Documentation CLIP

1501

Configuration réseau

Version	Date	Auteur	Commentaires
2.6	02/12/2008	Vincent Strubel	Correction des délais de renouvellement de SA IKE. A jour pour CLIP v03.00.24.
2.5.1	28/11/2008	Vincent Strubel	Correction de coquilles.
2.5	26/11/2008	Vincent Strubel	Séparation du filtrage IPsec et IKE des autres règles de filtrage, introduction de *pass_ipsec_if()* et compléments sur *force_ipsec_*()*. Compléments sur la définition d'un port temporaire, et filtrage des paquets SYN sur la boucle locale. A jour pour CLIP v03.00.23.
2.4	14/10/2008	Vincent Strubel	Ajout des variables *NOLOG_** pour *netfilter*. Ajout des privilèges *CLSM_PRIV_XFRM**. A jour pour *clip-generic-net-1.0.13*(CLIP v03.00.21).
2.3	29/09/2008	Vincent Strubel	Ajout des ports temporaires pour les règles passantes sur la boucle locale. A jour pour CLIP v03.00.19.
2.2	11/09/2008	Vincent Strubel	Remaniement des règles de filtrage *force_ipsec* (passage en *mangle*). ETH0_OUT activé aussi sur les passerelles UPDATE. A jour pour CLIP v03.00.16.
2.1	09/09/2008	Vincent Strubel	Secret temporaire pour l'authentification auprès de *spmd*. Ajustement des durées de vie des SA. A jour pour CLIP v03.00.15.
2.0.1	30/07/2008	Vincent Strubel	Convention plus lisible pour les références.
2.0	03/07/2008	Vincent Strubel	Réécriture complète suite aux nombreuses évolutions des scripts réseau. A jour pour *clip-generic-net-1.0.10*.
1.0	06/07/2007	Vincent Strubel	Version initiale CLIP-RM + CLIP-single. A jour pour *clip-subs-1.3.0*, *clip-net-1.3.0* et *clip-single-net-1.1.0*.

Table des matières

Introduction....4
1 Conception générale....5
1.1 Scripts génériques et scripts spécifiques....5
1.2 Fichiers de configuration....6
1.3 Mode sans échec....11
2 Scripts génériques....13
2.1 Principe de la configuration générique....13
2.2 Configuration du pare-feu....13
2.2.1 Script générique....13
2.2.2 Gestion des privilèges....21
2.2.3 Traitement à l'arrêt....21
2.2.4 Hooks....21
2.3 Configuration des politiques de sécurité....21
2.3.1 Script générique....21
2.3.2 Gestion des privilèges....23
2.3.3 Traitement à l'arrêt....23
2.3.4 Hooks....23
2.4 Configuration des interfaces réseau....24
2.4.1 Script générique....24
2.4.2 Gestion des privilèges....25
2.4.3 Traitement à l'arrêt....26
2.4.4 Hooks....26
2.5 Configuration des associations de sécurité....27
2.5.1 Script générique....27
2.5.2 Gestion des privilèges....28
2.5.3 Traitement à l'arrêt....29
2.5.4 Hooks....29
3 Configurations spécifiques....30
3.1 Configuration CLIP-RM (app-clip/clip-net)....30
3.2 Configuration CLIP-single (app-clip/clip-single-net)....34
3.3 Configuration CLIP-GTW (app-clip/clip-gtw-net)....36
Annexe A Références....41
Annexe B Liste des figures....42
Annexe C Liste des tableaux....42
Annexe D Liste des remarques....42

Introduction

Le cloisonnement réseau réalisé par CLIP prolonge sur le réseau le cloisonnement lourd établi localement entre les cages. Ses deux principaux objectifs sont, d'une part, d'éviter la rupture du cloisonnement local par des canaux de communication réseau (écoute passive d'une cage par une autre ou collaboration des deux cages par des communications client-serveur), et ce y compris sur la boucle locale, et d'autre part d'établir les compartiments réseau qui lient ces compartiments locaux à leur zones de services respectives. La distinction entre compartiments réseau se fait selon l'adresse source des paquets émis et l'adresse destination des paquets reçus, adresse qui est fixée dans chaque cage par l'adresse du contexte réseau *vserver* associé à la cage. Outre la configuration initiale des interfaces, le cloisonnement réseau repose sur deux fonctions de sécurité principales : le filtrage des paquets d'une part, et l'encapsulation IPsec de certains flux d'autre part.

L'essentiel des scripts de configuration réseau CLIP est factorisé dans un paquetage commun à tous les types de systèmes CLIP, complété d'un paquetage spécifique à chaque type de système. La section 1 décrit le principe général de cette séparation, tandis que la section 2 détaille le fonctionnement des scripts génériques. Enfin, la section 3 explicite les adaptations de cette configuration générique réalisées pour les différentes configurations déployées du système CLIP : CLIP-RM, CLIP-single et CLIP-GTW.

1 Conception générale

1.1 Scripts génériques et scripts spécifiques

La configuration réseau d'un poste CLIP est réalisée par une succession de quatre scripts, exécutés, dans l'ordre suivant, au démarrage du poste (cf. [CLIP_1301]) :

- *netfilter* : configuration du pare-feu local
- *spm* : configuration initiale des politiques de sécurité (SP) IPsec, et lancement éventuel d'un démon de mise à jour de ces SP.
- *networking* : configuration et activation des interfaces réseau (y compris la boucle locale)
- *kmpd* : lancement d'un démon de négociation d'associations de sécurité (SA) IPsec.

Les scripts *netfilter, spm* et *networking* sont invoqués avant le verrouillage du système réalisé par le script *reducecap*. En revanche, le script *kmpd* n'est lancé qu'après ce verrouillage (dont il dépend). Les quatre scripts sont de plus tous lancés après le script *veriexec*, dont ils dépendent directement ou indirectement, et qui permet d'attribuer aux utilitaires de configuration réseau les privilèges nécessaires à leur fonctionnement, en particulier les privilèges CLIP-LSM d'accès au réseau (cf. [CLIP_1201]).

On notera que les scripts *spm* et *kmpd* sont virtuels : il n'existe pas de scripts ainsi nommés dans */etc/init.d*. Ces scripts virtuels peuvent être fournis (par le mécanisme de *provide* des scripts de démarrage *Gentoo*) par d'autres scripts, afin de satisfaire les dépendances d'autres scripts de démarrage. Sur un poste CLIP standard, ils sont fournis respectivement par :

- *spmd* : configuration statique des SP, puis lancement du démon de mise à jour de SP de *racoon2*, *spmd*.
- *iked* : lancement du démon IKEv2 de *racoon2*, *iked*.

Cependant, l'utilisation de scripts virtuels permet au besoin de remplacer ces scripts par d'autres, afin par exemple de négocier les SA à l'aide d'un démon IKEv1[1].

Les quatre scripts de configuration réseau sont fournis par un paquetage générique, commun à l'ensemble des types de systèmes CLIP (CLIP-RM, CLIP-GTW, etc...) : *app-clip/clip-generic-net*. Ces scripts génériques réalisent les opérations de base communes à l'ensemble des postes CLIP, comme décrit en 2. Une partie de leur code est partagé au sein d'un fichier unique */etc/init.d/network-common*. Ils sont de plus paramétrables par des fichiers de configuration et scripts secondaires, normalement installés par un second paquetage, cette fois spécifique à un type de système CLIP, par exemple *app-clip/clip-net* pour un poste CLIP-RM, *app-clip/clip-gtw-net* pour un poste de type CLIP-GTW ou *app-clip/clip-single-net* pour un poste CLIP-single. Les fichiers de configuration installés par un tel paquetage définissent un certain nombre de variables obligatoires ou optionnelles qui conditionnent le fonctionnement des scripts génériques, par exemple le nombre d'interfaces réseau externes, ou l'interface à laquelle associer la cage $USER_{clip}$ (cf. 1.2 pour une liste complète). Par ailleurs, un script secondaire spécifique au type de système CLIP est associé à chaque script générique, et « sourcé » par

[1] Une telle configuration, reposant sur les outils *setkeys* et *racoon* de *net-firewall/ipsec-tools*, est supportée par *clip-generic-net* et les différents paquetages secondaires lorsque ceux-ci sont générés sans le drapeau USE *clip-racoon2*. Cette configuration n'est pas la configuration nominale d'un poste CLIP, et n'est pas décrite dans le présent document.

ce dernier lors de chacune de ses invocations. Un script secondaire peut définir un certain nombre de fonctions de type *hook*, aux noms prédéfinis, qui sont appelées, dès lors qu'elles sont définies, par le script générique à certaines étapes de son traitement. Les scripts secondaires supportés par les scripts génériques de *app-clip/clip-generic-net* sont les suivants :

- *netfilter_extra* : sourcé par *netfilter,*
- *spmd_extra* : sourcé par *spmd,*
- *networking_extra* : sourcé par *networking*
- *iked_extra* : sourcé par *iked*

Le détail des fonctions qui peuvent être définies par les différents scripts secondaires est donné en section 2.

1.2 Fichiers de configuration

Les scripts de configuration réseau utilisent deux types de fichiers de configuration :

- Les fichiers de configuration dits **statiques,** qui sont fournis par un paquetage (typiquement le paquetage secondaire de configuration réseau), et ne sont pas modifiables localement. Ces fichiers de configuration sont installés sur la partition racine CLIP, et sont donc protégés contre les écritures en dehors de la séquence initiale de démarrage du poste (cf. [CLIP_1301]). Ils sont considérés comme de confiance, et sont directement sourcés par les scripts de démarrage.
- Les fichiers de configuration dits **dynamiques**[2]**,** qui sont fournis comme des *conffiles* (cf. [CLIP_1101]) par un paquetage, et sont ensuite modifiables localement par les administrateurs CLIP. Ces fichiers sont installés sur une partition non intégralement protégée en écriture, et sont exposés avec des droits en écriture au sein du socle CLIP et de la cage $ADMIN_{clip}$ (cf. [CLIP_1304]). Ils ne sont de ce fait pas considérés comme de confiance, et les variables qu'ils définissent sont uniquement lues à l'aide des fonctions d'import sécurisé de variables de configuration (cf. [CLIP_1301]).

Les scripts génériques supportent un fichier de configuration statique, */etc/conf.d/net*, et deux fichiers dynamiques, */etc/admin/conf.d/net* et */etc/admin/conf.d/netfilter*, ainsi qu'un certain nombre de variables de configuration, qui sont strictement réparties en deux catégories, variables statiques et variables dynamiques, selon le type de fichier de configuration susceptible de les définir. Ces variables sont énumérées dans les Tableau 1, Tableau 2 et Erreur : source de la référence non trouvéeTableau 3. Les variables listées dans ces tableaux doivent obligatoirement être définies sur le poste. Par ailleurs, les scripts secondaires sont susceptibles de définir d'autres variables, aussi bien statiques que dynamiques, et éventuellement des fichiers de configuration supplémentaires. Ces paramètres spécifiques sont décrits dans la section 3.

[2] On notera bien que la notion de statique / dynamique fait référence aux possibilités d'évolution des fichiers de configuration après leur installation, et non pas à la dynamique de prise en compte de leurs modifications : les modifications de fichiers de configuration, quels qu'ils soient, ne sont toujours prises en compte qu'au démarrage suivant.

Nom de variable	Valeurs possibles	Obligatoire	Signification
ETHx_IN	Quelconque	Non	La définition de cette variable, pour x entre 0 et IF_NUMBER – 1, autorise les connexions entrantes sur l'interface x.
ETHx_OUT	Quelconque	Non	La définition de cette variable, pour x entre 0 et IF_NUMBER – 1, autorise les connexions sortantes sur l'interface x.
IF_NUMBER	Entier supérieur ou égal à 1	Oui	Nombre d'interfaces réseau du poste, en plus de la boucle locale. Ces interfaces sont ensuite numérotées de 0 à IF_NUMBER – 1.
UPDATE_IF	Entier entre 0 et IF_NUMBER – 1	Oui	Numéro de l'interface associée à la cage $UPDATE_{clip}$.
UPDATE_NOIPSEC	Quelconque	Non	La définition de cette variable désactive la mise en oeuvre d'IPsec pour les flux réseau issus et à destination de $UPDATE_{clip}$.
USER_IF	Entier entre 0 et IF_NUMBER – 1	Oui	Numéro de l'interface associée à la cage $USER_{clip}$.
USER_IN	Quelconque	Non	La définition de cette variable autorise les connexions entrantes pour la cage $USER_{clip}$.
USER_OUT	Quelconque	Non	La définition de cette variable autorise les connexions sortantes pour la cage $USER_{clip}$.

Tableau 1: Variables de configuration réseau génériques statiquement définies dans *<u>/etc/conf.d/net</u>*, ou directement dans les scripts génériques ou secondaires concernés.

Nom de variable	Valeurs possibles	Obligatoire	Signification
DEFAULT_ROUTE	Adresse IP (aaa.bbb.ccc.ddd)	Oui	Adresse IP de la passerelle par défaut. Doit être atteignable depuis l'une des adresses ETHx_ADDR, dans le sous-réseau associé (ETHx_MASK).
ETHx_ADDR	Adresse IP (aaa.bbb.ccc.ddd)	Oui pour chaque interface du système	Pour tout x entier entre 0 et IF_NUMBER – 1, adresse IP principale de la x-ième interface réseau.
ETHx_MASK	Entier entre 0 et 32	Oui pour chaque interface du système	Longueur de préfixe du sous-réseau associé à ETHx_ADDR.
UPDATE_ADDR	Adresse IP (aaa.bbb.ccc.ddd)	Oui	Adresse IP de la cage $UPDATE_{clip}$.
UPDATE_GW	Adresse IP (aaa.bbb.ccc.ddd)	Oui sauf si UPDATE_NOIPSEC est définie	Adresse IP de la passerelle IPsec du VPN de mise à jour.
UPDATE_MASK	Entier entre 0 et 32	Oui	Longueur de préfixe du sous-réseau associé à UPDATE_ADDR.
USER_ADDR	Adresse IP (aaa.bbb.ccc.ddd)	Oui	Adresse IP de la cage $USER_{clip}$.
USER_MASK	Entier entre 0 et 32	Oui	Longueur de préfixe du sous-réseau associé à USER_ADDR.
USE_NATT	'yes' ou 'no'	Oui	Si positionné à 'yes', active le support du mode *NAT-traversal* pour la négociation de SA IPsec.

