Documentation CLIP

1302

Fonctions d'authentification CLIP

Version	Date	Auteur	Commentaires
1.1.2	17/09/2008	Vincent Strubel	Ajout des utilisateur et groupe *dhcp*. A jour pour CLIP_v03.00.17.
1.1.1	11/09/2008	Vincent Strubel	Ajout de précisions sur le *padding* des chiffrements symétriques *openssl*.
1.1	27/08/2008	Vincent Strubel	Ajout de précisions sur la génération et le renouvellement de clés SSH.
1.0	30/07/2008	Vincent Strubel	Version initiale, à jour pour CLIP_v03.00.07.

Table des matières

Introduction

Le présent document décrit les différents mécanismes d'authentification locale des utilisateurs CLIP. Outre un mécanisme classique de vérification de mot de passe (section 1), l'ouverture d'une session utilisateur dans CLIP est conditionnée à la possibilité de déchiffrer une ou plusieurs partitions chiffrées propres à l'utilisateur (section 2). Ces deux mécanismes s'appuient largement sur le système de modules d'authentification PAM, dont la configuration pour CLIP est détaillée en section 3. Par ailleurs, outre cette authentification principale, CLIP peut dans certains cas requérir des utilisateurs une ré-authentification, en particulier pour accéder aux rôles d'administration ou d'audit (section 4), ou déverrouiller une session graphique (section 4.3).

On notera bien que seuls les mécanismes d'authentification portant sur les utilisateurs sont décrits dans ce document. Le système CLIP met par ailleurs en oeuvre d'autres mécanismes d'authentification, notamment de systèmes distants (IKE, IPsec, HTTPS), de supports amovibles USB, ou de paquetages de mise à jour (signatures), qui sont détaillés dans d'autres documents.

1 Authentification UNIX

1.1 Hachage *bcrypt* des mots de passe

1.1.1 Principe de *bcrypt*

Le mécanisme d'authentification principal d'un système CLIP repose sur la saisie d'un mot de passe utilisateur, qui est comparée avec l'empreinte du mot de passe d'un compte local. CLIP calcule les empreintes de mots de passe à l'aide de l'algorithme *bcrypt*, issu d'*OpenBSD*, plutôt qu'à l'aide de l'algorithme standard des distributions Linux, basé sur MD5.

L'algorithme *bcrypt* est décrit en détail dans le document [BCRYPT]. Il est basé sur un algorithme de chiffrement *eksblowfish*, qui réalise un chiffrement similaire à *blowfish*, mais avec une étape de chargement de clé (c'est-à-dire de génération de sous-clés et de *S-boxes*) adapté à dessein de manière à présenter un coût important - et configurable - en temps de calcul. La fonction de chargement de clé accepte trois paramètres : un *coût,* un *sel* de 128 bits généré aléatoirement et une *clé* d'au plus 72 octets, cette dernière correspondant typiquement au mot de passe d'un l'utilisateur. Pour un *coût* donné, la fonction réalise $2^{coût}$ étapes d'expansion de clé à partir du *sel* et de la *clé*. A l'issue de ce chargement de clé, le hachage *bcrypt* est obtenu par un chiffrement *blowfish* (répété 64 fois), avec l'état obtenu par le chargement de la clé et du sel, de la chaîne de caractères constante *"OrpheanBeholderScryDoubt"*. L'empreinte obtenue en sortie d'un appel *bcrypt* est la concaténation du préfixe *"$2a$"* (identifiant l'algorithme *bcrypt*, comme *"1"* identifie l'algorithme standard basé sur MD5), du coût suivi d'un *'$'*, du sel encodé en *base64*, et du chiffré de la chaîne de caractère (184 octets, encodés en *base64*). Étant donné une telle empreinte, la vérification d'un mot de passe consiste à lui appliquer *bcrypt* avec le sel et le coût stockés dans l'empreinte, et à comparer le résultat avec l'empreinte. Le coût peut être ajusté entre 4 et 31.

La lenteur de calcul de l'algorithme (due essentiellement au temps de chargement de clé) lui assure une bonne protection contre les attaques par force brute. De plus, l'algorithme employé se prête très mal à une implémentation "cablée" matérielle, qui permettrait d'optimiser ce calcul, et le coût ajustable permet d'augmenter la complexité de calcul au fur et à mesure de l'évolution des performances des matériels informatiques. Par ailleurs, la taille importante du sel apporte, sous réserve d'une utilisation correcte de l'ensemble de l'espace des sels possibles, une très bonne protection contre les attaques en force brute à l'aide de tables précalculées (*rainbow tables*).

1.1.2 Intégration dans CLIP

Au sein de CLIP, *bcrypt* est intégré sous la forme d'un *patch* issu de la distribution *Open Wall Linux*, appliqué au paquetage *sys-libs/glibc* de manière à ajouter *bcrypt* aux algorithmes supportés par l'interface standard *crypt(3)*. Par ailleurs, un autre *patch Open Wall* est appliqué à *sys-apps/shadow*, de manière à permettre l'utilisation de cet algorithme lors de la vérification d'un mot de passe ou de la création d'une empreinte. Ce *patch* ajoute notamment une variable *CRYPT_ROUNDS* à définir dans */etc/login.defs*, qui correspond au coût par défaut à utiliser lors de la création d'une empreinte *bcrypt*.

Cependant, au sein de CLIP, les fonctionnalités *shadow* ne sont pas utilisées pour la génération des empreintes de mots de passe, qui est confiée à un utilitaire développé spécifiquement pour CLIP, *cryptpasswd* (*app-clip/clip-useradmin*). Celui-ci s'appelle selon la forme :

```
cryptpasswd [options]
```

les options supportées étant :

- *--password* <*pass*> : utiliser la chaîne <*pass*> comme mot de passe à hacher.
- *--passvar* <*var*> : utiliser le contenu de la variable <*var*> comme mot de passe à hacher. Cette options est une alternative à *--password*, qui permet d'éviter que le mot de passe n'apparaisse sur la ligne de commande du processus[1].
- *--salt* <*salt*> : utiliser la chaîne <*salt*> comme sel (encodé en *base64*). Cette chaîne est au besoin *"*paddée*"* avec des zéros pour atteindre les 128 bits nécessaires.
- *--saltfile* <*file*> : utiliser le contenu de <*file*> comme sel, alternativement à *--salt*. Le fichier <*file*> doit contenir au moins 128 bits, car aucun *padding* n'est réalisé dans ce cas. En cas d'impossibilité de lire 128 bits, l'appel échoue.
- *--rounds* <*count*> : utiliser un coût <*count*>.
- *--settings* <*set*> : extraire le coût et le sel de la chaîne <*set*>. Cette option est incompatible avec les options *--salt, --saltfile* et *--rounds*. La chaîne <*set*> doit prendre la forme d'un préfixe de configuration d'empreinte *bcrypt*, c'est-à-dire *"$2a$<coût>$<sel>"*, avec un <*sel*> de 128 bits exactement en *base64.*

Lorsqu'aucun sel n'est spécifié (par *--salt, --saltfile* ou *--settings*), 128 bits de sel aléatoire sont lus sur */dev/urandom*. Lorsqu'aucun coût n'est précisé (par *--rounds* ou *--settings*), un coût par défaut de 12 est utilisé. En cas de succès, la commande affiche sur sa sortie standard l'empreinte *bcrypt* calculée, sous sa forme complète, c'est-à-dire précédée du motif *$2a$*, du coût et du sel.

1.1.3 Génération des empreintes de mots de passe utilisateur

Lors de la création d'un compte ou de sa modification (changement de mot de passe), l'empreinte utilisée pour vérifier le mot de passe du compte est créée par un appel à la fonction *hash_password()* de */lib/clip/userkeys.sub* (*clip-libs/clip-sub*), qui appelle *cryptpasswd* en lui passant le mot de passe à hacher par l'environnement, avec un coût égal à la définition de *CRYPT_ROUNDS* si une telle définition est trouvée dans */etc/login.defs*, ou 12 par défaut. Aucun sel n'étant spécifiquement passé à l'utilitaire, celui-ci lit 128 bits d'aléa sur */dev/urandom* et les utilise comme sel.

A ce stade, la configuration de *CRYPT_ROUNDS* dans les système CLIP spécifie un coût de 12 lors de la création d'une telle empreinte. Des versions ultérieures du système pourraient redéfinir cette variable pour augmenter la complexité du calcul et ainsi s'adapter aux évolutions des performances des matériels informatiques. A titre de référence, le calcul d'une empreinte de mot de passe avec un coût de

[1] La ligne de commande d'un processus CLIP est toujours consultable par les autres processus du même utilisateur et de la même cage et / ou vue, dans */proc/<pid>/cmdline.* En revanche, l'accès à l'environnement d'un processus par */proc/<pid>/environ* est soumis à un contrôle plus strict, qui exige que le processus observateur dispose de privilèges au moins égaux à ceux de l'observé. Ainsi, il est préférable du point de vue de la sécurité de passer le mot de passe par l'environnement.

12 (2^{12} itérations de l'expansion de clé) prend en moyenne 800 millisecondes sur le matériel client type au moment de la rédaction du présent document[2].

Après ce calcul par *cryptpasswd*, l'empreinte est ajoutée au fichier *shadow* de l'utilisateur (cf. 1.2) par une commande *usermod* ou *useradd* selon les cas. On notera que d'autres empreintes du mot de passe utilisateur, générées de la même manière mais jamais stockées sur le disque, sont par ailleurs utilisées pour la gestion des partitions chiffrées de l'utilisateur, comme décrit en section 2.

1.2 Module d'authentification *pam_tcb*

1.2.1 Principe de TCB

Au sein de la plupart des systèmes Linux, les empreintes des mots de passe de tous les utilisateurs sont stockées dans un fichier unique, */etc/shadow*. Ce fichier est accessible en lecture uniquement par l'utilisateur *root*. On interdit ainsi à tout autre utilisateur de copier ces empreintes pour tenter une attaque par force brute "hors-ligne", c'est-à-dire basée sur le calcul direct des empreintes de mots de passe possibles, plutôt que sur la mise en oeuvre des interfaces légitimes d'authentification du système (qui mettent généralement en oeuvre des mesures complémentaires contre les attaques par force brute, en particulier l'introduction de délais artificiels). La contrepartie de ce choix est que la vérification d'une empreinte n'est possible que pour l'utilisateur *root*, et que tous les mécanismes d'authentification, ou de mise à jour des mots de passe doivent s'exécuter sous cette identité, éventuellement obtenue à l'aide d'un *bit setuid* sur l'exécutable concerné. Ainsi, toute vulnérabilité d'un logiciel réalisant l'authentification conduit à la compromission de l'ensemble du système. Même en présence de mécanismes de réduction des privilèges de l'utilisateur *root*, une telle vulnérabilité se traduit au minimum par la compromission de l'ensemble des empreintes de mots de passe utilisateur.

Au sein du système CLIP, ce mécanisme standard de stockage de mot de passe est remplacé par le mécanisme *TCB*, issu de la distribution *Open Wall Linux* ([TCB]). Ce mécanisme vise à permettre à chaque utilisateur d'accéder, sous sa propre identité mais à travers un groupe particulier, à son empreinte de mot de passe, et à celle-ci uniquement. Il repose sur la mise en oeuvre d'un fichier *shadow* par utilisateur, stockant uniquement l'empreinte de cet utilisateur. Les permissions UNIX placées sur ce fichier en limitent l'accès (en lecture et en écriture) au seul utilisateur propriétaire. Le fichier est par ailleurs stocké dans un répertoire accessible (droit en lecture et exécution) uniquement de l'utilisateur, dont le répertoire parent n'est accessible (en exécution) que par *root* et l'utilisateur *shadow*. Le répertoire de stockage du fichier *shadow* d'un utilisateur est par ailleurs propriété du groupe *shadow,* qui ne se voit pas pour autant attribuer les droits en lecture ou en exécution sur le répertoire. En revanche, un *bit setgid* placé sur ce répertoire garantit que les nouveaux fichiers créés dans celui-ci (typiquement des copies temporaires du fichier *shadow* réalisées lors de la mise à jour du mot de passe) appartiendront au groupe *shadow,* là encore, sans droit en lecture a priori. En résumé, l'arborescence de stockage d'une empreinte de mot de passe prend la forme suivante, pour un utilisateur *toto* :

```
/etc                        root:root       drwxr-xr-x
/etc/tcb                    root:shadow     drwx--x---
/etc/tcb/toto               toto:shadow     drwx--S---
/etc/tcb/toto/shadow        toto:shadow     -rw-------
```

Cette répartition des droits implique les restrictions suivantes :

[2] Portable DELL Latitude D530, avec un processeur *Intel Core2 Duo* cadencé à 2.0 Ghz.

