryptographique à laquelle ils sont soumis du fait de 2.3. Ces exécutables ne sont en particulier pas exposés au sein des vues ainsi configurées – il n'est de ce fait pas possible de créer une vue depuis l'intérieur d'une autre vue.
- Les techniques permettant normalement d'échapper à une prison *chroot* sont contrées par différentes mesures issues de *Grsecurity* ([CLIP_1203]) ou CLIP-LSM ([CLIP_1201])
    - Les techniques reposant sur la prise de contrôle d'un processus hors *chroot*, ou des ressources qu'il expose à travers */proc/<pid>*, sont contrées par *GRKERNSEC_FINDTASK*. De plus, les processus d'une vue sont bridés par *grsecurity* dans les interactions indirectes qu'ils pourraient avoir avec des processus extérieurs à leur vue via des *sockets* abstraites ou des segments de mémoire partagée.

- Les techniques reposant sur la création d'une deuxième prison *chroot*, dans laquelle un descripteur de répertoire situé en-dessous de la racine de la prison serait récupéré et utilisé pour sortir des deux prisons sont bloquées par l'impossibilité de faire un appel *chroot* dans une vue (2.3). De plus :
  - CLIP-LSM interdit à tout processus de faire un appel *chroot* en conservant des descripteurs de répertoire ouverts
  - CLIP-LSM interdit la transmission par *socket UNIX* d'un descripteur de répertoire hors prison à un processus enfermé dans une prison *chroot*.
  - Le bridage des interactions avec des processus extérieurs à la prison *chroot*, exposé plus haut, s'applique aussi dans ce cas.
- Les techniques reposant sur l'utilisation de *devices* existants, ou la création de nouveaux *devices*, pour accéder aux fichiers hors vue, sont bloquées par 2.8.

Traitement des erreurs

Le cloisonnement est réalisé au niveau du noyau vis-à-vis de la couche utilisateur. Les accès interdits sont donc refusés au niveau des appels système correspondants. Le contrôle d'accès réalisé par la fonction prend pour l'essentiel la forme d'un contrôle de la visibilité des objets du système. Ainsi, une tentative d'accéder à un objet non autorisé (typiquement, le répertoire *parent "..."* à la racine de la vue, ou les répertoires du */proc* correspondant à des processus hors-vue) se traduira par un message d'erreur équivalent à ce qu'il serait si l'objet concerné n'existait pas : retour d'un code d'erreur *-ENOENT*. En revanche, l'envoi de signaux à un processus hors-vue (par *kill()* ou *fcntl()*) sera refusé avec une erreur *-EPERM*, de même que les différents autres types d'opérations spécifiquement interdits par *grsecurity* dans une prison *chroot*, tels que détaillés dans le document de référence [CLIP_1203]. De même, le transfert par une *socket* de descripteurs de fichiers ouverts hors de la vue sera refusé avec une erreur *-EPERM* renvoyée à la réception du descripteur.

Seul ce dernier type d'erreur (transferts interdits de descripteurs de fichiers) est journalisé, dans le fichier *clsm.log*.

3.3 Cloisonnement graphique

Description de la fonction

Cette fonction permet de définir plusieurs niveaux de clients non privilégiés du serveur X11, cloisonnés (au sens des interactions X11: copier-coller, capture d'écran, envoi d'événements) entre eux et vis-à-vis des clients privilégiés.

Elle est utilisée pour labelliser les clients graphiques réalisant l'affichage des bureaux des différentes cages RM (visionneuses VNC), à des niveaux distincts. Son implémentation repose sur une modification spécifique à CLIP de l'extension *Xsecurity* (cf. [CLIP_1303]), pour permettre d'associer un label à un *cookie* d'authentification *Xauthority*, couplée à l'enfermement des clients non privilégiés dans des vues (cf. 3.2) dédiées.

Objectif de la fonction

La fonction permet de prolonger au niveau X11 le cloisonnement réalisé au niveau noyau entre cages par la fonction 3.1, et est de ce fait essentielle pour interdire le contournement de cette dernière.

Protection et non-contournement de la fonction

- La fonction de contrôle d'accès proprement dite est implémentée au sein du serveur X11, qui s'exécute dans une cage dédiée, et est donc protégé en intégrité (de l'environnement d'exécution) par 3.1, vis-à-vis des clients graphiques lancés dans $USER_{clip}$. De plus, 2.7 assure la protection en intégrité des exécutables associés au serveur X11 vis-à-vis de ces mêmes clients.
- Les clients labellisés non privilégiés sont enfermés dans une vue dédiée par niveau, qui contient uniquement le *cookie Xauthority* associé à leur niveau. La fonction de cloisonnement 3.2 leur interdit ainsi l'accès aux *cookies* des autres niveaux, qui leur permettraient de modifier leur propre label. De plus, la minimalisation des arborescences de fichiers exposées dans ces vues (essentiellement limitées au client, à un *shell*, et aux bibliothèques associées) limite les possibilités d'attaque sur le reste du système, et en particulier le serveur X11, qu'aurait un client corrompu. On notera en revanche que la fonction de cloisonnement des vues n'est pas suffisante pour bloquer la transmission d'un *cookie Xauthority* entre deux clients non privilégiés de niveaux distincts qui coopéreraient entre eux.
- Les clients labellisés non privilégiés sont bridés dans leur accès aux ressources et fonctions du serveur X11, ce qui leur interdit notamment de générer un *cookie Xauthority* pour un niveau différent du leur, et donc de modifier leur propre label. De plus, ce bridage réduit sensiblement la capacité de ces clients à exploiter d'éventuelles vulnérabilités dans le serveur X11 afin de contourner le contrôle d'accès.
- Les clients labellisés sont exécutés dans des vues de $USER_{clip}$, et non dans les cages RM directement, ce qui limite du fait de 3.1 leur exposition aux processus éventuellement corrompus de ces cages, et évite l'exposition directe du serveur X11 à ces mêmes processus.

Traitement des erreurs

Les accès X11 interdits (accès d'un client non-privilégié à une ressource privilégiée, ou à un client d'un

niveau de sécurité différent) sont rejetés par le serveur X11, qui retourne au client fautif une erreur X11 *BadRequest* ou *BadAccess*, selon le type de requête. Ces erreurs ne sont pas journalisées.

3.4 Cloisonnement cryptographique des données locales utilisateur

Description de la fonction

La fonction réalise le chiffrement des données de chaque utilisateur, de telle sorte que ces données ne soient accessibles en clair que lorsque l'utilisateur possède une session ouverte sur le système. De manière complémentaire, elle assure la non rémanence de toutes les données en clair de chaque utilisateur lorsque sa session se termine.

La fonction est implémentée à l'aide d'un module *PAM* spécifique à CLIP, *pam_exec_pwd*, qui assure le déchiffrement et le montage de partitions chiffrées propres à l'utilisateur lorsque celui-ci ouvre sa session, en réutilisant à cette fin le mot de passe saisi à l'ouverture de session. Ces montages sont complétés par des montages de type *tmpfs*, créés à chaque ouverture de session par le même module *PAM*, et supprimés en fin de session. Ces montages couvrent l'ensemble des répertoires accessibles en écriture par l'utilisateur, en dehors de ses partitions chiffrées, et garantisse l'effacement en fin de session de toutes les données qui ne sont pas sauvegardées sous forme chiffrée. Les partitions utilisateur sont chiffrées en AES-256-LRW-BENBI, avec une clé par partition, elle-même chiffrée en AES-256-CBC par un haché *bcrypt* du mot de passe utilisateur, réalisé avec 2^12 itérations *eksblowfish* et un sel de 128 bits spécifique à chaque clé (cf. [CLIP_1302]).