Tableau 2: Variables de configuration réseau génériques dynamiquement définies dans */etc/admin/conf.d/net*.

Nom de variable	Valeurs possibles	Obligatoire	Signification
ETHx_IN_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si ETHx_IN est définie	Ports source et destination des connexions TCP autorisées en entrée sur l'interface x, avec un port source égal au port destination.
ETHx_IN_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'. Les ports 500 et 4500 sont automatiquement ignorés.	Oui si ETHx_IN est définie	Ports source et destination des connexions UDP autorisées en entrée sur l'interface x, avec un port source égal au port destination.
ETHx_IN_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si ETHx_IN est définie	Ports destination des connexions TCP autorisées en entrée sur l'interface x, avec un port source supérieur ou égal à 1024.
ETHx_IN_UDP	Liste de ports, séparés par des virgules, ou '-'. Les ports 500 et 4500 sont automatiquement ignorés.	Oui si ETHx_IN est définie	Ports destination des connexions UDP autorisées en entrée sur l'interface x, avec un port source supérieur ou égal à 1024.
ETHx_OUT_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si ETHx_OUT est définie	Ports source et destination des connexions TCP autorisées en sortie sur l'interface x, avec un port source égal au port destination.
ETHx_OUT_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'. Les ports 500 et 4500 sont automatiquement ignorés.	Oui si ETHx_OUT est définie	Ports source et destination des connexions UDP autorisées en sortie sur l'interface x, avec un port source égal au port destination.
ETHx_OUT_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si ETHx_OUT est définie	Ports destination des connexions TCP autorisées en sortie sur l'interface x, avec un port source supérieur ou égal à 1024.
ETHx_OUT_UDP	Liste de ports, séparés par des virgules, ou '-'. Les ports 500 et 4500 sont automatiquement ignorés.	Oui si ETHx_OUT est définie	Ports destination des connexions UDP autorisées en sortie sur l'interface x, avec un port source supérieur ou égal à 1024.
ILLEGAL_LOGLEV	Mot de 2 à 6 lettres (a-z)	Oui	Niveau de journalisation des paquets rejetés par le pare-feu (par exemple, *info* ou *debug*).
ILLEGAL_LOGLIM	Nombre entre 0 et 999 suivi de */minute*, par exemple *12/minute*	Oui	Limite de fréquence de journalisation des paquets rejetés par le pare-feu.
NOLOG_FW_TCP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination TCP rejetés sans journalisation par les règles par défaut en routage (FORWARD).
NOLOG_FW_UDP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination UDP rejetés sans journalisation par les règles par défaut en routage (FORWARD).
NOLOG_IN_TCP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination TCP rejetés sans journalisation par les règles par défaut en entrée (INPUT).
NOLOG_IN_UDP	Liste de ports, séparés par	Non	Ports destination UDP rejetés sans

	des virgules, ou '-'.		journalisation par les règles par défaut en entrée (INPUT).
NOLOG_LO_TCP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination TCP rejetés sans journalisation par les règles par défaut sur la boucle locale.
NOLOG_LO_UDP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination UDP rejetés sans journalisation par les règles par défaut sur la boucle locale.
NOLOG_OUT_TCP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination TCP rejetés sans journalisation par les règles par défaut en sortie (OUTPUT).
NOLOG_OUT_UDP	Liste de ports, séparés par des virgules, ou '-'.	Non	Ports destination UDP rejetés sans journalisation par les règles par défaut en sortie (OUTPUT).
UDPATE_OUT_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui	Ports source et destination des connexions TCP autorisées en sortie de la cage $UPDATE_{clip}$ avec un port source égal au port destination.
UDPATE_OUT_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'.	Oui	Ports source et destination des connexions UDP autorisées en sortie de la cage $UPDATE_{clip}$ avec un port source égal au port destination.
UDPATE_OUT_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui	Ports source et destination des connexions TCP autorisées en sortie de la cage $UPDATE_{clip}$ avec un port source supérieur ou égal à 1024.
UDPATE_OUT_UDP	Liste de ports, séparés par des virgules, ou '-'.	Oui	Ports source et destination des connexions UDP autorisées en sortie de la cage $UPDATE_{clip}$ avec un port source supérieur ou égal à 1024.
USER_IN_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_IN est définie	Ports source et destination des connexions TCP autorisées en entrée de la cage $USER_{clip}$ avec un port source égal au port destination.
USER_IN_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_IN est définie	Ports source et destination des connexions UDP autorisées en entrée de la cage $USER_{clip}$ avec un port source égal au port destination.
USER_IN_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_IN est définie	Ports destination des connexions TCP autorisées en entrée de la cage $USER_{clip}$ avec un port source supérieur ou égal à 1024.
USER_IN_UDP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_IN est définie	Ports destination des connexions UDP autorisées en entrée de la cage $USER_{clip}$ avec un port source supérieur ou égal à 1024.
USER_OUT_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_OUT est définie	Ports source et destination des connexions TCP autorisées en sortie de la cage $USER_{clip}$ avec un port source

			égal au port destination.
USER_OUT_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_OUT est définie	Ports source et destination des connexions UDP autorisées en sortie de la cage $USER_{clip}$ avec un port source égal au port destination.
USER_OUT_TCP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_OUT est définie	Ports destination des connexions TCP autorisées de la cage $USER_{clip}$ avec un port source supérieur ou égal à 1024.
USER_OUT_UDP	Liste de ports, séparés par des virgules, ou '-'.	Oui si USER_OUT est définie	Ports destination des connexions UDP autorisées en sortie de la cage $USER_{clip}$ avec un port source supérieur ou égal à 1024.

Tableau 3: Variables de configuration réseau génériques dynamiquement définies dans /etc/admin/conf.d/netfilter.

1.3 Mode sans échec

Les scripts génériques de configuration réseau supportent un mode sans échec, déclenché automatiquement en cas d'échec de la configuration réseau nominale. Ce mode est nécessaire dans la mesure où la mise en oeuvre des fonctions d'administration locales n'est possible qu'après une authentification réseau sur la boucle locale, de la cage $USER_{clip}$ vers la cage $ADMIN_{clip}$ (cf. [CLIP_1304]). Ainsi, en l'absence de mode sans échec, une erreur d'administration qui entraînerait un échec de configuration réseau au démarrage suivant ne serait plus corrigeable, faute de pouvoir créer la boucle locale réseau nécessaire aux fonctions d'administration.

Le mode sans échec peut être déclenché par l'un des scripts génériques de configuration réseau, suite à la détection d'une erreur. Lors d'un tel déclenchement, le script concerné crée un fichier */var/run/nonetwork* (vide), à fin de signaler aux scripts suivants le passage en mode sans échec. La détection de ce fichier entraîne automatiquement le basculement en mode sans échec des autres scripts de configuration réseau, mais aussi des scripts de configuration des cages $ADMIN_{clip}$ et du script de lancement du démon XDM qui crée la cage $USER_{clip}$, ces derniers ajustant dans ce cas l'adresse obligatoire des cages. On notera que le fichier */var/run/nonetwork* est automatiquement supprimé au début de la séquence de démarrage suivante.

Le mode sans échec limite la configuration réseau aux éléments suivants (quel que soit le type de système CLIP) :

- Politique de filtrage bloquante sur toutes les interfaces réseau, règle spécifique autorisant les flux ADMIN et AUDIT (SSH ports 22 et 23) sur la boucle locale, avec comme adresse source et destination 127.0.0.1.
- Politique de sécurité IPsec passante en clair pour ces mêmes flux ADMIN et AUDIT pour l'adresse source et destination 127.0.0.1, et bloquante pour tous les autres flux.
- Configuration de la boucle locale avec l'adresse 127.0.0.1/8. Désactivation des autres interfaces réseau.
- Aucun démon de mise à jour de SP ou de négociation de SA n'est lancé.
- Les cages $ADMIN_{clip}$ et $USER_{clip}$ sont lancées avec l'adresse 127.0.0.1/8.

La cage $AUDIT_{clip}$ est lancée avant l'invocation du premier script de configuration réseau et ne peut par conséquent pas être paramétrée spécifiquement pour le mode sans-échec. Cependant, la configuration par défaut de la cage l'autorise à écouter sur l'adresse 127.0.0.1 de la boucle locale, ce qui permet dans tous les cas son utilisation en mode sans échec.

Enfin, on notera aussi que le mode sans échec est totalement générique : les *hooks* et paramètres définis par le paquetage secondaire spécifique au type de système CLIP considéré ne sont jamais pris en compte.

***Remarque 1 : déclenchement du mode sans échec par le script* kmpd**

Le mode sans échec peut en l'état être correctement déclenché par n'importe lequel des trois premiers scripts de configuration réseau : netfilter, spmd *ou* networking. *Au besoin, le script* networking *peut recharger les politiques de sécurité du mode sans échec, et définir la règle de filtrage autorisant les flux ADMIN et AUDIT sur la boucle locale 127.0.0.1. En revanche, le déclenchement du mode sans échec par le dernier script,* kmpd, *après un traitement nominal par les trois scripts précédents, n'entraîne que l'échec du lancement du démon de négociation de SA. On notera que cette exception n'est pas réellement problématique, dans la mesure où la configuration nominale réalisée par les trois premiers scripts permet dans ce cas la mise en oeuvre des fonctionnalités d'administration locale.*

2 Scripts génériques

2.1 Principe de la configuration générique

Les scripts de configuration générique installés par le paquetage *app-clip/clip-generic-net* permettent d'établir une configuration réseau de base comportant les éléments suivants :

- Boucle locale réseau (127.0.0.1/8), et une ou plusieurs interfaces *ethernet*, avec chacune une adresse IPv4 et un masque de sous-réseau principaux, et des flux autorisés en entrée et/ou en sortie (au sens de l'établissement de connexion).
- Une cage $USER_{clip}$ associée à une adresse et un masque réseau propres (communs à $USER_{clip}$, $AUDIT_{clip}$ et $ADMIN_{clip}$), ainsi qu'à l'une des interfaces *ethernet* du système et éventuellement des flux autorisés en entrée et / ou en sortie.
- Une cage $UPDATE_{clip}$ associée à une adresse et un masque réseau propres, ainsi qu'à l'une des interfaces *ethernet* du système, et à des flux autorisés en sortie (uniquement), à travers un VPN IPsec (sauf si UPDATE_NOIPSEC est définie).

Les autres cages du système, si elles existent, doivent en revanche être gérées spécifiquement par les scripts secondaires propres au type de système considéré.

Ces différents scripts utilisent un ensemble de fonctions *shell* communes, « sourcées » à partir de deux fichiers fournis par le paquetage *clip-libs/clip-sub* :

- */lib/clip/net.sub* : contient des fonctions de manipulation de paramètres réseau (calcul des adresses de réseau et de *broadcast*, transformations sur les représentations d'adresses et de masques, recherche d'intersection entre deux sous-réseaux,etc...), ainsi que des fonctions de configuration des interfaces réseau (activation / désactivation, attribution d'adresses et d'alias, configuration des routes).
- */lib/clip/netfilter.sub* : contient des fonctions de définition de règles de pare-feu *netfilter*, permettant les différentes opérations décrites en 2.2.

On notera de plus que la fonction commune d'import des adresses de base depuis */etc/admin/conf.d/net,* définie dans */etc/init.d/network-common* et appelée initialement par chacun de ces scripts, vérifie systématiquement l'absence d'intersection entre les sous-réseaux associés aux cages $USER_{clip}$ et $UPDATE_{clip}$. La détection d'une telle intersection déclenche immédiatement le basculement en mode sans échec.

2.2 Configuration du pare-feu

2.2.1 Script générique

Le script */etc/init.d/netfilter* génère plusieurs règles par défaut applicables à l'ensemble des paquets traités par le système, puis définit un filtrage *stateful* par ports spécifique pour chaque adresse du système (adresses des interfaces et des cages). Ces règles peuvent être regroupées en plusieurs

catégories, décrites ici dans l'ordre de leur génération par le script[3] :

Journalisation

Cette catégorie ne correspond pas à une fonction bien identifiée de *netfilter.sub*, il s'agit plutôt d'une caractéristique commune à toutes les règles générées au titre des autres catégories. Le rejet d'un paquet par une règle du pare-feu est journalisé en utilisant les fonctionnalités *printk* du noyau à l'aide de la cible *iptables LOG*. Les messages créés par le pare-feu sont tous précédés du préfixe *FW:* , suivi d'une indication permettant d'identifier la chaîne qui est à l'origine du message. Un filtrage par préfixe réalisé par l'utilitaire de collecte de journaux (cf. [CLIP_1304]) permet de concentrer tous les messages de journalisation du pare-feu dans le fichier */var/log/fw.log* (*/log/fw.log* dans la cage $AUDIT_{clip}$). Ces journaux sont limités en débit par la variable *ILLEGAL_LOGLIM* paramétrable dans */etc/admin/conf.d/netfilter*, avec une valeur par défaut de dix messages par minute.

Règles par défaut

La fonction *set_policy()* définit des politiques *DROP* par défaut pour les trois chaînes *OUTPUT*, *INPUT* et *FORWARD* de la table *filter*. Elle est systématiquement appelée avant toute manipulation des règles, afin d'éviter des états transitoires non sûrs.

La fonction *set_default_rules()* crée une règle par défaut de rejet (*DROP*) avec journalisation des paquets dans un état *INVALID*, ainsi qu'une chaîne *loopback*, initialement vide, vers laquelle sont redirigés tous les paquets circulant sur la boucle locale (redirection depuis les chaînes *OUTPUT* et *INPUT* de la table *filter*). Une règle est de plus ajoutée à la chaîne *OUTPUT* de la table *mangle*, ajoutant une marque spécifique aux paquets destinés à être encapsulés en IPsec (paquets identifiés à l'aide du module *policy* d'*iptables*). Cette marque est destinée à empêcher ces paquets d'être ultérieurement modifiés par une règle de *SNAT*, ce qui pourrait les faire « échapper » à la politique de sécurité qui leur était initialement associée.

