Documentation CLIP

1203

PaX et Grsecurity

Version	Date	Auteur	Commentaires
1.1.2	28/11/2008	Vincent Strubel	Correction de coquilles.
1.1.1	25/11/2008	Vincent Strubel	Clarifications sur le traitement de *mmap (...MAP_FIXED)* par *PAX_RANDMMAP*.
1.1	05/09/2008	Vincent Strubel	Terminaison de tous les *threads* d'un processus en cas de faute PaX. A jour pour *clip-kernel-2.6.22.24-r10*.
1.0.2	31/07/2008	Vincent Strubel	Convention plus lisible pour les références.
1.0.1	17/07/2008	Vincent Strubel	Option GRKERNSEC_CHROOT_{MKNOD,MOUNT} supprimées de la configuration *clip-kernel-2.6.22.24-r6*
1.0	05/03/2008	Vincent Strubel	Version initiale, à jour pour *clip-kernel-2.6.22.19 (grsec-2.1.11)*.

Table des matières

Introduction

Le patch *Grsecurity* introduit dans le noyau Linux plusieurs mécanismes de durcissement générique du système. Il s'agit d'un patch public (*http://grsecurity.net*), développé en parallèle du processus de développement du noyau. Le patch Grsecurity est construit autour de PaX, patch noyau qui apporte plusieurs mesures de durcissement à bas niveau de la gestion mémoire réalisée par le noyau. Grsecurity ajoute à PaX plusieurs mécanismes génériques complémentaires, de plus haut niveau, ainsi qu'un moteur de gestion de listes de contrôle d'accès (ACL) permettant de définir, processus par processus, des règles de contrôle d'accès très détaillées, et d'ajouter une gestion de rôles *RBAC* (*Role Based Access Control)* aux droits discrétionnaires standards du noyau Linux.

L'objet du présent document est de décrire les différents mécanismes ajoutés par PaX et Grsecurity au noyau Linux, et l'usage qui en est fait dans les systèmes CLIP, aussi bien en termes de configuration que de modification du code. PaX et Grsecurity sont configurables de manière très fine à la compilation du noyau, à travers un nombre important d'options de configuration statiques. La description des différents mécanismes est structurée selon ces différentes options. Sauf mention explicite du contraire, les options décrites sont activées par défaut dans les noyaux CLIP.

1 Durcissement de la gestion mémoire par PaX

1.1 Mémoire des processus utilisateur

1.1.1 PAGEXEC et SEGMEXEC

PAGEEXEC et SEGMEXEC correspondent aux deux mécanismes proposés par PaX afin de rendre non exécutables les zones de mémoire qui ne contiennent pas de code légitime. Une partie des traitements est commune aux deux options, et consiste à marquer, dans la gestion mémoire du noyau, ces zones mémoires comme non exécutables. En particulier, suite à l'activation de l'une ou l'autre de ces options, la pile et le tas de chaque processus auquel s'appliquent ces protections (c'est-à-dire ceux qui ne sont pas marqués explicitement comme exempts de protection, cf. 1.3) sont automatiquement alloués comme non-exécutables[1], respectivement par fs/exec.c:setup_arg_pages() et mm/mmap.c:do_mmap_pgoff(). Les deux options diffèrent en revanche quant à la technique utilisée pour interdire matériellement l'exécution de code dans ces mêmes zones. Une option commune PAX_NOEXEC peut être utilisée pour tester l'activation de l'un ou l'autre des mécanismes PAGEEXEC ou SEGMEXEC.

PAGEEXEC avec support de pages non exécutables (CLIP)

Le mécanisme le plus simple est celui mis en oeuvre par l'option *PAGEEXEC*, lorsque le ou les processeurs du système supportent un marquage de pages mémoire non-exécutables. Un tel support existe nativement sur plusieurs architectures (*SPARC, Alpha, AMD64, HPPA)*, et a été ajouté sur les processeurs récents de la famille *x86* sous la forme du « *bit NX/XD* », utilisable uniquement en mode pagination étendue (*PAE)*. Dans ce cas, l'implémentation de *PAGEEXEC* se résume essentiellement au code commun décrit plus haut, ainsi qu'à un traitement particulier des *pages faults* à des fins de journalisation des erreurs suspectes. La non-exécutabilité des zones mémoires non explicitement exécutables (c'est-à-dire sans drapeau *VM_EXEC*) est quant à elle assurée directement par le noyau non modifié, qui positionne dans ce cas automatiquement le drapeau non-exécutable sur toutes les pages qui n'appartiennent pas à une projection portant le drapeau *VM_EXEC* (fonction *mm/mmap.c:vm_get_page_prot()*).

Ce mode est celui mis en oeuvre par défaut au sein du système CLIP. La configuration noyau active l'option *CONFIG_PAX_PAGEEXEC*, ainsi que l'option *CONFIG_X86_PAE* nécessaire à l'utilisation du drapeau NX, et pas l'option *CONFIG_PAX_SEGMEXEC*. Par ailleurs, les architectures matérielles de déploiement de CLIP sont supposées supporter ce drapeau. On notera cependant que la décision par *PAGEXEC* d'utiliser ou non le drapeau NX est dynamique, en fonction des propriétés du processeur déterminées au démarrage. En particulier, en cas de démarrage sur un processeur ne supportant pas ce drapeau, PaX utilisera automatiquement le mode *PAGEEXEC* sans support de pages non exécutables, décrit ci-dessous, sans impact sur la sécurité autre que la complexité accrue du traitement. On peut par ailleurs signaler que lorsque les options *PAGEEXEC* et *SEGMEXEC* sont simultanément activées dans le noyau (ce qui n'est pas le cas pour CLIP), *PAGEXEC* est utilisé en priorité si le processeur supporte des pages non exécutables, tandis que *SEGMEXEC* est utilisé dans le cas contraire.

[1] Les structures *vm_area_struct* correspondantes perdent le drapeau *VM_EXEC*.

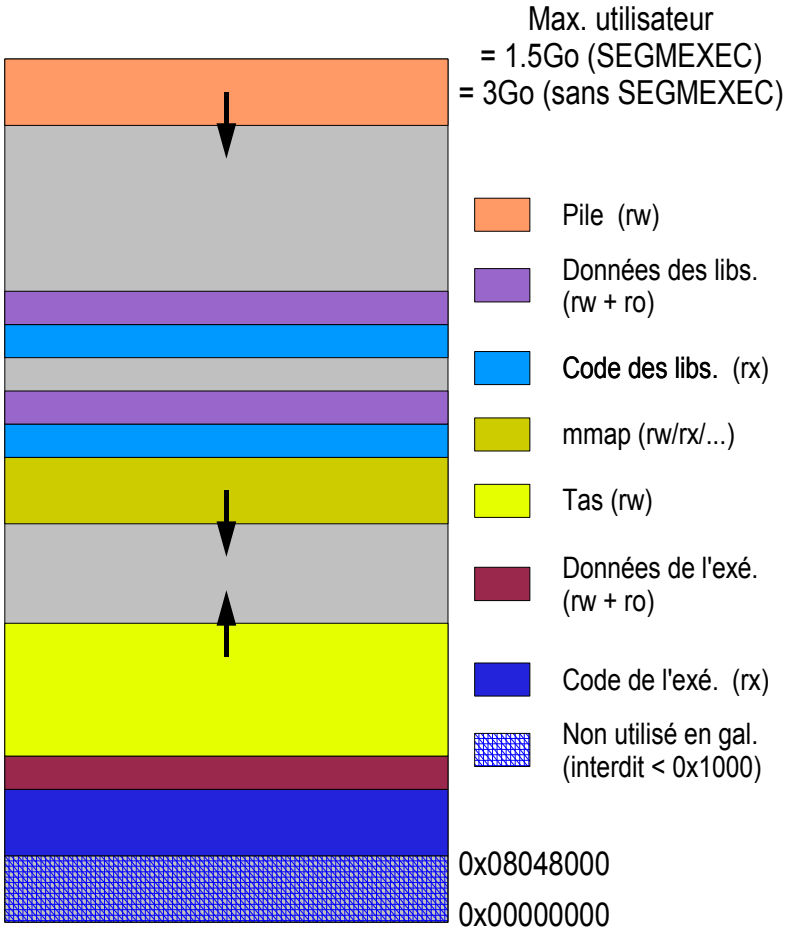

Figure 1: Espace mémoire d'un processus sous Linux.

NB : pour plus de simplicité, ce schéma ne tient pas compte du comportement en cas de très importante occupation mémoire, par exemple l'allocation d'une partie du tas en dessous du code de l'exécutable ou entre les bibliothèques et la pile.

PAGEXEC sans support de pages non exécutables (pour mémoire)

En l'absence de support matériel pour des pages non exécutables, *PAGEXEC* assure une émulation logicielle de cette fonctionnalité en utilisant le drapeau « superviseur » des pages mémoires. Afin de diminuer l'impact en termes de performances d'une telle émulation, cette technique est complétée sur les architectures qui le permettent (notamment *x86*) par un mécanisme léger reposant sur la limite supérieure du segment de code. Lorsque *PAGEXEC* est actif (et sans support de drapeau NX), le segment de code utilisateur des processus protégés est réduit, en ramenant sa limite supérieure à la fin

de la plus haute page légitimement exécutable, au lieu de la fin de l'espace d'adressage. Ainsi, toute tentative d'accès en exécution aux pages mémoires situées au dessus de cette dernière page (typiquement la pile du processus) déclenche une faute de segmentation qui permet de bloquer la tentative avec un faible impact sur les performances des accès légitimes[2]. En revanche, ce mécanisme « léger » ne permet pas de détecter les accès en exécution aux pages de données situées sous la limite haute du segment de code, en particulier le tas et les sections de données du processus dès lors que celui-ci utilise des bibliothèques dynamiques (dont les segments de code sont chargés au dessus du tas et des données de l'exécutable, cf. Figure 1).

Les pages non exécutables situées sous la limite supérieure du segment de code sont protégées par positionnement du bit superviseur, ce qui entraîne automatiquement une faute de page pour tout accès à ces pages depuis le code utilisateur. La routine de traitement des fautes de pages est aménagée pour traiter spécifiquement de telles fautes, selon que la cause de la faute est un *« fetch »* d'instruction (adresse de la faute == EIP fautif), ou un autre type d'accès. En cas d'accès légitime (de type donnée, et compatible avec les permissions en lecture / écriture de la page), la routine retire le bit superviseur de la page fautive, y accède afin de forcer un chargement dans la TLB[3], puis en repositionne le bit superviseur afin de déclencher à nouveau la faute pour un prochain accès (sauf pour les accès en lecture/écriture qui restent régis par les permissions chargées dans la TLB de données, pendant la durée de validité de celle-ci). Ce mécanisme permet de compléter l'utilisation de la limite de segment de code, afin d'offrir une couverture complète de l'espace d'adressage, avec la granularité d'une page. Il impose en revanche une réduction potentiellement sensible des performances.

SEGMEXEC (pour mémoire)

Le mécanisme *SEGMEXEC,* non activé dans un noyau CLIP, repose largement sur les propriétés de la segmentation *x86*, et n'est supporté par PaX que sur cette architecture. Il est basé sur une réduction de moitié de la taille de l'espace d'adressage utilisateur (typiquement 1,5 Go au lieu de 3 Go), permettant de rendre totalement disjoints les segments de code (1,5 Go à 3 Go) et de données (0 à 1,5 Go). Une technique de *VMA mirroring* permet d'exposer une même page physique à deux adresses linéaires différentes, correspondant au même *offset* à l'intérieur de chacun de ces deux segments. Une telle page est alors accessible aussi bien pour une lecture/écriture (à son adresse virtuelle A, convertie par le segment de données en A_l linéaire) qu'en exécution (à la même adresse A, convertie en l'adresse linéaire A_l + 1,5 Go par le segment de code). Le segment de donnée étant positionné sur les adresses basses, toutes les adresses utilisateur (entre 0 et 1,5 Go) sont valides, du point de vue de la segmentation, pour des accès en lecture/écriture (ce qui ne préjuge pas de l'autorisation d'un tel accès du point de vue de la pagination). En revanche, l'accès en exécution à une telle adresse n'est possible que si la page correspondante est aussi exposée à son adresse linéaire « miroir » 1,5 Go plus haut. Ainsi, en n'incluant que les pages légitimement exécutables dans l'image miroir exposée au sein du segment de données, le noyau est en mesure de forcer une faute de pagination pour tout accès illégitime en exécution, et ainsi de repérer un tel accès.