Objectif de la fonction

La fonction assure la confidentialité des données utilisateur en cas de perte ou de vol du poste, ainsi que la confidentialité au sein du système des données d'un utilisateur vis-à-vis de tous les autres utilisateurs.

Protection et non-contournement de la fonction

- La fonction est largement implémentée par des utilitaires du coeur du socle, dont l'intégrité est garantie par 2.4. On notera cependant que le module *PAM pam_exec_pwd* qui lance les scripts de déchiffrement de partitions est lui-même invoqué au sein du démon *XDM*, qui ne bénéficie pas de cette protection systématique en intégrité. Ce démon n'est néanmoins modifiable que depuis le socle CLIP ou la cage $\text{UPDATE}_{\text{clip}}$, du fait de 2.7, tandis que 3.1 garantit son intégrité vis-à-vis de toutes les cages du système (*XDM* étant exécuté dans le socle CLIP).
- Les mécanismes cryptographiques utilisés pour le chiffrement de disque sont conformes au référentiel [CRYPTO].
- Les clés cryptographiques utilisées pour ce chiffrement sont stockées dans une partition visible uniquement du socle, et ne sont manipulées que par des processus du socle. Elles sont donc protégées en confidentialité vis-à-vis des cages du système par 3.1.
- Ces clés cryptographiques sont stockées uniquement chiffrées par un haché *bcrypt* du mot de passe utilisateur, avec un chiffrement conforme au référentiel [CRYPTO]. Leur déchiffrement par un attaquant du socle présente une complexité au moins équivalente à la recherche du mot de passe de l'utilisateur.

- Les clés cryptographiques déchiffrées ne sont manipulées au sein du socle que par des processus privilégiés au sens du LSM-CLIP, dont l'environnement d'exécution est protégé en intégrité et confidentialité vis-à-vis de tout processus moins privilégié du socle, du fait de 2.3, et vis-à-vis de tout processus hors socle du fait de 3.1.
- La récupération des clés ou données claires dans le swap est interdite par 2.10.
- Le module *PAM pam_exec_pwd* refuse l'ouverture de session lorsque l'un des montages utilisateur échoue à l'ouverture de session, ce qui interdit l'ouverture d'une session dans laquelle l'utilisateur aurait accès à des montages lui permettant un stockage non volatile de données claires. De manière complémentaire à cette mesure, les permissions et droits de montage sont ajustées sur les répertoires sur lesquels doivent être montées les partitions (chiffrées et temporaires) des utilisateurs, de manière à interdire à tout utilisateur non *root* l'accès en écriture à ces répertoires lorsque les montages ne sont pas réalisés.
- Les montages réalisés à l'ouverture de session ne peuvent pas être démontés depuis une cage du système, du fait du bridage systématique des privilèges nécessaires en dehors du socle CLIP (3.1). L'opération de démontage n'est possible dans le socle CLIP que pour les démons les plus privilégiés du système du fait de 2.3.
- Les montages réalisés à l'ouverture de session sont tracés dans un fichiers *mtab* propre à chaque cage, et géré entièrement par le socle. Un script lancé par *pam_exec_pwd* interdit tout ouverture de session lorsque l'un de ces montages est encore actif, ce qui interdit l'accès aux données d'un utilisateur précédent dont les partitions n'auraient pas été correctement démontées. De manière complémentaire (et redondante) à cette mesure, les points de montage utilisés sont les mêmes quelque soit l'utilisateur, de telle sorte que les montages d'un nouvel utilisateur « couvrent » les montages précédents s'il n'ont pas été démontés et que l'ouverture d'une nouvelle session a malgré cela été permise par une erreur du système.

Traitement des erreurs

Cette fonction de sécurité peut générer deux types d'erreur :

- Une erreur lors d'une tentative d'accès non autorisée aux données privées (chiffrées) d'un utilisateur CLIP. Lorsque cette tentative passe par les interfaces légitimes du système (authentification XDM), l'ouverture de session est refusée est un message est journalisé dans le fichiers de journaux *auth.log* ou *daemon.log*, signalant l'échec de la vérification du mot de passe fourni ou du déchiffrement d'une des partitions chiffrées. Naturellement, lorsque la tentative d'accès ne passe pas par ces interfaces légitimes du système (par exemple, tentative de lecture directe suite à l'extraction du disque), aucune erreur ni journalisation ne peuvent être produites, mais les données, toutes chiffrées, sont simplement inintelligibles.
- Une erreur lors du démontage d'une partition (chiffrée ou temporaire) d'un utilisateur, qui laisserait cette partition exposée dans la session suivante, appartenant potentiellement à un autre utilisateur. Une telle erreur est journalisée dans le fichier *daemon.log*, et bloque toutes les possibilités d'ouverture de session par XDM jusqu'au démontage de la partition fautive ou au redémarrage du système.

3.5 Cloisonnement cryptographique des supports amovibles

Description de la fonction

La fonction réalise le chiffrement de l'ensemble des données d'un support amovible, de manière spécifique à un utilisateur et un niveau (CLIP, ou niveaux RM).

Le chiffrement est réalisé en AES-256-CBC:ESSIV, avec une clé générée aléatoirement lors de l'initialisation du périphérique, et stockée sur les premier et dernier secteurs de ce périphérique sous forme chiffrée pour une clé privée RSA 1536 bits propre à l'utilisateur et au niveau. L'initialisation d'un support amovible vierge est réalisée par un script du socle sur commande de l'utilisateur. Le déchiffrement d'un support initialisé est réalisé lors de la connexion de ce support, après avoir vérifié l'authenticité du secteur de méta-données contenant la clé symétrique chiffrée (cf. 4.6), et demandé à l'utilisateur courant de $USER_{clip}$ le mot de passe de la clé RSA de déchiffrement (cf. [CLIP_DCS_15088]).

Objectif de la fonction

La fonction prolonge sur les supports amovibles à la fois le cloisonnement inter-niveau de 3.1 et le cloisonnement inter-utilisateurs de 3.4.

Protection et non-contournement de la fonction

- Le chiffrement / déchiffrement des supports amovibles est réalisé par des utilitaires du coeur du socle CLIP, intègres du fait de 2.4.
- Le chiffrement / déchiffrement des supports amovibles est réalisé au sein du socle CLIP, et s'effectue donc dans un environnement d'exécution protégé en intégrité et confidentialité vis-à-vis de toutes les cages du système, du fait de 3.1, ce qui interdit depuis ces cages l'espionnage des clés stockées dans la mémoire des processus effectuant le chiffrement / déchiffrement.
- Le chiffrement / déchiffrement des supports amovibles est réalisé par un utilitaire privilégié au sens du LSM CLIP, ce qui protège aussi en intégrité et confidentialité son environnement d'exécution vis-à-vis des processus moins privilégiés du socle CLIP (cf. 2.3).
- La récupération des clés ou données claires dans le swap est interdite par 2.10.
- La clé privée utilisée pour le déchiffrement est stockée sur la partition chiffrée de niveau CLIP de l'utilisateur, ce qui lui assure une protection partielle en confidentialité :
  - la clé cryptographique est protégée à tout moment vis-à-vis des cages autres que $USER_{clip}$, du fait de la non exposition de la partition
  - la clé cryptographique est protégée vis-à-vis du socle CLIP et de la cage $USER_{clip}$ lorsque l'utilisateur n'est pas connecté, du fait du chiffrement de la partition.
  - L'export des clés cryptographiques permet éventuellement de rompre la confidentialité inter-utilisateurs, mais maintient le cloisonnement inter-niveaux, dans la mesure où :
    - L'export réseau est soumis au cloisonnement inter-niveau assuré par 3.7 et 3.6.
    - L'export sur support amovible est accompagné d'un chiffrement spécifique à chaque

niveau (mais pas chaque utilisateur).