- L'ajout et la suppression d'un utilisateur ne peuvent être réalisés que par *root*.
- La vérification d'une empreinte ne peut être réalisé que par *root* ou l'utilisateur propriétaire de l'empreinte. Dans le deuxième cas, le processus qui réalise la vérification doit de plus être membre du groupe *shadow*.
- De même, la modification d'une empreinte n'est possible que pour *root* ou pour l'utilisateur propriétaire, à condition pour ce dernier que le processus modifiant l'empreinte appartienne aussi au groupe *shadow*.

Ainsi, les utilitaires permettant à l'utilisateur de modifier son mot de passe (typiquement, */bin/passwd*) peuvent se passer de *bit setuid root*, qui peut être remplacé par un *bit setgid shadow*. Par ailleurs, TCB permet à un utilisateur de vérifier son propre mot de passe, par un appel à */usr/libexec/chkpwd/tcb_chkpwd*, lui aussi *setgid shadow*. Cependant, l'exécution de cet utilitaire est restreinte au groupe *chkpwd*.

Remarque 1 : Utilisation du groupe* auth *par TCB

Le mode de fonctionnement décrit ici est celui de TCB sans mise en oeuvre du groupe 'auth'*, c'est-à-dire lorsque la variable* TCB_AUTH_GROUP *est définie à* 'no' *dans le fichier* /etc/login.defs*, ce qui est le cas sous CLIP. Lorsque cette variable est positionnée à* 'yes'*, le groupe* 'auth' *se voit conférer le droit de lire les fichiers* shadow *de* /etc/tcb*, sous réserve d'accéder au contenu de ce dernier répertoire, qui reste limité au groupe* shadow*. Les permissions sont dans ce cas adaptées de la manière suivante :*

```
/etc                    root:root       drwxr-xr-x
/etc/tcb                root:shadow     drwx--x---
/etc/tcb/toto           toto:auth       drwx--s---
/etc/tcb/toto/shadow    toto:auth       -rw-r-----
```

Ainsi, tout processus membre des groupes auth *et* shadow *peut vérifier le mot de passe de n'importe quel utilisateur.*

Ce mode de fonctionnement n'est pas mis en oeuvre dans CLIP, car tous les processus qui vérifient un mot de passe utilisateur le font soit sous l'identité root*, soit sous l'identité de l'utilisateur. Cependant, son utilisation dans des évolutions futures n'est pas exclue, et le groupe* auth *est inclus de manière prévisionnelle dans les groupes standards du système CLIP (cf. 1.3).*

1.2.2 Mise en oeuvre sous CLIP

Le mécanisme TCB est intégré à CLIP par deux paquetages issus de la distribution *Gentoo* : *sys-apps/tcb*, et *sys-apps/shadow*. Le premier de ces paquetages apporte deux composants :

- Un module PAM *pam_tcb*, qui peut être inclus dans les configurations PAM de divers services en remplacement de *pam_unix,* pour réaliser une authentification (ainsi que les autres opérations supportées par PAM, par exemple le changement de mot de passe) TCB. On notera que ce module supporte, et utilise par défaut, le hachage *bcrypt* des mots de passe dès lors que celui-ci est supporté par la *glibc*.
- Une bibliothèque *libnss_tcb.so*, qui fournit le support TCB pour les fonctions de la famille

getspnam(3), dès lors que *"tcb"* apparaît sur la ligne *"shadow:"* de */etc/nsswitch.conf*, ce qui est le cas sous CLIP.

Le paquetage *shadow* se voit quant à lui appliquer dans CLIP un *patch* issu de la distribution *Open Wall Linux* (moyennant un portage sur des versions plus récentes de *shadow*), de manière à inclure le support de TCB dans les différents utilitaires de ce paquetage. Ce mode est ainsi autorisé par défaut dès lors que *USE_TCB* est définie à *yes* dans */etc/login.defs*, ce qui est le cas sous CLIP.

Le module *pam_tcb* est utilisé sous CLIP en remplacement de *pam_unix* dans la configuration PAM *system-auth* (qui est utilisée pour toutes les authentifications PAM du système, cf. 3). Le succès de l'appel à ce module est requis pour toute authentification. Le module *pam_tcb* est appelé avec les options supplémentaires suivantes :

- *shadow* : force la recherche des empreintes de mots de passe dans des fichiers de type *shadow*, plutôt que dans les fichiers de type *passwd*. Ce mode est normalement le mode par défaut.
- *prefix=$2a$* : force l'utilisation de *bcrypt* pour le hachage des mots de passe. Ce mode est aussi le mode par défaut.
- *fork* : tous les accès aux fichiers *shadow*, et la manipulation des empreintes qu'ils contiennent, sont réalisés dans un fils dédié du processus faisant appel au module, et ce fils est terminé lors de l'appel à *pam_end()*. Cette option permet de ne pas manipuler les données *shadow* dans le processus principal qui réalise l'authentification, et ainsi d'éviter une éventuelle rémanence en mémoire de ces données.

Les authentifications réussies ou échouées sont journalisées dans la *facility syslog AUTHPRIV*, et sont donc consultables dans le fichier */log/auth.log* au sein de la cage $AUDIT_{clip}$ ([CLIP_1304]).

1.3 Comptes et groupes utilisateurs

Un certain nombre de comptes et groupes utilisateurs sont définis sur tous les systèmes CLIP, indépendamment de leur configuration locale. Ces comptes sont pour la plupart non interactifs, et utilisés uniquement par des services du système. Ils sont résumés dans les Tableau 1 et Tableau 2

.

Compte	*uid*	*shell*	Nature du compte
root	0	*/bin/bash*	*Root* du système. Non interactif.
nobody	65534	*/bin/false*	Utilisateur non privilégié par défaut.
cron	16	*/bin/false*	Accès en écriture à */var/spool/cron*.
sshd	22	*/bin/false*	Identité du fils non privilégié de *sshd* (en mode séparation de privilèges).
syslog	300	*/bin/false*	Identité du processus *syslog-ng* après enfermement dans une cage / vue.
racoon	320	*/bin/false*	Identité réservée pour un démon *ike* non privilégié. Utilisée dans une configuration utilisant IKEv1, mais pas à ce stade dans les configurations standards (IKEv2).
dhcp	321	*/bin/false*	Identité réservée à un éventuel démon *dhcp*.
hsqldb	340	*/bin/false*	Identité non privilégiée utilisée par la base de donnée *hsqldb* (mise en oeuvre par *OpenOffice*).
_admin	4000	*/bin/login_shell.sh*	Compte d'administration locale (dans $ADMIN_{clip}$ et les vues ADMIN de cages RM).
_audit	5000	*/bin/login_shell.sh*	Compte de consultation des journaux dans $AUDIT_{clip}$.

Tableau 1: Comptes utilisateurs standards sur un système CLIP.

Groupe	*gid*	Nature du groupe
root	*0*	Groupe *root*.
nobody	*65534*	Groupes non privilégiés par défaut.
nogroup	*65533*	
wheel	*10*	Autorisation du *su* (non utilisé à ce stade dans CLIP).
tty	*5*	Accès à différents fichiers de */dev*
disk	*6*	
kmem	*9*	
floppy	*11*	
console	*17*	
audio	*18*	
cdrom	*19*	
video	*27*	
usb	*85*	
cron	*16*	Groupe associé à l'utilisateur *cron*. Permet la lecture des fichiers *crontab* des différents utilisateurs.
sshd	*22*	Groupe associé à l'utilisateur *sshd*.
syslog	*300*	Groupe associé à l'utilisateur *syslog*. Permet la lecture des journaux de */var/log*
racoon	*320*	Groupe associé à l'utilisateur *racoon*.
dhcp	*321*	Groupe associé à l'utilisateur *dhcp*.
hsqldb	*340*	Groupe associé à l'utilisateur *hsqldb*.
interp	*201*	Groupe autorisant l'accès en exécution à certains interpréteurs, par exemple *java*.
utmp	*406*	Groupe autorisant l'accès en écriture à *utmp* et *wtmp*.
auth	*407*	Groupe réservé pour la mise en oeuvre de TCB, mais non utilisé au sein de CLIP à ce stade.
chkpwd	*408*	Groupe permettant d'exécuter */usr/libexec/chkpwd/tcb_chkpwd*, qui est lui-même *setgid shadow*.
shadow	*409*	Groupe permettant l'accès en lecture aux entrées *shadow* TCB des différents utilisateurs du système.
crypthomes	*2000*	Groupes des utilisateurs pour lesquels une ou plusieurs partitions chiffrées doivent être montées

		à l'ouverture de session. Ces utilisateurs sont aussi les seuls autorisés à ouvrir une session graphique CLIP.
rm_admin	*3000*	Groupe des utilisateurs ayant le profil administrateur RM. Ce groupe n'est utilisé qu'au sein des systèmes de type CLIP-RM.
core_admin	*3001*	Groupe des utilisateurs ayant le profil administrateur CLIP.
core_audit	*3002*	Groupe des utilisateurs ayant le profil auditeur CLIP.
admin	*4000*	Groupe principal de l'utilisateur *_admin*.
audit	*5000*	Groupe principal de l'utilisateur *_audit*

Tableau 2: Groupes utilisateurs standards sous CLIP.

On notera plus particulièrement, parmi ces groupes :

- Le groupe *interp*, permettant de limiter l'accès à certains exécutables de type interpréteur, en forçant (par des permissions UNIX du type *root:interp rwxr-x---*) pour les utilisateurs normaux le passage par un exécutable intermédiaire, lui même *setgid interp*. Une telle configuration est utile pour forcer des vérifications sur les entrées (par exemple, ouverture des fichiers interprétés avec le drapeau *O_MAYEXEC,* cf. [CLIP_1201]) avant de passer la main à un interpréteur, en particulier dans le cas où le code source de ce dernier n'est pas disponible pour y intégrer directement les vérifications. Le seul exemple d'une telle configuration dans CLIP est l'interpréteur de *bytecode java*, bien que l'exécutable intermédiaire ne réalise à ce stade aucune vérification.
- Le groupe *crypthomes* contient tous les utilisateurs 'réels' (c'est-à-dire correspondant à un utilisateur physique, qui ouvre des sessions sur le poste), et aucun utilisateur 'virtuel'. L'appartenance à ce groupe force le montage de partitions chiffrées et temporaires lors de l'ouverture de session (cf. 2). Ce groupe est attribué automatiquement à la création d'un compte utilisateur quelconque depuis $\text{ADMIN}_{\text{clip}}$ ([CLIP_1304]).
- De même l'appartenance à *core_admin, rm_admin* et *core_audit* est réservée aux utilisateurs réels, dont elle détermine le profil. Un utilisateur membre de *crypthomes* mais d'aucun de ces trois groupes est un utilisateur 'normal', capable d'ouvrir une session d'utilisation standard. Ces différents groupes sont eux aussi attribués automatiquement à la création d'un compte utilisateur, en fonction du profil choisi.
- Bien qu'un seul compte utilisateur soit à ce stade utilisé au sein de chacune des cages $\text{ADMIN}_{\text{clip}}$ (et vues ADMIN des éventuelles cages RM) et $\text{AUDIT}_{\text{clip}}$ (*_admin* et *_audit* respectivement), la définition des groupes *admin* et *audit* permet d'envisager à terme la création de plusieurs profils administrateurs et auditeurs distincts, tous membres du groupe commun, qui conditionnerait en particulier la possibilité d'accéder à ces cages par une connexion *ssh* locale[3]. Ces comptes

[3] Les serveurs *sshd* de ces cages autoriserait dans ce cas les authentifications vers des comptes membres de ces groupes, plutôt que de les limiter aux seuls utilisateurs *_admin* ou *_audit*.

devront eux aussi utiliser */bin/login_shell.sh* comme *shell,* afin de positionner certaines variables d'environnement (en particulier LC_ALL et LANG) à l'ouverture de session dans les cages $ADMIN_{clip}$ et $AUDIT_{clip}$.