EMUTRAMP (pour mémoire)

L'option *PAX_EMUTRAMP* introduit une exception à l'interdiction d'exécuter la mémoire non

[2] Ce mécanisme est similaire à celui mis en oeuvre par *ExecShield* dans un noyau Linux non patché PaX, cf. [NONSELSEC].

[3] En exploitant la séparation entre les TLB de données (DTLB) et d'instruction (ITLB) : ce chargement dans la DTLB rend la page accessible comme donnée, mais pas pour un *fetch* d'instruction.

exécutable apportée par *SEGMEXEC* ou *PAGEXEC*, de façon à permettre l'exécution de *trampolines* GCC, qui peuvent être générés par le compilateur afin par exemple d'appeler par pointeur une fonction définie de manière imbriquée au sein d'une autre fonction (cf. [CLOSURE]). De tels *trampolines* sont typiquement implémentés à l'aide de code généré dynamiquement sur la pile, dont l'exécution est normalement interdite par PaX. Lorsque l'option *PAX_EMUTRAMP* est activée, la routine de traitement des fautes en exécution de PaX va analyser en premier lieu les premiers octets situés après l'adresse fautive, les comparer à deux motifs typiques de *trampolines* GCC, et autoriser spécifiquement l'exécution si l'un de ces motifs est repéré.

Cette option n'est en aucun cas activée sous CLIP.

Traitement des fautes

Quelle que soit la méthode retenue pour assurer la non-exécutabilité des données, la violation de cette protection PaX se traduit par une faute CPU, qui est spécifiquement détectée par le gestionnaire de faute du noyau, modifié par PaX. Une telle violation est alors journalisée (cf. section 1.4), puis la tâche fautive est terminée par envoi du signal *SIGKILL*. Le comportement standard de PaX dans ce cas est de terminer uniquement la tâche (c'est-à-dire, au niveau utilisateur, le *thread* plutôt que le processus) fautive. Cependant, un *patch* spécifique à CLIP, intégré dans le module *clip-patches*, entraîne dans ce cas la terminaison de l'ensemble du groupe de *threads* (c'est-à-dire tous les *threads* du même processus) contenant la tâche fautive, par un appel *do_group_exit()* plutôt que *do_exit()*. Cette modification est justifiée au plan de la sécurité, dans la mesure où une violation PaX traduit normalement une corruption mémoire, qui affecte a priori tous les *threads* du processus. Par ailleurs, la terminaison du *thread* principal peut dans la pratique faire passer certains processus (typiquement *firefox*) dans un état zombie, tandis que les autres *threads* restent actifs.

1.1.2 MPROTECT

Les contrôles activés par l'option *CONFIG_PAX_MPROTECT* permettent d'interdire la création de projections mémoire (*mmap()*) accessibles, simultanément ou successivement, en écriture et en exécution. Ces contrôles sont réalisés lors de la création de nouvelles projections (*mmap()* et appels associés – *brk()* ou *sbrk()* notamment, fichier source *mm/mmap.c*) et lors de la modification des permissions d'une projection existante (*mprotect()*, fichier source *mm/mprotect.c*).

Lors de la création d'une nouvelle projection, les droits en exécution ne sont pas attribués sur cette projection dès lors que les droits en écriture ont été demandés, et ce y compris dans le cas où les droits en exécution auraient aussi été explicitement demandés[4]. Dans ce cas, le drapeau *VM_MAYEXEC* est de plus retiré à la projection, afin d'interdire l'attribution ultérieure de tout droit en exécution. Si au contraire les droits en exécution avaient été demandés explicitement, mais pas les droits en écriture, la projection est simplement marquée non inscriptible, et le drapeau *VM_MAYWRITE* lui est retiré afin d'interdire tout accès futur en écriture. Lors d'une extension du tas par un appel *brk()*, la projection associée est réalisée automatiquement sans les permissions en exécution, et sans le drapeau *VM_MAYEXEC*. De même, la pile de chaque processus auquel *PAX_MPROTECT* s'applique se voit

[4] On notera qu'au sein d'un noyau CLIP, cette modification éventuelle des droits effectivement accordés sur la projection est réalisée avant l'interdiction par le LSM CLIP (cf. [CLIP_1201]) des accès en écriture au fichier associé à une projection exécutable, ce qui évite de réaliser cette interdiction pour une projection sur laquelle le droit en exécution ne serait finalement pas accordé.

retirer le drapeau *VM_MAYEXEC* lors de sa création. On notera en particulier que *VM_MAYEXEC* n'est laissé à une projection que lorsque celle-ci dispose bien initialement des droits en exécution. Il est ainsi impossible de donner par *mprotect()* les droits en exécution à une projection créée initialement sans *PROT_EXEC*.

PAX_MPROTECT ne modifie en revanche pas les contrôles réalisés lors d'un *mprotect()*, dans la mesure où la prise en compte normale des drapeaux VM_MAYWRITE et VM_MAYEXEC, ainsi que le positionnement ajusté pour PaX de ces derniers lors du *mmap()* initial suffisent à interdire toute transition qui donnerait successivement les droits en écriture et en exécution à la projection. En effet, deux cas de figure seulement sont à envisager :

- soit la projection était originalement exécutable, auquel cas elle n'a jamais eu le drapeau VM_MAYWRITE et ne pourra donc jamais être projetée en écriture.
- soit la projection n'était pas exécutable à l'origine, auquel cas elle n'a jamais eu le drapeau VM_MAYEXEC, et ne pourra donc jamais être re-projetée en exécution, qu'il y ait eu ou non accès en écriture entre temps.

Le seul traitement spécifique (non réalisé au sein de CLIP) à *PAX_MPROTECT* lors d'un appel *mprotect()* est la gestion d'une exception au contrôle de *VM_MAYWRITE* permettant de supporter les relocalisations ELF dans la section *.text (TEXTREL)*. Si l'option *CONFIG_PAX_NOELFRELOCS* n'a pas été activée, le premier *mprotect(...PROT_WRITE...)* d'une projection non anonyme possédant le drapeau *VM_MAYEXEC* entraîne une recherche d'en-têtes ELF dans le fichier associé à cette projection. Si le noyau détecte des en-têtes ELF dans ce fichier, comportant le drapeau *DT_TEXTREL* qui signale la présence de relocalisations dans le texte, le premier accès en écriture est autorisé par ajout temporaire du drapeau *VM_MAYWRITE*, ainsi que d'un drapeau spécifique *VM_MAYNOTWRITE* (non pris en compte par les fonctions standards du noyau). Le prochain appel *mprotect()* sans option *PROT_WRITE* sur cette même projection détectera le drapeau *VM_MAYNOTWRITE*, ce qui entraînera la suppression de *VM_MAYWRITE*. Cette exception est adaptée au comportement de l'éditeur de lien dynamique ELF (*ld-linux.so*), qui, en cas de *TEXTREL*, va temporairement projeter en écriture la section *.text* de l'exécutable (première projection, autorisée par l'exception à *PAX_MPROTECT*), afin d'y écrire les relocalisations, avant de supprimer les droits en écriture par un deuxième appel à *mprotect()* (ce qui supprime définitivement *VM_MAYWRITE*). L'ajout temporaire de *VM_MAYWRITE* n'est réalisé qu'en l'absence de drapeau *VM_MAYNOTWRITE*, et celui-ci n'est pas supprimé en même temps que *VM_MAYWRITE* lors du second *mprotect()*, ce qui garantit que seule la première projection en écriture sera autorisée. On notera bien que les noyaux CLIP sont normalement compilés avec l'option *PAX_NOELFRELOCS*, qui désactive ce traitement d'exception, mais peut s'avérer problématique pour des exécutables contenant effectivement des *TEXTREL*[5].

On peut enfin signaler que les différents traitements réalisés par *PAX_MPROTECT* n'ont de sens que lorsque l'un des mécanismes PaX permettant d'assurer la non-exécutabilité des zones mémoires de données (*PAGEEXEC* et *SEGMEXEC*) est aussi actif, et garantit que l'absence de drapeau *VM_EXEC* sur une projection interdit bien l'exécution de code dans cette projection. De ce fait, ces traitements *PAX_MPROTECT* ne sont en général activés que par la combinaison de *CONFIG_PAX_MPROTECT* et d'au moins l'une des deux options *CONFIG_PAX_SEGMEXEC* ou *CONFIG_PAX_PAGEEXEC*.

[5] Typiquement, des exécutables disponibles uniquement sous forme binaire, et qui n'ont de ce fait pas pu être corrigés par la distribution Gentoo qui sert de base à CLIP.

1.1.3 ASLR

L'option *CONFIG_PAX_ASLR* donne accès à plusieurs mécanismes de randomisation des différentes zones mémoires composant l'espace d'adressage des processus. Deux de ces options, *PAX_RANDMMAP* et *PAX_RANDUSTACK*, concernent l'espace d'adressage utilisateur, tandis que la dernière concerne uniquement l'espace noyau, et est décrite en 1.2.2. Les biais aléatoires introduits dans les espaces d'adressage de processus par ces deux options sont résumés dans la Figure 2, tandis que le Tableau 1 récapitule les nombres de bits aléatoires significatifs pour ces différents biais sous CLIP.

PAX_RANDMMAP

Cette première option active, pour tous les exécutables *ELF*[6] qui ne la désactivent pas explicitement (cf. 1.3), la randomisation de la base des projections *mmap*. Le biais aléatoire est déterminé lors de chaque appel *exec()*, par un appel à la fonction interne du noyau *get_random_bytes()*[7] lors de l'exécution de *fs/binfmt_elf.c:load_elf_binary()*. Ce biais est ajouté à l'adresse de base des projections mémoire par la fonction *include/linux/sched.h:arch_pick_mmap_layout()*, et est de plus sauvegardé dans un champ *delta_mmap* ajouté à la structure *mm_struct* associée au processus. Cette première randomisation ajoute un biais constant (mais aléatoire d'une exécution à l'autre) à toutes les projections réalisées à l'aide de l'appel système *mmap()*, en particulier les chargements de bibliothèques. Ce premier biais compte 16 bits significatif sous une architecture *x86* 32 bits sans *SEGMEXEC* (cas de CLIP), ou 15 bits sur la même architecture lorsque *SEGMEXEC* est actif.

De manière complémentaire, l'option *PAX_RANDMMAP* désactive (fonctions *arch_get_unmapped_area() et arch_get_unmapped_area_topdown()* de *mm/mmap.c*), pour les processus qui disposent de cette randomisation, la prise en compte du premier argument de tout appel *mmap()* sans option *MAP_FIXED*, c'est-à-dire l'adresse de projection souhaitée, de manière à toujours forcer la prise en compte du biais aléatoire. On notera en revanche que le comportement standard reste maintenu dans le cas d'appels *mmap()* avec l'option *MAP_FIXED*[8] : la projection est réalisée à l'adresse demandée par l'appelant, si cette adresse est disponible.

L'option *PAX_RANDMMAP* peut aussi introduire un deuxième biais aléatoire, indépendant du premier (c'est-à-dire généré par un autre appel à *get_random_bytes()*), dans l'adresse de chargement de l'exécutable principal (par ajustement du paramètre *load_bias* dans *fs/binfmt_elf.c:load_elf_binary()*). Ce traitement n'est réalisé normalement que dans le cas où cet exécutable est lui-même relocalisable, c'est-à-dire typiquement de type *ET_DYN* (aussi appelés PIE - *Position Independant Executable*), ce qui est le cas de la quasi-totalité des exécutables déployés au sein d'un système CLIP[9] du fait de la mise en oeuvre d'une chaîne de compilation durcie (cf. [TOOLCHAIN]). Ce biais est ajouté à l'adresse standard de chargement des exécutables *ELF* (0x08048000), et affecte la projection en mémoire de

---

6 On pourra remarquer que les exécutables d'un format autre que *ELF* ne sont en aucun cas randomisés. Cependant, les noyaux CLIP ne supportent dans tous les cas que les exécutables au format *ELF*.

7 Il est rappelé que cette fonction fournit du pseudo-aléa issu du « *pool* » non-bloquant du noyau, c'est-à-dire produit par un générateur logiciel initialisé par le « *pool* » bloquant, lui-même peuplé par les événements disque, clavier et réseau. Le résultat de *get_random_bytes()* est équivalent à une lecture de */dev/urandom* en couche utilisateur.

8 Il serait très complexe de supprimer cette exception à la randomisation (c'est-à-dire de supprimer le support de *MAP_FIXED*), dans la mesure où cette option est très largement utilisée, en particulier par l'interpréteur ELF */lib/ld-linux.so*, pour projeter les données d'une bibliothèque immédiatement après son code (qui est quant à lui chargé à une adresse aléatoire).