Traitement des erreurs

Toute tentative de déchiffrement d'un support amovible sans la clé correspondante aboutit à l'échec soit du déchiffrement, soit du montage du support clair (dans le cas peu probable ou un déchiffrement incorrect de la clé symétrique du support maintiendrait le format de *padding openssl* de celle-ci, ce qui ne permet pas de détecter l'erreur lors du déchiffrement). Ces échecs sont signalés par une fenêtre graphique *"pop-up"* dans la session $USER_{clip}$ de l'utilisateur, et journalisés dans le fichier *messages*.

Par ailleurs, l'échec du démontage d'un support amovible en fin de session, qui laisserait le support clair exposé dans la session suivante, est traité comme l'échec du démontage d'une partition chiffrée locale de l'utilisateur concerné, par journalisation dans *daemon.log* et interdiction de toute nouvelle ouverture de session jusqu'au démontage du support ou au redémarrage du poste.

3.6 Filtrage des flux

Description de la fonction

Cette fonction réalise un filtrage des flux réseaux sur la boucle locale et la ou les interfaces réseaux externes du systèmes, en fonction d'un certain nombre de règles, dont certaines sont configurables par l'administrateur.

Elle est réalisée par configuration du filtre de paquets natif du noyau Linux, *netfilter*, par un script de démarrage qui applique en premier lieu des règles prédéfinies et non modifiables, avant de générer un ensemble de règles à portée limitée en fonction d'un fichier de configuration modifiable par l'administrateur local (depuis la cage $ADMIN_{clip}$). Le détail de cette configuration est donné dans [CLIP_1501].

Objectif de la fonction

Cette fonction assure en premier lieu un filtrage des flux réseaux sur la boucle locale, nécessaire pour interdire le contournement par cette voie du cloisonnement entre cages (3.1). Elle effectue aussi des vérifications de cohérence des flux émis avec la politique de chiffrement IPsec, et contribue de ce fait à la robustesse de la fonction 3.7. Enfin, elle permet de contrôler, de manière partiellement configurable par l'administrateur local, les flux applicatifs autorisés en émission et réception, et ainsi de participer à la défense en profondeur du système en limitant l'exposition à des attaques réseau et en bridant, de manière complémentaire à 2.11, les possibilités de pilotage à distance d'une application locale compromise.

Protection et non-contournement de la fonction

- Le filtrage des paquets, une fois configuré, est réalisé entièrement par le noyau, et est donc protégé en intégrité par 2.1.
- La configuration du filtre de paquets est réalisée lors du démarrage du poste uniquement, à l'aide d'exécutables et de fichiers de configurations entièrement compris dans le coeur du socle CLIP à l'exception d'un fichier de configuration modifiable par l'administrateur local du poste, fichier qui est lu à l'aide de fonctions d'import sécurisé. L'intégrité de cette séquence de configuration est donc assurée par 2.4 et 2.6.
- Une fois le système verrouillé, aucun exécutable ne se voit plus accorder par 2.3 les privilèges nécessaires à la configuration de *netfilter* (en particulier, *CAP_NET_RAW*), ce qui interdit toute reconfiguration de la fonction après sa configuration initiale.
- Le cloisonnement lourd 3.1, couplé à l'authentification 4.2, limitent aux seuls administrateurs locaux la possibilité de modifier le fichier de configuration des règles de filtrage importé au démarrage.
- L'import sécurisé de ce même fichier, et le traitement qui en est fait, limitent les possibilités de l'administrateur à la création de règles d'autorisation de certains flux TCP et UDP, cage par cage et sur les interfaces externes uniquement (cf. [CLIP_1501]). Ce traitement ne permet en particulier pas à l'administrateur de modifier le filtrage effectué sur la boucle locale, ni les règles vérifiant la cohérence des flux par rapport à la politique de chiffrement IPsec. Ces deux

types de règles sont de toutes manières générées avant les règles configurables par l'administrateur, ce qui garantit que même si l'administrateur parvenait à faire charger des règles arbitraires, celles-ci ne seraient pas évaluées avant les règles prédéfinies.

- La configuration du filtre de paquet est systématiquement réalisée avant l'activation des interfaces réseau (boucle locale comprise) dans la séquence de démarrage (cf. [CLIP_1301]), ce qui empêche toute *race condition* dans l'application des politiques.

Traitement des erreurs

Les paquets réseau non autorisés sont supprimés par le noyau, sans signalisation au niveau réseau (aucun paquet d'erreur, par exemple ICMP, n'est renvoyé à l'émetteur en cas de réception d'un paquet non autorisé). Ce rejet est journalisé, dans le fichier de journaux dédié *fw.log*.

3.7 Chiffrement IPsec des flux

Description de la fonction

Cette fonction réalise un chiffrement IPsec des flux réseaux émis depuis certaines cages sur les interfaces externes du système. La politique de filtrage est prédéfinie pour un type de système CLIP, et non modifiable à l'exception de la prise en compte des adresses IP propres au poste et à ses cages.

Le chiffrement est réalisé avec le protocole ESP en mode tunnel entre le poste et une ou plusieurs passerelles. La politique prédéfinie du système impose normalement le chiffrement des flux de la cage $UPDATE_{clip}$[3], auquel peut éventuellement s'ajouter le chiffrement des flux des cages RM (RM_H sur une configuration CLIP-RM standard). Cette politique peut aussi, selon les cas, autoriser des flux clairs (obligatoirement pour le socle CLIP, et éventuellement pour $USER_{clip}$ et / ou une cage RM).

Les algorithmes cryptographiques utilisés pour ce chiffrement sont les algorithmes symétriques de la bibliothèque CCSD incluse dans le noyau (cf. [CLIP_1205]), assurant à la fois la confidentialité (mode CTR) et l'authenticité (mode SHMAC) des paquets. Les clés symétriques IPsec sont négociées à l'aide du protocole IKEv2 par un démon exécuté dans le socle CLIP, qui met en oeuvre les fonctions de négociation authentifiée décrites en 4.3. Les politiques de sécurité sont initialement créées de manière statique au démarrage du système, mais peuvent être modifiées par un autre démon du socle CLIP.

Le détail de cette configuration est donné dans [CLIP_1501].

Objectif de la fonction

Cette fonction prolonge sur le réseau le cloisonnement local entre cages, en particulier en confidentialité, et est donc essentielle au non contournement de 3.1. Elle assure aussi l'authenticité des flux de téléchargement de mises à jour, complétant ainsi l'authentification individuelle des paquetages de mise à jour (4.5). Enfin, elle permet éventuellement d'interdire tout flux réseau depuis certaines cages, de manière complémentaire au filtrage réseau (3.6).