1.4 Gestion de la qualité des mots de passe

La création d'un compte utilisateur, ainsi que la modification du mot de passe d'un compte utilisateur existant, sont réalisés sous CLIP par des scripts spécifiques, qui appellent *cryptpasswd* puis *usermod* (cf. 1.1.3). Il n'est donc pas possible de vérifier la qualité des mots de passe par la méthode standard consistant à inclure le module *pam_cracklib* dans la configuration PAM du service de mise à jour de mot de passe (cf. 3). En revanche, une vérification de qualité est réalisée dans ces scripts par un appel direct à l'utilitaire */usr/sbin/cracklib-check* (*sys-libs/cracklib*). Cet appel vérifie la longueur et la qualité (nombre de lettres de types différents) du mot de passe, et vérifie qu'il est absent du dictionnaire *cracklib*. Ce dictionnaire est installé dans CLIP par le paquetage *cracklib*, et contient la concaténation de plusieurs dictionnaires issus de [COTSE], notamment les dictionnaires *cracklib, allwords,* et *french*. Lors de la saisie d'un nouveau mot de passe par un utilisateur, le mot de passe est passé en argument d'un appel à *cracklib-check*. Lorsque ce dernier retourne une erreur, le message d'erreur (localisé en français) est affiché pour l'utilisateur, et un nouveau mot de passe lui est demandé. La procédure est répétée tant que le mot de passe saisi n'est pas satisfaisant.

Cette vérification est systématiquement réalisée lors de la saisie d'un mot de passe de compte utilisateur à travers l'interface de gestion des comptes utilisateurs, utilisable à l'aide du client *userclt* depuis $ADMIN_{clip}$ (cf. [CLIP_1304]). Elle n'est en revanche pas appliquée aux mots de passe des clés *ssh* de ré-authentification des profils administrateur et auditeur (cf. 4), ni à ceux des clés de chiffrement et de signature de clés USB sécurisées ([CLIP_DCS_15088]), ni encore aux mots de passe des comptes initiaux créés lors de l'installation du système ([CLIP_2001]).

Remarque 2 : Comparaison d'un nouveau mot de passe à l'ancien

Contrairement à la mise en oeuvre du module pam_cracklib*, un appel direct à* cracklib-check *ne permet pas de comparer le nouveau mot de passe saisi lors de la mise à jour du mot de passe d'un compte existant à l'ancien mot de passe de ce compte. Ainsi, il n'est pas possible sous CLIP à ce stade d'imposer un nouveau mot de passe suffisamment différent du précédent.*

2 Partitions chiffrées et montages temporaires

Plusieurs montages peuvent être réalisés lors de l'ouverture d'une session interactive sur un système CLIP. Ces montages sont de deux types :

- Partitions chiffrées : des partitions de données pérennes, spécifiques à un utilisateur et chiffrées pour celui-ci, contenant ses données privées. Les systèmes de fichiers chiffrés correspondant à ces montages sont stockés dans des fichiers images de la partition */home* du système, plutôt que dans des partitions dédiées. Le chiffrement est réalisé à l'aide d'une clé de chiffrement symétrique, qui est elle même stockée sur le disque, chiffrée à l'aide du mot de passe utilisateur.
- Montages temporaires : des montages de systèmes de fichiers en mémoire (de type *tmpfs*), permettant de stocker les données temporaires de l'utilisateur, sans jamais les écrire en clair sur le disque dur, et en assurant leur disparition au démontage du système de fichiers. On notera que le *swap* sur le disque du contenu de ces systèmes de fichiers est possible, et ne pose pas de problème de sécurité dans la mesure ou le *swap* lui même est chiffré.

Ces différents systèmes de fichiers sont montés automatiquement à l'ouverture de session, et démontés à la fermeture de session. Ils sont définis de telle sorte que l'utilisateur d'une session "normale"[4] ne puisse (au regard des permissions discrétionnaires) écrire nulle part ailleurs dans son arborescence de fichiers que sur ces montages de session. Il en résulte qu'aucune donnée écrite ou lue par l'utilisateur ne peut subsister en clair sur le disque du poste une fois la session de cet utilisateur terminée.

Ces différents montages, ainsi que diverses opérations complémentaires à l'ouverture et à la fermeture de session, sont réalisés sous CLIP par un module PAM spécifique, *pam_exec_pwd*.

2.1 Principe du chiffrement de partitions utilisateur CLIP

Les partitions chiffrées d'utilisateurs CLIP sont réalisées à l'aide de systèmes de fichiers projetés par *dm-crypt* ([DMCRYPT]) depuis des *devices loop* associés à des fichiers de */home*. Les partitions virtuelles claires correspondantes sont formatées en *ext2*[5].

Le chiffrement *dm-crypt* est réalisé en AES-256, dans le mode LRW ([NMHDE], [IEEE1619]), en utilisant les primitives cryptographiques du noyau Linux. Les vecteurs d'initialisation (IV) (c'est-à-dire les 'index' du mode LRW) sont générés selon le mode BENBI, c'est-à-dire en prenant l'index depuis le début de la partition en commençant à 1, codé en *big endian* sur 64 bits, du bloc de clair à chiffrer, soit :

$$idx = be64\left(\left[numéro\ secteur\right] \times \frac{512}{128} + \frac{\left[offset\ bloc\ dans\ secteur\right]}{128} + 1\right)$$

et en stockant cet index sur les 64 derniers octets de l'IV, les 64 premiers octets étant laissés à zéro. La

[4] Cette propriété s'applique uniquement aux utilisateurs de la cage $USER_{clip}$, et des vues RM des éventuelles cages RM. Les utilisateurs des cages $ADMIN_{clip}$ et $AUDIT_{clip}$ en revanche sont en mesure d'écrire sur des montages de plus longue durée de vie dans leurs cages respectives.

[5] L'utilisation d'un système de fichiers journalisé, comme *ext3*, n'est pas recommandée sur une partition virtuelle montée ou projetée depuis un fichier. En effet, la journalisation repose, pour sa fiabilité, sur des hypothèses de séquencement des opérations qui ne sont respectée que lors de l'écriture directe sur un disque physique, et pas lorsque l'écriture sur la partition virtuelle entraîne une écriture potentiellement retardée sur un autre fichier.

clé utilisée pour ce chiffrement a une taille de 384 bits : 256 bits pour le chiffrement AES-256, et 128 pour le *"tweak"* LRW.

La clé de chaque partition est générée aléatoirement pour chaque partition lors de la création d'un compte utilisateur, par lecture de */dev/urandom*. Afin de fournir les 384 bits de clés, la lecture de 24 caractères aléatoires serait théoriquement suffisante. Cependant, dans la mesure où cette clé doit être manipulée par des scripts *shell*, il est nécessaire de se limiter à des caractères imprimables, c'est-à-dire à ceux qui valident *isgraph(3)*. On réduit ainsi le nombre de valeurs possibles pour un caractère à 94 au lieu de 256, soit environ 6,55 bits d'entropie par caractère au lieu de 8. Afin de conserver réellement au moins 384 bits d'entropie pour la clé de chiffrement, la clé lue sur */dev/urandom* est une chaîne de 119 caractères imprimables, soit un peu plus de 768 bits d'entropie. La clé de 384 bits effectivement passée à *dm-crypt* est dérivée de cette clé "imprimable" par l'utilitaire *cryptsetup-luks* utilisé pour créer les projections chiffrées. Cette dérivation de clé repose sur l'algorithme *sha256* implémenté dans la bibliothèque *gcrypt*, utilisée de la manière suivante (en pseudo-code) :

```
char key[klen]
char passwd[119]

for (n = 0; n < klen / 256; n++)
      key[n*256 -> (n+1)*256] = SHA256("AA....A" + passwd)

(avec "AA...A" une chaîne de n caractères 'A')
```

La clé "imprimable" est stockée sur le disque chiffrée à l'aide du mot de passe utilisateur. L'algorithme utilisé pour ce chiffrement est l'AES-256 en mode CBC implémenté par *openssl*. Un chiffrement direct en utilisant le mot de passe utilisateur est à éviter, car il permettrait de tester un mot de passe par un simple déchiffrement AES de cette clé, beaucoup plus rapide que le hachage *bcrypt* utilisé pour protéger l'empreinte du mot de passe utilisateur stockée dans le fichier *shadow* de l'utilisateur. Une recherche par force brute des mots de passe utilisateurs pourrait dans ce cas être lancée sur les clés de chiffrement de partitions noircies plutôt que sur les empreintes de mot de passe, et donner des résultats très rapidement.

Afin d'éviter une telle attaque, la clé utilisée pour le chiffrement AES des clés de chiffrement de partition est une empreinte *bcrypt* du mot de passe utilisateur, calculée avec le même facteur de coût que l'empreinte stockée dans le fichier *shadow* de cet utilisateur, mais avec un sel différent. Pour ce faire, lors de la création d'un utilisateur, ou de la modification du mot de passe d'un utilisateur existant, un sel de 128 bits (différent de celui utilisé pour l'empreinte *shadow*) est généré pour chaque clé de partition de l'utilisateur, par lecture de */dev/urandom*[6]. Ce sel est utilisé pour hacher avec *bcrypt* le mot de passe de l'utilisateur. L'empreinte résultante (sous sa forme complète, comprenant le sel, le coût, et 184 bits de "secret" correspondant au hachage du mot de passe) est passée à *openssl*, qui en dérive la clé AES-CBC à utiliser pour chiffrer la clé de partition. Le sel est ensuite stocké en clair au côté de la clé de partition chiffrée, ce qui permet de déchiffrer celle-ci à partir du mot de passe, en régénérant l'empreinte *bcrypt* avec le même sel.

[6] Ce sel est lu directement sous une forme encodée en *base64*, en lisant 22 caractères compatibles avec la *base64* sur */dev/urandom*. Ainsi, le sel peut être manipulé sans problème par un script *shell*.

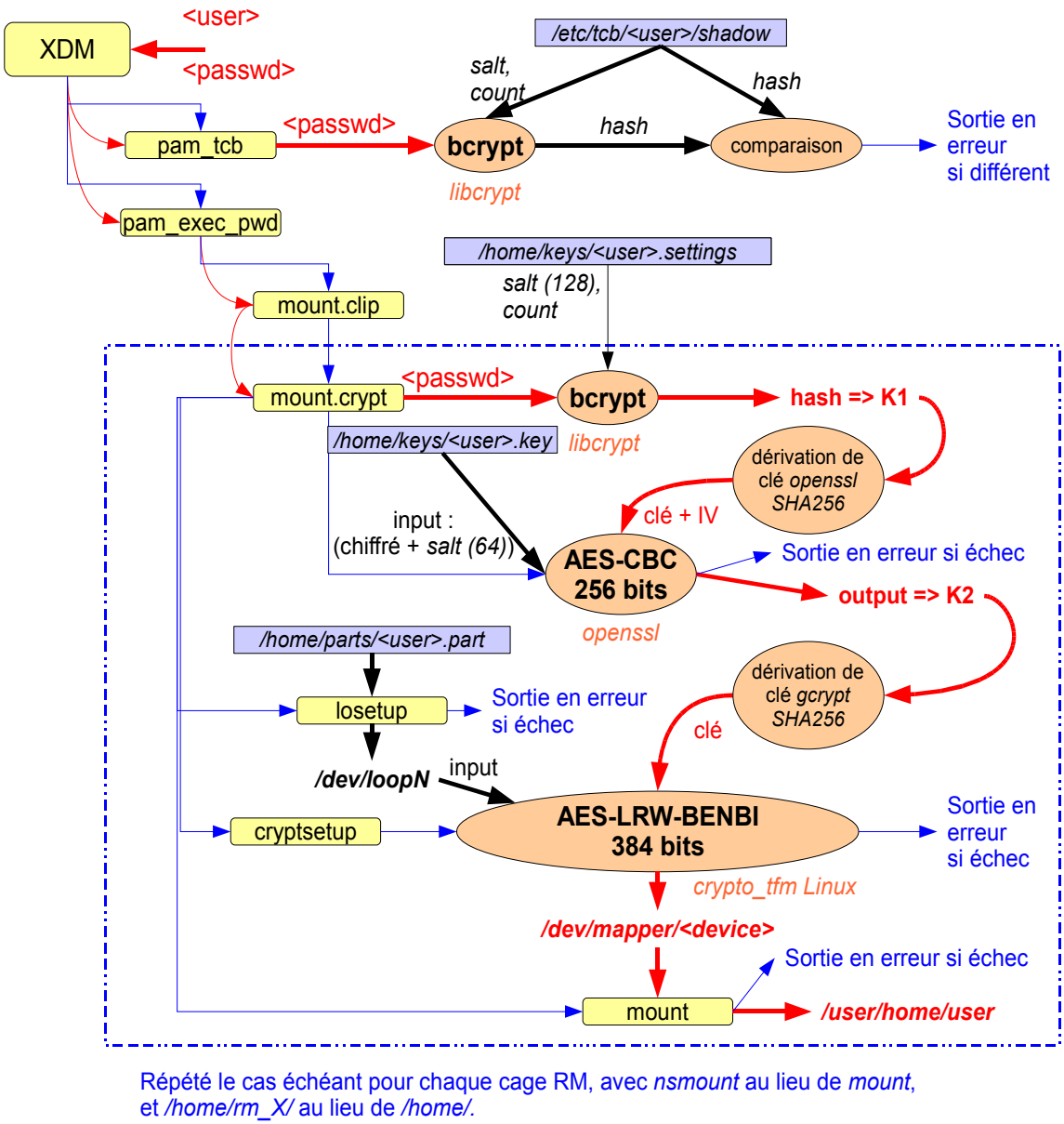

Figure 1: Authentification CLIP : vérification du mot de passe et montage des partitions chiffrées.