Une fonction *set_tcp_checks()* permet d'ajouter une chaîne spécifique *tcpcheck*, vers laquelle sont envoyés tous les paquets TCP reçus en *filter INPUT* sur une interface autre que la boucle locale. Cette chaîne est peuplée d'un ensemble de règles de vérification de cohérence des drapeaux TCP. Elle ne prend en aucun cas de décision *ACCEPT*, mais peut rejeter, avec journalisation, des paquets dont les drapeaux TCP ne seraient pas cohérent entre eux, ou cohérents avec l'état de la connexion tel qu'il est perçu par le module *conntrack*.

Gestion d'IPSec

La mise en oeuvre d'IPsec est permise pour une paire d'associations (entrante et sortante) donnée par la fonction *pass_ipsec_if(),* appelée selon le prototype suivant :

```
pass_ipsec_if <if> <addr> <remote> <use_nat>
```

avec :

- <*if*> le nom de l'interface utilisée par les flux IPsec (ESP) et IKE.

[3] La description qui suit fait plusieurs fois référence aux "ports temporaires". La définition de ceux-ci diffère selon les règles :
- pour un port fixé par une machine distante, un port temporaire est un port compris en 1024 et 65535,
- pour un port fixé localement, un port temporaire est un port compris dans les limites données par la variable *sysctl net.ipv4.ip_local_port_range,* soit typiquement entre 32768 et 61000.

- <*addr*> l'adresse locale des flux ESP et IKE.
- <*remote*> l'adresse distante des flux ESP et IKE, ou "-" pour des règles ouvertes.
- <*use_nat*> *"yes"* ou *"no"* selon que les flux ESP et IKE en mode *NAT-Traversal* doivent être autorisés ou non.

Chaque appel à cette fonction autorise en *INPUT* et *OUTPUT* tous les flux ESP dont l'adresse destination (respectivement source) correspond à l'adresse locale <*addr*>. Optionnellement, l'adresse distante du flux ESP (adresse source en *INPUT* et destination en *OUTPUT*) peut être spécifiée à l'aide du paramètre <*remote*>. Par ailleurs, la fonction *pass_ipsec_if()* définit aussi les règles passantes en *INPUT* et *OUTPUT*, pour les flux IKE permettant la négociation des associations IPsec. Les règles correspondantes sont *stateless*[4], et autorisent tous les flux UDP entre le ou les ports IKE locaux, et les ports IKE et temporaires[5] des pairs distants, dont les adresses sont également fixées par le paramètre <*remote*>, s'il est différent de *"-"*. Les ports IKE ainsi ouverts comportent systématiquement le port 500, complété du port 4500 si le paramètre <*use_nat*> est défini à *yes*, dénotant l'utilisation du mode *NAT-Traversal.*

Par ailleurs, cette mise en oeuvre d'IPsec peut être contrôlée à l'aide de deux autres fonctions, *force_ipsec_if()* et *force_ipsec_all()*, appelées selon les prototypes suivants :

```
force_ipsec_if <if> <addr> [<src> [<dst>]]

force_ipsec_all <addr> [<src> [<dst>]]
```

avec :

- <*if*> le nom de l'interface concernée.
- <*addr*> l'adresse locale ("claire", avant encapsulation) des flux qui doivent être encapsulés.
- <*src*> l'adresse optionnelle de l'extrémité locale du tunnel par lequel les flux doivent être encapsulés.
- <*dst*> l'adresse optionnelle de l'extrémité distante du tunnel par lequel les flux doivent être encapsulés.

Ces fonctions ajoutent des règles de rejet avec journalisation aux chaînes *PREROUTING* et *POSTROUTING* de la table *mangle* (sur l'interface <*if*> pour la première de ces fonctions, et sur toutes les interfaces pour la deuxième), pour tous les paquets dont l'adresse destination (respectivement source) correspond à <*addr*> et auxquels n'est pas associée une politique (identifiée par le module *policy* d'*iptables*) IPsec de chiffrement ESP en mode tunnel. Les extrémités du tunnel peuvent être spécifiées à l'aide des arguments optionnels <*src*> et <*dst*> de ces deux fonctions. Par défaut, en l'absence de tels arguments supplémentaires, un tunnel quelconque est accepté. Chaque appel à l'une de ces deux fonctions crée deux règles dans chacune des tables *PREROUTING* et *POSTROUTING* :

- Une première règle passante pour tous les paquets dont l'adresse destination (resp. source) correspond à <*addr*>, et auxquels est associée une politique de sécurité acceptable par la

[4] En effet, le protocole IKE (ou IKEv2) se prête mal au suivi de connexion, car il repose sur des paquets UDP qui peuvent être émis indifféremment par l'un ou l'autre des pairs. Ainsi, un client CLIP, même s'il se comporte comme un *initiator* IKE, peut recevoir des paquets émis par la passerelle, de sa propre initiative, par exemple pour réaliser une mise à jour de SA sur le point d'expirer sur la passerelle, ou pour réaliser le *nat keepalive*. De tels paquets sont potentiellements vus comme des paquets en état *NEW* en *INPUT* par le filtre de paquets du client, et doivent néanmoins être acceptés.

[5] L'acceptation des flux émis depuis un port temporaire à destination du port IKE est nécessaire, dans la mesure où le port source IKE initial du paquet peut être modifié par une NAT.

fonction.

- Une deuxième règle de rejet avec journalisation, pour tous les autres paquets dont l'adresse destination (resp. source) correspond à <*addr*>, et qui n'ont pas été acceptés par la première règle.

Lorsque UPDATE_NOIPSEC n'est pas définie, le script générique appelle *pass_ipsec_if()* et *force_ipsec_all()* pour, respectivement :

- Autoriser les flux ESP et IKE sur l'interface associée à la cage $UPDATE_{clip}$ (de numéro UPDATE_IF), avec l'adresse principale de cette interface (ETHx_ADDR, avec *x* correspondant à UPDATE_IF). Le port IKE 4500 est ouvert si et seulement si l'administrateur local a défini la variable USE_NATT à *yes* dans */etc/admin/conf.d/net*. Ces autorisations sont limitées à l'adresse distante correspondant à la passerelle de mise à jour (UPDATE_GW).
- Forcer (sur toutes les interfaces) l'encapsulation des paquets dont l'adresse correspond à cette même cage (UPDATE_ADDR), dans un tunnel ESP entre l'adresse principale de l'interface associée à la cage, et l'adresse UPDATE_GW.

On notera que les fonctions *force_ipsec* définissent des règles de filtrages dans une table (*mangle*) et des chaînes (*PREROUTING / POSTROUTING*) qui ne sont généralement pas employées pour ce type d'opération. Cette utilisation peu orthodoxe de *netfilter* remplit un double objectif :

- Unifier le traitement des flux émis ou reçus localement, et des flux routés, sur une configuration de type passerelle. L'utilisation de *PREROUTING* et *POSTROUTING* se substitue à celle de *INPUT, FORWARD* et *OUTPUT*, et permet d'identifier plus trivialement les flux routés entrants ou sortants.
- Garantir le non contournement de ce filtrage, qui est primordial pour la sécurité du système, par d'autres règles de filtrage, définies quant à elles dans la table *filter* classique. En effet, les règles de *mangle PREROUTING* sont systématiquement les premières à être évaluées sur tout paquet entrant, tandis que celles de *mangle POSTROUTING* sont évaluées après toutes les autres règles en sortie, à l'exception de la *NAT* (laquelle ne s'applique pas, sous CLIP, aux flux destinés à être chiffrés, comme décrit plus bas), et indépendamment de l'*ACCEPT* prononcé en table *filter*. Ainsi, quel que soit l'ordre de définition des règles de la table *filter* susceptibles de prononcer un *ACCEPT* (et ainsi de "court-circuiter" les règles suivantes de la table), les paquets violant les définitions *force_ipsec()* sont systématiquement rejetés, aussi bien en entrée qu'en sortie.

Règles ICMP

Une fonction *set_icmp_rules_if()* permet d'ajouter deux règles génériques pour les paquets de type ICMP à la chaîne *filter INPUT* (pour une interface donnée), *OUTPUT* ou *FORWARD*. Ces deux règles entraînent l'acceptation des paquets ICMP de type *destination-unreachable* et *time-exceeded* uniquement, lorsqu'ils sont dans un état *RELATED* du point de vue du moteur *stateful*. Ces règles ICMP permettent la réception de messages d'erreur valides, et ne prennent pas en compte les adresses des paquets, ce qui les rend communes à l'ensemble des cages.

Le script générique appelle cette fonction pour la chaîne *INPUT* de chacune des interfaces *ethernet* du système.

Règles SNAT / DNAT

La déclaration de transformations SNAT peut être réalisée à l'aide de la fonction *snat_cleartext_if()*, qui crée une règle de SNAT applicable à l'ensemble des flux émis par une interface avec une adresse source donnée, à l'exception de ceux qui portent la marque créée par la règle *mangle* installée par *set_default_rules()*. Cette dernière exception permet de ne pas modifier les adresses de paquets pour lesquels une encapsulation IPsec est prévue.

Réciproquement, des règles de DNAT peuvent être définies avec la fonction *dnat_if()*, qui réalise une transformation des flux DNAT arrivant sur une interface avec une adresse destination donnée. Contrairement à *snat_cleartext_if()*, les règles créées par *dnat_if()* s'appliquent sans vérification de marque *mangle*, et sont spécifiques à un protocole TCP ou UDP, et à un port (ou un segment de ports) destination.

Le script générique peut définir plusieurs types de transformation NAT pour les cages $USER_{clip}$ et $UPDATE_{clip}$, en fonction de ses variables de configuration :

- Si les connexions sortantes sont autorisées pour $USER_{clip}$ (USER_OUT définie), celles-ci (portant l'adresse source USER_ADDR) sont transformées par SNAT sur l'interface associée à la cage (USER_IF) pour leur donner l'adresse principale de cette interface.
- Si les connexions entrantes sont autorisées pour $USER_{clip}$ (USER_IN définie), celles-ci (portant comme adresse destination l'adresse de l'interface) sont transformées par DNAT sur l'interface associée à la cage (USER_IF) pour leur donner l'adresse de la cage pour adresse destination. Cette transformation est réalisée aussi bien pour les flux TCP que UDP, et pour tous les ports destination autorisés en entrée de la cage (USER_IN_SAME_TCP, USER_IN_TCP, USER_IN_SAME_UDP, USER_IN_UDP)[6].
- Si les connexions sortantes de $UPDATE_{clip}$ ne sont pas soumises à une encapsulation IPsec (UPDATE_NOIPSEC définie), elles sont transformées par SNAT sur l'interface associée à la cage (UPDATE_IF) pour leur donner l'adresse principale de cette interface.

Règles pour la boucle locale

La fonction *pass_local_lo()* permet de définir des autorisations de communication (*stateless*) sur la boucle locale, pour une adresse spécifique passée en argument (adresse qui doit être à la fois la source et la destination du paquet), et ce entre un port non privilégié (*>= 1024*) et un ou plusieurs ports passés en argument. L'appel à cette fonction entraîne l'ajout de deux règles *ACCEPT* dans la chaîne *loopback* (créée par *set_default_rules()*, et qui s'applique aussi bien en *OUTPUT* qu'en *INPUT* sur la boucle locale), une pour le sens « port non privilégié vers port(s) passé(s) en argument », et une pour le sens inverse (qui exclut les paquets SYN). Dans la mesure où les paquets issus d'un contexte réseau *vserver* utilisent automatiquement l'adresse principale de ce contexte même sur la boucle locale, l'adresse passée en premier argument permet de spécifier à quelle cage s'applique la règle de filtrage local ainsi définie. Les règles définies par cette fonction utilisent le module *multiport* d'*iptables*, de telle sorte qu'une règle unique peut être utilisée même lorsque plusieurs ports sont passés en argument de la fonction.

Le script générique utilise cette fonction pour autoriser les flux d'accès à $ADMIN_{clip}$ et $AUDIT_{clip}$ sur la boucle locale, avec l'adresse de $USER_{clip}$ et les ports associés à ces flux (TCP sur les ports 22 et 23).

[6] On notera que dans ce cas, les ports autorisés en entrée pour $USER_{clip}$ masquent les ports équivalents sur l'adresse principale de l'interface.

Filtrage cage par cage

En dehors des règles génériques, le filtrage est réalisé cage par cage. Une chaîne spécifique est créée pour chaque adresse locale (destination ou source), mais aussi pour chaque état (*NEW* ou *ESTABLISHED,* les connexions *RELATED* ne sont à ce stade pas autorisées en dehors des paquets ICMP) et chaque direction, *INPUT* ou *OUTPUT* (ou alternativement *FORWARD*). Ainsi, jusqu'à quatre chaînes peuvent être associées à une adresse locale : deux pour l'état *ESTABLISHED* dans les directions *INPUT* et *OUTPUT*, plus une pour l'état *NEW* en *OUTPUT* et/ou l'état *NEW* en *INPUT*, selon les directions de connexion qui sont autorisées pour cette adresse. Chaque chaîne est peuplée de règles *stateless* acceptant les paquets en fonction de leur protocole et de leurs ports source et destination. Ces règles utilisent le module *iptables multiport* afin de limiter le nombre de règles nécessaires lorsque plusieurs ports sont ouverts. L'ouverture d'un port en sortie ou en entrée nécessite l'ajout des règles correspondantes dans trois chaînes *stateless*, deux pour les états *ESTABLISHED* (en *INPUT* et *OUTPUT*, en inversant les ports source et destination), et une pour un état *NEW* (*INPUT* ou *OUTPUT*, selon la direction de la connexion). Une fois les différentes règles ajoutées à ces chaînes *stateless*, celles-ci sont « terminées » par une règle explicite de rejet avec journalisation. Des règles de redirection vers ces chaînes, en fonction de l'adresse source ou destination des paquets et de l'état de la connexion associée, sont ensuite ajoutées aux chaînes *OUTPUT* et *INPUT* de la table *filter*.