Protection et non-contournement de la fonction

- L'application des politiques IPsec, une fois configurées, est réalisé par le noyau, et est donc protégée en intégrité par 2.1.
- Le contournement de la fonction par des moyens cryptographiques est supposé impossible, par hypothèse sur la bibliothèque CCSD.
- La configuration des politiques IPsec, et la négociation des associations de sécurité, sont réalisées par deux démons du coeur du socle (*spmd* et *iked*), lancés au démarrage. L'intégrité au lancement de ces services est assurée par 2.4 et 2.6 (cette dernière ne s'appliquant que partiellement à *iked*, qui n'est lancé qu'après le verrouillage du système). De manière complémentaire au lancement de *spmd*, une configuration initiale des politiques de sécurité est réalisée statiquement au démarrage par appel à l'utilitaire *setkey*, avec les mêmes garanties d'intégrité. De plus, les cages non explicitement autorisées à sortir en clair ne peuvent pas accéder au réseau sans politique de sécurité IPsec, du fait de l'absence du drapeau *NXF_NO_SP*

[3] Sauf sur une passerelle de type UPDATE, sur laquelle la cage $UPDATE_{clip}$ accède directement au serveur de mise à jour sur le réseau interne protégé par la passerelle.

de leur configuration *vserver* (cf. [CLIP_1202]).

- Le cloisonnement lourd 3.1, couplé à l'authentification 4.2, limitent aux seuls administrateurs locaux la possibilité de modifier le fichier de configuration des adresses IP du poste importé au démarrage des démons. De plus, les informations importées depuis ce fichier ne remettent jamais en cause les politiques et associations IPsec générées par les démons (en particulier, la décision de chiffrer ou non les flux issus d'une cage donnée, et les algorithmes utilisés).
- Une fois le système verrouillé, seuls les démons *spmd* et *iked* disposent des privilèges nécessaires à la configuration IPsec, du fait de 2.3. L'environnement d'exécution de ces démons est protégé en intégrité, vis-à-vis des processus des cages par 3.1, et vis-à-vis des autres processus (moins privilégiés) du socle par 2.3. Le démon *spmd* est de plus adapté de manière à ne supprimer aucune politique de sécurité en cas de terminaison impromptue.
- La configuration des politiques de sécurité IPsec est systématiquement réalisée avant l'activation des interfaces réseau dans la séquence de démarrage (cf. [CLIP_1301]), ce qui empêche toute *race condition* dans l'application de ces politiques.
- Le chiffrement du swap (2.10) interdit la récupération dans ce derniers des clés IPsec négociées par *iked*.
- L'authentification des passerelles (4.3) garantit la négociation correcte de ces clés.

Traitement des erreurs

L'échec d'un chiffrement ou déchiffrement (y compris la vérification d'authenticité ESP) entraîne la suppression par le noyau du paquet réseau fautif, avant émission (dans le sens émission) ou transmission à la couche utilisateur (dans le sens réception). Dans le cas d'un paquet émis localement, un code d'erreur (correspondant au code d'erreur de la fonction de chiffrement : *-EINVAL*, *-EFAULT*, *-ENOMEM* par exemple, selon les cas d'erreur) est retourné en sortie de l'appel système ayant demandé l'envoi, par exemple *sendmsg()*.

De même, le non-respect de la politique de sécurité IPsec par un paquet entrant (paquet arrivé en clair alors que la politique de sécurité l'impose chiffré, ou le contraire) entraîne le rejet du paquet par le noyau. Le cas symétrique n'est pas possible en sortie, dans la mesure où la politique de sécurité est systématiquement appliquée en émission. En revanche, il est possible que dans un tel cas un chiffrement IPsec demandé par la politique de sécurité ne puisse pas être effectué, faute de pouvoir établir l'association de sécurité correspondante. Dans un tel cas, l'émission est refusée, et l'appel système correspondant retourne un code d'erreur *-EAGAIN*.

Aucun de ces rejets de paquets n'est journalisé.

3.8 Cloisonnement des rôles

Description de la fonction

Cette fonction réalise la stricte séparation des privilèges d'utilisation et de configuration du système selon les différents rôles utilisateurs. Elle repose sur deux éléments complémentaires de la construction d'un système CLIP:

- La répartition des différents rôles entre plusieurs cages (et vues, dans le cas d'un poste CLIP-RM) distinctes, réalisée automatiquement lors de l'ouverture de session dans un rôle donné. Ce point est détaillé dans les documents de référence [CLIP_1302], [CLIP_1304] et [CLIP_1401].
- La configuration des montages VFS constituant ces différentes cages, de manière à n'accorder que les droits en lecture ou écriture aux différentes sous-arborescences du système qui correspondent aux prérogatives des rôles associés. Ainsi, les droits en écriture sur les répertoires contenant des fichiers de configuration modifiables par un rôle administrateur (CLIP ou, le cas échéant RM), ne sont montés en écriture que dans la cage ou vue associée à ce rôle (cage $ADMIN_{clip}$ ou vues ADMIN des cages RM). De même, le répertoire contenant les journaux du système n'est monté que dans la cage $AUDIT_{clip}$, associée au rôle d'auditeur. Enfin, les répertoires contenant les applicatifs dévolus au rôle utilisateur ne sont exposés que dans la cage $USER_{clip}$ (respectivement, les vues USER des cages RM, dans le cas d'un système CLIP-RM). Ces configurations de montages, et les droits qui en découlent, sont décrites en détail dans les documents de référence [CLIP_1304] et [CLIP_1401].

Objectif de la fonction

La séparation des rôles constitue un objectif de sécurité fondamental du système. Elle contribue par ailleurs à la défense en profondeur des autres fonctions du système, en limitant notamment l'exposition de leurs interfaces de configuration et de journalisation aux différents rôles utilisateurs. En particulier, elle assure un degré de protection supplémentaire à la fonction 2.6, en limitant au profil administrateur CLIP l'accès aux fichiers de configuration depuis lesquels des paramètres de démarrage sont importés de manière sécurisée.

Protection et non-contournement de la fonction

- La configuration des cages et vues, conditionnant les droits respectifs de celles-ci, est réalisée au démarrage du poste, à partir de scripts et de fichiers de configuration stockés dans le coeur du socle CLIP. Elle est donc protégée en intégrité par 2.4 et 2.6.
- Une fois enfermé dans une cage ou vue, l'accès aux droits accordés par une autre cage ou vue est interdit par les fonctions 3.1 et 3.2.
- L'usurpation d'un rôle est interdite par les fonctions d'authentification : la fonction 4.1 interdit l'ouverture de session sous un compte (et donc dans un rôle) autre que celui de l'utilisateur concerné, tandis que la fonction 4.2 interdit l'accès par ré-authentification à un rôle autre que celui légitimement attribué à l'utilisateur.

Traitement des erreurs

La fonction ne génère pas directement d'erreurs. Les tentatives de contournement sont traitées comme des erreurs des fonctions 4.1, 4.2, 3.1 et 3.2 sur lesquelles la fonction s'appuie.