La clé et l'IV de chiffrement AES-CBC sont dérivés de l'empreinte par *openssl* à l'aide de sa fonction *EVP_BytesToKey()*, (*crypto/evp/evp_key.c* dans les sources *openssl*, documentée dans *doc/ssleay.txt*), en utilisant *SHA256* comme fonction de hachage. Vu les tailles respectives des clés et des hachés, cette dérivation réalise en pratique les opérations suivantes :

$$K_{AES} = SHA256 (H + S)$$
$$IV_{AES} = SHA256 (K_{AES} + H + S)$$

en notant K_{AES} la clé de chiffrement, IV_{AES} l'IV, H l'empreinte *bcrypt* du mot de passe et S un sel de taille

compatible PKCS#5 (64 bits, cf. [RFC2898]) généré aléatoirement (par lecture sur */dev/urandom*) lors du chiffrement et stocké avec le chiffré de la clé. Ce sel est inscrit en tête du "chiffré" de la clé. Par ailleurs, *openssl* ajoute à la clé chiffrée un *padding* compatible avec la méthode PBES1 décrite dans [RFC2898]. Ce *padding* est ajouté même lorsque le clair a une longueur multiple de la taille de bloc, afin de faciliter la détection d'erreur. En particulier, dans le cas des clés AES+LRW de 384 bits, le *padding* ajouté avant le chiffrement AES consiste en 16 octets, contenant chacun la valeur 16.

L'ensemble de ces transformations est résumé dans la Figure 1.

2.2 Module PAM *pam_exec_pwd*

Les montages de session, qu'il s'agisse de partitions chiffrées ou de systèmes de fichiers temporaires, sont tous réalisés par un module PAM (cf. [PAM_MWG]) spécifiquement développé pour CLIP, *pam_exec_pwd*. Ce module permet l'exécution de commandes à l'ouverture et la fermeture de session utilisateur, en passant optionnellement le mot de passe de l'utilisateur à certaines des commandes lancées à l'ouverture de session.

Le module définit deux services PAM, qui peuvent être inclus dans une pile de configuration PAM :

- Un service d'authentification (*'auth'*), qui n'exécute aucune commande, ni interaction avec l'utilisateur, mais se contente de sauvegarder le mot de passe utilisateur dans le contexte PAM courant. Ce service doit obligatoirement être inclus dans toute pile PAM qui utilise par ailleurs le service de session *pam_exec_pwd*, en passant le mot de passe utilisateur à certaines des commandes lancées par celui-ci.
- Un service de session (*'session'*), qui est appelé lors de l'ouverture et de la fermeture de session, et réalise à cette occasion l'exécution des commandes définies dans son fichier de configuration.

Le service d'authentification n'utilise aucun fichier de configuration. Le service de session, lorsqu'il est appelé (aussi bien à l'ouverture qu'à la fermeture de session), lit le fichier de configuration */etc/security/exec.conf*. Ce dernier contient une liste de commandes à exécuter, à raison d'une par ligne selon le format suivant :

```
<condition>  <drapeaux>  <commande>  [arguments]
```

avec :

- <*condition*> une condition sur l'utilisateur ouvrant ou fermant la session, qui définit les utilisateurs pour lesquels cette commande est exécutée. Cette condition peut être de l'une des trois formes suivantes :
  - un nom <*nom*>, ne commençant pas par les caractères '!' ni '@' : cette condition n'est vérifiée que si le nom du compte utilisateur de la session est <*nom*>.
  - un nom précédé d'un '@', @<*nom*> : cette condition est vérifié par les utilisateurs membres du groupe <*nom*> (au sens du *gid* ou des groupes supplémentaires).
  - une des deux formes précédentes, précédées d'un '!', *!*<*condition*> : cette condition est vérifiée par les utilisateurs qui ne vérifient pas <*condition*>.
- <*drapeaux*> une chaîne de drapeaux, définissant la manière dont la commande est exécutée, lorsque <*condition*> est vérifiée. Ces drapeaux sont exprimés par une chaîne de lettres, choisies parmi les suivantes:

- *'o'* : la commande est lancée à l'ouverture de session.
- *'c'* : la commande est lancée à la fermeture de session. Au moins une des deux lettres, *'o'* ou *'c'*, doit être présente sur chaque ligne. Les deux lettres peuvent être présentes simultanément, auquel cas la commande sera exécutée aussi bien à l'ouverture de session qu'à la fermeture de session.
- *'u'* : la commande est lancée sous l'identité (*uid, gid*) de l'utilisateur de la session, plutôt que sous l'identité du processus qui ouvre la session et fait appel au module PAM (c'est-à-*dire root* en général).
- *'p'* : la commande est lancée en lui passant le mot de passe de l'utilisateur à travers la variable d'environnement *PASSWD*. Cette option n'est supportée qu'à l'ouverture de session, dans la mesure où le mot de passe utilisateur n'est pas disponible dans l'environnement PAM lors de la fermeture de session. Ainsi, elle n'est acceptée que sur les lignes comportant le drapeau *'o'*, et pas le drapeau *'c'*. Par ailleurs, son utilisation nécessite l'inclusion du service *auth* de *pam_exec_pwd* dans la pile PAM d'authentification. Dans le cas contraire, le service de session sort en erreur sur toute commande *'p',* du fait de son incapacité à lire le mot de passe utilisateur dans l'environnement PAM.

- <*commande*> est la commande à lancer, sous la forme d'un chemin absolu complet d'exécutable.
- <*argument*> un argument optionnel à passer à la commande. Par mesure de simplicité, un unique argument (sans espace) est supporté. Lorsqu'il est présent, cet argument est passé sur la ligne de commande de <*commande*>, c'est-à-dire comme *argv[1]* de l'appel *execve()*.

Indépendamment de la définition ou non de *PASSWD*, toutes les commandes sont lancées avec une variable d'environnement *USER* positionnée au nom de l'utilisateur pour lequel une session est en cours d'ouverture ou de fermeture.

Le service de session lit */etc/security/exec.conf* ligne par ligne, en lançant successivement les commandes pour lesquelles les conditions d'exécution sont satisfaites. Le module attend le retour de chaque commande avant de passer à la lecture de la ligne suivante. Il interrompt entièrement son traitement dès qu'une erreur de syntaxe est détectée dans une ligne du fichier de configuration, ou dès qu'une commande lancée renvoie un code d'erreur non nul. Ce dernier point peut être ajusté, pour la fermeture de session uniquement, par une option *close_run_all* du module. Lorsque ce mot clé est présent sur la ligne *session pam_exec_pwd* d'un fichier de configuration de pile PAM, le module exécute toutes les commandes qui doivent être lancées lors d'une fermeture de session, même si certaines de ces commandes rendent un code d'erreur non nul. Le comportement à l'ouverture de session n'est pas changé dans ce cas.

Le module journalise ses traitements à travers l'interface *syslog*, dans la *facility AUTHPRIV*, et avec le niveau de priorité *LOG_ERR* pour les erreurs et *LOG_INFO* pour les autres messages. Un message est en particulier journalisé au niveau *LOG_INFO* en cas de traitement correct à l'ouverture ou à la fermeture de session. Ces messages sont consultables sur CLIP dans le fichier */log/auth.log* de la cage $\text{AUDIT}_{\text{clip}}$.

2.3 Mise en oeuvre de *pam_exec_pwd*

Le module *pam_exec_pwd* est appelé par la configuration PAM du démon *xdm* (cf. 3) pour fournir les services *auth* et *session*. Il lance ainsi les commandes définies dans */etc/security/exec.conf* lors de l'ouverture et de la fermeture de session. Ces commandes assurent la création des montages de session, ainsi qu'un certain nombre d'actions complémentaires. L'option *close_run_all* est passée au module, afin de garantir l'exécution de toutes les commandes de fermeture de session.

On notera que, au même titre que le démon *xdm* qui fait appel à *pam_exec_pwd,* toutes les commandes ainsi lancées le sont initialement dans le socle CLIP. Certaines de ces commandes peuvent néanmoins s'enfermer dans une cage après leur lancement.

2.3.1 Montage et démontage de partitions chiffrées

Montage

Les montages de session sont configurés par un script unique, */sbin/mount.clip* (*app-clip/clip-user-mount*), qui est appelé sous l'identité *root* et avec l'argument *"mount"* lors de toute ouverture de session par un utilisateur membre du groupe *crypthomes*, et lancé sous l'identité *root* avec le mot de passe utilisateur passé dans la variable d'environnement *PASSWD*. Ce script lit un fichier de configuration unique, */etc/fstab.session*, qui définit l'ensemble des montages à réaliser pour un utilisateur de ce type, à raison d'un montage par ligne selon le format :

```
<type> <cage> [<source> [<clé>]] <destination> <options> <mkdir> <namespace>
```

avec:

- <*type*> le type de montage, parmi *crypt* (montage de partition chiffrée CLIP, selon les principes décrits en 2.1), *bind* (montage *bind*) et *tmp* (montage *tmpfs*)
- <*cage*> la cage destination dans laquelle le montage doit être réalisé. Les valeurs supportées sont les noms des cages tels qu'ils apparaissent dans les sous-répertoires de */etc/jails*, notamment *rm_h* (RM_H sur un poste CLIP-RM), *rm_b* (RM_B) et *user* ($USER_{clip}$).
- <*source*> la source du montage. Ce paramètre n'apparaît que pour les types de montages pour lesquels il a un sens, c'est-à-dire les montages *bind* et *crypt* (pour lesquels il représente le fichier image chiffré associé à un montage).
- <*clé*> la clé du montage. Ce paramètre n'apparaît que pour les montages de type *crypt*, pour lesquels il correspond au chemin du fichier contenant la clé de chiffrement de partition, elle même chiffrée avec le mot de passe utilisateur (cf. 2.1). Un chemin de type */chemin/vers/fichier.key* est attendu, tandis que les paramètres de hachage du mot de passe utilisateur en vue de déchiffrer la clé sont cherchés automatiquement dans le fichier */chemin/vers/fichier.settings*.
- <*destination*> le chemin du point de montage. Ce chemin est relatif soit à la racine du socle CLIP (lorsque <*namespace*> est nul), soit à la racine de la cage concernée.
- <*options*> les options de montage. Lorsque <*namespace*> est non nul, ces options doivent être supportées par l'utilitaires *nsmount* (cf. [CLIP_1202]).
- <*mkdir*> un paramètre défini à *"yes"* ou *"no"*. Lorsque ce paramètre vaut *"yes"*, le point de

montage est créé par un *mkdir* s'il n'existe pas lors de l'appel au script. Dans le cas contraire, l'inexistence d'un point de montage entraîne la sortie en erreur du script. La création de répertoire est réalisée dans la cage concernée, à l'aide d'un appel *vsctl enter*.