Une fonction unique de *netfilter.sub* permet de créer un tel ensemble de chaînes et de règles pour une adresse de cage donnée (et une interface donnée). Cette fonction, *pass_compartment_if()*, s'appelle selon le prototype suivant :

```
pass_compartment_if <int_if> <ext_if> <base name> <addr> <direction> \
                    <ports1> <ports2> ... <portsn>
```

avec :

- <*int_if*> le nom de l'interface interne du compartiment.
- <*ext_if*> le nom de l'interface externe du compartiment (différente de l'interface d'entrée uniquement dans le cas d'une direction *forward*[7])
- <*base name*> le nom de base à donner à toutes les chaînes *custom* créées par la fonction. Cette base est complétée en fonction du type de chaîne, par exemple <*base name*>*outnew* pour la chaîne *OUTPUT* en état *NEW* créé au titre de <*base name*>.
- <*addr*> l'adresse du compartiment (adresse source des paquets sortants, et destination des paquets entrants)
- <*direction*> une spécification des types de connexions à créer, choisie parmi :
  - *out* : connexions locales sortantes : entraîne la création de chaînes *OUTPUT/NEW*, *OUTPUT/ESTABLISHED* et *INPUT/ESTABLISHED*
  - *in* : connexions locales entrantes (chaînes *INPUT/NEW, INPUT/ESTABLISHED* et *OUTPUT/ESTABLISHED*)
  - *inout* : connexions locales entrantes et sortantes (chaînes *INPUT/NEW, INPUT/ESTABLISHED, OUTPUT/NEW* et *OUTPUT/ESTABLISHED*)
  - *forward* : connexions routées d'une interface vers l'autre (chaîne *NEW* dans le sens sortant - <*in_if*> vers <*out_if*> - et chaînes *ESTABLISHED* dans les deux sens)

[7] On notera que dans ce cas, l'interface 'externe' *ext_if* est l'interface d'entrée des nouvelles connexions en FORWARD, tandis que l'interface 'interne' *int_if* constitue leur interface de sortie.

- *<portsX>* un nombre arbitraire de spécifications de ports autorisés, séparés par des espaces. Chaque spécification prend la forme *<keyword>:<ports>* avec *<ports>* une liste de ports (ou de couples de ports) séparés par des virgules, et *<keyword>* un mot clé définissant le type de règles *stateless* à créer, choisi parmi :
  - *in_tcpsame, in_udpsame, out_tcpsame, out_udpsame* : autorisation de connexions (respectivement entrantes ou sortantes, TCP ou UDP) avec le même port source que destination. *<ports>* est dans ce cas une liste de ports.
  - *in_tcp, in_udp, out_tcp, out_udp* : autorisation de connexions (respectivement entrantes ou sortantes, TCP ou UDP) avec les ports destination passés en argument, et un port source quelconque entre 1024 et 65535 (inclus). *<ports>* est dans ce cas une liste de ports.
  - *in_tcpsrc, in_udpsrc, out_tcpdst, out_udpdst*: autorisation de connexions (respectivement entrantes ou sortantes, TCP ou UDP) avec des ports destinations et sources différents et arbitraires. *<ports>* est dans ce cas une liste de couples de ports, chacun sous la forme *<port source>:<port destination>*

Le script générique appelle cette fonction pour les flux locaux pour chaque interface *ethernet* du système, ainsi que pour les cages $USER_{clip}$ et $UPDATE_{clip}$. Pour chaque interface *X*, les connexions sortantes et entrantes sont autorisées ou non en fonction de la définition ou non de ETH*X*_OUT et ETH*X*_IN, respectivement. De même, les connexions entrantes ou sortantes sont autorisées ou non pour $USER_{clip}$ en fonction de USER_IN et USER_OUT. En revanche, les connexions sortantes sont systématiquement autorisées pour $UPDATE_{clip}$, et aucune connexion entrante ne peut être autorisée pour cette cage. Le script générique ne définit en aucun cas de règles *FORWARD*, qui sont entièrement du ressort du script secondaire *netfilter_extra.* Les différents ports autorisés par ces règles génériques sont configurables à travers les variables listées dans le Tableau 3Erreur : source de la référence non trouvée.

Finalisation des chaînes *filter*

Une fois configuré l'ensemble des chaînes par cage et état, les chaînes principales *OUTPUT, FORWARD* et *INPUT* de la table *filter* peuvent être finalisées par la fonction *set_final_rules()*, qui ajoute une règle de rejet avec journalisation à la fin de chacune de ces règles, afin de générer des traces (à la différence de ce qui serait fait par la politique par défaut, qui n'assure aucune journalisation) de paquets qui ne correspondraient à aucune cage, ainsi que, à ce stade, d'éventuels paquets non-ICMP appartenant à une connexion dans l'état *RELATED.*

Optionnellement, ces règles de rejet par défaut avec journalisation peuvent être immédiatement précédées de règles de rejet sans journalisation, pour certains ports destination TCP ou UDP définis par l'administrateur local (à travers les variables facultatives *NOLOG_*_** listées dans le Tableau 3). Chaque variable définit une liste de ports destination pour un protocole et une chaîne par défaut (*INPUT, OUTPUT, FORWARD*, et la chaîne *loopback* qui assure le traitement par défaut pour la boucle locale). Lorsque l'une de ces variables est définie, une règle de rejet sans journalisation est automatiquement ajoutée juste avant la règle de journalisation des rejets de la chaîne concernée, de manière à rejeter les paquets sans journalisation. La règle ainsi définie utilise comme seuls critères de choix les ports destination et le protocole, et aucune autre information. Lorsqu'une variable

*NOLOG_*_** donnée n'est pas définie dans */etc/admin/conf.d/netfilter*, la règle correspondante de rejet sans journalisation n'est pas créée. La vocation de ces variables est de permettre à un administrateur d'éviter la pollution des journaux du pare-feu CLIP par des flux réseaux courants et non problématiques, mais néanmoins rejetés par CLIP, par exemple les paquets NETBIOS envoyés en *broadcast* par des postes *Windows* du même LAN.

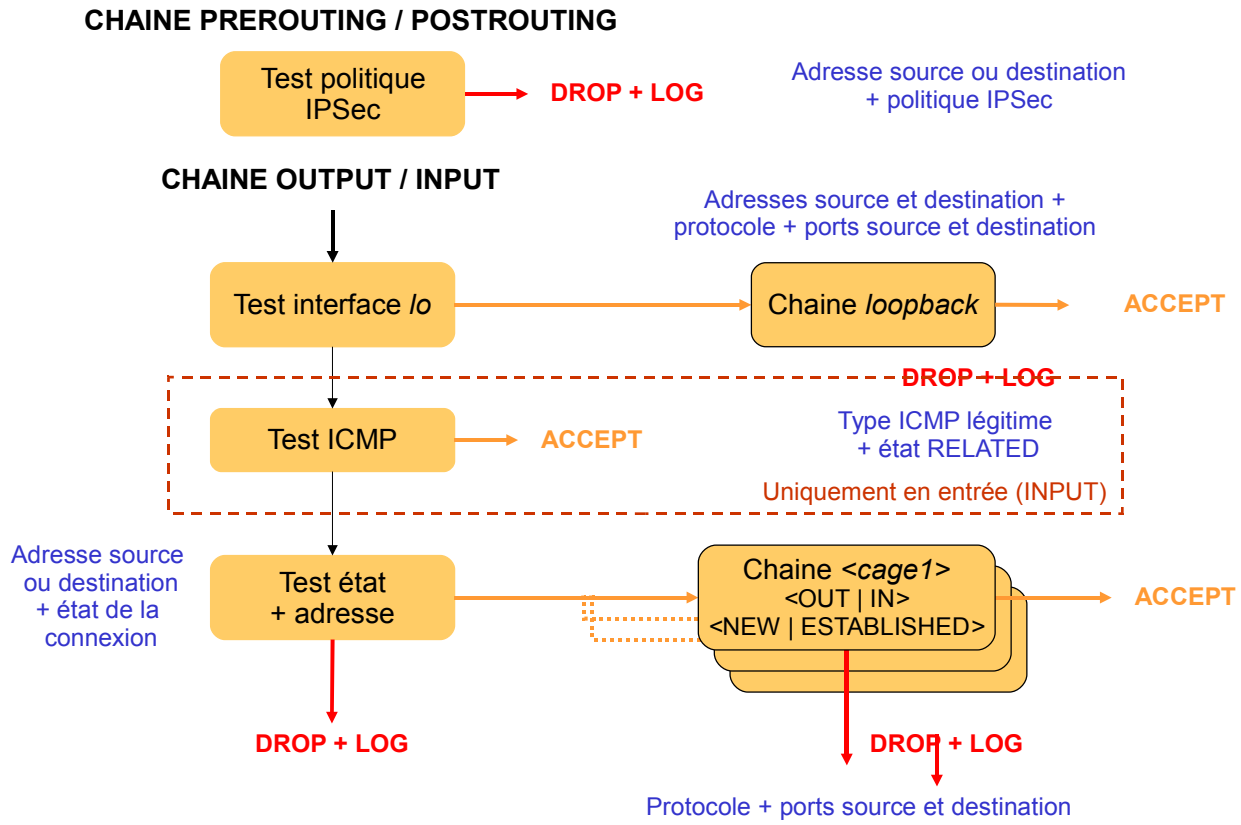

Figure 1: Principe de configuration du filtrage au sein d'un système CLIP

***Remarque 2 : prise en compte des ports IKE dans les appels* pass_compartment_if()**

Le filtrage des flux IKE est réalisé de manière spécifique, à l'aide des fonctions pass_ipsec_if(). *Afin de ne pas contourner ce traitement, les ports IKE doivent être exclus des appels* pass_compartment_if() *sur les interfaces correspondantes. A cette fin, le script générique* netfilter *exclut automatiquement les ports 500 et 4500 des différentes variables* ETHx_{IN,OUT}_UDP{,_SAME} *qui sont importées depuis les fichiers modifiables par l'administrateur local.*

2.2.2 Gestion des privilèges

L'utilitaire *iptables* se voit attribuer par *veriexec* (cf. [CLIP_1201]) le privilège *CLSM_PRIV_NETCLIENT* nécessaire à son fonctionnement, mais pas les capacités *CAP_NET_ADMIN* et *CAP_NET_RAW* dont il a aussi besoin. De ce fait, il ne peut être invoqué avec des privilèges suffisants que par *root* avant l'exécution de */etc/init.d/reducecap* (ce qui est bien le cas du script *netfilter*). Une fois la séquence de démarrage terminée, la configuration *netfilter* ne peut ainsi plus être modifiée sans redémarrage (cf. [CLIP_1301]).

2.2.3 Traitement à l'arrêt

Le script *netfilter,* lorsqu'il est invoqué avec le paramètre *stop* au cours de la séquence d'arrêt du système, ne réalise aucun traitement spécifique, laissant ainsi les politiques de filtrage en place jusqu'à l'arrêt complet du système. Ce traitement n'est pas modifiable par le script secondaire.

2.2.4 *Hooks*

Le script générique *netfilter* appelle, lorsqu'elles sont définies dans le script secondaire *netfilter_extra*, les fonctions *hook* suivantes :

- *config_extra* : appelée après l'import des variables de configuration et avant toute création de règle, permet d'importer des variables de configuration complémentaires.
- *init_extra* : appelée après la définition des règles par défaut par *set_default_rules(), pass_ipsec_if()* et *force_ipsec_*()*, et avant la définition des règles *icmp* et des règles *stateless* par cage. Permet de définir des règles complémentaires en particulier par des appels *pass_ipsec_if()* et *force_ipsec_*()*.
- *start_extra* : appelée après les appels *pass_local_lo(), pass_compartment_if()* et les créations de règles NAT par défaut, et avant la finalisation des règles. Permet de compléter ces appels par défaut par d'autres appels du même type, typiquement associés à des cages supplémentaires ou à des règles en *FORWARD*.

2.3 Configuration des politiques de sécurité

2.3.1 Script générique

La configuration des politiques de sécurité IPsec est réalisée par le script générique *spmd*, qui fournit le script virtuel *spm*. Ce script réalise un traitement en deux temps : en premier lieu, des SP initiales sont statiquement définies par invocation de l'utilitaire *setkey* (*net-firewall/ipsec-tools*), puis le démon *spmd* (*net-firewall/racoon2*) est lancé afin éventuellement de mettre à jour ces SP, et dans tous les cas d'interroger la base de SP (SPD) au profit du démon de négociation de SA lancé par *iked* (cf. 2.5). Ces

deux traitements sont paramétrés par l'import des adresses de base, et de la variable USE_NATT, depuis le fichier de configuration dynamique */etc/admin/conf.d/net*. Après l'import sécurisé de ces variables de configuration, les opérations sont réalisés dans l'ordre comme suit :

Paramétrage statique

Le paramétrage statique est réalisé en « sourçant » en fichier de commandes *setkey*, */etc/ipsec.conf*, qui est entièrement fourni par le paquetage de configuration secondaire. Ce fichier lance des commandes *setkey* sous la forme d'un *here document bash*, selon le principe suivant :

```
/usr/sbin/setkey -k -c << EOF 1>/dev/null
<commande setkey 1>;
<commande setkey 2>;
...
<commande setkey n>;
EOF
```

avec des commandes utilisant des variables *shell* pour spécifier leurs adresses, par exemple :

```
spdadd ${UPDATE_ADDR} 0.0.0.0/0 any -P out \
     ipsec esp/tunnel/${ETH0_ADDR}-${UPDATE_GW}/require;
```

ce qui permet de prendre en compte automatiquement les variables importées par le script générique. Le script */etc/ipsec.conf* nominal est fourni par le paquetage secondaire propre à chaque type de système CLIP. En revanche, le paquetage générique fournit un script alternatif, */etc/ipsec_default.conf*, utilisé dans le mode sans échec réseau (suite à la détection de */var/run/nonetwork*, ou à l'échec de l'import des variables de configuration par le script *spmd*). Ce script par défaut définit dans cet ordre les politiques suivantes :

- Autorisation des flux TCP en clair sur l'adresse 127.0.0.1 (source et destination, dans les sens *in* et *out*), ce qui correspond au flux ADMIN et AUDIT dans le mode sans échec.
- Rejet de tous les autres flux (politique *discard* en *in* et *out*).