9 Les exceptions correspondant pour l'essentiel à des exécutables disponibles uniquement sous forme binaire, qu'il est impossible de recompiler en *PIE*.

segments de texte et de donnée de l'exécutable, ainsi que le début du tas du processus, ces différentes zones restant par ailleurs contiguës en mémoire. Le biais introduit dans ce cas est de même taille que celui introduit pour la base des projections *mmap*, soit 16 bits significatifs sur *x86* 32 bits sans *SEGMEXEC*, ou 15 bits avec.

Enfin, un troisième biais aléatoire indépendant est introduit au chargement de l'exécutable, cette fois dans l'adresse de fin du tas initial (paramètre *elf_brk* dans *fs/binfmt_elf.c:load_elf_binary()*). Ce dernier biais comprend *PAGE_SHIFT* bits significatifs, soit 12 bits sur une architecture *x86* 32 bits.

Ces différents biais de randomisation ne sont insérés qu'à condition que les marquages PaX de l'exécutable spécifient bien une randomisation de la base *mmap()* (drapeau *ELF PF_RANDMMAP*, cf. 1.3), et que la variable *sysctl kernel.randomize_va_space* (non spécifique à PaX) soit positionnée à une valeur non-nulle (ce qui est le cas par défaut, et toujours vrai sous CLIP, cf. 1.3).

Type de biais	Nombre de bits significatifs sous CLIP *x86*	Source du biais
Adresse de base des projections *mmap()* (*PAX_RANDMMAP*)	16	Générateur pseudo-aléatoire du noyau (PRNG)
Adresse de base d'un exécutable PIE (*PAX_RANDMMAP*)	16	PRNG
Taille initiale du tas (*PAX_RANDMMAP*)	12	PRNG
Sommet de la pile utilisateur (*PAX_RANDUSTACK*)	10	PRNG
Sommet de la pile utilisateur en contexte noyau (*PAX_RANDKSTACK*)	4	*Timestamp counter* du processeur courant

Tableau 1: Nombre de bits significatifs des aléas introduits par PAX_ASLR.

PAX_RANDUSTACK

Cette deuxième option réalise simplement une randomisation de la pile (utilisateur) de chaque processus affecté par *PAX_ASLR*, en rabaissant le sommet de cette pile d'un biais aléatoire, toujours issu de *get_random_bytes()*, lors de la création de l'espace mémoire du processus (dans *fs/exec.c:do_execve()*[10]). La taille du biais généré est au maximum d'une page, ce qui correspond à *PAGE_SHIFT* bits significatifs, auxquels il faut encore retirer la contrainte d'alignement à la taille d'un *void* *, qui est imposée pour le sommet de la pile. Sur une architecture *x86* 32 bits, le biais compte ainsi 10 bits significatifs (12 – 2).

[10] On notera qu'un autre biais aléatoire (plus important) est aussi généré par *fs/binfmt_elf.c:load_elf_binary()* lorsque *PAX_RANDUSTACK* est actif, et stocké dans le champ *delta_stack* de la structure *mm_struct* associée au processus. Ce champ n'est cependant pas utilisé en pratique, et n'affecte en rien l'espace d'adressage du processus. Il s'agit probablement d'une « scorie » d'une implémentation antérieure.

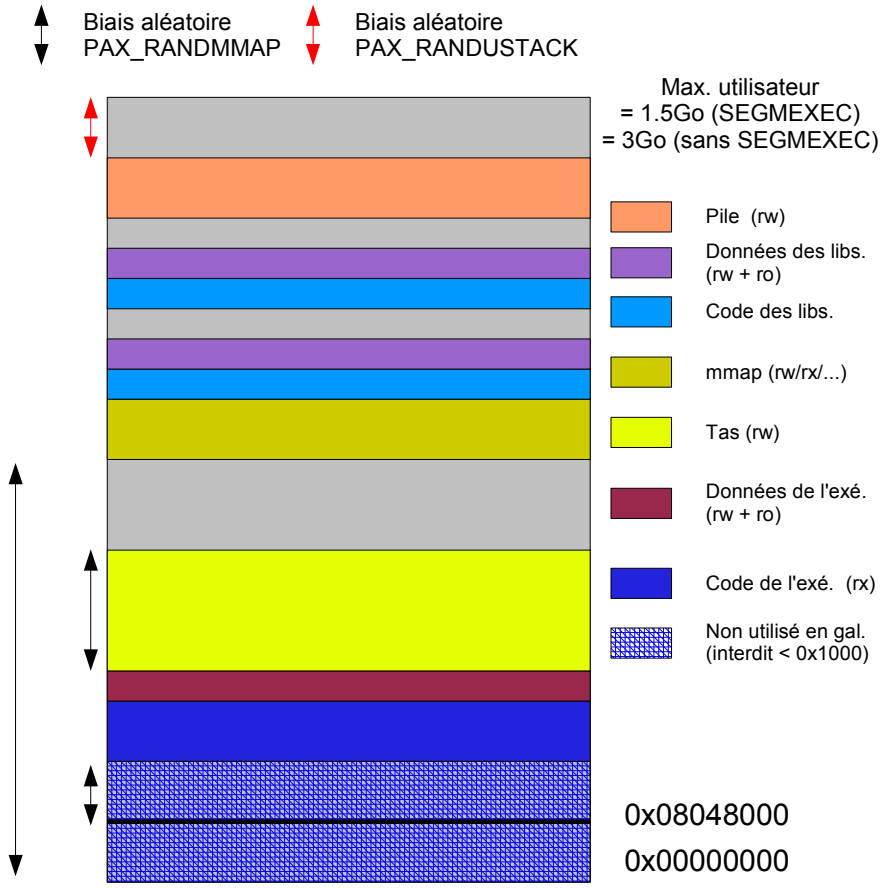

Figure 2: Biais aléatoires introduits dans l'espace d'adressage utilisateur par PAX_ASLR.

Ce biais aléatoire n'est ajouté que lorsque la variable *sysctl kernel.randomize_va_space* est positionnée à une valeur non-nulle (ce qui est toujours le cas sous CLIP, cf. 1.3). Contrairement à *PAX_RANDMMAP*, il n'existe pas de moyen de désactiver la randomisation de la pile pour un exécutable spécifique.

1.1.4 SANITIZE

L'option *PAX_MEMORY_SANITIZE* réalise une mise à zéro explicite de chaque page de mémoire physique dès qu'elle n'est plus allouée, afin de réduire potentiellement la durée de vie en RAM physique d'éventuelles informations sensibles. On notera que dans tous les cas, le noyau Linux standard met à zéro les pages physiques lors de leur réallocation. La mise à zéro des pages par *SANITIZE* est réalisée lors de l'appel à *mm/page_alloc.c:__free_one_page()*, en réalisant un *kmap_atomic()* de la page concernée à l'aide d'un *« slot » kmap* spécifiquement ajouté par PaX : *KM_CLEARPAGE*.

1.2 Mémoire du noyau

1.2.1 KERNEXEC

L'option *PAX_KERNEXEC* apporte le pendant des fonctionnalités *PAGEXEC* ou *SEGMEXEC* pour l'espace mémoire du noyau, ainsi que plusieurs mesures complémentaires de durcissement de l'espace noyau. Ces différentes mesures reposant largement sur les propriétés de la segmentation *x86*, *KERNEXEC* n'est disponible que sur cette architecture.

La non-exécutabilité des zones de données de la mémoire noyau est assurée par une réduction du segment de code noyau (*KERNEL_CS*), afin que ce dernier ne couvre que le code noyau, et non l'ensemble de l'espace d'adressage. Cette mesure apporte en outre une interdiction pour le noyau d'exécuter du code en espace utilisateur, y compris dans des pages exécutables[11] en mode utilisateur. Lorsque le noyau ne supporte pas le chargement de module, la limite supérieure du segment de code est simplement ramenée à la fin de la section *.text* du noyau (symbole *_etext*), tandis que la limite inférieure du segment est « remontée » au début du code[12]. Dans le cas contraire, une adaptation spécifique permet de réserver une plage mémoire exécutable en mode noyau initialement libre, afin d'y charger les sections de code d'éventuels modules. Cette adaptation repose sur une modification du script de configuration utilisé pour l'édition de liens noyau (*arch/i386/kernel/vmlinux.lds.S*), afin de créer après la section *.text* du noyau[13] une section *.module.text* initialement vide, d'une taille prédéfinie à 4 Mo. Dans ce cas, le segment de code noyau est étendu jusqu'à la fin de cette section. Des fonctions spécifiques, *module_alloc_exec()* et *module_free_exec()*, sont ajoutées (*arch/i386/kernel/module.c*) afin de permettre au code de chargement de module l'allocation et la désallocation dynamiques de mémoire exécutable dans cette section, pour y charger les sections de code des modules.

De manière complémentaire, *KERNEXEC* interdit tout accès en écriture à plusieurs zones de la mémoire noyau, en retirant les permissions en écriture des pages correspondantes. Les éléments ainsi protégés sont :

- toute la mémoire noyau avant le début de la section *.data*, ce qui couvre notamment les sections *.text*, *.rodata.page_aligned* et *.tracedata*, et le cas échéant *.module.text,*
- la table de pages du noyau,
- la table des descripteurs d'interruptions (IDT),
- les tables globales de descripteurs de segments (GDT)

La table d'appels systèmes (*sys_call_table*, *arch/i386/kernel/syscall_table.S*) est par ailleurs (par le patch PaX en général, indépendamment de l'option *KERNEXEC*) définie dans la section *.rodata*, plutôt

[11] Par exemple dans le cas d'une redirection du flot de contrôle de l'exécution d'un appel système vers des sections de code utilisateur, suite à l'écrasement d'un pointeur de fonction dans les données du noyau. Une telle attaque resterait possible (sous réserve de vulnérabilité offrant la possibilité d'écraser un pointeur noyau), avec pour conséquence l'exécution de code utilisateur en mode noyau (*ring 0*) si la non-exécutabilité des données noyau était assurée uniquement par la pagination, typiquement à l'aide d'un drapeau NX.

[12] Ce qui introduit un biais dans les adresses linéaires du code noyau, qui est géré par une modification du script d'édition de liens.

[13] En réalité, la section *.modules.text* n'est pas déclarée immédiatement après la section *.text,* mais après les sections suivantes *.tracedata* et *.rodata.page_aligned*, qui sont de ce fait laissées exécutables en mode noyau.

que simplement *.data*. Ainsi, l'activation de *KERNEXEC* assure indirectement la protection de cette table contre les accès en écriture.

On notera qu'il reste nécessaire pour le noyau d'accéder occasionnellement à certaines des zones mémoire ainsi protégées, par exemple pour charger un module ou modifier l'IDT ou la GDT. Les fonctions réalisant ces accès légitimes sont modifiées par l'option *PAX_KERNEXEC*, de manière à désactiver temporairement la protection en écriture sur le processeur courant, en remettant à zéro le bit *WP* (bit 16) du registre *CR0*. Ainsi, les accès en écriture légitimes (c'est-à-dire identifiés a priori dans le code noyau) peuvent passer outre la protection en écriture des pages qu'ils modifient. La désactivation et la réactivation de la protection en écriture sont réalisées dans ces cas particuliers par un couple de fonctions spécifiques à PaX, *pax_open_kernel()* et *pax_close_kernel()*.

1.2.2 ASLR - RANDKSTACK

Cette option, supportée uniquement sous l'architecture *x86*, retire un biais « aléatoire » à l'adresse du sommet de la pile de chaque processus en espace mémoire noyau (pile d'une tâche en contexte d'appel système). Ce biais compte quatre bits significatif (plus deux bits d'alignement), générés à partir des bits de poids faibles du *time stamp counter* (compteur de *ticks* processeur depuis le dernier *reset*), lu au moment de la génération du biais. Il n'est donc pas réellement aléatoire. Un nouveau biais est tiré lors de chaque appel système, indépendamment du processus. Ce traitement n'est réalisé que si le *sysctl kernel.randomize_va_space* est positionné à une valeur non nulle (ce qui est toujours le cas sous CLIP, cf. 1.3).