- <*namespace*> un paramètre défini à *0* ou *1*. Lorsqu'il vaut 1, le montage est réalisé directement dans le *namespace VFS* de la cage concernée (qui doit avoir été créée au préalable), à l'aide d'une commande *nsmount*[7]. A contrario, lorsque le paramètre vaut 0, le montage est crée dans le *namespace VFS* du socle. Cette seconde approche est adaptée au cas de cages qui ne sont pas actives en permanence, et qui ne sont créées qu'après l'appel du script, ce qui est typiquement le cas de la cage $USER_{clip}$.

Le script *mount.clip* fait appel à un script spécifique pour chaque type de montage : */sbin/mount.crypt* pour les montages *crypt*, */sbin/mount.tmp* pour les montages *tmp* et */sbin/mount.bind* pour les montages *bind*. Ces scripts sont appelés successivement sur chaque ligne de *fstab.session,* en leur passant en argument l'ensemble de la ligne, sauf le premier champ qui n'est exploité que par le script "maître". Par ailleurs, le script *mount.clip* ignore automatiquement les lignes de */etc/fstab.session* qui font référence à une cage autre qu'une cage CLIP, lorsque l'utilisateur en train d'ouvrir une session appartient à un profil administrateur ou auditeur CLIP (groupes *core_admin* et *core_audit*, respectivement).

Le script *mount.clip,* ainsi que tous les scripts secondaires qu'il appelle, journalisent leurs actions à travers l'interface *syslog* (utilisée à l'aide de l'utilitaire *logger*), dans la *facility LOG_USER*, et avec les niveaux *LOG_INFO* pour les succès (un message est en particulier généré pour chaque montage ou démontage réussi), et *LOG_ERR* pour les erreurs. Ces messages sont consultables sous CLIP dans le fichier */log/messages* de la cage AUDITindclip.

Fichiers *mtab*

Chaque montage ainsi réalisé est inscrit dans un fichier de type *mtab* spécifique à chaque cage, stocké dans */var/run/<cage>.mtab* (avec <*cage*> le même paramètre que sur la ligne de *fstab.session.* Ce fichier recense tous les montages temporaires réalisés dans la cage pour une session donnée[8], sous la forme d'un montage par ligne (dans l'ordre chronologique de montage) avec le format suivant :

```
<source>  <destination>  <type>  <options>  [<namespace>]
```

avec

- <*source*> la source du montage, potentiellement différente de celle apparaissant dans la ligne correspondante de *fstab.session*. Par exemple, pour un montage de type *crypt*, la source qui apparaîtra dans le fichier *mtab* sera de la forme */dev/mapper/<device>*, alors qu'elle sera de la forme */home/parts/<user>.part* par exemple dans *fstab.session*.
- <*destination*> la destination du montage. Contrairement à la destination qui apparaît dans *fstab.session*, celle-ci est toujours exprimée comme un chemin absolu par rapport à la racine de la cage concernée, même lorsque ce montage a initialement été créé dans le *namespace* du socle CLIP.
- <*type*> le type du montage. Il s'agit du type réel du système de fichiers monté, plutôt que du type "virtuel" de *fstab.session*. Par exemple, un montage *crypt* aura ici un type *ext2*.

[7] Au besoin, en copiant temporairement le *device* associé au montage (*device dm-crypt*, dans le cas d'un montage *crypt*) dans le */dev* de la cage par un appel *tar*.

[8] C'est-à-dire les montages créés par *pam_exec_pwd* lors de l'ouverture de session, mais aussi d'éventuels montages de supports amovibles réalisés dans la cage au cours de la session, cf. [CLIP_DCS_15088].

- <*options*> les options de montage.
- <*namespace*> un champ supplémentaire optionnel, qui est défini (à une valeur arbitraire) uniquement lorsque le montage a été réalisé à l'origine dans le *namespace VFS* du socle (et devra par conséquent être démonté dans ce *namespace*, en plus de celui de la cage).

L'accès à chaque *mtab* est protégé par un verrou *flock(2)* (verrou BSD) sur le fichier, afin d'interdire les accès concurrents en écriture sur le fichier. Ce verrou est automatiquement posé par *nsmount* lorsque les opérations de montage/démontage sont réalisées à l'aide de cet utilitaire (cf. [CLIP_1202]). Lorsque *mount* est employé, la manipulation du fichier *mtab* est réalisée séparément, en protégeant les accès par des commandes *flock(1)*.

Les différents fichiers *mtab* de montages de session sont testés à l'ouverture de session par le script *mount.clip*. Celui-ci sort immédiatement en erreur, sans réaliser un seul montage dans la cage concernée, lorsqu'un fichier *mtab* est non vide (ce qui signifie a priori que certains montages de la session précédente n'ont pas pu être démontés).

Démontage

Le démontage des montages de session est réalisé par le script *mount.clip,* appelé avec l'argument *umount,* sous l'identité *root,* lors de la fermeture de session *xdm* de tout utilisateur membre du groupe *crypthomes*. Ce script ce contente dans ce cas de lancer un deuxième script, *umount.clip,* pour chaque cage susceptible de contenir des montages de session. La liste de ces cages est inscrite dans le script *mount.clip* lui-même, et définie à ce stade comme *admin, audit* et *user*, complétées de *rm_h* et *rm_b* dans le cas d'un poste CLIP-RM.

Lorsqu'il est invoqué pour une cage <*cage*> donnée, le script *umount.clip* lit le fichier *mtab* de session de cette cage, et procède au démontage du montage inscrit sur chaque ligne. Le fichier *mtab* est lu dans l'ordre inverse de sa création (c'est-à-dire ligne par ligne, mais du bas vers le haut), de manière à réaliser les démontages dans l'ordre chronologique inverse des montages, ce qui permet de supporter des montages interdépendant (par exemple, montage de */tmp* puis de */tmp/.X11-unix*). Pour chaque montage, le script procède systématiquement au démontage dans la cage concernée, en réalisant les opérations suivantes :

- Test d'activité de la cage. Si la cage entière s'est terminée, il n'est pas nécessaire de procéder au démontage dans celle-ci, et les opérations ci-dessous ne sont pas réalisées.
- Terminaison des processus de la cage qui utilisent le montage, en lançant *fuser* dans la cage pour récupérer la liste de ces processus, puis en les terminant par des appels *kill* (lancés dans la cage sous l'identité *root*).
- Nouveau test d'activité de la cage (qui peut avoir été terminée par un effet de bord de l'opération précédente). Si la cage est terminée, les opérations ci-dessous ne sont pas réalisées.
- Démontage du montage par un appel *nsmount,* en supprimant en cas de succès la ligne correspondante du fichier *mtab* grâce à l'option *-m* de *nsmount.*
- En cas d'échec de *nsmount*, les quatre opérations précédentes sont répétées au maximum douze fois, avec des délais de 0.5 seconde (deux premières tentatives), puis 1 seconde. Le signal envoyé par la deuxième opération est un *SIGTERM* lors de la première tentative, et un *SIGKILL* lors des suivantes.

Lorsque le montage ainsi démonté avait initialement été réalisé dans le socle CLIP, c'est-à-dire lorsque

le champ <*namespace*> de la ligne *mtab* est défini, le script procède ensuite au démontage dans le socle, par les opérations suivantes :

- Terminaison des processus du socle qui utilisent le montage, par un appel *fuser -km*.
- Démontage sans modifier le *mtab*, par un appel *umount*.
- En cas d'échec de *umount*, les deux opérations précédentes sont répétées au maximum six fois à des intervalles de 0.5 secondes (en utilisant un signal *SIGTERM* pour la première, et *SIGKILL* pour les suivantes).

Enfin, le script réalise la suppression des éventuelles projections *dm-crypt* et *loop* associées au montage. Lorsque la source du montage (extraite du fichier *mtab*) est reconnue par le *device-mapper* (c'est-à-dire par une commande *cryptsetup status*), la projection correspondante est supprimée par un appel *cryptsetup remove*. Par ailleurs, lorsque la source de cette projection *device-mapper* (donnée par *cryptsetup status*) est elle-même de la forme */dev/loopX*, la projection *loop* correspondante est aussi supprimée par un appel *losetup -d*.

Remarque 3 : Démontage dans le socle et mise à jour du mtab

La suppression d'une ligne du fichier mtab *d'une cage est effectuée dès que le montage correspondant est supprimé dans la cage. Dans le cas d'un montage réalisé à l'origine dans le socle, le démontage dans le socle peut néanmoins échouer après cette suppression, et ainsi ne pas être tracé dans le fichier* mtab. *Une solution permettant d'assurer la traçabilité d'un tel montage serait préférable.*

Montages réalisés

Le paramétrage d'un poste CLIP impose systématiquement les montages suivants pour tous les utilisateurs membres du groupe *crypthomes* :

- Un montage chiffré sur */user/home/user* dans le *namespace* du socle. Ce montage correspond au *$HOME* de tous les utilisateurs de la cage $USER_{clip}$. Il est réalisé depuis le fichier image */home/parts/<user>.part* dans l'arborescence du socle (avec <*user*> le nom de l'utilisateur), en utilisant */home/keys/<user>.key* comme clé, chiffrée avec l'empreinte *bcrypt* du mot de passe utilisateur, elle-même générée avec les paramètres (coût, sel) définis dans */home/keys/<user>.settings*. Le principe de ce montage est résumé dans la Figure 1.
- Un montage de type *tmpfs* sur */user/tmp* dans le *namespace* du socle, afin d'éviter la rémanence en fin de session d'informations temporaires de l'utilisateur de $USER_{clip}$.
- Un montage de type *bind* en lecture seule de */tmp/.X11-unix* sur */user/tmp/.X11-unix* dans le *namespace* du socle. Ce montage permet d'exposer la *socket unix* du serveur X11 dans $USER_{clip}$.

On notera qu'aucun montage n'est réalisé à l'ouverture de session dans les cages $ADMIN_{clip}$ et $AUDIT_{clip}$. Cependant, le script de démontage *umount.clip* est tout de même lancé pour chacune de ces deux cages lors de la fermeture de session, afin de démonter d'éventuels supports USB. Ces différents montages sont détaillés dans le document [CLIP_1304].

Par ailleurs, dans un système de type CLIP-RM, deux montages de session supplémentaires s'ajoutent pour chaque cage RM *rm_X* :

- Un montage chiffré sur */user/home/user* dans le *namespace* de la cage *rm_X*. Ce montage

correspond au *$HOME* de tous les utilisateurs de la vue USER de la cage. Il est réalisé depuis le fichier image */home/rm_X/parts/<user>.part* dans l'arborescence du socle (avec *<user>* le nom de l'utilisateur), en utilisant */home/rm_X/keys/<user>.key* comme clé, chiffrée avec l'empreinte *bcrypt* du mot de passe utilisateur, elle-même générée avec les paramètres (coût, sel) définis dans */home/rm_X/keys/<user>.settings*.

- Un montage de type *tmpfs* sur */user/tmp* dans le *namespace* de la cage.

Ces montages, ainsi que les éventuels montages USB qui les complètent, sont décrits plus en détail dans le document [CLIP_1401].

2.3.2 Autres commandes lancées par *pam_exec_pwd*

Le module *pam_exec_pwd* est par ailleurs utilisé pour lancer plusieurs autres commandes lors de l'ouverture et de la fermeture de session *xdm*.

Interdiction des utilisateurs sans montages chiffrés

Le script */usr/bin/fail-login* (*app-clip/clip-user-mount*) est lancé en premier lieu lors d'une ouverture de session *xdm*, pour les utilisateurs qui ne sont pas membres du groupe *crypthomes* (condition *! @crypthomes*). Ce script se contente de journaliser (dans la *facility auth*, avec la priorité *LOG_ERR*) un message signalant qu'une ouverture de session interdite a été bloquée, avant de se terminer avec un code d'erreur 1, ce qui interrompt le traitement par *pam_exec_pwd*, et l'ouverture de session en général.