4 Fonctions d'authentification

4.1 Authentification des utilisateurs

Description de la fonction

Cette fonction réalise l'authentification des utilisateurs avant de leur donner accès à toute interface interactive du système. L'authentification nécessite la saisie d'un nom d'utilisateur valide (apparaissant dans */etc/passwd*) et autorisé à ce connecter interactivement (c'est-à-dire membre du groupe *crypthomes*, et disposant sur le système de partitions chiffrées), ainsi que du mot de passe de ce compte, vérifié à l'aide des fichiers *shadow* et des clés chiffrées de chiffrement de partitions utilisateur.

Un service secondaire consiste à réaliser la ré-authentification d'utilisateurs des cages, à l'aide du démon *pwcheckd*, en vérifiant le mot de passe d'utilisateurs qui en font la demande (par exemple, pour déverrouiller une session graphique).

Les différents mécanismes associés à cette fonction sont décrits plus en détail en [CLIP_1302].

Objectif de la fonction

Cette fonction vise à interdire les accès illégitimes au système, et à permettre une traçabilité des activités authentifiées.

Protection et non-contournement de la fonction

- L'authentification auprès de XDM est, par construction de CLIP, le seul moyen d'obtenir une session interactive sur le système (en l'absence notamment de connexions distantes ou sur la console). XDM réalise cette authentification à l'aide d'une pile PAM configurée de manière à réaliser, dans cet ordre, les opérations suivantes (tout échec intermédiaire entraînant l'échec de l'ouverture de session) :
  - vérification de l'existence du compte (*pam_tcb*)
  - vérification du mot de passe par comparaison avec l'empreinte stockée dans le fichier *shadow* de l'utilisateur (*pam_tcb*)
  - vérification de l'appartenance de l'utilisateur au groupe *crypthomes* (*pam_exec_pwd*)
  - déchiffrement des clés de partitions et montage des partitions utilisateur (*pam_exec_pwd*)
- La fonction est assurée par des éléments du coeur de CLIP (*pam* et sa configuration) protégés en intégrité par 2.4, et par le démon *XDM*, protégé en intégrité sur le disque (sauf vis-à-vis du coeur CLIP et de $\text{UPDATE}_{\text{clip}}$) par 2.7, et protégé en mémoire en tant que démon privilégié par 2.3. En particulier, il est impossible pour un utilisateur interactif du système de modifier la fonction d'authentification.
- La récupération des mots de passe dans le swap est interdite par 2.10.

- La capacité d'un attaquant à deviner le mot de passe d'un utilisateur du système est largement fonction de la complexité des mots de passes choisis par les utilisateurs. CLIP effectue un contrôle de qualité des mots de passe utilisateur lors de leur définition (cf. [CLIP_1302]), qui se veut plus une aide au choix de mots de passe corrects qu'une solution complète. Par ailleurs, la recherche exhaustive de mot de passe est dans tous les cas compliquée par les mesures techniques suivantes :
  - les interfaces normales d'authentification (XDM et *pwcheckd*) introduisent des délais importants (3 secondes) entre un échec d'authentification et la tentative suivante, et journalisent tous les échecs.
  - L'interface XDM ne peut pas être facilement *« scriptée »* de manière à automatiser la recherche. L'interface *pwcheckd* le peut, mais ne permet dans tous les cas que la vérification du mot de passe de l'utilisateur qui l'appelle, et jamais d'un autre utilisateur.
  - La recherche exhaustive directe, à partir des empreintes stockées sur le disque, n'est possible que depuis le socle CLIP, les fichiers permettant cette recherche (fichiers *shadow* et clés de partitions chiffrées) n'étant pas montés dans les arborescences des cages. De plus, une telle recherche exhaustive est compliquée par la résistance intrinsèque du mécanisme *bcrypt* : un hachage *bcrypt* avec un nombre important d'itérations *eksblowfish* (2^12) et un sel conséquent (128 bits) est nécessaire pour tester aussi bien l'empreinte *shadow* qu'une clé chiffrée.

Traitement des erreurs

Les erreurs d'authentification entraînent l'échec de l'ouverture (ou du déverrouillage) de session. Lorsque l'erreur correspond à un mot de passe incorrect (et non, par exemple, à un compte incorrect), un délai de 3 secondes est de plus imposé avant toute nouvelle tentative d'authentification. Ces échecs sont journalisés dans le fichier *auth.log*.

4.2 Authentification des administrateurs et auditeurs

Description de la fonction

La fonction réalise une seconde authentification des utilisateurs ayant un profil administrateur ou auditeur, par ailleurs déjà authentifiés par la fonction générique 4.1, avant de leur donner accès aux cages $ADMIN_{clip}$, $AUDIT_{clip}$ depuis la cage $USER_{clip}$, ou éventuellement aux vues ADMIN des cages RM depuis la vue USER de ces mêmes cages.

L'authentification est réalisée par bi-clé *ssh* sur la boucle locale, respectivement sur les adresses communes des cages $ADMIN_{clip}$, $AUDIT_{clip}$ et $USER_{clip}$, et sur l'adresse locale de chaque cage RM.

Objectif de la fonction

Cette fonction limite l'accès aux fonctionnalités d'administration et d'audit à ceux des utilisateurs (des cages et vues USER) qui y sont spécifiquement autorisés.

Protection et non-contournement de la fonction

- L'authentification sur la boucle locale réseau est le seul moyen pour un utilisateur d'accéder aux cages et vues auxquelles elle donne accès.
    - Il n'existe pas d'autre moyen que la boucle réseau locale pour accéder à la cage $ADMIN_{clip}$ depuis les autres cages du système, du fait de 3.1.
    - De même, il n'existe pas d'autre moyen que la boucle réseau locale pour accéder à la vue ADMIN d'une cage RM, du fait de 3.2.
    - La seule interaction autre que la boucle locale réseau autorisée par 3.1 entre la cage $AUDIT_{clip}$ et les autres cages du système est la socket de collecte de journaux créée par le démon *syslog-ng* de $AUDIT_{clip}$ dans chaque cage. Cette interface n'est utilisable que par le démon *syslog-ng* lancé au démarrage du système (intègre du fait de 2.6), dans la mesure où ces *sockets* de collecte ne peuvent être créées qu'avant l'enfermement du démon dans $AUDIT_{clip}$. Le protocole *syslog-ng* ne permettant que la remontée de message de journaux, l'utilisation de cette interface pour créer une session dans $AUDIT_{clip}$ nécessiterait donc l'exploitation d'une vulnérabilité spécifique dans *syslog-ng*, ce qui est largement bloqué par 2.2, 3.1 et 2.3 (*syslog-ng* étant un démon privilégié au sens de CLIP-LSM).
- Les flux *ssh* légitimes (c'est-à-dire entre $ADMIN_{clip}$, $AUDIT_{clip}$ et $USER_{clip}$ d'une part, et sur la boucle locale propre à chaque cage RM d'autre part) sont les seuls autorisés par le filtrage réseau (3.6). Il est donc impossible par exemple de se connecter à $ADMIN_{clip}$ depuis une cage RM.
- Les démons *sshd* sont les seuls exécutables au sein des cages $ADMIN_{clip}$, $AUDIT_{clip}$, $USER_{clip}$ et RM autorisés à se mettre en écoute sur le port *ssh*, du fait :
    - de la non attribution du privilège *CLSM_PRIV_NETSERVER* aux autres exécutables de ces cages (2.3) ;

- de la non attribution de la capacité *CAP_NET_BIND_SERVICE* aux autres exécutables de ces cages (2.3).