1.2.3 UDEREF

L'option *PAX_UDEREF* apporte un cloisonnement strict, vis-à-vis des accès en lecture-écriture, des espaces mémoire noyau et utilisateur, afin d'éviter le déréférencement par le noyau d'un pointeur utilisateur alors qu'un pointeur noyau est attendu, ou réciproquement le déréférencement d'un pointeur noyau alors qu'un pointeur utilisateur est attendu[14]. Ce cloisonnement repose sur la segmentation *x86*, et l'option n'est supportée que sur cette architecture. Lorsque l'option *PAX_UDEREF* est activée, la limite supérieure de *__USER_DS* (segment de données utilisateur, chargé dans *%ss, %ds* et *%es* en mode utilisateur) est abaissée à *__PAGE_OFFSET* (typiquement, 3Go) au lieu du sommet de l'espace d'adressage, de manière à ne plus couvrir la plage d'adresses réservée au noyau, tandis que la limite inférieure de *__KERNEL_DS* (segment de données noyau, chargé dans *%ss, %ds* et *%es* en mode noyau) est remontée à *__PAGE_OFFSET*, de manière à ne plus couvrir la plage d'adresses utilisateur. Cette première modification permet d'interdire tout déréférencement de pointeurs utilisateur par le code noyau.

Une seconde adaptation est réalisée par *PAX_UDEREF* afin de préserver la possibilité d'effectuer des copies entre espaces mémoire noyau et utilisateur à l'aide uniquement des fonctions légitimes pour de telles copies (par exemple *copy_from_user()* et *copy_to_user()*). L'adaptation consiste dans ce cas à charger temporairement *__USER_DS* dans l'un des registres de segment (source ou destination, selon le sens du transfert) utilisés par la fonction de copie, afin de rendre la copie valide. Enfin, une dernière adaptation est rendue nécessaire par le fait que certaines de ces mêmes fonctions de copie entre espaces

[14] Un tel déréférencement non contrôlé peut typiquement être exploité, dans le premier cas (pointeur utilisateur utilisé à la place d'un pointeur noyau) pour réaliser une escalade de privilèges ou faire fuir des données noyau, et dans le second cas pour modifier de manière arbitraire la mémoire noyau.

utilisateur et noyau peuvent exceptionnellement être appelées afin de réaliser une copie purement interne à l'espace noyau, lorsque certaines fonctions de l'interface d'appels système sont appelées directement par le noyau (par exemple, le code associé au traitement de *sys_fork*, appelé aussi lors de la création d'un nouveau *thread* noyau). Afin d'éviter une faute de segmentation dans ce cas, due au fait que *__USER_DS* utilisé comme source ou destination ne permet pas l'accès à une source ou destination en mémoire noyau, la fonction *set_fs()* est redéfinie (*arch/i386/lib/usercopy.c*) de manière à ajuster temporairement la limite supérieure de *__USER_DS* en plus de la limite d'adresse utilisée spécifiquement par les fonctions de copie (*current_thread_info()->addr_limit,* qui ne correspond normalement à aucune limite matérielle). Ainsi, un appel *set_fs(KERNEL_DS)* permet bien une copie interne à l'espace noyau à l'aide des fonctions de copie entre espaces mémoire, avec la spécificité qu'un *__USER_DS* adapté est employé à la place de *__KERNEL_DS* comme source ou destination. On notera bien que cette solution laisse une fenêtre de vulnérabilité aux déréférencements illégitimes de pointeurs noyau (au lieu d'utilisateur) entre l'appel à *set_fs()* et le rétablissement du *__USER_DS* d'origine.

1.3 Configuration dynamique de PaX

Au-delà de la configuration globale et statique du noyau, différents mécanismes PaX peuvent être sélectivement activés ou désactivés pour chaque exécutable, grâce à la possibilité d'inclure dans les en-têtes *ELF* un masque d'options spécifiques à PaX. Ce masque peut-être stocké de deux manières distinctes, selon les options configurées dans le noyau :

- Dans une zone réservée de l'en-tête *ELF* standard, lue par le noyau si l'option *CONFIG_PAX_EI_PAX* a été sélectionnée. Ce mode de fonctionnement est considéré comme obsolète.
- Dans un en-tête *ELF* spécifique à PaX, *PT_PAX_FLAGS,* lu par le noyau si l'option *CONFIG_PT_PAX_FLAGS* a été sélectionnée. Ce mode de fonctionnement nécessite une adaptation de l'éditeur de lien utilisé pour la génération des exécutables, afin de leur ajouter cet en-tête.

Seul le deuxième mode est supporté par les noyaux CLIP. Les en-têtes *PT_PAX_FLAGS* sont automatiquement ajoutés par la chaîne de compilation *Gentoo Hardened* (cf. [TOOLCHAIN]) utilisée pour la compilation des paquetages *CLIP.* Les exécutables qui ne sont pas recompilés avant leur inclusion dans CLIP (typiquement, des applications propriétaires) peuvent se voir si nécessaire ajouter a posteriori un en-tête *PT_PAX_FLAGS* à l'aide d'une commande *paxctl -C* (appelé automatiquement par *portage* au besoin, cf. [CLIP_1101]).

Dans un mode de fonctionnement de PaX dit « *hard mode* », les options PaX activées dans la configuration noyau sont toutes appliquées par défaut à chaque exécutable (y compris en l'absence de tout en-tête PaX). Certains bits du masque d'options PaX d'un exécutable permettent de désactiver certaines options uniquement pour les processus exécutant cet exécutable. C'est ce mode de fonctionnement qui est employé dans CLIP. Alternativement, un mode dit *« soft mode »* est aussi supporté par PaX, et activé quand, d'une part, l'option *PAX_SOFTMODE* a été sélectionnée à la compilation du noyau (ce qui n'est pas le cas sur un noyau CLIP), et, d'autre part, l'argument *pax_softmode=1* est passé au noyau au démarrage ou la variable *sysctl kernel.pax.softmode*[15] est positionnée à une valeur non-nulle. Dans ce cas, aucune option PaX n'est appliquée par défaut, et seuls

[15] Variable qui n'existe que si *PAX_SOFTMODE* a été sélectionnée.

les exécutables qui demandent explicitement l'application de protections PaX par le positionnement de bits dans leur masque d'options PaX bénéficieront de ces protections. Les différents bits significatifs du masque d'options PaX sont résumés dans le Tableau 2.

Option PaX	Signification	Lettre-clé	Position par défaut dans CLIP
PF_PAGEEXEC	Activer *PAGEXEC* (mode *soft*)	P	0 (ignoré)
PF_NOPAGEEXEC	Désactiver PAGEXEC	p	0
PF_SEGMEXEC	Activer *SEGMEXEC* (mode *soft*)	S	0 (ignoré)
PF_NOSEGMEXEC	Désactiver *SEGMEXEC*	s	0 (ignoré – *SEGMEXEC* non supporté)
PF_MPROTECT	Activer *MPROTECT* (mode *soft*)	M	0 (ignoré)
PF_NOMPROTECT	Désactiver *MPROTECT*	m	0
PF_EMUTRAMP	Activer *EMUTRAMP* (mode *soft*)	E	0 (ignoré)
PF_NOEMUTRAMP	Désactiver *EMUTRAMP*	e	1 (ignoré – *EMUTRAMP* non supporté)
PF_RANDMMAP	Activer RANDMMAP (mode *soft*)	S	0 (ignoré)
PF_NORANDMMAP	Désactiver *RANDMMAP*	s	0

Tableau 2: Signification des options PaX et positionnement par défaut dans CLIP.

Les options d'activation de mécanismes PaX ne sont utilisées qu'en mode « soft ». Ce mode n'étant pas supporté par les noyaux CLIP, ces options sont systématiquement ignorées.

Par ailleurs, ces mêmes options PaX peuvent être contrôlées dynamiquement par un mécanisme de contrôle d'accès obligatoire (MAC), comme par exemple les ACL Grsecurity (cf. 2.5). PaX fournit à cette fin une interface aux développeurs de tels mécanismes, par l'intermédiaire d'une fonction *pax_set_initial_flags()* (option *PAX_HAVE_ACL_FLAGS*) ou d'un pointeur de fonction *pax_set_initial_flags_fun()* (option *PAX_HOOK_ACL_FLAGS*), qui peuvent être définis par le système de contrôle d'accès. Cette fonction est appelée lors du chargement d'un exécutable (*fs/binfmt_elf.c:load_elf_binary()*), en lui passant en argument la structure *linux_bprm* associée à l'exécutable, et est en mesure de positionner ou non les différents drapeaux PaX décrits dans le Tableau 2. Grsecurity définit ainsi une fonction *pax_set_initial_flags()*, permettant d'attribuer ou de retirer des options PaX aux processus en fonction de la base d'ACL Grsecurity. Ce mécanisme peut cependant être entièrement désactivé (auquel cas aucune fonction *pax_set_initial_flags()* n'est jamais appelée), en sélectionnant l'option de compilation *PAX_NO_ACL_FLAGS*. Dans la mesure où les ACL Grsecurity ne sont pour l'heure pas mises en oeuvre sous CLIP, cette option est activée dans les noyaux CLIP de

manière à interdire toute modification dynamique des drapeaux PaX.

Enfin, les différents mécanismes de randomisation mémoire de PaX (*RANDMMAP, RANDUSTACK* et *RANDKSTACK*) peuvent être globalement désactivés par mise à zéro du *sysctl kernel.randomize_va_space*. Un patch spécifique à CLIP désactive entièrement ce *sysctl* dès lors que *CONFIG_PAX_ASLR* est définie (ce qui est le cas dès qu'une option de randomisation PaX est active). Il est ainsi impossible sous CLIP de désactiver la randomisation mémoire, à part par modification des en-têtes d'un exécutable.

1.4 Journalisation PaX

PaX réalise une journalisation spécifique des attaques potentielles détectées par ses mécanismes de protection. Cette journalisation signale notamment :

- Les tentatives d'exécution dans une projection mémoire non exécutable, accompagnés du nom commun de la tâche fautive, de celui du fichier exécutable associé, des *euid* et *uid* du processus fautif, et des adresses de début et de fin de la projection où s'est produite la faute (dépend de *PAGEEXEC* ou *KERNEXEC*, définition dans *fs/exec.c:pax_report_fault()*), suivie d'un *dump* des vingt premiers octets situés après le *program counter* et le *stack pointer*.
- Les tentatives d'accès en écriture au code noyau[16], accompagnées du nom commun, du *pid* et des *euid* et *uid* de la tâche fautive, et de l'adresse de la faute (dépend de *KERNEXEC*, définition dans *arch/i386/mm/fault.c*).
- Les erreurs de segmentation causées par du code noyau, typiquement révélatrices d'une violation *KERNEXEC* ou *UDEREF* (dépend de *KERNEXEC*, définition dans *arch/i386/kernel/traps.c*).

Ces différents messages sont préfixés dans les journaux noyau du mot-clé « PAX: ». Le démon syslog-ng est configuré dans CLIP pour collecter ces messages dans un fichier distinct, /var/log/pax.log (/log/pax.log dans la cage $AUDIT_{clip}$).

Les protections PaX actives pour une tâche donnée peuvent par ailleurs être consultées dans le champ *PaX* de */proc/<pid>/status*, avec *<pid>* le *pid* de la tâche. Ce champ contient une chaîne de lettres-clés, comme décrites dans le Tableau 2, représentant les options effectivement actives ou inactives pour la tâche considérée.

[16]Qui sont ainsi distinguées des autres fautes de protection noyau, généralement rapportée sous la forme *« BUG: unable to handle kernel paging request »*.

2 Grsecurity

2.1 Compléments à la protection mémoire

Grsecurity apporte plusieurs mesures de durcissement complémentaires des protections mémoire offertes par PaX. Ces compléments concernent principalement la randomisation mémoire et la protection de l'espace mémoire du noyau.

2.1.1 Compléments à la randomisation mémoire

Masquage des adresses dans *proc*

L'option *GRKERNSEC_PROC_MEMMAP* supprime l'affichage des adresses de projections mémoire des différents fichiers du */proc* qui les incluent normalement, lorsque ces fichiers sont lus par une tâche autre que celle à laquelle ils font référence. Cette mesure vise à éviter le contournement de la randomisation de l'espace mémoire des processus, dès lors que l'attaquant est en mesure d'exécuter des commandes au sein du système[17]. Plus précisément, l'option a les effets suivants, pour toute tâche utilisateur[18], de *pid* (*tgid* en pratique sous Linux) <*pid*>, et protégée par l'une ou l'autre des options *PAX_RANDMMAP* ou *PAX_SEGMEXEC* (on notera que *PAX_RANDUSTACK* seule n'est pas prise en compte) :

- Suppression des adresses de début et de fin, et des tailles, de toutes les projections mémoire affichées dans */proc/<pid>/maps*. Les champs correspondant sont remplacés par des zéros.
- Suppression des informations *eip, esp* et *wchan* (adresse d'attente de la tâche au sein du noyau, si la tâche est bloquée sur un appel système), ainsi que des adresses de début et de fin de code et de l'adresse de début (sommet) de la pile, du fichier */proc/<pid>/stats*. Les champs correspondants sont remplacés par des zéros.