Affichage d'un message d'ouverture de session

Le script */usr/bin/xdm-issue* est lancé sous l'identité *root* à l'ouverture de session *xdm* pour tous les utilisateurs membres du groupe *crypthomes*. Ce script permet d'afficher un message dans une fenêtre graphique, qui doit être validé par l'utilisateur afin de procéder à l'ouverture de session. Il teste la présence et la taille du fichier */etc/admin/issue* (modifiable par l'administrateur local, cf. [CLIP_1304]). Lorsque ce fichier est présent et non vide, son contenu (textuel) est affiché à l'écran dans une fenêtre *Xdialog*, proposant deux choix, "Accepter" / "Refuser". Si l'utilisateur choisit de refuser le texte ainsi affiché, l'ouverture de session est interrompue. S'il l'accepte, le message disparaît et l'ouverture se poursuit.

Suppression des projections *dm-crypt*

Le script */sbin/dmcleanup* est lancé sous l'identité *root* à la fermeture de session *xdm* pour tous les utilisateurs membres du groupe *crypthomes*. Il réalise le démontage de toutes les projections *device-mapper* autres que celle associée au *swap* chiffré du poste, par des appels *cryptsetup remove*. On notera que ce script n'est appelé qu'après le script *mount.clip umount*, qui se charge normalement de supprimer les projections associées à des montages actifs. Le rôle de *dmcleanup* est donc principalement de supprimer les projections qui ne sont pas montées lors de la fermeture de session, ce qui est typiquement le cas d'un support amovible qui a été connecté (ce qui entraîne sa projection par *dm-crypt*), mais jamais monté, ou monté puis démonté avant la fin de la session, mais pas déconnecté. Ainsi, un support USB chiffré qui aurait été laissé connecté au poste par son propriétaire à la fin de la session de ce dernier ne sera pas exposé en clair aux utilisateurs suivants du même poste.

3 Configuration PAM d'un système CLIP

Cette section détaille la configuration des piles de modules PAM (cf. [PAM_SAG]) associées aux différents services d'un système CLIP, telles qu'elles sont définies dans les fichiers de */etc/pam.d*. L'essentiel de la configuration PAM d'un système CLIP est fourni dans une pile générique *system-auth*, qui est incluse par la plupart des autres fichiers.

Configuration system-auth

La configuration */etc/pam.d/system-auth* n'est associée à aucun service en particulier, mais est incluse par la plupart des configurations de services du système. Elle active en particulier l'utilisation de TCB pour l'authentification. Plus précisément, cette configuration utilise les modules suivants :

- Pour l'authentification :
  - Module *pam_env.so*, *required*. Ce module standard permet de définir ou de supprimer des variables d'environnement dans la session d'un utilisateur. Il n'est pas utilisé à ce stade dans CLIP, le fichier de configuration */etc/security/pam_env.conf* qui lui est associé restant vide. Le module est néanmoins inclus dans la configuration afin de permettre facilement des utilisations futures.
  - Module *pam_tcb.so*, *required*. Ce module réalise l'authentification par mot de passe pour les différents services du système. Sa configuration est détaillée en 1.2.2.
- Pour la gestion de compte (*account*), uniquement le module *pam_tcb.so* (*required*).
- Pour la gestion de mot de passe :
  - Le module *pam_cracklib.so* (*required*), permettant de contrôler la qualité des nouveaux mots de passe. On notera que le changement de mot de passe des utilisateurs CLIP ne fait pas appel à l'authentification PAM, ce qui signifie que cet appel à *pam_cracklib* n'est pas significatif au regard de la qualité des mots de passe CLIP. Voir aussi 1.4.
  - Le module *pam_tcb.so* (*required*) avec l'argument supplémentaire *use_authtok* lui permettant de récupérer le mot de passe saisi par *pam_cracklib*.
- Pour la gestion de session :
  - Le module *pam_limits* (*required*), qui permet de limiter les ressources (*rlimits*) accordées à un utilisateur. Il n'est pas utilisé à ce stade dans CLIP, le fichier de configuration */etc/security/limits.conf* qui lui est associé restant vide. Le module est néanmoins inclus dans la configuration afin de permettre facilement des utilisations futures.
  - Le module *pam_tcb.so* (*required*).

Cette configuration est notamment utilisée telle qu'elle (réduite aux services *auth, account* et *password*, sans *session*) pour le service *password*. On notera que ce dernier n'est normalement pas utilisé sous CLIP pour changer le mot de passe d'un utilisateur (*usermod* est utilisé à la place, en conjonction avec un appel à *cryptpasswd*).

Configuration other

La configuration *other* est importante, dans la mesure où elle est celle utilisée par défaut lorsque aucun fichier de configuration n'est trouvé pour un service faisant appel à PAM. Dans CLIP, elle est définie de manière complètement restrictive, et réduite à *pam_deny.so* (*required*) pour les quatre services *auth, account, password* et *session.* Cette configuration entraîne un refus systématique des opérations PAM pour lesquelles aucun fichier de configuration dédié n'est présent dans le système.

Configuration xdm

Cette configuration est celle utilisée pour l'authentification principale et l'ouverture de session des utilisateurs finaux du système, à travers la fenêtre d'accueil XDM. Elle inclut *system-auth* pour les quatre services, mais y ajoute par ailleurs :

- Pour l'authentification, après les modules de *system-auth* :
  - Le module *pam_exec_pwd* décrit en 2 (*required*). Cette inclusion du module dans le service d'authentification lui permet de sauvegarder le mot de passe utilisateur dans l'environnement PAM, afin de le transmettre aux commandes exécutées à l'ouverture de session. On notera que ce mot de passe a déjà été vérifié par *pam_tcb* lors de sa sauvegarde.
  - Le module standard *pam_nologin.so* (*required*) permettant d'interdire l'authentification de certains utilisateurs. Il n'est pas utilisé à ce stade dans CLIP, le fichier de configuration */etc/nologin* qui lui est associé étant absent du système. Le module est néanmoins inclus dans la configuration afin de permettre facilement d'éventuelles utilisations futures.
- Pour la gestion de session, après les modules :
  - Le module *pam_exec_pwd*, avec l'option *close_run_all*.

Authentification sur la console

La configuration *login* est utilisée pour l'authentification et l'ouverture de session sur la console. On notera bien qu'elle n'est à ce stade pas mise en oeuvre sur un poste CLIP (hors configurations de test et de développement), dans la mesure où aucun gestionnaire de session (*getty*) n'est lancé par *init*. Cependant, cette configuration est maintenue afin de faciliter d'éventuels développements futurs qui utiliseraient la console (en y ajoutant au besoin *pam_jail,* cf. [CLIP_1202]). Cette configuration inclut *system-auth*, mais y ajoute :

- Pour l'authentification, avant les modules de *system-auth* :
  - Le module *pam_securetty* (*required*), pour limiter l'accès de *root* aux consoles locales. Le fichier de configuration associé, */etc/securetty*, limite dans CLIP cet accès aux six premières consoles locales (*tty1* à *tty6*).
  - Le module *pam_shells* (*required*) limite cette authentification aux utilisateurs auxquels est associé un *shell* de *login* valide (c'est-à-dire mentionné dans */etc/shells,* qui est limité à */bin/sh* et */bin/bash* à ce stade dans CLIP).
  - Le module *pam_nologin* (*required*) permet d'interdire sélectivement et temporairement certains utilisateurs.

- Pour la gestion de compte (*account*), avant *system-auth* :
  - Le module *pam_access* (*required*), qui permet de limiter les droits d'ouverture de session en fonction du contenu du fichier */etc/security/access.conf*. Sous CLIP, ce fichier fourni par *sys-libs/pam* interdit à ce stade toute ouverture de session locale.
- Pour l'ouverture de session, avant *system-auth* :
  - Le module *pam_env* (*required*) pour ajuster l'environnement des utilisateurs (inutilisé à ce stade).
  - Le module *pam_lastlog* (*optional*) pour afficher l'heure et la date de la dernière connexion de l'utilisateur.
  - Le module *pam_motd* (*optional*) pour afficher le contenu d'un éventuel fichier */etc/motd*.

Configuration *pwcheckd*

Cette configuration est utilisée par le démon *pwcheckd,* qui permet en particulier la ré-authentification d'un utilisateur pour déverrouiller sa session X11 (cf. 4.3). Cette configuration inclut *system-auth* uniquement pour le service d'authentification, et complète ce service par un service *account* autorisant à ce stade toutes les opérations (service réduit au module *pam_permit*).

Configuration générique

Une configuration standard commune est utilisée pour les autres services supportés. Cette configuration est conçue de manière à limiter les opérations pour lesquelles elle est invoquée à l'utilisateur *root*. Ainsi, ses services *auth* et *password* incluent *pam_rootok* en mode *sufficient*, afin d'autoriser systématiquement l'*uid* 0, puis *pam_deny* en mode *required* de manière à interdire tous les autres accès. Son service *account* est limité à celui de *system-auth*, et elle n'offre pas de service *session*, ce qui limite son usage à des services sans session.

Cette configuration standard est utilisée pour les services suivants :

- *chage*
- *chfn*
- *chsh*
- *groupadd, groupdel, groupmod*
- *useradd, userdel, usermod*

Cette configuration est beaucoup plus stricte que celle normalement employée au sein d'un système Linux pour ces différents services. En effet, au sein d'une distribution Linux standard, n'importe quel utilisateur est autorisé à faire appel à *chfn* par exemple pour modifier ses paramètres personnels. Il utilise pour cela un exécutable *chfn* installé avec un bit *setuid root*, qui lui permet de modifier */etc/passwd*. Une telle utilisation est impossible sous CLIP, en l'absence d'exécutables *setuid root* (le bit correspondant n'est pas honoré par le système, cf. [CLIP_1201]), et dans la mesure où l'accès en écriture à */etc/passwd* est de toutes manières interdit depuis les cages d'utilisation principales. Les opérations correspondantes ne peuvent être réalisées que par *root* dans le socle, généralement suite à une requête issue de $\text{ADMIN}_{\text{clip}}$ et traitée par le démon *useradmin* ([CLIP_1304]). La configuration PAM est donc adaptée à ce mode de fonctionnement.

On notera que la mise en oeuvre de *usermod* pour mettre à jour le mot de passe utilisateur (cf. 1.1.3), combinée avec la simplicité de cette configuration PAM, font que l'ancien mot de passe utilisateur n'est pas vérifié lors de la mise à jour de l'empreinte *shadow* du mot de passe utilisateur. Cependant, cette vérification est réalisée lors du transchiffrement des clés de partition chiffrées (cf. 2.1) qui est réalisée avant la mise à jour *shadow* dans le traitement d'un changement de mot de passe utilisateur.

4 Ré-authentification des profils administrateur et auditeur

4.1 Authentification

Trois groupes d'utilisateurs sont associés à des profils spéciaux, qui leur permettent d'ouvrir des sessions dans des cages et vues autres que $USER_{clip}$ / USER (cf. [CLIP_1304] et [CLIP_1401]) :

- Le groupe *core_admin*, qui permet l'ouverture de session dans $ADMIN_{clip}$ depuis $USER_{clip}$.
- Le groupe *core_audit*, qui permet l'ouverture de session dans $AUDIT_{clip}$ depuis $USER_{clip}$.
- Le groupe *rm_admin*, qui permet l'ouverture de sessions dans les vues ADMIN des cages RM, depuis les vues USER de ces mêmes cages. Ce groupe n'est reconnu qu'au sein d'un système de type CLIP-RM.

Ces trois groupes sont ici énoncés dans leur ordre de priorité : un compte qui serait membre à la fois de *core_audit* et *core_admin* par exemple (cas de figure qui ne peut a priori pas résulter de l'utilisation des outils légitimes de création de comptes utilisateurs) serait considéré comme simplement membre de *core_admin*, et son appartenance à *core_audit* serait ignorée par les différents services du système. Les utilisateurs membres de ces groupes ne peuvent pas mettre en oeuvre de session $USER_{clip}$ / USER standard, mais uniquement des sessions spécifiques réduites à l'ouverture de session dans une autre cage ou vue.