- Les démons *sshd* sont lancés depuis des exécutables intègres (grâce à 2.4 pour les démons de $ADMIN_{clip}$ et $AUDIT_{clip}$, et grâce à 2.7 dans les cages RM), par des scripts de démarrage intègres (2.6). Leurs fichiers de configuration ne sont pas modifiables depuis la cage dans laquelle ils s'exécutent, pas plus que leur clés privées de serveur.
- Les démons *sshd* sont configurés de manière à n'autoriser que les connexions sous le compte légitime pour la cage ou vue dans laquelle ils sont exécutés (*_admin* ou *_audit*), et ce uniquement par la méthode *PubkeyAuthentication* du protocole *SSH-2*.
- Les clés publiques autorisées pour l'authentification *ssh* dans un compartiment sont modifiables par l'utilisateur légitime (*_admin* ou *_audit*) de ce compartiment. En revanche, les clés privées associées sont accessibles uniquement par leur utilisateur propriétaire, du fait de 3.4 (ces clés étant stockées sur une partition chiffrée de l'utilisateur). La récupération de ces clés ou de leurs mots de passe dans le swap est interdite par 2.10.
- Toutes les connexions réussies ou échouées sont journalisées. De plus, les démons *ssh* affichent, lors de chaque connexion, la date et l'heure de la précédente connexion réussie, ce qui aide un utilisateur légitime à détecter des utilisations non autorisées du compartiment.
- L'exploitation de vulnérabilité dans les démons *sshd* est complexifiée par :
  - la non exposition de ces démons aux autres cages par d'autres interfaces que la boucle locale, du fait de 3.1 et 3.2.
  - Le durcissement de ces démons par 2.2.
  - La protection spécifique de ces démons, privilégiés au sens de CLIP-LSM, du fait de 2.3.
  - L'absence d'exécutables autres que *ssh* utilisables pour se connecter à *sshd* depuis une autre cage, du fait de 2.11.

Traitement des erreurs

Les erreurs d'authentification entraînent un échec de l'ouverture de session dans le profil administrateur ou auditeur, et, dans le cas des sessions dans $ADMIN_{clip}$ et $AUDIT_{clip}$, une terminaison immédiate de la session utilisateur dans son ensemble. Les échecs sont journalisés dans le fichier *auth.log* (cages CLIP), ou *rm_X_auth.log* (cages RM), avec *X=h* ou *b* selon la cage RM concernée, lorsqu'ils sont effectivement détectés par le serveur *sshd*, c'est-à-dire lorsqu'une clé incorrecte est utilisée par le client. On notera que la saisie d'un mot de passe *ssh* incorrect entraîne pour le client l'échec du déchiffrement de la clé privée *ssh*, et que dans ce cas le serveur *sshd* ne voit pas la demande de connexion. Un tel échec n'est donc pas journalisé.

4.3 Authentification des passerelles IPsec

Description de la fonction

La fonction réalise l'authentification des passerelles IPsec avec lesquelles les clés de chiffrement et d'authentification IPsec sont négociées à l'aide du protocole IKEv2, ainsi que l'authentification de la négociation de clé proprement dite. Ces authentifications sont réalisées par le démon *iked* préalablement à toute injection de clé IPsec dans la couche noyau, en s'appuyant sur les fonctionnalités de négociation de clé authentifiée de la bibliothèque CCSD ([CCSD]). Le protocole mis en oeuvre et son interface avec CCSD est décrit en détail dans le document de référence [CLIP_1502] .

Objectif de la fonction

Cette authentification vise à éviter le contournement de la fonction de chiffrement des flux IP (3.7) par une attaque active de type *man in the middle*.

Protection et non-contournement de la fonction

- Le contournement de la fonction par des moyens cryptographiques est supposé impossible, par hypothèse sur la bibliothèque CCSD.
- Le démon *iked* et ses fichiers de configuration sont inclus dans le coeur du socle CLIP, et sont donc intègres au lancement du fait de 2.4 et 2.6.
- Les clés privées d'authentification du démon *iked* local ne sont visibles que dans le socle CLIP et dans la cage $\text{ADMIN}_{\text{clip}}$, et ne sont accessibles qu'à *root* de ces cages. 3.1 interdit donc tout accès à ces clés depuis les autres cages du système.
- Les clés publiques des passerelles sont injectées lors de l'installation initiale du système, et stockée dans un répertoire du coeur du socle CLIP. 2.4 garantit que ces clés ne sont plus modifiables (hors séquence de mise à jour du coeur, intègre du fait de), et ce y compris depuis la cage $\text{ADMIN}_{\text{clip}}$ ou le socle CLIP. Ainsi, l'administrateur local d'un poste CLIP peut injecter une clé privée frauduleuse (et ainsi par exemple faire passer le poste pour un autre poste CLIP, sous réserve qu'il ait accès à la clé privée de ce dernier), mais ne peut pas pour autant contourner l'authentification des passerelles et faire authentifier une passerelle arbitraire (dont la clé privée ne correspondrait pas à la clé publique connue par le poste).
- Le démon *iked* est exécuté dans le socle CLIP, il est donc impossible, du fait de 3.1, d'interférer avec son fonctionnement depuis une cage du système.
- Le démon *iked* est privilégié du point de vue du LSM CLIP (capacités et privilèges CLSM supplémentaires par rapports à ceux attribués normalement à un processus *root*). Il est donc très difficile d'interférer avec son fonctionnement depuis le socle CLIP (seuls les démons *iked* et *spmd* disposent des privilèges nécessaires, hors séquence d'arrêt du poste).
- Les démons *iked* et *spmd* sont les seuls exécutables à disposer des privilèges nécessaires à l'injection de clés IPsec dans le noyau, hors séquence de démarrage du poste. Il est donc impossible de contourner l'authentification *iked* pour injecter des clés arbitraires, sauf à exploiter une vulnérabilité interne de *iked*, ou une vulnérabilité de *spmd*.

Traitement des erreurs

En cas d'échec d'authentification, aucune association de sécurité IPsec n'est établie avec la passerelle concernée, et l'erreur est journalisée dans le fichier *daemon.log*.

4.4 Authentification des serveurs de mise à jour

Description de la fonction

La fonction réalise l'authentification HTTPS des serveurs de téléchargement des mises à jour, par rapport à une autorité de certification dont le certificat est installé localement par l'administrateur du poste.

Objectif de la fonction

La fonction complète l'authentification des flux réseau de mise à jour principalement réalisée à l'aide du tunnel IPsec de mise à jour (cf. 3.7 et 4.3), en offrant une authentification de niveau applicatif.

Protection et non-contournement de la fonction

- La fonction est assurée par les outils de téléchargement de mise à jour, qui appartiennent au coeur de la cage $UPDATE_{clip}$, et sont donc intègres du fait de 2.5.
- Le chemin du certificat d'autorité racine à utiliser est fixe, et pointe sur un fichier modifiable uniquement par l'administrateur local depuis la cage $ADMIN_{clip}$, du fait de la configuration des droits en écriture sur les montages et de 3.1.
- Les mécanismes cryptographiques mis en oeuvre pour cette authentification sont supposés corrects et robustes, par hypothèse sur la bibliothèque *openssl* qui les fournit.