Ces restrictions ne sont appliquées que lorsque la tâche qui consulte les fichiers du */proc* a un espace mémoire différent de celui consulté. Ainsi, un processus (ou le cas échéant, tout *thread* d'un processus donné) peut toujours consulter les informations qui le concernent, y compris à travers */proc/self/{maps,stats}*. Cette exception n'a pas d'impact réel sur la sécurité, dans la mesure où le modèle d'attaque contré est bien l'attaque d'un processus par un autre, et non une attaque interne au processus (qui pourrait de toutes façons obtenir les informations recherchées par d'autres moyens, par exemple lecture de registres). Par ailleurs, une telle exception est nécessaire au fonctionnement de certaines applications, par exemple certaines machines virtuelles Java.

Protection contre les attaques en force brute

La randomisation mémoire des processus utilisateur n'est réalisée par PaX que lors d'un appel *exec()*, mais pas lors d'un *fork()*. De ce fait, plusieurs processus créés par *fork()* à partir d'un processus maître,

[17] Ce cas de figure correspond typiquement à celui d'un attaquant qui serait déjà parvenu à exécuter du code arbitraire non privilégié au sein du système, et qui tenterait d'exploiter localement une deuxième vulnérabilité dans un processus plus privilégié, afin de réaliser une escalade de privilèges.

[18] Les *threads* noyaux ne sont pas affectés par cette option, car il n'ont de toutes façons pas d'espace mémoire propre.

et qui exécutent le même code que ce dernier, partageront tous les mêmes biais aléatoires dans les différentes adresses randomisées par PaX. Cette propriété expose potentiellement des démons *« forkant »* (c'est à dire des démons en écoute, qui créent un fils par *fork()* pour traiter chaque nouvelle connexion, par exemple *apache* ou *sshd*) à une attaque en force-brute contre la randomisation mémoire : il suffit à l'attaquant de lancer une nouvelle connexion pour obtenir une nouvelle copie de l'espace mémoire à attaquer, sur laquelle il pourra lancer une attaque (qui entraînera en cas d'échec la terminaison du processus fils, mais pas du processus maître). Afin de contrer, ou du moins de ralentir de telles attaques, Grsecurity propose une option *GRKERNSEC_BRUTE*, dont le rôle est d'introduire dans le cas d'une telle attaque supposée, un délai incompressible de 30 secondes dans les appels à *fork()* réalisés par le processus maître. Plus concrètement, cette option repose sur l'ajout d'un champ spécifique, *brute*, aux structures *task_struct*. Ce champ est normalement laissé à zéro. Lorsqu'un processus se termine anormalement, suite à la réception d'un signal *SIGKILL* ou *SIGILL*[19], la fonction *fs/exec.c/do_coredump()* appelle *grsecurity/grsec_sig.c:gr_handle_brute_attach()*, qui va à son tour comparer le fichier exécuté par le processus fautif et par son père. Si ce fichier est le même pour le père et le fils (cas typique d'un *fork()* non suivi par un *exec()*), la fonction met à un le champ *brute* de la structure *task_struct* du père. Dans le cas contraire, aucun traitement n'est réalisé. Par ailleurs, la fonction *grsecurity/gr_sig.c:gr_handle_brute_check()*, appelée lors de tout appel *fork()*, vérifie ce même champ *brute*, et bloque l'appel pendant une durée minimale de 30 secondes (sommeil en état *TASK_UNINTERRUPTIBLE*) lorsque ce champ est non nul.

On notera que la fonction *fs/exec.c:do_coredump()* est systématiquement appelée en cas de terminaison imprévue d'un processus, même si aucun *coredump* n'est finalement généré. En revanche, le positionnement de l'appel *gr_handle_brute_attach()*, tel qu'il est réalisé par le patch Grsecurity, ne permet pas l'appel de cette fonction dans le cas où aucun *coredump* ne doit être généré, soit parce que ces derniers ne sont pas supportés (désactivation de l'option *ELF_CORE_DUMP* à la compilation, ce qui est le cas d'un noyau CLIP), soit parce que la tâche fautive n'est pas *dumpable*. Un patch spécifique à CLIP vient corriger ce problème, qui empêcherait sinon toute mise en oeuvre sur CLIP de ces protections contre les attaques en force brute.

Augmentation du *pool* d'aléa noyau

L'option *GRKERNSEC_RANDNET* permet de quadrupler la taille des réservoirs (*pools*) d'entropie utilisés par le générateur de nombres pseudo-aléatoires du noyau pour stocker l'entropie « matérielle » issue des événements clavier, réseau et disque. Ainsi, la taille du *pool* de collecte est portée à 2048 bits au lieu de 512 bits, tandis que celles des deux *pools* de sortie (utilisés par le générateur bloquant et le générateur non bloquant) est portée à 512 bits au lieu de 128. Ces modifications ont pour objectif la prise en compte des besoins accrus en génération de nombres aléatoires amenés notamment par la randomisation mémoire.

[19] Ces deux signaux sont des conséquences probables d'une mauvaise tentative de deviner les adresses d'un processus, qui entraînerait l'exécution de données qui ne correspondent pas à des instructions valides. On notera que *SIGSEGV* n'est en revanche pas pris en compte par *GRKERNSEC_BRUTE,* alors qu'il constitue une autre conséquence probable d'un mauvais essai dans le cadre d'une attaque par force brute.

2.1.2 Compléments à la protection de l'espace noyau

Protection contre les entrées-sorties privilègiées

Les possibilités de modifications arbitraires de l'espace mémoire noyau depuis la couche utilisateur par patch binaire de l'image mémoire noyau sont contrées par une option *GRKERNSEC_KMEM*. Cette option est complémentaire de l'option *PAX_KERNSEC*, dans la mesure où cette dernière interdit bien l'injection de code arbitraire dans le noyau (les pages exécutables sont non inscriptibles) ou la modification du code existant (les pages inscriptibles sont non exécutables), mais ne bloque en revanche pas la modification des données du noyau. Il est ainsi possible, même avec *PAX_KERNSEC*, de modifier une structure des données noyau contenant des pointeurs de fonctions, pour faire pointer ces derniers vers d'autres fonctions arbitraires du noyau. *GRKERNSEC_KMEM* supprime les possibilités de réaliser de telles opérations depuis la couche utilisateur en interdisant de manière systématique les accès suivants :

- écriture sur */dev/mem*
- écriture sur */dev/kmem*
- projection mémoire avec les droits en écriture de */dev/mem* et */dev/kmem*, à l'exception des plages correspondant à la mémoire vidéo/ISA (adresses physiques *0x000a0000* à *0x00100000* et *0x00000000* à *0x00001000*). Cette exception est maintenue afin de permettre le fonctionnement d'un serveur X11 standard.
- Ouverture de */dev/port* (indépendamment des droits d'accès demandés)

Ces différentes interdictions sont inconditionnelles, contrairement au fonctionnement sans *GRKERNSEC_KMEM*, qui autorise ces opérations si l'appelant dispose de la capacité *CAP_SYS_RAWIO*. Elles sont de plus accompagnées de la génération d'un message *grsec* dans le journal noyau (cf. 2.4).

Par ailleurs, une deuxième option, *GRKERNSEC_IO*, vient interdire tout accès aux ports permettant le pilotage à bas niveau des périphériques (en mode *PIO*), en rendant inconditionnellement une erreur sur les appels *sys_iopl()* et *sys_ioperm()* qui augmenteraient le niveau d'accès de l'appelant à ces ports[20]. En effet, l'accès à ces ports permet par plusieurs techniques la modification arbitraire de la mémoire noyau (en contournant au besoin les mécanismes de pagination et de segmentation (cf. notamment [SMM]). Cette interdiction n'est, contrairement au contrôle d'accès normalement réalisé par le noyau sur ces appels système, pas contournable par la capacité *CAP_SYS_RAWIO*. Elle est de plus accompagnée de la génération d'un message *grsec* dans le journal noyau. L'option *GRKERNSEC_IO* n'est normalement pas compatible de l'utilisation d'un serveur X11, qui a besoin d'un accès complet par *iopl()*. Cependant, le serveur X11 utilisé dans CLIP est adapté de manière à ne pas nécessiter ces privilèges (cf. [CLIP_1303]), ce qui permet l'activation de cette option sur les clients CLIP.

Enfin, un patch spécifique à CLIP vient étendre le rôle de *GRKERNSEC_IO*, qui complète dans ce cas *GRKERNSEC_KMEM*, en interdisant de manière inconditionnelle toute ouverture de */dev/mem* et */dev/kmem* (ouverture qui resterait sinon autorisée avec *CAP_SYS_RAWIO*[21]), y compris en lecture, et en

[20] C'est-à-dire *sys_iopl(level)* avec *level* plus grand que le niveau courant, ou *sys_ioperm*() utilisé pour augmenter les permissions (troisième paramètre non null).

[21] Cette mesure relève essentiellement de la défense en profondeur, dans la mesure où *CAP_SYS_RAWIO* est normalement masquée sur CLIP par le mécanisme du *cap_bound* (cf. [CLIP_1201]). L'interdiction des accès en lecture, et non seulement en écriture, à ces périphériques est essentielle pour préserver la confidentialité des clés cryptographiques

supprimant l'exception autorisée par *GRKERNSEC_KMEM* sur les accès par *mmap()* à la plage de mémoire vidéo. Ces interdictions sont aussi accompagnées d'une journalisation *grsecurity* des tentatives d'accès. Là encore, ces restrictions ne permettent pas le fonctionnement d'un serveur graphique X11 standard, mais restent compatibles avec le serveur X11 modifié qui est utilisé au sein des systèmes CLIP.

Désactivation du chargement de modules

L'option *GRKERNSEC_MODSTOP* permet de désactiver dynamiquement le chargement et le déchargement de modules noyau. Lorsque cette option (disponible uniquement si le noyau supporte le chargement de modules) est activée, la fonction *gr_handle_modstop()*, appelée au chargement et au déchargement de modules (*sys_init_module()* et *sys_delete_module()* dans *kernel/module.c*) interdit ces opérations dès lors qu'une variable *grsec_modstop* est non nulle, indépendamment de la capacité *CAP_SYS_MODULE*, qui permet normalement ces opérations. La variable *grsec_modstop,* initialement nulle, peut être mise à un par le *sysctl kernel.grsecurity.disable_modules*, et ne peut ensuite plus être modifiée. Il est ainsi possible, lors du démarrage du système, de désactiver définitivement les opérations sur les modules après une phase initiale de chargement de modules.

Cette option n'est pas utilisée dans les configurations noyau CLIP entièrement statiques. Elle l'est en revanche dans les configurations modulaires, où elle permet à l'*initrd* utilisé dans ce cas de désactiver le chargement de modules avant de passer la main à *init*, sans pour autant nécessiter le masquage immédiat de *CAP_SYS_MODULE* (cf. [CLIP_1301]).

Protection des symboles noyau

Plusieurs options Grsecurity permettent de limiter les informations sur l'espace d'adressage noyau qui sont consultables par l'interface */proc* depuis la couche utilisateur. On notera bien que de telles mesures constituent un mécanisme de défense en profondeur relativement faible, dans la mesure où les informations protégées (adresses des fonctions du noyau) ne font l'objet d'aucune randomisation spécifique, et sont de ce fait les mêmes pour tous les postes CLIP dotés d'une version donnée du noyau CLIP.

Une première option*, GRKERNSEC_HIDESYM*, interdit l'affichage des *eip, esp,* et *wchan* des tâches dans */proc/<pid>/stats*, et ce de manière inconditionnelle (contrairement à *GRKERNSEC_PROC_MEMMAP*, qui ne masque pas ces informations pour les *threads* noyaux, ni pour les processus dont l'espace d'adressage n'est pas randomisé, cf. 2.1.1). On notera que contrairement à ce que pourrait laisser croire l'aide fournie par l'outil de configuration noyau pour cette option, *GRKERNSEC_HIDESYM* n'empêche pas la lecture du fichier */proc/kallsyms*, qui affiche tous les symboles du noyau courant[22].

Par ailleurs, l'option *GRKERNSEC_PROC_ADD* (aussi décrite en 2.2.2) interdit la création du fichier */proc/kcore*, qui exposerait sinon l'espace mémoire du noyau à la couche utilisateur, et permettrait notamment la lecture des adresses de symboles noyaux.

manipulées par le noyau (chiffrement de disque et IPsec notamment).