L'ouverture de session dans une autre cage ou vue est réalisée par une connexion *ssh*, sur la boucle réseau locale (sur l'adresse commune aux cages $ADMIN_{clip}$, $AUDIT_{clip}$ et $USER_{clip}$ pour *core_admin* et *core_audit*, et sur les adresses des différentes cages RM pour *rm_admin*), auprès de serveurs *sshd* dédiés (un serveur par cage ou vue). Cette connexion s'accompagne d'un changement de compte utilisateur, et d'une ré-authentification. Les comptes utilisateurs standards mis en oeuvre par ces sessions spéciales sont à ce stade les suivants :

- *_admin* (groupe *admin*) dans $ADMIN_{clip}$, pour tout utilisateur d'origine membre de *core_admin*.
- *_audit* (groupes *audit* et *syslog*) dans $AUDIT_{clip}$, pour tout utilisateur d'origine membre de *core_audit*.
- *_admin* (groupe *admin*) dans les vues ADMIN de cages RM, pour tout utilisateur d'origine membre de *rm_admin*.

Ces utilisateurs sont les seuls autorisés par les différentes configurations *sshd* (directive *AllowUsers*, cf. *sshd_config(5)*). Des ouvertures de session spéciales sous d'autres comptes pourraient être envisagées à l'avenir. Dans ce cas, ces nouveaux comptes resteraient membres du même groupe principal que le compte unique utilisé à ce stade.

Les configurations *sshd* imposent aussi l'affichage de la date et de l'heure de la dernière connexion depuis le compte courant, extraites du fichier */var/log/lastlog* de la cage ou vue du serveur (directive *PrintLastLog yes*).

La ré-authentification lors du changement de cage ou de vue est réalisée uniquement avec le mode *Pubkey Authentication* du protocole SSH v2, mode qui est décrit dans le document de référence [RFC4252], et qui repose sur la signature par le client d'un message, en utilisant une clé cryptographique privée à laquelle est associée une clé publique connue du serveur. La configuration des différents serveurs *sshd* interdit tous les autres modes d'authentification. Cette configuration spécifie par ailleurs de la manière suivante les algorithmes mis en oeuvre pour la protection des flux SSH, en remplacement de la configuration par défaut qui laisse un choix beaucoup plus libre d'algorithmes :

- Chiffrement symétrique AES en mode CBC avec des clés dédiées à la confidentialité de 256 bits (*Ciphers aes-256-cbc*)
- Authenticité et intégrité assurée par des motifs HMAC-SHA1, avec des clés dédiées de 128 bits: (*MACs hmac-sha1*)

Ces algorithmes sont décrits plus en détail dans le document de référence [RFC4253]. Les clés associés sont générées par un échange Diffie-Hellman, en utilisant le mode *diffie-hellman-group-exchange-sha256* défini dans le document de référence [RFC4419]. Le paramétrage par défaut est conservé pour le renouvellement des clés, ce qui entraîne un renouvellement des clés après le traitement d'un maximum de $2^{\frac{[taille\,bloc(bits)]}{4}}$ blocs, soit 64 Go.

En revanche, il n'est pas possible de spécifier de manière plus précise dans cette configuration les algorithmes utilisés pour la signature sur laquelle repose l'authentification par clé publique. Dans la pratique, le seul algorithme utilisé sous CLIP est RSA (mode *ssh-rsa*, cf. [RFC4253]), avec des clés de 2048 bits.

4.2 Génération et protection des clés

Les clés d'hôtes (clés des serveurs) sont générées pour chaque serveur lors de la première installation (typiquement, lors de l'installation du poste) d'un démon *sshd* sur le système, par un script *postinst* qui fait appel à l'utilitaire *ssh-keygen*. Les clés publiques associées sont ajoutées automatiquement, lors de chaque création d'un compte utilisateur spécifique associé au serveur concerné, au fichier *.ssh/known_hosts* de la partition *$HOME* de l'utilisateur au sein de la cage ou des vues d'origine de la connexion. Ainsi, lors de la création d'un utilisateur membre de *core_admin*, la clé publique du serveur *sshd* de $ADMIN_{clip}$ est ajoutée au *known_hosts* de la partition *$HOME* de l'utilisateur dans $USER_{clip}$. Lors de la création d'un utilisateur membre de *rm_admin*, la clé publique du serveur *sshd* de la vue ADMIN de RM_X est ajoutée, pour chaque cage RM RM_X, au *known_hosts* de la partition *$HOME* de l'utilisateur dans la vue USER de RM_X. Ces partitions *$HOME* sont systématiquement des montages chiffrés de session, réservés à un compte utilisateur (cf. 2.1).

Réciproquement, les clés de clients (clés des utilisateurs) sont générées lors de la création des comptes utilisateurs concernés, là aussi par un appel à *ssh-keygen*. La clé privée d'un utilisateur est stockée dans le fichier *.ssh/id_rsa* de la ou des partitions *$HOME* associées au profil (c'est à dire celle de $USER_{clip}$ pour les profils *core_admin* et *core_audit,* et celles de chaque vue USER de cage RM pour le profil *rm_admin*). La clé publique correspondante est automatiquement ajoutée au fichier *.authorized_keys* du répertoire *$HOME* de l'utilisateur destinataire de la connexion (*_admin* ou *_audit*) dans la cage ou vue

de destination. Ces répertoires ne constituent pas des montages chiffrés, et ne contiennent pas de clés privées.

Des mesures particulières sont nécessaires pour s'adapter au fait que deux installations CLIP cohabitent généralement sur un même poste ([CLIP_1301]). Les partitions *$HOME* chiffrées, ainsi que les répertoires *$HOME* des utilisateurs *_admin* et *_audit*, sont tous projetés depuis la partition */home* commune à ces deux installations ([CLIP_1304] et [CLIP_1401]) . En revanche, les clés d'hôtes sont stockées dans les arborescences des différentes cages et vues, et donc spécifiques à chaque installation. Cependant, la procédure d'installation complète (cf. [CLIP_2001]) garantit que les clés d'hôtes de la première installation CLIP sont copiées dans la seconde installation sur le même poste, ce qui permet aux utilisateurs de ne jamais voir que le même hôte, quelle que soit l'installation CLIP active à un moment donné.

L'utilitaire *ssh-keygen* génère des clés RSA aléatoires en faisant appel à la fonction *openssl RSA_generate_key()* (*docs/ssleay.txt* dans les sources *openssl*), qui appelle à son tour la fonction *crypto/rsa/rsa_gen.c:rsa_builtin_keygen()*. Cette génération repose sur la lecture de nombres aléatoires sur */dev/random*, en ne retenant que ceux qui vérifient des tests de primalité. On notera que le */dev/random* des cages CLIP et RM n'est en fait qu'un lien symbolique (cf. [CLIP_1304] et [CLIP_1401]) vers */dev/urandom* (générateur pseudo-aléatoire), plutôt que vers le véritable */dev/random* Linux (cf. [CLIP_1206]). En pratique, le générateur aléatoire utilisé diffère donc selon le type de clé :

- Les clés d'hôtes sont générées par un script *postinst* dans la cage $\text{UPDATE}_{\text{clip}}$ ou la vue UPDATE d'une cage RM, et tirent donc leur aléa de */dev/urandom*.
- Les clés d'utilisateurs sont générées par le script */sbin/create_user.sh* (*app-clip/clip-useradmin*) dans le socle CLIP, et tirent leur aléa du vrai */dev/random*.

Les clés ainsi générées sont ensuite sauvegardées au format PEM *openssl,* optionnellement protégées par un chiffrement 3DES utilisant un mot de passe comme clé. Plus précisément, ce chiffrement est réalisé par la fonction *openssl PEM_write_RSAPrivateKey()*, implémentée par *crypto/pem/pem_lib.c:PEM_ASN1_write_bio()*, avec l'algorithme de chiffrement *des-ede3-cbc* (triple DES EDE à trois clés en mode CBC), avec un IV aléatoire (qui est ensuite stocké avec le chiffré dans le fichier *id_rsa)* et une clé dérivée du mot de passe par la fonction *EVP_BytesToKey()*, qui réalise dans ce cas les opérations suivantes :

$$K_{3DES}\ [0:127] = MD5\ (\ P + IV\ )$$
$$K_{3DES}\ [128:167] = \{\ MD5\ (\ K_{3DES}\ [0:127] + P + IV\ )\ \}\ [0:39]$$

où K_{3DES} désigne la clé de chiffrement triple DES, P le mot de passe, et IV l'IV aléatoire aussi utilisé pour le chiffrement, qui est ensuite stocké en tête du chiffré. Le clair se voit par ailleurs ajouter un *padding* selon la méthode PBES1 décrite dans [RFC2898]. Les clés RSA utilisées étant de longueurs multiples de 64 bits, le *padding* réalisé dans ce cas consiste en 8 octets, contenant chacun la valeur 8.

Au sein d'un système CLIP, les clés d'hôte sont générées sans mot de passe, et figurent donc en clair sur le disque. Les clés utilisateur sont en revanche protégées par un mot de passe utilisateur, qui est saisi lors de la création du compte, par l'administrateur qui crée ce compte (donc pas nécessairement par l'utilisateur final du compte). On notera cependant que ce chiffrement des clés privées *ssh* ne constitue qu'une protection complémentaire, le noircissement des clés sur le disque étant avant tout assuré par le chiffrement des partitions où elles sont stockées (cf. 2.1).

Remarque 4 : Gestion avancée des clés ssh

Il n'existe à ce stade pas dans CLIP de moyen de modifier (pour régénérer la clé, ou simplement changer son mot de passe) une clé de ré-authentification SSH. Par ailleurs, en cas de suppression d'un compte à profil administrateur ou auditeur, les clés publiques associées à ce compte doivent être manuellement retirées de la liste authorized_keys *du compte* _admin *ou* _audit*, par un autre utilisateur du compartiment logiciel concerné. Ces opérations devraient à terme être supportées de manière plus automatique.*

4.3 Verrouillage de session X11 et ré-authentification

Les sessions graphiques CLIP se verrouillent automatiquement au bout de 3 minutes d'inactivité. Elles peuvent de plus être verrouillées à la demande explicite d'un utilisateur d'une session $USER_{clip}$ normale (mais pas depuis les sessions spécifiques *core_admin* et *core_audit*). La seule action possible au sein d'une session verrouillée est le déverrouillage de la session, en saisissant le mot de passe de l'utilisateur propriétaire de la session dans une fenêtre graphique. La session n'est déverrouillée qu'en cas de ré-authentification correcte. Cette authentification s'appuie sur deux composants : le démon *pwcheckd* d'une part, et *xscreensaver* d'autre part.

4.4 Démon *pwcheckd*

4.4.1 Fonctionnement du démon

Le démon *pwcheckd* est un démon développé spécifiquement pour CLIP, qui écoute sur une *socket* UNIX, en attente de connexions clientes. Lorsqu'un tel client se connecte, le démon lit sur la *socket* connectée un mot de passe utilisateur, qu'il vérifie en s'appuyant sur PAM, avant de transmettre le résultat de cette vérification au client. Le but d'un tel traitement est de permettre à des clients exécutés dans des cages n'ayant pas d'accès direct au système PAM (en particulier, pas d'accès aux fichiers d'empreintes de mots de passe) de réaliser une authentification indirecte, en dialoguant avec un démon *pwcheckd* exécuté dans le socle.

Le démon *pwcheckd* est supposé lancé sous l'identité *root*. Il prend l'identité de groupe *shadow* (cf. 1.2), puis se détache de son terminal de contrôle, et crée une *socket unix* d'écoute. Lorsqu'une connexion est reçue sur cette *socket*, le démon l'accepte et crée un processus fils pour dialoguer sur la *socket* connectée, puis se met en attente de la terminaison de ce fils (ce qui signifie en particulier qu'une seule connexion est traitée à la fois). Le fils réalise, dans cet ordre, les opérations suivantes :

- Lecture de l'identité du client connecté, par un appel à *clip_getpeereid()* (fonction de la bibliothèque *clip-libs/clip-lib*, qui réalise un appel *getsockopt(SO_PEERCRED)*).
- Appel *setuid()* pour prendre l'identité du client. A l'issue de cet appel, le processus s'exécute sous l'identité du client et dans le groupe *shadow*, ce qui lui donne accès, dans un modèle TCB, à l'empreinte de mot de passe de l'utilisateur associé au client, et à celle-ci uniquement.
- Appel *getpwnam()* pour lire le nom d'utilisateur associé au client. Si l'identité du client n'est pas référencée dans */etc/passwd*, l'authentification est directement refusée.
- Lecture d'un mot de passe d'au plus 64 caractères sur la *socket* connectée.