Traitement des erreurs

Une erreur d'authentification du serveur de mise à jour entraîne l'arrêt du téléchargement, et la journalisation de l'échec dans le fichier de journaux *messages.*

4.5 Authentification des paquetages de mise à jour

Description de la fonction

La fonction réalise l'authentification des paquetages de mise à jour en vérifiant la double signature CCSD de chaque paquetage. Cette vérification est opérée deux fois : une fois immédiatement après le téléchargement des paquetages (dans la cage $UPDATE_{clip}$), et une fois juste avant leur installation (dans $UPDATE_{clip}$ ou éventuellement dans la vue UPDATE d'une cage RM). Tout échec de vérification entraîne la suppression du paquetage affecté, et la journalisation de l'incident.

Objectif de la fonction

La fonction assure, de manière complémentaire à l'authentification des flux réseau de téléchargement de mises à jour (3.7), l'authenticité des logiciels installés sur le système. Cette authenticité est primordiale pour le maintien dans la durée de toutes les autres fonctions de sécurité du système.

Protection et non-contournement de la fonction

- La vérification est opérée par des utilitaires du coeur de la cage $UPDATE_{clip}$, et éventuellement du coeur de la vue UPDATE d'une cage RM. L'intégrité de ces utilitaires est assurée par 2.4 (pour les utilitaires de $UPDATE_{clip}$ uniquement) et 2.5.
- Les clés de vérification de signature, qui fournissent les paramètres cryptographiques et l'autorité de certification à utiliser pour la vérification, sont installées dans le coeur du socle CLIP, et sont donc intègres du fait de 2.4. Il est donc impossible de configurer le système de manière à accepter des signatures créées à l'aide de clés privées frauduleuses.
- Dans le cas particulier de la mise à jour du coeur CLIP, la vérification significative (celle réalisée avant l'installation des paquetages, et non celles réalisée après leur téléchargement) est réalisée au cours de la séquence de démarrage, ce qui garantit son exécution dans un environnement intègre du fait de 2.6.
- Dans le cas des mises à jour de paquetages secondaires CLIP, la vérification significative est réalisée au sein de la cage $UPDATE_{clip}$, et son environnement d'exécution est protégée contre un attaquant situé dans toute autre cage du fait de 3.1.
- Dans le cas des mises à jour RM, la vérification est réalisée dans la vue UPDATE de la cage RM concernée, et sont environnement d'exécution est protégé vis-à-vis des autres cages du système par 3.1, et vis-à-vis des autres vues de la cage RM concernée par 3.2.
- En revanche, en dehors du cas des mises à jour de coeur CLIP, il est toujours possible à un attaquant disposant de privilèges *root* au sein du compartiment logiciel qui effectue la vérification ($UPDATE_{clip}$ ou UPDATE, selon les cas) d'interférer avec cette vérification, et potentiellement de faire accepter un paquetage non signé. Cette limitation n'est pas réellement significative, dans la mesure ou un attaquant disposant de tels privilèges pourrait de toutes façons installer directement les exécutables concernés par la mise à jour, sans passer par le système de gestion de paquetages.
- La robustesse des mécanismes cryptographiques employés pour la vérification est supposée

avérée, par hypothèse sur la bibliothèque CCSD qui les fournit ([CCSD]).

Traitement des erreurs

L'échec d'une vérification de signature de paquetage interdit l'inclusion dans le miroir local ou l'installation (selon que l'erreur est détectée lors du téléchargement ou de l'installation) du paquetage concerné, et l'inscription d'un message d'erreur dans le fichier de journaux *messages* (si l'erreur se produit lors du téléchargement, ou lors de l'installation d'un paquetage CLIP), ou *rm_X_messages.log* (si l'erreur se produit lors de l'installation du paquetage dans la cage RM_X).

4.6 Authentification des supports amovibles

Description de la fonction

Cette fonction assure l'authenticité des supports amovibles connectés au système, en vérifiant la signature RSA du secteur de méta-données spécifiques stocké en début et fin de support. La signature est réalisée – et vérifiée – à l'aide d'une clé privée propre à chaque utilisateur et chaque niveau.

Objectif de la fonction

Cette fonction protège le système contre l'utilisation de supports amovibles arbitraires, et contribue donc à protéger l'intégrité générale de celui-ci. Elle authentifie le niveau de chaque support et permet d'en limiter l'utilisation à un niveau et un utilisateur, ce qui complète le cloisonnement (inter-niveaux et inter-utilisateurs) assuré par le chiffrement de ces mêmes supports (cf. 3.5). Enfin, elle assure l'authenticité des chiffrés des clés symétriques utilisées par ce même chiffrement, contribuant ainsi à la robustesse de ce dernier.

Protection et non-contournement de la fonction

- La vérification de signature est réalisée par des utilitaires du coeur du socle CLIP, intègres du fait de 2.4.
- La vérification est réalisée au sein du socle CLIP, et s'effectue donc dans un environnement d'exécution protégé vis-à-vis de toutes les cages du système, du fait de 3.1.
- La vérification est réalisée par un utilitaire privilégié au sens du LSM CLIP, ce qui protège aussi son environnement d'exécution vis-à-vis des processus moins privilégiés du socle CLIP (cf. 2.3).
- La clé publique utilisée pour la vérification de signature est stockée au sein de la partition chiffrée de niveau CLIP de l'utilisateur, ce qui lui assure une protection partielle en intégrité :
  - Sa modification est impossible depuis une cage autre que $USER_{clip}$ ou le socle CLIP, du fait de la non exposition de cette partition (sous forme chiffrée ou claire), et grâce à 3.1.
  - Sa modification est difficile depuis $USER_{clip}$ ou le socle CLIP lorsque l'utilisateur n'est pas connecté, du fait du chiffrement 3.4. Cependant, une telle modification n'est pas totalement impossible, le mode de chiffrement mis en oeuvre n'offrant aucune garantie d'intégrité cryptographique.

- La menace liée à la non protection en intégrité des clés publiques vis-à-vis de l'utilisateur propriétaire est jugée acceptable, car :
  - sur un poste CLIP non multi-niveaux (pas de cages RM), la menace porte uniquement sur l'intégrité du système. Cette menace est non significative au sein de $USER_{clip}$ – qui doit déjà être compromise pour sa mise en oeuvre – et modérée pour le reste du système, du fait des options de montages restrictives imposées sur les supports amovibles (cf. [CLIP_DCS_15088] et [CLIP_1201]).
  - sur un poste CLIP multi-niveaux, la concrétisation de la menace permet éventuellement de contourner l'authentification du niveau d'un support amovible, et fait donc peser un risque sur le cloisonnement inter-niveaux. Cependant, une telle attaque est difficile à mettre en oeuvre, du fait de la faible interactivité de la cage $USER_{clip}$ dans une telle configuration. De plus, le cloisonnement inter-niveau resterait maintenu dans un tel cas de figure par le chiffrement des supports (3.5).
- La clé privée utilisée pour créer les signatures est stockée sur la partition chiffrée de niveau CLIP de l'utilisateur, ce qui lui assure une protection partielle en confidentialité, selon des principes similaires à ceux valables pour l'intégrité des clés publiques :
  - la clé est protégée à tout moment vis-à-vis des cages autres que $USER_{clip}$, du fait de la non exposition de la partition
  - la clé est protégée vis-à-vis du socle CLIP et de la cage $USER_{clip}$ lorsque l'utilisateur n'est pas connecté, du fait du chiffrement de la partition.
  - L'export des clés permet éventuellement de rompre la confidentialité inter-utilisateurs, mais maintient le cloisonnement inter-niveaux, dans la mesure où :
    - L'export réseau est soumis au cloisonnement inter-niveau assuré par 3.7 et 3.6.
    - L'export sur support amovible est accompagné d'un chiffrement spécifique à chaque niveau (mais pas chaque utilisateur).
  - La récupération des clés privées dans le swap est interdite par 2.10.