[22] Cette lecture n'est cependant pas permise dans CLIP, car le fichier n'est même pas créé, l'option *CONFIG_KALLSYMS* n'étant pas activée dans les noyaux CLIP. Cette protection en confidentialité – avec un objectif de défense en profondeur – est complétée par le masquage du fichier image du noyau et de son *System.map*, qui sont stockés sur une partition distincte, non montée et dont le montage n'est en aucun cas permis par *devctl* (cf. [CLIP_1201]).

2.1.3 Autres compléments

Désallocation des segments de mémoire partagée

L'option *GRKERNSEC_SHM*, lorsqu'elle est activée, entraîne la destruction immédiate des segments de mémoire partagée qui ne sont plus utilisés par aucun autre processus, lorsque leur processus créateur (identifié par le champ *shm_cprid* des *struct shmid_kernel* associées à ces segments) se termine. Ces segments restent présents en mémoire sur un noyau standard s'ils ne sont pas explicitement détruits par un processus. Cette mesure peut être vue comme un complément à l'option *PAX_SANITIZE* (1.1.4).

2.2 Contrôle d'accès supplémentaire sur les fichiers

Plusieurs options Grsecurity viennent compléter les permissions UNIX classiques pour réaliser un contrôle d'accès plus strict sur les fichiers. Ces options activent des mécanismes génériques, qui ne nécessitent aucune configuration spécifique. Elles peuvent cependant être complétées par des ACL Grsecurity propres à chaque sujet et exécutable (cf. 2.5).

2.2.1 Accès en exécution

L'option *GRKERNSEC_TPE* active un mécanisme de *Trusted Path Execution*, permettant de restreindre l'exécution de fichiers. Quand cette option est sélectionnée, il devient possible d'interdire l'exécution de fichiers installés dans un répertoire qui n'appartient pas à *root* ou qui est inscriptible par des utilisateurs autres que *root*. Ces restrictions s'appliquent aussi bien à l'exécution proprement dite qu'à la projection en mémoire[23] avec des droits en exécution (donc en particulier au chargement de bibliothèques dynamiques). Les accès refusés en exécution sont journalisés. Le principal objectif du mécanisme TPE est d'assurer une forme d'exclusivité des droits en écriture et en exécution sur les fichiers[24], afin d'éviter la pérennisation d'une attaque par installation locale de code malveillant.

Le mode de fonctionnement par défaut de TPE consiste à n'appliquer ces restrictions que lorsque l'utilisateur appelant *exec()* appartient à un groupe *Untrusted* dont le *gid* est défini à la compilation du noyau. L'option *GRKERNSEC_TPE_INVERT* permet d'inverser cette condition, et d'appliquer ces restrictions à tout utilisateur qui n'est pas membre d'un groupe *Trusted*. L'option *GRKERNSEC_TPE_ALL* généralise les restrictions à tout utilisateur autre que *root*, indépendamment

[23] La vérification des propriétés TPE sur les projections *mmap* est réalisé aussi bien lors d'un *mmap(...PROT_EXEC...)* que d'un *mprotect(...PROT_EXEC...)*, respectivement par *grsecurity/gracl.c/gr_acl_handle_mmap()* et *grsecurity/gracl.c/gr_acl_handle_mprotect()*, et ce même lorsque les ACL Grsecurity ne sont pas utilisées. On notera que les vérifications ainsi réalisées le sont juste après les appels aux *hooks* LSM présents dans ces fonctions, donc avant que le LSM CLIP ne désactive l'accès en écriture aux fichiers sous-jacents (cf. [CLIP_1201]). De ce fait, TPE joue aussi un rôle dans la prévention des dénis de service par projection exécutable de fichiers, qui constituent un risque potentiel introduit par le LSM CLIP.

[24] On peut souligner qu'une telle mesure relève au sein de CLIP essentiellement de la défense en profondeur, dans la mesure où l'exclusivité des droits en écriture et en exécution est généralement assurée dans ce système par les options de montage, et ce indépendamment de l'utilisateur concerné. TPE peut cependant s'avérer utile, non seulement comme protection de secours en cas d'erreur ou de vulnérabilité dans la prise en compte des options de montage, mais aussi au sein des cages et vues de mise à jour, où les options de montage ne peuvent pas assurer une exclusivité totale des droits en écriture et en exécution – du fait de la nécessité d'exécuter les *maintainers scripts* des paquetages installés.

des groupes. Un patch spécifique à CLIP ajoute une option plus stricte encore, *GRKERNSEC_TPE_STRICT*, qui applique les restrictions TPE à tout utilisateur, y compris *root*, de manière indiscriminée. Ainsi, l'exécution d'un fichier par un un utilisateur quelconque n'est possible que si le fichier est propriété de *root*, dans un répertoire propriété de *root*, et que répertoire et fichier ne sont inscriptibles que par *root*. Cette dernière option, qui est celle sélectionnée par défaut dans les noyaux CLIP, a pour vocation de protéger aussi le système contre les possibilités d'escalade de privilèges par chargement de bibliothèque malicieuse dans un processus *root*, ou redirection d'un appel *exec()* réalisé par un processus privilégié vers un fichier malicieux sous contrôle de l'attaquant.

2.2.2 Accès en lecture-écriture

Outre le contrôle des accès en exécution par le mécanisme TPE, plusieurs options Grsecurity permettent d'imposer des restrictions supplémentaires à certains accès en général, afin de contrer plusieurs familles d'attaques classiques en environnement UNIX.

Contrôle d'accès dans */proc*

L'option *GRKERNSEC_PROC* réalise un durcissement des permissions accordées sur des fichiers du */proc* qui sont normalement accessibles en lecture (voire lecture-écriture) à tous les utilisateurs du système. Le comportement exact est défini par le choix de l'une ou l'autre des sous-options liées : *GRKERNSEC_PROC_USER* limite en général l'accès au seul utilisateur propriétaire (aucune permission pour le groupe ou les autres utilisateurs), tandis que *GRKERNSEC_PROC_USERGROUP* limite l'accès à l'utilisateur propriétaire et aux membres d'un groupe « inspecteur » dont le *gid* est défini statiquement lors de la compilation. L'option plus restrictive, *GRKERNSEC_PROC_USER*, est celle qui est utilisée par les noyaux CLIP. Les restrictions supplémentaires imposées dans ce cas sont :

- Restriction des répertoires */proc/<pid>/* aux utilisateurs propriétaires, avec des permissions *0500* sur ces répertoires. On notera que les fichiers individuels situés à l'intérieur de chacun de ces répertoires ne sont pas affectés, et demeurent pour la plupart lisibles par tous les utilisateurs[25]. Il est rappelé que l'utilisateur propriétaire d'un répertoire */proc/<pid>* est normalement celui dont l'*uid* correspond à l'*euid* de la tâche identifiée par *<pid>*, sauf si cette tâche n'est pas *dumpable* (par exemple, suite à un bit 's' sur l'exécutable, ou à des privilèges augmentés par *veriexec* – cf. [CLIP_1201]), auquel cas le répertoire est systématiquement propriété de *root*.
- Non-visibilité des répertoires */proc/<pid>* n'appartenant pas à l'utilisateur appelant, sauf si ce dernier est *root* (*uid* de la tâche appelante nul). Cette mesure complète l'ajustement des droits sur ces répertoires, et interdit à un utilisateur non privilégié de lister les *pid* des processus exécutés par d'autres utilisateurs.

Ces mesures peuvent être complétées par une option supplémentaire, *GRKERNSEC_PROC*_ADD (activée sur un noyau CLIP), qui impose des permissions plus restreintes sur les fichiers génériques du */proc* (hors des répertoires */proc/<pid>*. Plus précisément, les modifications apportées par l'activation de cette option sont :

[25] A l'exception des fichiers sensibles pour la sécurité, auquel l'accès est généralement conditionné à la possibilité de tracer la tâche par *PTRACE* (et donc soumis non seulement à des restrictions sur l'utilisateur, mais aussi, avec le LSM CLIP, à des conditions sur les capacités et privilèges CLIP-LSM respectifs de l'appelant et de la tâche tracée – cf. [CLIP_1201]).

- Restriction à *root* (ou *root* et le groupe inspecteur avec *GRKERNSEC_PROC_USERGROUP)* seul des répertoires */proc/pci* et */proc/net* (mode *0500* au lieu de *0555*).
- Restriction à *root* (ou *root* et le groupe inspecteur) de */proc/kallsyms,* qui contient la table de symboles du noyau (mode *0400* au lieu de *0444*). On notera que ce fichier n'est de toutes manières pas créé sur un poste CLIP, l'option *CONFIG_KALLSYMS* étant désactivée dans le noyau.
- Restriction à *root* (ou *root* et le groupe inspecteur) de */proc/config.gz*, qui contient la configuration du noyau (mode *0400* au lieu de *0444*). On notera que ce fichier n'est de toutes manières pas créé sur un poste CLIP, l'option *CONFIG_IKCONFIG* étant désactivée dans le noyau.
- Restriction à *root* (ou *root* et le groupe inspecteur) de */proc/iomem, /proc/ioports, /proc/cmdline* et */proc/devices* (mode *0400* au lieu de *0444*).
- Restriction à *root* (ou *root* et le groupe inspecteur) de */proc/slabinfo* (mode *0600* au lieu de *0644*).
- Suppression de */proc/kcore*.

Ces différentes mesures ont pour vocation d'interdire à un attaquant local non privilégié l'accès à des informations sur le système ou les processus privilégiés qui pourraient lui être utiles pour monter une attaque en escalade de privilèges.

Restrictions sur les liens

L'option *GRKERNSEC_LINK* active des mesures préventives contre des familles d'attaques classiques en environnement UNIX, reposant sur des redirections illégitimes à l'aide de liens (symboliques ou non).

D'une part, cette option interdit le suivi d'un lien symbolique dans un répertoire en mode '+t' accessible en écriture par tous les utilisateurs (typiquement, */tmp* ou */var/tmp*), lorsque le propriétaire du lien symbolique est différent du propriétaire du répertoire (ce qui permet le suivi de liens symboliques créés par *root* dans */tmp*) et différent du *fsuid* de la tâche appelante (ce qui permet de suivre ses propres liens symboliques). Cette restriction s'applique à tous les utilisateurs de manière indiscriminée, et les résolutions de liens symboliques interdites font l'objet d'une journalisation.

D'autre part, la création d'un lien dur est interdite à tout utilisateur non *root* et ne disposant pas de la capacité *CAP_FOWNER*[26] lorsque l'*inode* vers lequel pointerait le lien ne lui appartient pas (*uid* de l'*inode* différent du *fsuid* de la tâche appelante) et que cet *inode* correspond à un fichier spécial (*device, fifo*, etc.) ou porte un bit 's' (*SETUID* ou *SETGID*) ou n'est pas accessible en lecture et écriture par l'utilisateur appelant. Les créations de liens interdites par cette mesure sont journalisées.

Restrictions sur les FIFO

L'option *GRKERNSEC_FIFO*, lorsqu'elle est activée, interdit l'ouverture de fichiers de type *FIFO* dans un répertoire '+t' accessible en écriture par tous les utilisateurs, lorsque le propriétaire de la *FIFO* est

[26] Ce double test est non standard pour le noyau Linux, et autorise potentiellement une action privilégiée à un processus *root* ne disposant pas de la capacité associée à cette action.

différent du propriétaire du répertoire et du *fsuid* de l'appelant. Les ouvertures interdites sont journalisées uniquement lorsque les droits UNIX discrétionnaires auraient autorisé l'accès (dans la mesure où ces droits sont vérifiés uniquement après la vérification des contraintes Grsecurity). Ces contraintes ne sont pas appliquées aux ouvertures avec le drapeau *O_EXCL*, pour lesquelles elles n'ont pas de sens (la non existence du fichier étant vérifiée après ces contraintes).

2.3 Durcissement des prisons *chroot*

Grsecurity propose un ensemble d'options de durcissement des prisons *chroot*, accessibles lorsque l'option globale *GRKERNSEC_CHROOT* est sélectionnée. La principale utilisation des mécanismes associés au sein de CLIP est d'améliorer le niveau de sécurité des « vues » CLIP, qui reposent sur le partitionnement d'une cage *vserver* en plusieurs prisons *chroot*. Ces mécanismes font l'objet d'adaptations spécifiques à CLIP, apportées par le patch CLIP-LSM (cf. [CLIP_1201]), afin d'une part d'utiliser les informations stockées dans les étiquettes de sécurités du LSM pour déterminer si une tâche est enfermée ou non dans une prison *chroot*, et d'autre part de comparer la racine d'une telle prison avec la racine réelle dans l'espace de nommage courant, plutôt que celle de l'espace de nommage principal. Ces adaptations sont nécessaires afin de prendre en compte les multiples espaces de nommage VFS, et les racines associées, mis en oeuvre par *vserver* (cf. [CLIP_1202]). En effet, les tests normalement réalisés par Grsecurity (comparaison de la racine de la tâche courante à celle du processus *init*) conduiraient à considérer toute tâche enfermée dans une cage *vserver* comme enfermée dans une prison *chroot*, et à comparer la racine de toute prison *chroot* à celle du système en contexte *vserver* ADMIN, ce qui ne permet pas de protéger spécifiquement une prison *chroot* à l'intérieur d'une cage *vserver*.