Démon *spmd*

Le démon *spmd* est ensuite lancé, sauf dans le mode sans échec. Les modifications apportées dans CLIP à *spmd*, et les détails de sa configuration, sont donnés dans [CLIP_1502]. Le script lance ce démon après avoir généré un fichier de configuration adapté en fonction des variables de configuration. Ce fichier est dérivé d'un squelette de fichier de configuration, dans lequel les adresses réelles et certaines lignes sont remplacées par des valeurs symboliques correspondant à des variables *shell*. Plus précisément, le squelette prend la forme d'un *here document bash*, selon le principe suivant :

```
cat >"${output_file}" <<EOF

interface
{
        ike {
                ${ETH0_ADDR} port 500;
                ${NATT_IF}
        };
```

```
...
EOF
```

avec NATT_IF une variable initialisée ou non à '*${ETH0_ADDR} port 4500;*' par le script générique. Ce squelette est installé dans un fichier */etc/ike2/racoon2.conf.skel*, utilisé aussi bien par le script *spmd* que par le script *iked* (cf. 2.5). Le script générique définit la variable *${output_file}* comme un fichier à chemin aléatoire généré par *mktemp(1)* et de la forme */var/run/racoon2.conf.XXXXXX,* puis « source » le fichier squelette afin de générer dans *${output_file}* un fichier de configuration temporaire par substitution des variables symboliques du squelette. Le démon *spmd* est lancé avec ce fichier de configuration (argument *-f ${output_file}*), qui est ensuite immédiatement supprimé.

Préalablement au lancement de *spmd*, le script génère dans */tmp/spmd.psk* un secret pré-partagé, nécessaire à l'authentification de *iked* auprès de *spmd* sur l'interface *socket UNIX /var/run/racoon/spmif*. Ce secret est généré par lecture de 32 octets aléatoires sur */dev/urandom*, qui sont ensuite stockés dans */tmp/spmd.psk*, lui-même accessible en lecture uniquement par *root*. Ce fichier est ensuite supprimé immédiatement après le lancement de *iked.*

2.3.2 Gestion des privilèges

Tout comme pour le script *netfilter* et l'utilitaire *iptables*, les utilitaires et démons invoqués par *spmd* ne se voient attribuer par *veriexec* que les privilèges permettant leur fonctionnement au cours de la séquence de démarrage, avant le lancement du script *reducecap*, mais pas après cela[8]. En particulier, il est impossible, au terme de la séquence de démarrage, d'invoquer à nouveau *setkey* ou *spmd* pour modifier la base de SP. En revanche, une telle modification reste possible pour le démon *spmd* lancé au cours de la séquence de démarrage.

On notera par ailleurs que le démon *spmd*, lancé avant le verrouillage du système par *reducecap*, dispose initialement de toutes les capacités POSIX. Cependant, ce démon est adapté dans CLIP ([CLIP_1502]) de manière à réduire dès son lancement ses masques de capacités, de manière à ne conserver que celles qui sont nécessaires à son fonctionnement (*CAP_NET_ADMIN* et *CAP_DAC_OVERRIDE*).

2.3.3 Traitement à l'arrêt

Le script *spmd,* lorsqu'il est invoqué avec le paramètre *stop* au cours de la séquence d'arrêt du système, termine le démon *spmd* par une commande *killall spmd*. En revanche, les SP IPsec sont laissées telles quelles, le démon *spmd* étant lui-même modifié pour ne pas supprimer les SP lors de sa terminaison (cf. [CLIP_1502]). Ce traitement n'est pas modifiable par le script secondaire.

2.3.4 *Hooks*

Le seul *hook* qui peut être défini dans le script secondaire *spmd_extra* est une fonction *config_extra*,

[8] Plus précisément, les privilèges *CLSM_PRIV_NETCLIENT, CLSM_PRIV_NETLINK* et *CLSM_PRIV_XFRMSP* sont attribués à *setkey* et *spmd*, mais la capacité *CAP_NET_ADMIN*, aussi nécessaire à leur fonctionnement, ne leur est pas attribuée par *veriexec*. Comme cette capacité est ensuite supprimée, par *reducecap,* du masque de capacités attribuées par défaut à *root*, ces utilitaires ne peuvent plus être invoqués avec des capacités suffisantes une fois le système verrouillé.

permettant d'importer des variables de configuration secondaires, après l'import des variables génériques et avant la configuration des politiques proprement dites. Les principales adaptations au comportement du script sont réalisées à travers l'installation par le paquetage secondaire de fichiers */etc/ipsec.conf* et */etc/ike2/racoon2.conf.skel* adaptés à un type de système CLIP donné.

2.4 Configuration des interfaces réseau

2.4.1 Script générique

Le script générique *networking* assure la configuration et l'activation des interfaces réseau. Il utilise pour ce faire les fonctions de *net.sub*, pour l'attribution d'adresses IP aux interfaces et la configuration du routage. Une description de ces fonctions, qui offrent une couche d'abstraction pour des appels à l'utilitaire *ip* du paquetage *sys-apps/iproute2*, est donnée ici, préalablement à la description du traitement réalisé par le script générique.

Configuration des interfaces

Le fichier *net.sub* définit deux fonctions permettant de configurer les interfaces réseau. La fonction *net_addaddr()*, d'une part, permet d'attribuer une adresse IP à une interface, en lui associant un nom d'alias spécifique. La fonction vérifie au préalable que ce même alias n'est pas déjà défini sur l'interface. La fonction s'appelle selon le prototype suivant :

```
net_addaddr <interface> <netaddr> <alias>
```

avec:

- <*interface*> le nom de l'interface, par exemple *eth0*
- <*netaddr*> l'adresse et le sous-réseau à attribuer à l'interface, sous la forme <*addresse IP*>/<*longueur de préfixe*>, par exemple '*192.168.10.3/24*'
- <*alias*> le nom alias à associer à cette adresse sur l'interface. Le nom réellement attribué prendra la forme <*interface*>:<*alias*>

La fonction *net_startif()*, d'autre part, réalise une configuration complète d'une interface, en lui attribuant une ou plusieurs adresses, avant de l'activer. Son prototype prend la forme suivante :

```
net_startif <interface> <addr1> <addr2> ... <addrn>
```

avec :

- <*interface*> le nom de l'interface, par exemple *eth0*.
- <*addr1*>, <*addr2*>, ... un nombre arbitraire de spécification d'adresses à attribuer à l'interface, chacune spécifiée sous la forme <*alias*>:<*netaddr*> avec <*alias*> le nom d'alias à attribuer à cette adresse, et <*netaddr*> une définition de l'adresse et de la longueur de préfixe, sous la même forme <*adresse*>/<*prefixe*> que celle utilisée par *net_addaddr()*.

Enfin, une troisième fonction, *net_stopif()*, permet de désactiver une interface, puis de supprimer toutes les adresses IP qui lui sont attribuées.

Configuration du routage

Le routage peut être configuré à l'aide de trois fonctions principales. La fonction *net_route_default()*, d'une part, permet de définir la passerelle par défaut du système, tandis que *net_route_dev()* permet de créer une route spécifique pour un sous-réseau à travers une interface donnée du système, et *net_route_gtw()* permet de créer une route spécifique pour un sous-réseau à travers une passerelle donnée. La fonction *net_route_deldefault()* permet de supprimer la route par défaut.

Le script générique *networking* utilise ces primitives afin de réaliser (en mode nominal) les opérations suivantes, dans cet ordre :

- Attribution de l'adresse *127.0.0.1/8* à la boucle locale, et activation de cette interface (appel à *net_startif()*).
- Création d'une route pour le sous-réseau *127.0.0.0/8* à travers l'interface *lo* (*net_route_dev()*).
- Activation de chacune des interfaces *ethernet* du système (de 0 à IF_NUMBER - 1), en lui attribuant pour seule adresse initiale l'adresse principale de l'interface (ETH*X*_ADDR/ETH*X*_MASK). Cette adresse se voit attribuer l'alias *ethX:core* (appels à *net_startif*).
- Attribution complémentaire des adresses USER_ADDR et UPDATE_ADDR aux interfaces concernées (USER_IF et UPDATE_IF respectivement), avec pour alias respectifs *ethX:user* et *ethY:update* (appels à *net_addaddr()*).
- Définition de la passerelle par défaut DEFAULT_ROUTE (*net_route_default()*).

Lorsque le mode sans échec a été déclenché avant le lancement du script (typiquement par l'un des scripts *netfilter* ou *spmd,* créant */var/run/nonetwork*), ou lors de sa phase de configuration (pendant l'import des variables de configuration, avant toute opération de configuration), le script générique se contente de configurer la boucle locale réseau. Par ailleurs, le déclenchement du mode sans échec est possible à tout moment pendant le traitement du script (échec d'une opération de configuration, par exemple parce que la carte *ethernet* concernée n'est pas connectée), et entraîne dans ce cas le retour à une politique IPsec sans-échec (en « sourçant » */etc/ipsec_default.conf*) et la définition au besoin d'un règle de filtrage passante pour les flux ADMIN et AUDIT sur la boucle locale *127.0.0.1*.

2.4.2 Gestion des privilèges

L'utilitaire */sbin/ip* (*sys-apps/iproute2*), qui est utilisé pour toutes les opérations de configuration des interfaces et du routage, se voit attribuer par *veriexec* les privilèges *CLSM_PRIV_NETCLIENT, CLSM_PRIV_NETSERVER* et *CLSM_PRIV_NETLINK* nécessaires à son fonctionnement. L'exécutable *ip* dispose de surcroît de la capacité héritable *CAP_NET_ADMIN*, activable par les capacités héritables du processus qui exécute le fichier (cf. [CLIP_1201]). Cette capacité héritable n'est pas nécessaire lors du démarrage du système, dans la mesure où *CAP_NET_ADMIN* est alors encore attribué par défaut à tous les processus *root*. En revanche, elle est nécessaire au fonctionnement de l'utilitaire pendant la

séquence d'arrêt du système, où elle activée par la capacité héritable transmise par le démon *init* à tous les scripts d'arrêt (cf. [CLIP_1301]). En dehors de ces séquences de démarrage et d'arrêt, la capacité héritable de l'exécutable ne peut pas être activée, ce qui interdit toute reconfiguration des paramètres réseau.

2.4.3 Traitement à l'arrêt

Le script *networking,* lorsqu'il est invoqué avec le paramètre *stop* au cours de la séquence d'arrêt du système, procède à la désactivation de toutes les interfaces *ethernet* et de la boucle locale, ainsi qu'à la suppression des adresses attribuées à ces interfaces, par des appels à *net_stopif()*. La route par défaut est aussi supprimée par un appel à *net_route_deldefault()*.

2.4.4 *Hooks*

Les *hooks* suivants peuvent être définis par le script secondaire */etc/init.d/networking_extra*, afin de compléter la configuration générique :

- *config_extra* : appelée après l'import des variables de configuration et avant toute opération de configuration, permet d'importer des variables de configuration complémentaires.
- *startif_extra* : appelée après le démarrage des interfaces *ethernet* et l'attribution des adresses USER_ADDR et UPDATE_ADDR, et avant la configuration de la route par défaut, permet principalement d'attribuer des adresses supplémentaires aux différentes interfaces.
- *route_extra* : appelée après la configuration de la route par défaut, permet de définir d'éventuelles routes spécifiques.

2.5 Configuration des associations de sécurité

2.5.1 Script générique

Le script générique *iked* fournit la dépendance virtuelle *kmpd* et lance le démon de négociation de SA IPsec *iked* (*net-firewall/racoon2*). Les modifications apportées dans CLIP à *iked*, et les détails de sa configuration, sont donnés dans [CLIP_1502]. Le script lance ce démon après avoir généré un fichier de configuration adapté en fonction des variables de configuration. Ce fichier est dérivé d'un squelette de fichier de configuration commun à *iked* et *spmd* (cf. 2.3), qui se présente comme un script copiant dans un fichier destination *${output_file}* le contenu d'un *here document bash*, en procédant à la substitution des différentes variables de ce document à partir de l'environnement. Le script *iked* procède en premier lieu à l'import des différentes variables, en particulier adresses IP, depuis les fichiers de configuration statiques et dynamiques. Il définit de plus deux variables complémentaires lorsque la variable USE_NATT est positionnée à *'yes'* :

- NATT_IF, positionnée à *'${ETH0_ADDR} port 4500;'*, à moins qu'elle ne soit déjà définie (typiquement par le script de configuration secondaire). Cette variable fournit une directive d'écoute sur le port de négociation *NAT-Traversal*, à inclure dans la section '*interface'* du squelette de configuration.
- NATT_LINE, définie à *'nat_traversal on;'*. Cette variable fournit la directive d'activation du mode *NAT-Traversal,* à inclure dans les sections *'ikev2'* du squelette de configuration.

Le script *iked* génère ensuite un nom de fichier de configuration temporaire, par *mktemp* sous la forme */var/run/racoon2/racoon2.conf.XXXXXX*, nom qui est placé dans *${output_file}* avant de « sourcer » le squelette de configuration, afin de générer le fichier de configuration adapté. Le démon *iked* est ensuite lancé avec le paramètre *-f ${output_file}* de manière à utiliser ce fichier de configuration, qui est ensuite immédiatement supprimé.

Le squelette de configuration, */etc/ike2/racoon2.conf.skel*, n'est pas intégré au paquetage *app-clip/clip-generic-net*, mais fourni par le paquetage de configuration secondaire. Il définit un ensemble de SA et SP à établir, de manière cohérente avec les SP statiquement créées par *spmd* (cf. 2.3). Le détail de cette configuration est donné dans la section 3 et dans le document [CLIP_1502]. On notera simplement ici les points généraux suivants :

- Les algorithmes employés sont ceux de la bibliothèque CCSD ([CCSD]), aussi bien pour les transformations IPsec (en couche noyau, cf. [CLIP_1205]) que pour le protocole IKEv2 ([CLIP_1502]). En particulier :
  - Les SA négociées correspondent à une encapsulation ESP, avec une confidentialité assurée par *ccsd_ctr* et une authenticité assurée par *ccsd_shmac*.
  - La négociation IKEv2 est protégée par les modes symétriques *ccsd_ctr* (confidentialité) et *ccsd_shmac* (authenticité et fonction pseudo-aléatoire).
  - La négociation de clé est réalisée par les fonctions de négociation de clé CCSD (protocole *ccsdaka* dans le fichier de configuration *racoon2*)
  - L'authentification IKEv2 est réalisée par l'authentification de la négociation de clé CCSD (*ccsdsig* dans la configuration *racoon2*), reposant sur des bi-clés CCSD.