- Ouverture d'une session PAM, sous le nom de service *pwcheckd* (ce qui signifie que la pile PAM utilisée est celle définie dans le fichier */etc/pam.d/pwcheckd*).
- Authentification et gestion de compte PAM, par des appels *pam_authenticate()* puis *pam_acct_mgmt()*. Fermeture de la session PAM.
- Ecriture sur la *socket* connectée d'un unique caractère informant le client du résultat de la vérification de mot de passe. Un *'Y'* est envoyé pour signaler un succès, et un *'N'* pour un échec.
- Fermeture de la *socket* connectée et terminaison du processus, pour permettre au démon père de se remettre en attente sur sa *socket* d'écoute.

Ce principe de fonctionnement est résumé dans la Figure 2.

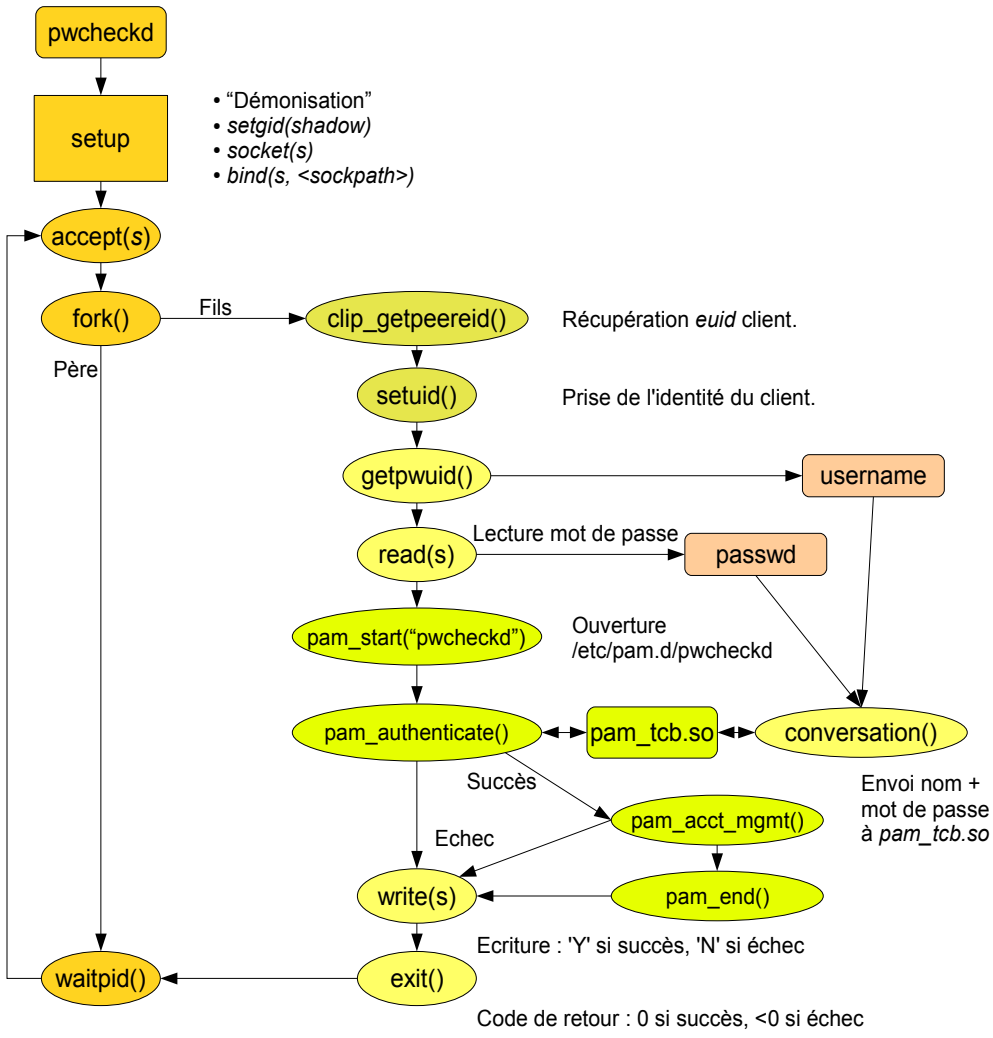

Figure 2: Fonctionnement de *pwcheckd*.

La vérification du mot de passe est réalisée à l'aide d'une fonction de "conversation" PAM (cf.

[PAM_ADG]), qui transmet à la bibliothèque PAM :

- Le nom d'utilisateur en réponse à une requête *PAM_PROMPT_ECHO_ON*.
- Le mot de passe en réponse à une requête *PAM_PROMPT_ECHO_OFF*.

On notera que, par construction, le démon *pwcheckd* ne permet pas de tester le mot de passe d'un autre compte utilisateur. Par ailleurs, en cas d'échec de vérification, le démon n'introduit pas de délai spécifique, mais se voit imposer le délai propre au système PAM sous-jacent, avant de répondre au client, ce qui limite les possibilités d'attaque en force brute, même sur le mot de passe de l'utilisateur courant.

4.4.2 Mise en oeuvre dans CLIP

Le démon *pwcheckd* est installé par un paquetage primaire du socle CLIP. Il est lancé au démarrage sous l'identité *root* dans le socle par le script */etc/init.d/clip_user* ([CLIP_1301]), et se met en écoute sur la *socket /mounts/user_priv/var/run/pwcheckd,* c'est-à-dire */var/run/pwcheckd* dans l'arborescence de $USER_{clip}$ ([CLIP_1304]).

Le fichier */etc/pam.d/pwcheckd* (installé par le même paquetage *sys-auth/pwcheckd*) fait appel à la configuration PAM standard *system-auth* pour l'authentification (cf. 3), et est entièrement permissif pour la gestion du compte (qui n'est pas utilisée à ce stade dans CLIP). Ainsi, la vérification d'un mot de passe par *pwcheckd* se résume à son traitement par *pam_tcb.so* c'est-à-dire à la vérification de l'empreinte *bcrypt* du mot de passe. Le module *pam_env.so*, qui est aussi appelé pour l'authentification par *system-auth*, est sans effet sur *pwcheckd*, dans la mesure où les variables d'environnement qu'il positionne appartiennent à l'environnement du processus fils de *pwcheckd*, qui se termine immédiatement.

Le démon *pwcheckd* est configuré dans CLIP de manière à journaliser les succès et échec de vérification par l'interface *syslog,* dans la *facility DAEMON*. Ces événements sont ainsi consultables dans le fichier */log/daemon.log* de la cage $AUDIT_{clip}$.

4.5 *Xscreensaver*

Le verrouillage de session proprement dit est assuré par un démon *xscreensaver*, qui est lancé dans $USER_{clip}$ à chaque ouverture de session, sous l'identité de l'utilisateur de la session. Ce lancement est effectué directement par le script de session */usr/local/etc/X11/Sessions/CLIP* (*x11-apps/xinit*). Le démon est systématiquement terminé en fin de session, comme tous les autres processus de $USER_{clip}$.

Le paquetage *x11-misc/xscreensaver* est compilé dans CLIP de manière minimaliste, en excluant tout les économiseurs d'écrans autres que l'écran noir (et en évitant ainsi la nécessité d'un *bit setuid root* pour le fonctionnement de l'économiseur), ainsi que les outils de configuration graphique. De plus, le démon *xscreensaver* lui-même est compilé sans le support de l'authentification PAM/*shadow* (options *--without-pam --without-shadow*), et avec pour seule méthode d'authentification l'appel d'un programme externe, *xscreensaver-pwcheck*. Enfin, deux patches spécifiques viennent, d'une part, apporter une localisation française de la fenêtre de déverrouillage, et, d'autre part, supprimer les restrictions d'accès imposées normalement à certains comptes (*adm, sys, bin, operator*, qui n'ont pas de signification particulière sous CLIP, et peuvent par ailleurs être utilisés de manière légitime, par

exemple pour un compte *core_admin*).

Le programme externe *xscreensaver-pwcheck* (*sys-auth/xscreensaver-pwcheck*) est un autre développement spécifique à CLIP, qui assure l'interface entre *xscreensaver* et *pwcheckd*. Il est appelé par *xscreensaver*, immédiatement après la saisie d'un mot de passe de déverrouillage, avec les arguments suivants :

```
xscreensaver-pwcheck "xscreensaver" <user>
```

avec <*user*> le nom de l'utilisateur propriétaire de la session. Immédiatement après son lancement, *xscreensaver* écrit sur son entrée standard le mot de passe saisi par l'utilisateur, puis attend sa terminaison. La session est déverrouillée si et seulement si le programme externe renvoie un code de terminaison nul.

Au lancement, *xscreensaver-pwcheck* vérifie que son premier argument est bien *xscreensaver* (et se termine immédiatement avec un code de retour négatif si ce n'est pas le cas), mais ignore complètement son deuxième argument. En effet, *pwcheckd* détermine l'identité de l'utilisateur à authentifier par ses propres moyens (récupération des *credentials* de la *socket* connectée), sans faire confiance à une quelconque identité transmise explicitement par son client. Une fois le mot de passe lu sur son entrée standard, *xscreensaver-pwcheck* se connecte à la *socket /var/run/pwcheckd*, et y écrit le mot de passe, puis tente d'y lire un caractère. S'il y parvient, et lit un caractère *'Y'*, il se termine avec un code d'erreur nul, ce qui déverrouille la session. Dans tous les autres cas, il se termine avec un code d'erreur négatif. On notera que dans le cas de l'appel à un programme vérificateur externe, *xscreensaver* impose son propre délai avant toute nouvelle tentative d'authentification lorsqu'un mot de passe saisi s'avère incorrect. Ce délai s'ajoute à celui imposé à *pwcheckd* par le module *pam_tcb*.

La configuration *xscreensaver* déployée dans CLIP déclenche un verrouillage automatique de la session après trois minutes d'inactivité. Par ailleurs, un verrouillage peut être explicitement demandé depuis une session $USER_{clip}$ "normale", par un appel à la commande *xscreensaver-cmd -lock,* lancée depuis le menu *fbpanel* ou *KDE* de cette session.

Annexe A Références

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

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

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

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

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

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

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

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

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

[CLIP_1206] *Documentation CLIP – 1206 – Génération d'aléa en couche noyau*

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

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

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

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

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

[CLIP_1502] *Documentation CLIP – 1502 – Mise en oeuvre de racoon2.*

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

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

[BCRYPT] Niels Provos, David Mazières, *A Future-Adaptable Password Scheme, USENIX 1999*

[TCB] *TCB - The alternative to shadow, http://www.openwall.com/tcb/*

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

(version PAM 1.0)

[PAM_SAG] Andrew G. Morgan, Thorsten Kukuk, *The Linux-PAM System Administrator's Guide* (version PAM 1.0)

[PAM_ADG] Andrew G. Morgan, Thorsten Kukuk, *The Linux-PAM Application Developper's Guide* (version PAM 1.0)

[DMCRYPT] *dm-crypt - a device-mapper crypto target, http://www.saout.de/misc/dm-crypt/*

[NMHDE] Clemens Fruhwirth, *New Methods in Hard Disk Encryption*

[IEEE1619] IEEE 1619 SISWG, *P1619 Narrow-Block Encryption, Draft 1.00.03* (octobre 2004)

[COTSE] *COTSE - Word Lists, http://www.cotse.com/tools/wordlists.htm*

[RFC2898] *PKCS #5: Password-Based Cryptography Specification Version 2.0, RFC 2898*

[RFC4252] *The Secure Shell (SSH) Authentication Protocol, RFC 4252*

[RFC4253] *The Secure Shell (SSH) Transport Layer Protocol, RFC 4253*

[RFC4254] *The Secure Shell (SSH) Connection Protocol, RFC 4254*

[RFC4344] *The Secure Shell (SSH) Transport Layer Encryption Modes, RFC 4344*

[RFC4419] *Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol, RFC 4419*

Annexe B Liste des figures

Annexe C Liste des tableaux

Annexe D Liste des remarques