Plusieurs mécanismes introduits par ces options font appel à une fonction *gr_is_outside_chroot()*, permettant de déterminer si un descripteur de fichier (représenté par un couple *struct dentry, struct vfsmount*) est situé en dehors d'une prison *chroot* donnée. L'implémentation de cette fonction repose sur une remontée récursive de descripteur en descripteur parent, à partir du descripteur à tester. La récursion s'arrête lorsqu'elle atteint soit le descripteur de fichier correspondant à la racine de la prison *chroot* courante[27], soit celui correspondant à la vraie racine de l'espace de nommage courant[28]. Dans le premier cas, le descripteur de fichier original est bien à l'intérieur de la prison *chroot*, tandis que dans le deuxième cas, il est à l'extérieur de celle-ci et peut être utilisé potentiellement pour en sortir.

Opérations interdites dans une prison *chroot*

Plusieurs options permettent d'interdire, de manière générale et sans prise en compte du niveau de privilège de la tâche appelante, certaines opérations dans une prison *chroot.* Les options retenues dans la configuration d'un noyau CLIP interdisent les opérations suivantes à toute tâche *chrootée* (d'après son étiquette de sécurité CLIP-LSM) :

- Appel *chroot()* sur un descripteur de fichier situé en dehors de la prison courante (*GRKERNSEC_CHROOT_DOUBLE*)
- Appel *pivot_root()* (*GRKERNSEC_CHROOT_PIVOT*)
- Augmentation de la priorité (réduction du *nice*) de la tâche courante par *sys_nice()* ou d'une

[27] Obtenue à partir des champs *root* et *rootmnt* de la structure *task_struct* de la tâche appelante, supposée *chrootée*

[28] Obtenue, avec CLIP-LSM, à partir de la structure *vx_info* du contexte *vserver* courant s'il existe – c'est à dire dans tout contexte autre que ADMIN ou WATCH, et sinon à partir de la *task_struct* du *child_reaper* de la tâche appelante – le patch Grsecurity non modifié utilise quant à lui uniquement cette deuxième approche.

autre tâche par *sys_setpriority()* (*GRKERNSEC_CHROOT_NICE*). On notera que la réduction de la priorité d'une tâche extérieure à la prison est possible sous réserve que les autres contraintes (*uid* / capacités) soient satisfaites.

- Modification d'une variable *sysctl* par l'appel *sys_sysctl* ou par écriture sur un fichier de */proc/sys (GRKERNSEC_CHROOT_SYSCTL*). Les accès en lecture restent possibles, mais sont conditionnés sous CLIP au montage de */proc* dans la prison, et à la visibilité de */proc/sys* dans le contexte *vserver* correspondant (cf. [CLIP_1202]), dans la mesure où l'appel système *sys_sysctl* n'est pas supporté par les noyaux CLIP.

Les opérations interdites par ces différentes options sont toutes journalisées.

Chdir()* automatique lors d'un *chroot

L'option *GRKERNSEC_CHROOT_CHDIR* ajoute un *chdir()* automatique à la nouvelle racine d'un *chroot* immédiatement après le changement de racine dans tout appel *chroot*, afin d'éviter au processus de conserver une référence à un répertoire hors du *chroot*. On notera que le patch CLIP-LSM interdit les appels *chroot* par un processus disposant de descripteurs de fichiers ouverts sur des répertoires, mais que cette option est complémentaire dans la mesure où le répertoire courant n'apparaît pas dans la table de descripteurs du processus et n'est donc pas contrôlé par CLIP-LSM.

Remarque 1 : Race condition potentielle dans GRKERNSEC_CHROOT_CHDIR

L'appel chdir() *imposé par cette option n'est réalisé qu'après le changement de racine. Dans le cas d'une tâche entrant dans un* chroot *existant, il existe donc un court intervalle de temps pendant lequel la tâche n'a pas modifié son répertoire courant, et n'est plus protégée vis-à-vis des autres processus du* chroot *par GRKERNSEC_CHROOT_FINDTASK (mécanisme décrit plus bas), dans la mesure où elle partage la même racine que ces processus. Il serait donc éventuellement possible pour ces autres processus d'accéder à des fichiers hors* chroot *pendant cet intervalle, en déréférençant le lien* /proc/<pid>/cwd *associé à la tâche entrante (sous réserve que celle-ci n'ait pas fait explicitement de* chdir() *avant d'appeler* chroot()*). Cependant, une telle exploitation n'est pas possible sur un noyau CLIP, car le déréférencement de* /proc/<pid>/cwd *nécessite les privilèges nécessaires à un attachement* ptrace*, ce qui est inconditionnellement interdit par CLIP-LSM pour des processus* chrootés.

Tests supplémentaires lors d'un *fchdir()*

Lorsque l'option *GRKERNSEC_CHROOT_FCHDIR* est activée, les appels *fchdir()* font l'objet de vérifications supplémentaires. Si la tâche appelante est enfermée dans un *chroot* et le descripteur de fichier passé en argument de *fchdir()* est situé hors de ce *chroot*, l'appel est refusé et journalisé.

Protection des processus hors-prison

L'option *GRKERNSEC_CHROOT_FINDTASK* rend invisibles pour une tâche enfermée dans un *chroot* toutes les tâches qui n'ont pas la même racine. Ces tâches n'apparaissent pas dans la vision qu'a la tâche enfermée de */proc* (les répertoires */proc/<pid>* correspondants ne sont pas créés) et leur *pid*, même devinés, ne sont pas utilisables pour les différents appels système acceptant des *pid* en argument (*setpgid(), setpriority(),* etc.), qui rendent dans ce cas une erreur compatible avec un *pid* inexistant. Il

n'est de plus explicitement pas autorisé de leur envoyer un signal[29] par *sys_kill* ou par les signaux liés aux entrées/sorties sur les fichiers (*sigio*).

Au sein de CLIP, cette mesure assure essentiellement une protection en disponibilité des processus situés hors de la prison vis-à-vis de ceux situés dans la prison, dans la mesure où le LSM CLIP bloque les possibilités de sortie de *chroot* liées à la visibilité des processus hors-*chroot*. Sans LSM CLIP ni *GRKERNSEC_CHROOT_FINDTASK*, il est en effet trivial pour un processus (même non-*root*) d'accéder aux fichiers situés hors du *chroot,* à travers */proc/<pid>/cwd* ou */proc/<pid>/root*, avec *<pid>* le *pid* d'un processus hors *chroot* du même utilisateur, ou en prenant le contrôle d'un tel processus par *ptrace*. Ces deux méthodes (résolution de liens dans */proc/<pid>* et attachement *ptrace*) sont soumises à la même condition : pouvoir s'attacher par *ptrace* au processus cible. Or le patch *CLIP-LSM* (cf. [CLIP_1201]) bloque d'ores et déjà toute opération *ptrace* depuis une prison *chroot* (y compris entre processus de la prison).

Protection des canaux IPC hors-prison

L'option *GRKERNSEC_CHROOT_SHMAT* interdit l'attachement de tâches enfermées dans une prison *chroot* à des segments de mémoire partagée (*shm System V*) créés ou utilisés hors de la prison. Dans la mesure où les structures associées à de tels segments ne conservent pas trace de leur racine VFS, deux méthodes sont utilisées par Grsecurity pour déterminer si un segment de mémoire partagée est en dehors de la prison. En premier lieu, le processus créateur du segment est recherché, en cherchant une tâche dont le *pid* corresponde au champ *shm_cprpid* de la *struct shmid_kernel* associée au segment. Si une telle tâche est trouvée, son heure de démarrage (donnée par le champ *start_time* de la *task_struct* associée) est comparée à l'heure de création du segment (donnée par un champ *shm_createtime* ajouté par Grsecurity à la *struct shmid_kernel*, et renseigné lors de la création d'un segment *shm*). Si la tâche est plus ancienne que le segment, elle est considérée comme la créatrice du segment, et sa racine est comparée à celle de la tâche qui tente de s'attacher au segment. Si les deux racines sont similaires, l'accès est accordé, sinon il est refusé et journalisé. L'accès est en revanche autorisé lorsque la tâche de *pid shm_cprpid* est plus jeune que le segment (ce qui signifie que la tâche créatrice du segment s'est terminée, et que son *pid* a été réattribué à une autre tâche). Lorsqu'aucune tâche de *pid shm_cprpid* n'est trouvée, une autre tâche est recherchée dont le *pid* égale le champ *shm_lapid* de la *struct shmid_kernel*, champ ajouté par Grsecurity pour mémoriser le *pid* de la dernière tâche à s'être attachée au segment, et renseigné lors de tout attachement. Si une tâche de *pid shm_lapid* est trouvée, sa racine est comparée à celle de la tâche qui tente l'attachement, et l'accès est accordé si les deux racines coïncident, et refusé et journalisé sinon. Si aucune tâche n'est trouvée dont le *pid* égale soit *shm_cprpid* soit *shm_lapid*, l'accès est par défaut autorisé.

Remarque 2 : Imprécision du contrôle d'attachement à un segment SHM

Les vérifications réalisées lors d'un attachement SHM par l'option GRKERNSEC_CHROOT_SHMAT *souffrent de nombreuses imprécisions, qui peuvent permettre leur contournement. La recherche de tâche par* pid *peut trouver une tâche différente de celle recherchée dès lors que cette dernière s'est terminée. Un attaquant situé dans le chroot pourrait dans ce cas appeler* fork() *jusqu'à créer un processus dont le pid soit le même que celui de la tâche créatrice du segment, mais enfermé dans la prison*

[29] L'erreur renvoyée dans ce cas est *-EPERM*, ce qui n'est pas consistent avec celle qui serait renvoyée si la tâche cible n'existait pas, *-ESRCH*. Il est donc possible de « deviner » les *pid* invisibles par un parcours systématique de l'espace des *pid* avec *sys_kill()*.

contrairement à cette dernière, et ainsi obtenir l'accès à un segment créé hors de sa prison. La comparaison des heures de création apporte une contre-mesure partielle (potentiellement annulée par un changement d'heure, et plutôt à vocation fonctionnelle, dans la mesure où l'attachement est autorisé lorsque le pid *du créateur a été réutilisé) dans le cas de la tâche créatrice (*shm_cprid*), mais aucune vérification n'est possible pour la dernière tâche attachée. Par ailleurs, ces contraintes n'étant imposées que lorsqu'une tâche enfermée dans un* chroot *tente un attachement, rien n'interdit à une tâche hors prison de s'attacher par erreur à un segment créé au sein d'une prison.*

Par ailleurs, lorsque l'option *GRKERNSEC_CHROOT_UNIX* est activée, la connexion à une *socket* UNIX abstraite créée (attachée par *bind()*) hors de la prison est interdite à toute tâche enfermée dans une prison *chroot*. A cette fin, la tâche en écoute sur la *socket* est recherchée à l'aide du *pid* sauvegardé dans les *credentials UNIX* de la *socket,* et sa racine comparée à celle de la tâche qui initie la connexion. La connexion est autorisée si les deux racines coïncident, refusée avec journalisation si elles diffèrent, et autorisée par défaut si la tâche en écoute n'est pas trouvée. On notera que la recherche par *pid* est dans ce cas a priori plus fiable que dans le cas d'un segment SHM, dans la mesure où la gestion des *credentials UNIX* assure que le *pid* recherché est valide. Le patch CLIP-LSM offre un mécanisme de durcissement complémentaire de ce dernier, en interdisant la transmission par les *credentials UNIX* (sur une *socket* abstraite ou non) vers une tâche enfermée dans un *chroot* de descripteurs de fichiers extérieurs à la prison.