- Les SA IPsec expirent après 3600 secondes sur les clients et 4500 secondes sur les passerelles. Les SA IKE expirent après 10100 secondes ou 11100 secondes, respectivement sur les clients (et les passerelles agissant en tant que client, typiquement pour télécharger leurs mises à jour) et les passerelles. Aucune expiration en fonction de la taille chiffrée n'est définie.
- Les SA réalisent un échange DPD (*Dead Peer Detection*) toutes les 60 secondes, afin de détecter les éventuels redémarrages de machines distantes.
- Le seul protocole de négociation autorisé est IKEv2.
- Le mode *NAT-Traversal* peut être optionnellement activé, par l'inclusion des variables NATT_IF et NATT_LINE dans les sections appropriées de la configuration.
- Les identifiants de négociation sont les champs *Subject-Name* des certificats associés aux clés d'authentification CCSD. Ces noms sont extraits de « *bundles* » de clés ACID (*.pvr, .ppr)* par la directive de configuration *ccsd_pxrsubject <chemin du bundle>*. Alternativement, des adresses IP peuvent être utilisées pour les identifiants des passerelles (vues des clients).
- Les clés CCSD sont disponibles localement sur chaque poste (y compris les clés privées des interlocuteurs), sous la forme de « *bundles* » ACID. Elles peuvent être lues par la directive de configuration *ccsdpxr <chemin du bundle>,* ou *ccsdid <répertoire de bundles publics>* pour lire une clé publique fonction de l'identifiant transmis par un initiateur distant.
- Les clés publiques et privées CCSD sont stockées dans deux répertoires :
  - */etc/admin/ike2/cert*, qui contient des fichiers pouvant être mis à jour par l'administrateur local du poste. Ce répertoire contient typiquement au moins la clé privée du poste et son mot de passe.
  - */etc/ike2/cert*, qui contient des fichiers installés lors de l'installation initiale du poste, et qui ne sont plus modifiables après cela. Ce répertoire contient typiquement les clés publiques des passerelles (en particulier passerelle UPDATE, sauf lorsque UPDATE_NOIPSEC est définie) avec lesquelles le système est appelé à établir des associations IPsec.

Le démon *iked*, une fois lancé, se connecte immédiatement au démon *spmd* à travers la *socket UNIX /var/run/racoon/spmif* crée par ce dernier. Il s'authentifie sur cette *socket* en utilisant le secret pré-partagé */tmp/spmd.psk* généré aléatoirement par le script *spmd*. Une fois *iked* lancé, le script *iked* supprime ce fichier de secret pré-partagé, de telle sorte qu'aucun processus lancé ultérieurement ne puisse plus interagir avec *spmd* par son interface légitime.

On notera enfin que le script générique *iked* ne lance pas le démon *iked* lorsque le système est en mode sans-échec (mode qui est soit détecté par la présence du fichier */var/run/nonetwork*, soit déclenché suite à l'échec de l'importation des variables de configuration).

2.5.2 Gestion des privilèges

Le démon *iked* est lancé après le verrouillage du système par *reducecap* ([CLIP_1301]). Il se voit attribuer par *veriexec* les privilèges nécessaires à son fonctionnement, c'est à dire *CLSM_PRIV_NETCLIENT* (connexion aux démons IKEv2 distants), *CLSM_PRIV_NETSERVER* (serveur IKEv2), *CLSM_PRIV_NETLINK* (opérations de résolution d'adresses) *et*

CLSM_PRIV_XFRMSA (ajout et suppression de SA) et les capacités *CAP_NET_ADMIN* et *CAP_NET_BIND_SERVICE* (pour l'écoute sur le port 500). Ces capacités sont attribués automatiquement lorsque le fichier */usr/sbin/iked* est exécuté par *root*, ce qui permet théoriquement à ce dernier de relancer arbitrairement le démon à tout moment du fonctionnement du système. Cependant, la suppression du fichier de secret */tmp/spmd.psk* lors du premier lancement fait qu'un démon *iked* lancé ultérieurement ne pourra pas se connecter au démon *spmd* (qui ne peut quant à lui pas être relancé une fois le système verrouillé), et sortira donc immédiatement en erreur. Tout comme *spmd*, le démon *iked* est modifié pour réduire ses capacités au strict minimum nécessaire (*CAP_NET_ADMIN, CAP_NET_BIND_SERVICE* et *CAP_DAC_OVERRIDE*) immédiatement après son démarrage ([CLIP_1502]). On notera de plus que le démon *iked* ne dispose pas du privilège *CLSM_PRIV_XFRMSP* nécessaire à la modification de la SPD, ce qui l'oblige à passer par *spmd* pour toute opération de ce type.

2.5.3 Traitement à l'arrêt

Le script *iked,* lorsqu'il est invoqué avec le paramètre *stop* au cours de la séquence d'arrêt du système, termine le démon *iked* par une commande *killall iked*. Ce comportement n'est pas modifiable par le script secondaire.

2.5.4 *Hooks*

Le seul *hook* qui peut être défini dans le script secondaire *iked_extra* est une fonction *config_extra*, permettant d'importer des variables de configuration secondaires, après l'import des variables génériques et avant la configuration des politiques proprement dites. Les principales adaptations au comportement du script sont réalisées à travers l'installation par le paquetage secondaire du fichier */etc/ike2/racoon2.conf.skel* adapté à un type de système CLIP donné.

3 Configurations spécifiques

3.1 Configuration CLIP-RM *(app-clip/clip-net)*

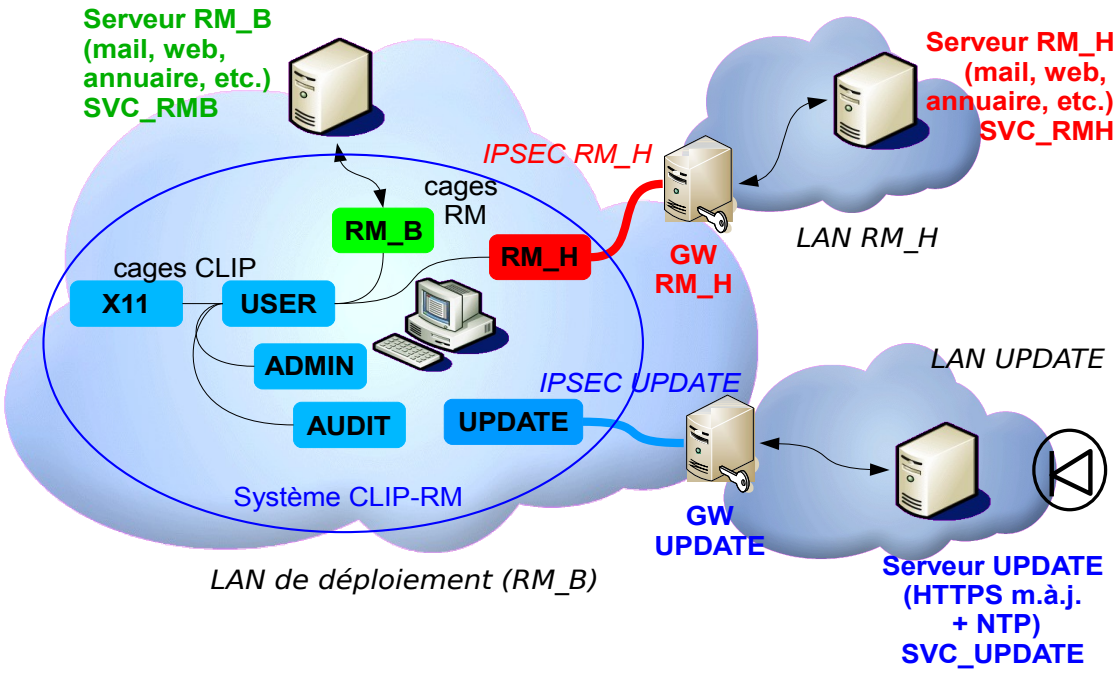

Figure 2: principe de déploiement réseau d'un poste CLIP-RM

La configuration CLIP-RM utilise une unique interface réseau externe, *eth0*, sur laquelle toutes les cages du système se voient attribuer une adresse. Les cages $USER_{clip}$, $ADMIN_{clip}$ et $AUDIT_{clip}$ partagent la même adresse, et n'ont aucun accès au réseau. La cage $UPDATE_{clip}$ met en oeuvre les services génériques de téléchargement de mises à jour et éventuellement de synchronisation horaire, à travers un VPN IPsec établi entre le poste et une passerelle GW_UPDATE. En plus de ces cages CLIP, le système incorpore deux cages RM spécifiques, RM_B et RM_H. RM_B accède directement, sans VPN, au réseau local de déploiement, tandis que RM_H accède à des serveurs de services à travers un VPN IPsec établi entre le poste et une passerelle GW_RM_H. Les connexions réseau entre les cages RM et leurs interlocuteurs sont systématiquement établies depuis les cages RM. Chaque cage RM utilise sa boucle locale pour son administration propre. Cette configuration est résumée dans la Figure 2.

Afin d'assurer cette configuration, les scripts génériques sont configurés par les paramètres statiques suivants :

- IF_NUMBER : 1 (une seule interface)
- USER_IF et UPDATE_IF définies à 0 (interface *eth0* pour ces cages)
- USER_IN et USER_OUT ne sont pas définies (pas d'accès réseau pour la cage $USER_{clip}$).
- UPDATE_NOIPSEC n'est pas définie ($UPDATE_{clip}$ est « branchée » sur un VPN)
- ETH0_OUT et ETH0_IN non définies (pas de flux en dehors des cages, et des flux IKE qui sont traités séparément).

Par ailleurs, les scripts secondaires installés par *app-clip/clip-net* sont définis de manière à réaliser la configuration réseau adaptée aux cages RM. Les *hooks config_extra()* des différents scripts réalisent l'import sécurisé des variables complémentaires listées dans le Tableau 4. Les autres *hooks* et fichiers de configuration réalisent les opérations décrites ci-dessous.

Nom de variable	Valeurs possibles	Signification
RMB_ADDR	Adresse IP (aaa.bbb.ccc.ddd)	Adresse de la cage RM_B
RMB_MASK	Entier entre 0 et 32	Longueur de préfixe du sous-réseau associé à RMB_ADDR.
RMB_OUT_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions TCP autorisées en entrée de la cage RM_B avec un port source égal au port destination.
RMB_OUT_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions UDP autorisées en entrée de la cage RM_B avec un port source égal au port destination.
RMB_OUT_TCP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions TCP autorisées en entrée de la cage RM_B avec un port source supérieur ou égal à 1024.
RMB_OUT_UDP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions UDP autorisées en entrée de la cage RM_B avec un port source supérieur ou égal à 1024.
RMH_ADDR	Adresse IP (aaa.bbb.ccc.ddd)	Adresse de la cage RM_H.
RMH_GW	Adresse IP (aaa.bbb.ccc.ddd)	Adresse de la passerelle RM_H (RM_H_GW).
RMH_MASK	Entier entre 0 et 32	Longueur de préfixe du sous-réseau associé à RMH_ADDR.
RMH_OUT_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions TCP autorisées en entrée de la cage RM_H avec un port source égal au port destination.
RMH_OUT_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions UDP autorisées en entrée de la cage RM_H avec un port source égal au port destination.
RMH_OUT_TCP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions TCP autorisées en entrée de la cage RM_H avec un port source supérieur ou égal à 1024.
RMH_OUT_UDP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions UDP autorisées en entrée de la cage RM_H avec un port source supérieur ou égal à 1024.

Tableau 4: Variables de configuration dynamiques complémentaires pour une configuration CLIP-RM.

Détection de conflits dans le plan d'adressage

Les *hooks config_extra()* de chacun des quatre scripts secondaires, en plus de l'import des variables complémentaires, assurent systématiquement une détection d'éventuelles intersections entre les sous-réseaux suivants :

- réseaux RM_B et RM_H
- réseaux RM_B et $\text{UPDATE}_{\text{clip}}$
- réseaux RM_H et $\text{UPDATE}_{\text{clip}}$

Lorsqu'une telle intersection est détectée, les *hooks* concernés sortent en erreur, ce qui déclenche le basculement en mode sans échec (dans lequel les cages RM ne sont pas lancées).

Filtrage réseau

Le *hook init_extra()* est utilisé pour forcer une encapsulation IPsec des flux associés à RM_H, par un appel *force_ipsec_if()* pour l'adresse RMH_ADDR (les extrémités du tunnel étant renseignées avec ETH0_ADDR et RMH_GW), et autoriser ces flux IPsec par un appel *pass_ipsec_if()*. Le *hook start_extra()* est utilisé pour définir les règles suivantes :

- Règle SNAT sur *eth0* (*snat_cleartext_if()*) pour donner à tous les flux issus de RM_B (RMB_ADDR) l'adresse principale du poste (ETH0_ADDR).
- Ensemble de règles passantes en sortie sur *eth0* pour la cage RM_B (*pass_compartment_if()* avec l'adresse RMB_ADDR, la direction *out*, et les ports importés dans les variables RMB_*).
- Règle passante pour le port 22 sur la boucle locale de RM_B (*pass_local_lo()*, port 22, avec l'adresse RMB_ADDR) (pour les connexions *ssh* d'administration RM), et sur les ports temporaires de la même boucle locale (pour les flux applicatifs, en particulier la communication entre le *plugin java* du navigateur *web* et la machine virtuelle *java*).
- Ensemble de règles passantes en sortie sur *eth0* pour la cage RM_H (*pass_compartment_if()* avec l'adresse RMH_ADDR, la direction *out*, et les ports importés dans les variables RMH_*).
- Règle passante pour le port 22 sur la boucle locale de RM_H (*pass_local_lo()*, port 22, avec l'adresse RMH_ADDR), et les ports temporaires de cette même boucle locale.