Traitement des erreurs

Une erreur de vérification de signature de support amovible interdit le déchiffrement du support, et donc son montage ultérieur. Cette erreur est signalée par une fenêtre graphique *"pop-up"* dans $USER_{clip}$ si une session utilisateur est ouverte lors de la connexion du support amovible, et dans tous les cas par un message d'erreur inscrit dans le fichier de journaux *messages*.

5 Initialisation et gestion sûres

La présente section représente des propriétés de sécurité qui découlent non pas de mécanismes techniques implantés au sein du système CLIP, mais de règles et procédures organisationnelles respectées lors de son installation et de sa gestion post-installation. Par conséquent, le non-contournement de ces différentes fonctions repose uniquement sur le respect des règles correspondantes, ce qui ne permet pas de décrire spécifiquement dans le présent document leur protection ou les éventuelles traitements d'erreurs associés. Ces différentes fonctions, assurées hors du périmètre du système CLIP lui-même, visent à interdire le contournement par des moyens extérieurs des fonctions de sécurité de ce dernier.

5.1 Installation depuis un support intègre

Description de la fonction

Le support d'installation utilisé pour installer un poste CLIP est protégé en intégrité durant son acheminement. Cette intégrité est vérifiée par un hachage cryptographique avant l'utilisation du support, selon la procédure décrite dans le document de référence [CLIP_2001].

Objectif de la fonction

Cette fonction garantit l'installation d'un système CLIP intègre, et est donc indispensable au non contournement de toutes les autres fonctions de sécurité du système.

5.2 Génération sûre et protection des secrets cryptographiques

Description de la fonction

Les paramètres cryptographiques importés sur un poste CLIP lors de son installation (et non générés sur le poste lui-même) sont générés correctement, sur un poste IGC hors-ligne et protégé en intégrité. Ces paramètres sont protégés en intégrité et confidentialité lors de leur acheminement en vue de l'installation. Les supports utilisés pour cet acheminement sont au besoin effacés, et gérés de manière à protéger la confidentialité des données rémanentes, après usage. Ces différentes procédures sont décrites dans le document de référence [CLIP_2001]. Les paramètres ainsi protégés sont :

- Les clés CCSD de vérification des signatures de paquetages.
- Les clés CCSD pour l'authentification IKE (clé privée du poste, clés publiques de ses pairs)
- Le certificat de l'autorité certifiant le serveur HTTPS de mise à disposition des mises à jour.
- Les clés RSA publiques (installées sur le poste CLIP) et privées (non installées sur le poste CLIP) permettant l'export de clés RSA (générées sur le poste CLIP) de signature et chiffrement de supports USB.

Une procédure similaire est appliquée lors de l'acheminement de nouveaux paramètres cryptographiques après installation, pour ceux des paramètres qui peuvent être mis à jour (clé privée

CCSD pour l'authentification IKE).

De plus, ceux de ces paramètres qui constituent des secrets au sens cryptographique (clés privées CCSD et RSA), ainsi que les éléments secrets générés et exportés ultérieurement depuis le poste CLIP (clés privées RSA pour le chiffrement de supports USB), sont protégés en confidentialité par des mesures techniques et organisationnelles adaptées lorsqu'ils sont stockés en dehors du poste CLIP, par exemple au sein d'une IGC ou sur un poste échangeant des données par support amovible avec le poste CLIP.

Objectif de la fonction

Cette fonction assure la protection, en dehors du périmètre du poste CLIP lui-même, des éléments cryptographiques mis en oeuvre par les fonctions de sécurité 4.3, 4.4, 4.5, 4.6, 3.7 et 3.5. Elle joue donc un rôle critique pour garantir le non contournement par des moyens externes de ces différentes fonctions.

5.3 Création des comptes utilisateurs initiaux

Description de la fonction

Immédiatement après l'installation d'un poste CLIP, les comptes utilisateurs suivants sont systématiquement créés, en plus de tout compte utilisateur final nécessaire à l'utilisation du poste :

- deux comptes administrateurs CLIP redondants, associés à deux personnes physiques différentes.
- un compte auditeur CLIP associé à une personne physique distincte des administrateurs.

Objectif de la fonction

Cette fonction garantit que les comptes nécessaires à la détection d'une erreur ou action illégitime d'un administrateur peut être détectée par un auditeur, et corrigée par un autre administrateur. Elle est essentielle au maintien de la disponibilité du poste dans un tel cas.

5.4 Configuration du BIOS

Description de la fonction

Immédiatement après l'installation d'un poste CLIP, le BIOS du poste est configuré de manière conforme au guide [CLIP_2002] applicable au type de matériel concerné. Cette configuration implique notamment de réaliser les opérations suivantes :

- Interdire le démarrage sur un support autre que le disque dur du poste.
- Désactiver les périphériques de communication (en particulier sans fil) non utilisés par le système CLIP.
- Verrouiller la configuration du BIOS, et la protéger par un mot de passe.

Objectif de la fonction

L'interdiction du démarrage sur un support externe ou sur le réseau interdit le contournement de la séquence de démarrage intègre (2.6). La désactivation des périphériques non utilisés réduit les interfaces du système exposées à des attaques potentielles (bien que le support de ces périphériques ne soit pas inclus dans le noyau CLIP, des attaques exploitant des vulnérabilités dans le *firmware* des périphériques plutôt que dans leur pilotes noyau resteraient sinon *a priori* possibles).

Annexe A Références

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

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

[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_1206] *Documentation CLIP – 1206 – Génération de nombres aléatoires*

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

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

[CLIP_1303] *Documentation CLIP – 1303 – X11 et cloisonnement graphique*

[CLIP_1304] *Documentation CLIP – 1304 – Socle et cages CLIP*

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

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

[CLIP_1502] *Documentation CLIP – 1502 – Racoon2*

[CLIP_2001] *Documentation CLIP – 2001 – Procédure d'installation*

[CLIP_2002] *Documentation CLIP – 2002 – Guide de configuration du BIOS (version adaptée au type de matériel concerné)*

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

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

[CLIP_DCS_15093] *Document de conception de l'étude sur les paramètres contrôlables par l'administrateur,*

CLIP-DC-15000-093-DCS

[CLIP_DCS_15088] *Document de conception de l'étude sur les supports amovibles, CLIP-DC-15000-088-DCS*

[CCSD] *Couche Cryptographique pour la Sécurité de Défense Document d'Interface Client version 3.2*

[CRYPTO] *Régles et recommendations concernant le choix et le dimmensionnement des mécanismes cryptographiques de niveau standard ou renforcé, 2379/SDGN/DCSSI/SDS/LCR du 19 décembre 2006.*

Annexe B Liste des figures

Annexe C Liste des remarques