Les flux typiquement autorisés (qui dépendent néanmoins de la configuration locale) sont :

Pour $\text{UPDATE}_{\text{clip}}$:

- flux de téléchargement HTTPS (sortant)
- flux de synchronisation NTP (sortant)

Pour RM_H et RM_B :

- HTTP et HTTPS sortant
- IMAP(S), POP(S) et SMTP sortant
- LDAP et LDAPS sortant

Politiques de sécurité IPsec

Le fichier de configuration */etc/ipsec.conf* définit, dans cet ordre, les politiques suivantes :

- Autorisation des flux clairs TCP sur la boucle locale $USER_{clip}$ (adresses source et destination USER_ADDR), pour les flux AUDIT et ADMIN.
- Rejet de tous les autres flux issus ou à destination de USER_ADDR.
- Autorisation des flux clairs TCP sur la boucle locale RM_H (adresses source et destination RMH_ADDR), pour les flux ADMIN de cette cage.
- Encapsulation IPsec obligatoire de tous les autres flux issus ou à destination de RMH_ADDR dans un tunnel ESP entre ETH0_ADDR et RMH_GW.
- Encapsulation IPsec obligatoire de tous les flux issus ou à destination de UPDATE_ADDR dans un tunnel ESP entre ETH0_ADDR et UPDATE_GW.

Par ailleurs, le squelette de configuration */etc/ike2/racoon2.conf.skel* définit les sélecteurs IKEv2 correspondant aux deux politiques d'encapsulation IPsec.

Configuration des interfaces

Le *hook start_extra()* appelle *net_addaddr()* pour ajouter à l'interface *eth0* les adresses RMB_ADDR/RMB_MASK (alias *eth0:rmb*) et RMH_ADDR/RMH_MASK (alias *eth0:rmh*).

Associations de sécurité IPsec

Le squelette de configuration */etc/ike2/racoon2.conf.skel* définit deux passerelles (sections *'remote'*) identifiées par les adresses IP RMH_GW et UPDATE_GW respectivement, et deux politiques associées (sections *'policy'*) pour l'établissement d'associations ESP en mode tunnel avec ces passerelles.

La même clé privée */etc/admin/ike2/cert/ccsd.pvr* est utilisée pour authentifier le poste dans les deux négociations. Le mot de passe de la clé privée est cherché dans le fichier */etc/admin/ike2/cert/ccsd.pwd*. Les passerelles sont identifiées par rapport à leurs clés publiques, respectivement */etc/ike2/cert/rmh.ppr* et */etc/ike2/cert/update.ppr*.

Les autres options sont conformes à celles listées en 2.5.

3.2 Configuration CLIP-single *(app-clip/clip-single-net)*

La configuration CLIP-single utilise une unique interface réseau externe, *eth0*, sur laquelle toutes les cages du système se voient attribuer une adresse. Les cages $USER_{clip}$, $ADMIN_{clip}$ et $AUDIT_{clip}$ partagent la même adresse, et ont accès au réseau local de déploiement, sans VPN et pour des connexions sortantes uniquement. Seule la cage $USER_{clip}$ utilise en pratique cet accès. La cage $UPDATE_{clip}$ met en oeuvre les services génériques de téléchargement de mises à jour et éventuellement de synchronisation horaire, à travers un VPN IPsec établi entre le poste et une passerelle GW_UPDATE. Cette configuration est résumée dans la Figure 3.

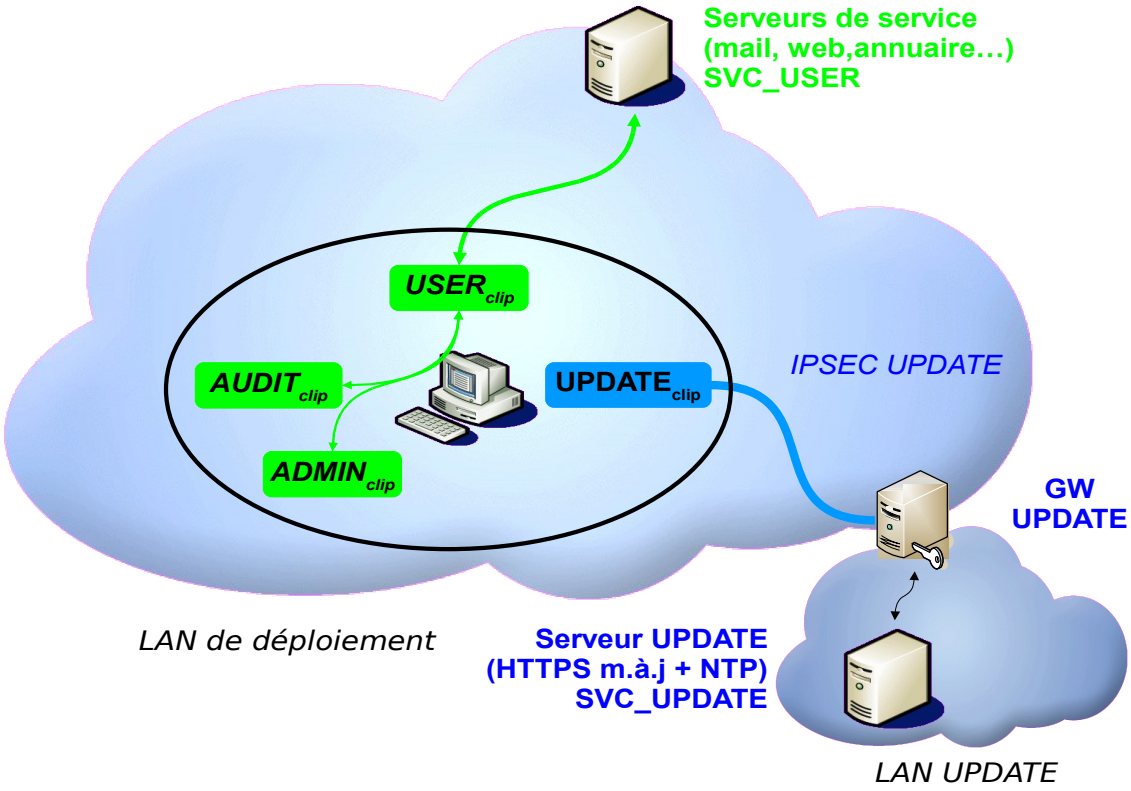

Figure 3: principe de déploiement réseau d'un poste CLIP-single

Afin d'assurer cette configuration, les scripts génériques sont configurés par les paramètres statiques suivants :

- IF_NUMBER : 1 (une seule interface)
- USER_IF et UPDATE_IF définies à 0 (interface *eth0* pour ces cages)

- USER_OUT est définie (autorisation des flux USER sortants)
- USER_IN n'est pas définie (pas d'accès réseau entrant pour la cage $USER_{clip}$).
- UPDATE_NOIPSEC n'est pas définie ($UPDATE_{clip}$ est « branchée » sur un VPN).

Ce paramétrage suffit pratiquement à assurer un fonctionnement adapté des scripts génériques. Le paquetage *app-clip/clip-single-net* n'installe qu'un seul script réseau secondaire, *netfilter_extra*. Les variables de configuration utilisées se limitent à celles listées en 1.2.

Filtrage réseau

Les flux typiquement autorisés (qui dépendent néanmoins de la configuration locale) sont :

Pour $UPDATE_{clip}$:

- flux de téléchargement HTTPS (sortant)
- flux de synchronisation NTP (sortant)

Pour $USER_{clip}$:

- HTTP et HTTPS sortant
- IMAP(S), POP(S) et SMTP sortant
- LDAP et LDAPS sortant

Par ailleurs, le script secondaire *netfilter_extra* ajoute une règle passante (*pass_local_lo*) sur la boucle locale de $USER_{clip}$ pour les ports TCP temporaires, afin d'autoriser certains flux applicatifs locaux (nécessaires en particulier au fonctionnement du *plugin java* intégré au navigateur *web*).

Politiques de sécurité IPsec

Le fichier de configuration */etc/ipsec.conf* définit une unique politique de sécurité (dans les deux directions *in* et *out*), imposant l'encapsulation IPsec obligatoire de tous les flux issus ou à destination de UPDATE_ADDR dans un tunnel ESP entre ETH0_ADDR et UPDATE_GW.

Par ailleurs, le squelette de configuration */etc/ike2/racoon2.conf.skel* définit les sélecteurs IKEv2 correspondant à cette politique .

Associations de sécurité IPsec

Le squelette de configuration */etc/ike2/racoon2.conf.skel* définit une unique passerelle (section *'remote'*) identifiée par l'adresse UPDATE_GW, et la politique associée pour l'établissement d'associations ESP en mode tunnel avec cette passerelle.

Le poste s'authentifie dans cette négociation à l'aide de sa clé privée */etc/admin/ike2/cert/ccsd.pvr* (dont le mot de passe est stocké dans */etc/admin/ike2/cert/ccsd.pwd*), tandis que la passerelle est authentifiée par rapport à sa clé publique */etc/ike2/cert/update.ppr*.

Les autres options sont conformes à celles listées en 2.5.

3.3 Configuration CLIP-GTW *(app-clip/clip-gtw-net)*

La configuration CLIP-GTW utilise deux interfaces réseau, *eth0* (externe) et *eth1* (interne). Elle se place en coupure entre le LAN interne accessible par *eth1* et le LAN externe accessible par *eth0,* sur lequel elle établit des VPN IPsec avec des clients CLIP. Les flux déchiffrés issus de ces VPN sont ensuite routés en clair vers le LAN interne, sur lequel ils ont accès à divers serveurs. Les cages $USER_{clip}$, $ADMIN_{clip}$ et $AUDIT_{clip}$ partagent la même adresse, et n'ont aucun accès au réseau. On distingue ensuite deux types de passerelles :

- Soit le LAN interne héberge des serveurs de service, au profit des cages RM de clients CLIP-RM par exemple. Dans ce cas, les mises à jour sont disponibles sur un serveur hébergé par un LAN distant, joignable par la passerelle à travers un VPN IPsec sur le LAN externe. La cage $UPDATE_{clip}$ met alors en oeuvre les services génériques de téléchargement de mises à jour et éventuellement de synchronisation horaire, à travers un VPN IPsec établi sur *eth0* entre le passerelle et une passerelle GW_UPDATE distante. Cette configuration, dite **CLIP-GTW-RM**, est résumée dans la Figure 4.

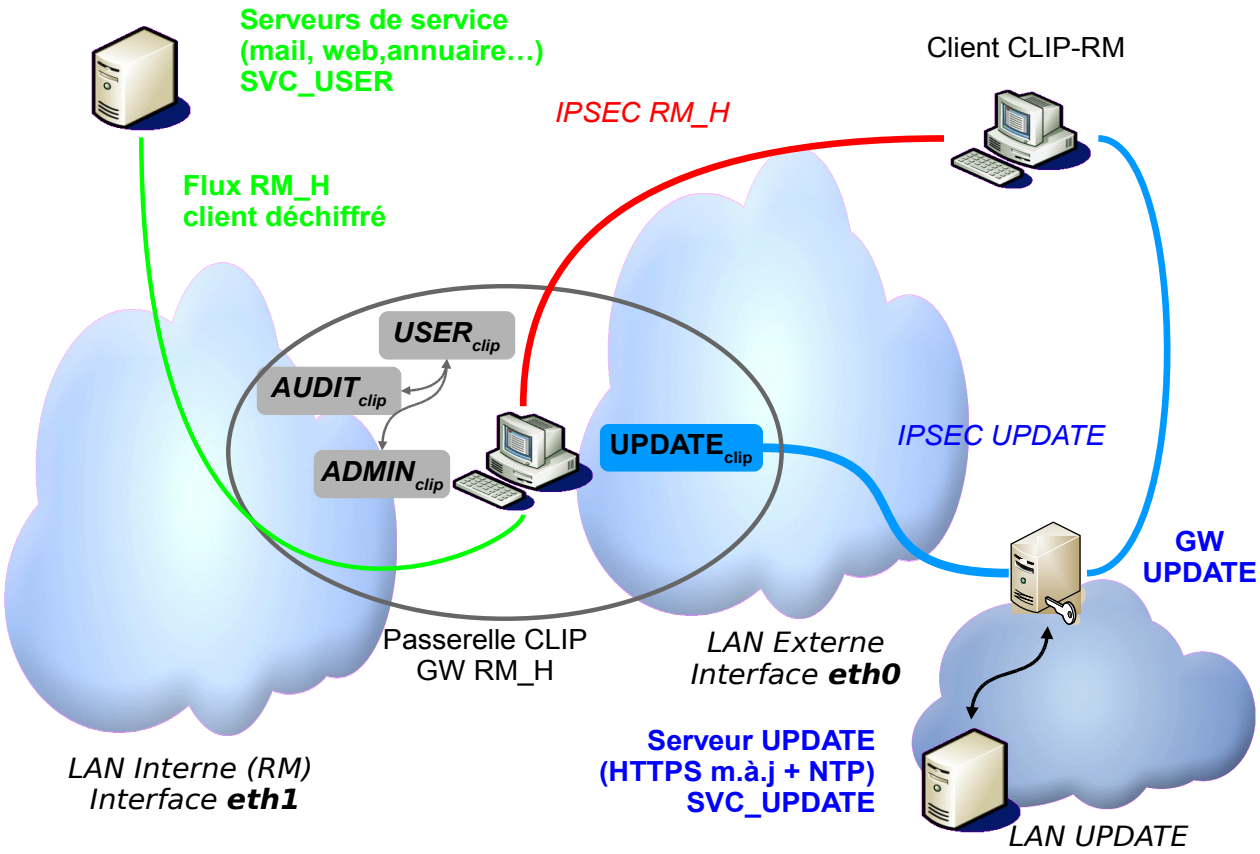

Figure 4: principe de déploiement réseau d'une passerelle CLIP (CLIP-GTW-RM).

- Soit le LAN interne héberge les serveurs UPDATE d'un déploiement CLIP. Dans ce cas, les

mises à jour de la passerelle sont disponibles directement sur le LAN interne, et la cage UPDATE$_{clip}$ met en oeuvre les services de téléchargement de mises à jour et éventuellement de synchronisation horaire directement (sans VPN), sur l'interface *eth1*. Cette configuration, dite **CLIP-GTW-UPDATE,** est résumée dans la Figure 5.

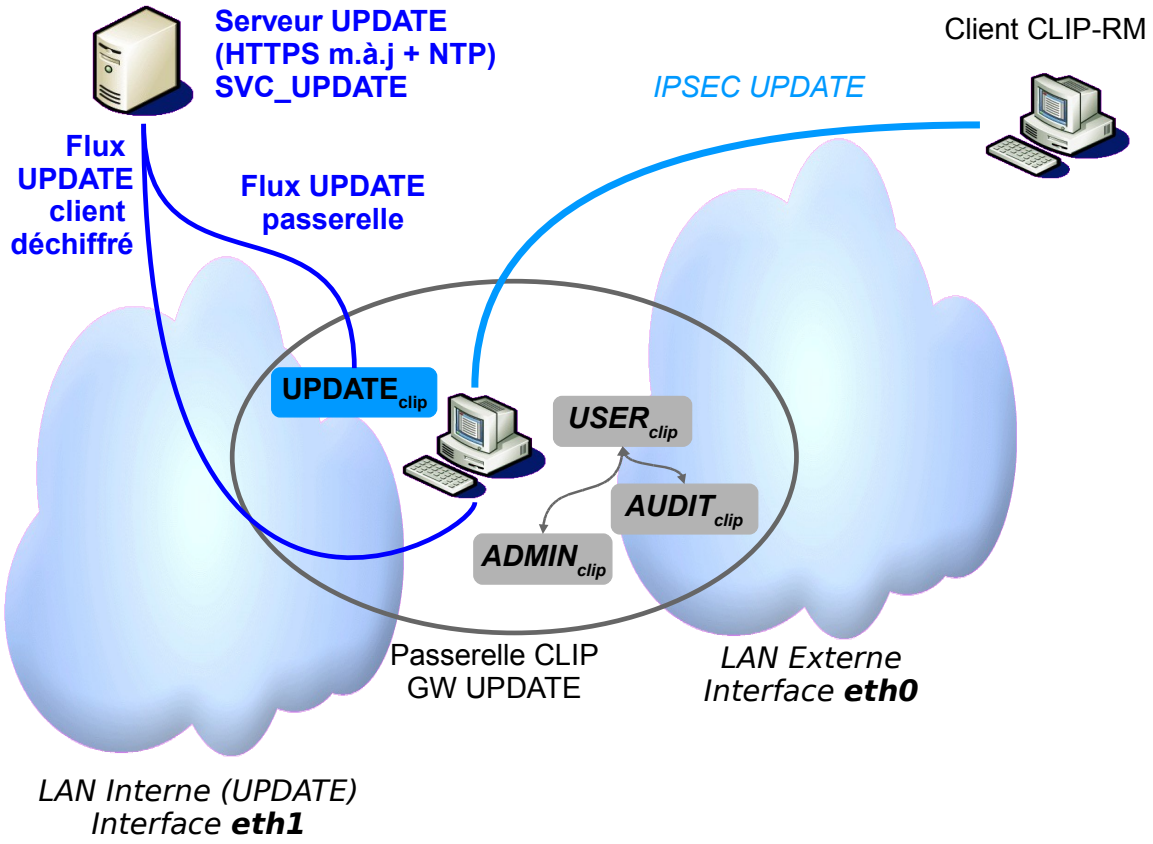

Figure 5: principe de déploiement réseau d'une passerelle UPDATE CLIP (CLIP-GTW-UPDATE).

Ces deux types de configuration sont gérés par un même paquetage secondaire, *app-clip/clip-gtw-net*, adapté pour une installation donnée par la modification des variables UPDATE_NOIPSEC et UPDATE_IF lors de la phase de *post-install* du paquetage binaire. Le paquetage comporte, au lieu d'un unique fichier */etc/conf.d/net*, deux fichiers adaptés respectivement aux configurations CLIP-GTW-RM et CLIP-GTW-UPDATE : */etc/conf.d/net-rm* et */etc/conf.d/net-update*. Le script *postinst* du paquetage lit le contenu d'un fichier */etc/gtw-type*, créé lors de l'installation du poste et jamais modifié ensuite. Lorsque ce fichier contient *'rm'*, le fichier *net-rm* est renommé en *net*, et le fichier *net-update* est supprimé. A contrario, lorsque le fichier contient *'update'*, le fichier *net-update* est renommé *net*, et le fichier *net-rm* est supprimé. Tous les autres fichiers du paquetage sont communs aux deux configurations.

Le paquetage secondaire définit les paramètres statiques suivants, pour les scripts génériques :

Pour les deux configurations :

- IF_NUMBER : 2 (deux interfaces).
- USER_IF : 1 (Cage $USER_{clip}$ sur l'interface interne, même si ce choix n'a pas de réelle signification en l'absence d'accès réseau depuis ou vers cette cage).
- USER_IN et USER_OUT non définies (aucun accès réseau depuis ou vers la cage $USER_{clip}$).
- ETH0_IN : no (les seuls flux entrants sont les flux IKE, qui sont gérés séparément)
- ETH0_OUT: no (les seuls flux sortants sont les flux IKE)

Pour la configuration CLIP-GTW-RM :

- UPDATE_IF : 0 ($UPDATE_{clip}$ sur l'interface externe).
- UPDATE_NOIPSEC non définie($UPDATE_{clip}$ est « branchée » sur un VPN).

Pour la configuration CLIP-GTW-UPDATE :

- UPDATE_IF : 1 ($UPDATE_{clip}$ sur l'interface interne).
- UPDATE_NOIPSEC : 1 ($UPDATE_{clip}$ accède directement au LAN UPDATE).

Les scripts secondaires de configuration réseau installés par *app-clip/clip-gtw-net* supportent un certain nombre de variables de configuration dynamiques, listées dans le Tableau 5.

Nom de variable	Valeurs possibles	Signification
CLIENT_NETWORK	Sous-réseau IP (aaa.bbb.ccc.ddd/nn)	Réseau de clients autorisés à passer en FORWARD de *eth0* vers *eth1* (adresse source des paquets déchiffrés).
FWD_IN_SAME_TCP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions TCP autorisées en FORWARD de *eth0* vers *eth1* avec un port source égal au port destination.
FWD_IN_SAME_UDP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions UDP autorisées en FORWARD de *eth0* vers *eth1* avec un port source égal au port destination.
FWD_IN_TCP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions TCP autorisées en FORWARD de *eth0* vers *eth1* avec un port source supérieur ou égal à 1024.
FWD_IN_UDP	Liste de ports, séparés par des virgules, ou '-'.	Ports destination des connexions UDP autorisées en FORWARD de *eth0* vers *eth1* avec un port source supérieur ou égal à 1024.

Tableau 5: Variables de configuration dynamiques complémentaires pour une configuration CLIP-GTW

Aucun script secondaire de configuration réseau (*networking_extra*) n'est nécessaire. Les trois autres scripts, et les fichiers de configuration IPsec, réalisent les traitements complémentaires décrits ci-

dessous.

Filtrage réseau

Le script *netfilter_extra* ajoute les règles suivantes :

Par le *hook init_extra()* :

- Règle *force_ipsec_if()* sur l'interface externe *eth0*, pour forcer l'encapsulation IPsec des flux issus ou à destination d'une adresse autre que l'adresse externe principale. Cette règle force l'encapsulation des flux clients. Seule l'adresse locale des tunnels est précisée, l'adresse distante étant fonction des clients, et inconnue a priori.
- Règle *pass_ipsec_if()* sur l'interface externe *eth0*, sans restriction sur l'adresse distante, et avec support de *NAT-Traversal*.
- Autorisation des paquets ICMP d'erreur (*set_icmp_rules_if()*) en FORWARD de *eth0* vers *eth1*, et de *eth1* vers *eth0*, et en sortie sur *eth0* et *eth1*.

Par le *hook start_extra()* :

- Création de chaînes et règles *stateless* pour les connexions en FORWARD du réseau externe (limité à CLIENT_NETWORK pour ce qui est des adresses source) vers le réseau interne, initialisées dans cette direction (pas de connexion initialisée par les serveurs) : appel *pass_compartment_if()* en direction *forward* avec *eth1* comme interface interne et *eth0* comme interface externe, et *${CLIENT_NETWORK}* comme adresse. Les types de flux (protocoles, ports) autorisés en FORWARD sont définis à travers les différentes variables *FW_** (cf. Tableau 5).

On notera que les différences entre configurations RM et UPDATE sont automatiquement gérées par le script *netfilter* générique, en fonction de la définition ou non de UPDATE_NOIPSEC.

Les flux typiquement autorisés (qui dépendent néanmoins de la configuration locale) sont :

Pour $UPDATE_{clip}$:

- flux de téléchargement HTTPS (sortant)
- flux de synchronisation NTP (sortant)

Pour FORWARD :

- HTTP et HTTPS
- IMAP(S), POP(S) et SMTP pour les configurations RM uniquement.
- LDAP et LDAPS pour les configurations RM uniquement.

Politiques de sécurité IPsec

Le fichier de configuration */etc/ipsec.conf* définit, dans cet ordre, les politiques suivantes :

- Autorisation des flux clairs TCP sur la boucle locale $USER_{clip}$ (adresses source et destination USER_ADDR), pour les flux AUDIT et ADMIN.

- Rejet de tous les autres flux issus ou à destination de USER_ADDR.
- Lorsque UPDATE_NOIPSEC n'est pas définie, encapsulation IPsec obligatoire de tous les flux issus ou à destination de UPDATE_ADDR dans un tunnel ESP entre ETH0_ADDR et UPDATE_GW.

Aucune politique statique n'est définie pour les clients. Les SP nécessaires aux connexions clientes sont générées à la volée par le démon *spmd*.

Le squelette de configuration *spmd /etc/racoon2.conf.skel* définit, lorsque UPDATE_NOIPSEC n'est pas définie (c'est-à-dire dans une configuration RM), les sélecteurs associés à la politique d'encapsulation des flux UPDATE. Par ailleurs, ce fichier définit dans tous les cas un couple de sélecteurs génériques pour le réseau distant CLIENT_NETWORK, associés à une politique 'IP_RW' (directives '*peers_sa_ipaddr IP_RW;*' dans la section *policy* et '*peers_ipaddr IP_RW;*' dans la section *remote*) pour la génération automatique de SP. On notera qu'une adaptation spécifique à CLIP du démon *spmd* (cf. [CLIP_1502]) fait que les SP ainsi générées sont plus « fines » que les sélecteurs CLIENT_NETWORK (c'est-à-dire limitées à l'adresse du client concerné, plutôt qu'à tout le réseau client), ce qui permet de n'utiliser que deux sélecteurs client, qui servent uniquement à borner au réseau CLIENT_NETWORK les SP pouvant être générées.

Associations de sécurité IPsec

Le squelette de configuration *iked /etc/racoon2.conf.skel* définit un interlocuteur (section *remote)* générique pour l'ensemble des clients, avec une adresse IP distante flottante (IP_RW). La passerelle s'authentifie vis-à-vis de ces interlocuteurs par la clé privée */etc/admin/ike2/cert/ccsd.pvr*, dont le mot de passe est cherché dans */etc/admin/ike2/cert/ccsd.pwd*. Les clients sont authentifiés par rapport à une clé publique flottante (directive *peers_public_key ccsdid /etc/admin/ike2/cert ;*), c'est-à-dire cherchée automatiquement sous le nom */etc/admin/ike2/cert/<ID>.ppr*, avec *<ID>* l'identifiant transmis par le client, correspondant au *SubjectName* de son certificat ACID. La politique associée à ces interlocuteurs spécifie la génération de SA d'adresse locale *ETH0_ADDR* et d'adresse distante flottante (IP_RW), accompagnées de la génération des SP correspondantes.

Par ailleurs, lorsque UPDATE_NOIPSEC n'est pas définie (configuration RM uniquement), le squelette de configuration définit aussi une passerelle (section *'remote'*) identifiée par l'adresse UPDATE_GW, et la politique associée pour l'établissement d'associations ESP en mode tunnel avec cette passerelle.

Le poste s'authentifie dans cette négociation à l'aide de la même clé privée */etc/admin/ike2/cert/ccsd.pvr*, tandis que la passerelle est authentifiée par rapport à sa clé publique */etc/ike2/cert/update.ppr*.

Les autres options sont conformes à celles listées en 2.5.

Annexe A Références

[CLIP_1001] Documentation CLIP – 1001 - Périmètre fonctionnel CLIP

[CLIP_1002] Documentation CLIP – 1002 – Architecture de sécurité

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

[CLIP_1201] Documentation CLIP – 1201 – Patch CLIP LSM

[CLIP_1202] Documentation CLIP – 1202 – Patch Vserver

[CLIP_1203] Documentation CLIP – 1203 – Patch Grsecurity

[CLIP_1204] Documentation CLIP – 1204 – Privilèges Linux

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

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

[CLIP_1304] Documentation CLIP – 1304 – Cages CLIP.

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

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

[RACOON2] The Racoon2 Project, http://www.racoon2.wide.ad.jp

Annexe B Liste des figures

Figure 1: Principe de configuration du filtrage au sein d'un système CLIP.....20
Figure 2: principe de déploiement réseau d'un poste CLIP-RM.....30
Figure 3: principe de déploiement réseau d'un poste CLIP-single.....34
Figure 4: principe de déploiement réseau d'une passerelle CLIP (CLIP-GTW-RM).....36
Figure 5: principe de déploiement réseau d'une passerelle UPDATE CLIP (CLIP-GTW-UPDATE).....37

Annexe C Liste des tableaux

Tableau 1: Variables de configuration réseau génériques statiquement définies dans /etc/conf.d/net, ou directement dans les scripts génériques ou secondaires concernés.....7
Tableau 2: Variables de configuration réseau génériques dynamiquement définies dans /etc/admin/conf.d/net.....8
Tableau 3: Variables de configuration réseau génériques dynamiquement définies dans /etc/admin/conf.d/netfilter.....11
Tableau 4: Variables de configuration dynamiques complémentaires pour une configuration CLIP-RM. 31
Tableau 5: Variables de configuration dynamiques complémentaires pour une configuration CLIP-GTW.....38

Annexe D Liste des remarques

Remarque 1 : déclenchement du mode sans échec par le script kmpd.....12
Remarque 2 : prise en compte des ports IKE dans les appels pass_compartment_if().....20