Options non retenues dans la configuration CLIP

Grsecurity offre aussi les options suivantes pour le durcissement des prisons *chroot*, options qui ne sont pas retenues dans la configuration des noyaux CLIP :

- *GRKERNSEC_CHROOT_CHMOD* : interdit l'ajout de bits 's' (*SETUID* ou *SETGID*) par *chmod()* ou *fchmod()* par un processus *chrooté*. Cette option serait incompatible avec le fonctionnement d'un rôle de mise à jour dans une prison *chroot* (vue UPDATE dans CLIP-RM), qui a besoin de créer de tels bits lors de l'installation de certains paquetages. On notera que les bits 's' suid root sont dans tous les cas ignorés par les noyaux CLIP (du fait d'une option CLIP-LSM, cf. [CLIP_1201]).
- *GRKERNSEC_CHROOT_CAPS* : force la suppression de certaines capacités particulièrement privilégiées[30] des trois masques de capacités POSIX immédiatement après un appel à chroot(), et exige *CAP_SYS_RAWIO* pour toute ouverture d'un fichier spécial en mode bloc au sein d'une prison (ce qui revient généralement à interdire une telle ouverture, du fait du masquage de cette capacité). Cette option n'est pas compatible avec le lancement de certaines cages CLIP, dans la mesure où la création d'une cage vserver nécessite un appel *chroot()* (cf. [CLIP_1202]), et doit pour certaines cages privilégiées conserver certaines capacités héritables incluses dans l'ensemble masqué par Grsecurity, afin de permettre la mise en oeuvre du mécanisme d'héritage de capacités permis par *veriexec*[31] (cf. [CLIP_1201]). On notera par ailleurs que les capacités les plus dangereuses (*CAP_SYS_MODULE, CAP_SYS_RAWIO*) sont globalement masquées sous CLIP par le mécanisme du cap-bound, et voient leur portée fortement réduite par la

[30] Plus précisément, *CAP_LINUX_IMMUTABLE, CAP_NET_ADMIN, CAP_SYS_MODULE, CAP_SYS_RAWIO, CAP_SYS_PACCT, CAP_SYS_ADMIN, CAP_SYS_BOOT, CAP_SYS_TIME, CAP_NET_RAW, CAP_SYS_TTY_CONFIG* et *CAP_IPC_OWNER.*

[31] Par exemple, le processus qui crée la cage X11 de CLIP doit conserver *CAP_SYS_TTY_CONFIG* dans son masque héritable, afin de donner ultérieurement cette capacité effective au serveur X11.

configuration noyau (absence du support de module, options *GRKERNSEC_KMEM* et *GRKERNSEC_IO*). Les autres capacités ne sont pas attribuées automatiquement à root sous CLIP, du fait de la mise en oeuvre de CLIP-LSM. Enfin, la construction de CLIP n'expose aucun périphérique de type bloc au sein des prisons *chroot*, et interdit leur création (par l'absence de *CAP_MKNOD*, mais aussi par l'utilisation systématique de l'option *nodev* pour les montages accessibles en écriture au sein des cages et vues).

- *GRKERNSEC_CHROOT_MKNOD* : interdit dans une prison *chroot* les appels *mknod()* pour créer un fichier autre que régulier[32] ou FIFO. On notera que cette option interdit aussi la création d'un fichier de type *socket* par *mknod* dans une prison *chroot*, opération qui n'est pas en soi dangereuse, mais est normalement réalisée par un appel *bind()* plutôt que par un appel mknod(). Cette option serait redondante dans CLIP, dans la mesure où un effet similaire est obtenu par le masquage de *CAP_SYS_MKNOD* dans toutes les cages, et entrerait de plus en conflit avec la procédure de mise à jour du coeur CLIP (cf. [CLIP_DCS_13006]), qui réalise l'installation de paquetages contenant potentiellement des devices au sein d'une prison chroot.
- *GRKERNSEC_CHROOT_MOUNT* : interdit toute opération de montage ou de re-montage VFS dans une prison *chroot*. Là encore, cette option est redondante dans CLIP du fait du masquage de *CAP_SYS_MKNOD* dans toutes les cages, et pourrait entrer en conflit avec le traitement réalisé par certains maintainer-scripts de paquetages, exécutés dans une prison chroot lors de la mise à jour du coeur CLIP.

2.4 Journalisation supplémentaire

Le patch *Grsecurity*, tel qu'il est configuré dans un noyau CLIP, apporte une journalisation spécifique des événements suivants :

- Montages et démontages VFS
- Création ou suppression de canaux de communication IPC *System V*
- Envoi a un processus d'un des signaux *SIGSEGV,SIGILL, SIGABRT* ou *SIGBUS*
- Échec d'un appel *fork()* (sauf pour cause d'interruption de l'appel par un signal)
- Modification de l'heure système par *settimeofday* ou *stime()*.

Les événements suivants pourraient aussi être journalisés, mais ne le sont pas par un noyau CLIP :

- Appels *exec()* (éventuellement, *exec()* dans un *chroot* uniquement) : une telle journalisation est rejetée dans CLIP car trop verbeuse.
- Échec d'un appel système du fait du dépassement d'une *rlimit*. Les *rlimit* n'étant pas mises en oeuvre dans CLIP à ce stade, cette option ne présente pas de réel intérêt.
- Appel *chdir(),* qui entraînerait aussi une journalisation trop verbeuse.
- Relocalisation ELF de type *TEXTREL* (nécessitant une modification de la section *.text* d'un exécutable en mémoire).

Cette journalisation est réalisée à destination du tampon de messages *printk* du noyau, par l'intermédiaire de primitives de journalisation spécifiques à Grsecurity (*grsecurity/grsec_log.c*). Ces

[32] On notera que l'appel système *sys_mknod()* permet la création d'un fichier régulier, mais pas d'un répertoire.

mêmes primitives sont aussi utilisées pour journaliser tous les accès refusés par les différents mécanismes de contrôle décrits en 2.2 et 2.3. La journalisation Grsecurity insère dans chaque message ainsi généré un préfixe et un suffixe génériques apportant des informations supplémentaires sur les circonstances de l'événement journalisé. Le préfixe contient a minima la chaîne de caractères *« grsec: »*, ce qui permet d'appliquer facilement un filtrage spécifique à ces messages Grsecurity[33]. Lorsque les ACL Grsecurity sont actives (cf. 2.5), le nom du fichier d'ACL associé à cette tâche est inclus dans le préfixe. De plus, Grsecurity assure un suivi de l'adresse IP de connexion associée à chaque tâche, à l'aide d'un champ *curr_ip* ajouté par le patch à la structure *signal_struct* associée à chaque processus. Ce champ, nul par défaut, est renseigné avec l'adresse IP de l'interlocuteur lorsqu'un processus réalise un *accept()* sur une socket réseau *TCP*. Lorsque ce champ *curr_ip* est non nul, il est ajouté au préfixe de chaque message de journalisation Grsecurity. Le suffixe contient quant à lui systématiquement les informations suivantes :

- Nom commun de la tâche à l'origine du message et de sa tâche parente.
- Chemin absolu (dans l'espace de nommage VFS de la tâche) de l'exécutable associé à la tâche et à son père.
- pid de la tâche et de son père.
- uid, euid, gid et egid de la tâche et de son père.

Entre ces deux éléments (préfixe et suffixe génériques) est inséré un message spécifique au type d'événement journalisé, contenant éventuellement des informations supplémentaires comme le nom du fichier associé à un refus d'accès.

Grsecurity peut par ailleurs modérer le flot de messages de journalisation généré par ses primitives, selon deux paramètres ajustés à la compilation : un nombre maximum de *CONFIG_GRKERNSEC_FLOODBURST* messages peut être généré dans un intervalle de *CONFIG_GRKERNSEC_FLOODTIME* secondes. Lorsqu'un message doit être journalisé, qui conduise à dépasser cette limite, un message d'avertissement est ajouté au tampon *printk*, et le message d'origine est ignoré. Tout autre message passé aux primitives de journalisation dans les *CONF_GRKERNSEC_FLOODTIME* secondes qui suivent ce premier refus est simplement ignoré. Ces limitations de débits s'appliquent uniquement aux messages signalant un refus d'accès; les messages d'audit supplémentaire décrits plus haut n'y sont pas soumis, pas plus que les événements audités spécifiquement pour un processus à l'aide des ACL Grsecurity décrites en 2.5.

2.5 ACL Grsecurity (non mises en oeuvre dans CLIP)

Grsecurity supporte un moteur d'ACL (*Access Control Lists*) permettant de définir des politiques détaillées régissant les interactions entre sujets (processus, identifiés en fonction de l'exécutable associé et du rôle qui leur est associé) et objets du système, selon un modèle de type *RBAC* (*Role Based Access Control*). Les interractions qui peuvent ainsi être contrôlées sont notamment :

- l'accès aux fichiers, selon plusieurs modes
- l'accès au réseau, éventuellement spécifique par protocole et adresse
- l'accès aux autres processus (signaux, *ptrace*)

[33] Ainsi, au sein de CLIP, le démon *syslog-ng* exécuté dans la cage $AUDIT_{clip}$ est configuré de manière à regrouper les messages Grsecurity dans un fichier propre, *grsecurity.log*, à l'exception des messages indiquant des montages ou démontages, qui sont redirigés, en raison de leur abondance, vers un fichier dédié, *grsec_mount.log*.

- les capacités POSIX (ajoutées ou retirées par rapport à celles attribuées par défaut – on notera que ce mécanisme permet de mettre en oeuvre un substitut aux bits *suid root*, comparable à celui apporté par *veriexec* (cf. [CLIP_1201]), mais sans la vérification d'empreinte cryptographique associée).
- les options PaX (cf. 1.3) applicables au processus, si PaX est configuré pour supporter une telle configuration dynamique (ce qui n'est pas le cas sous CLIP).

Les descriptions d'ACL sont chargées en mémoire noyau par un utilitaire *gradm*, qui utilise à cette fin des écritures sur un fichier spécial en mode caractère, */dev/grsec* (majeur 1, mineur 13). Les transitions entre rôles sont soumises à autorisation par l'éventuelle ACL active, et à authentification auprès du noyau, qui vérifie lui-même un mot de passe transmis par la couche utilisateur par écriture sur */dev/grsec*, l'empreinte du mot de passe étant chargée en mémoire noyau à l'initialisation des ACL. Cette initialisation se fait aussi par écriture sur */dev/grsec,* et s'accompagne d'une activation immédiate des ACL. La désactivation reste possible, là encore par écriture sur */dev/grsec*, mais nécessite une authentification dans le rôle administrateur Grsecurity. Enfin, le moteur d'ACL supporte un mode d'apprentissage, dans lequel le fonctionnement d'un processus est tracé par le noyau, afin de générer une ACL adaptée à ce fonctionnement, qui est ensuite lisible sur */dev/grsec*.

Ces ACL ne sont pas utilisées sous CLIP à ce stade. Il n'existe normalement aucun moyen de désactiver globalement le support des ACL Grsecurity à la compilation du noyau. Cependant, un patch spécifique à CLIP ajoute une option *GRKERNSEC_ACL*, dont la sélection conditionne la création du fichier spécial */dev/grsec*. Cette option n'est pas sélectionnée dans les noyaux CLIP, ce qui ne supprime pas le moteur d'ACL de ces noyaux, mais uniquement l'interface de configuration de ces ACL depuis la couche utilisateur. On notera qu'une utilisation future de ces ACL n'est pas exclue pour CLIP, afin par exemple de mieux contrôler les privilèges accordés aux démons du socle (démon IKE, XDM, etc.). Il serait cependant nécessaire au préalable d'assurer une prise en compte dans le moteur d'ACL des différents contextes *vserver*, afin de pouvoir disposer d'ACLs (et d'authentifications) totalement disjointes d'un contexte à l'autre.

2.6 Autres fonctionnalités Grsecurity non mises en oeuvre dans CLIP

Plusieurs autres fonctionnalités supportées par Grsecurity ne sont pas mises en oeuvre dans CLIP. Les options de compilation correspondantes ne sont pas sélectionnées dans les noyaux CLIP.

Restrictions sur les *sockets*

L'option *GRKERNSEC_SOCKET* donne accès à plusieurs sous-options permettant d'imposer des restrictions d'accès au réseau en fonction du groupe d'appartenance des utilisateurs.

- L'option *GRKERNSEC_SOCKET_ALL* permet d'interdire aux membres (au sens du *gid* ou des groupes supplémentaires) du groupe *GRKERNSEC_SOCKET_ALL_GID*, défini statiquement à la compilation, tout appel *socket()* pour créer une *socket* autre que *AF_UNIX/AF_LOCAL*.
- L'option *GRKERNSEC_SOCKET_CLIENT* permet d'interdire aux membres du groupe *GRKERNSEC_SOCKET_CLIENT_GID* tout appel *connect()* sur une *socket* de type autre que *AF_UNIX / AF_LOCAL*.
- L'option *GRKERNSEC_SOCKET_SERVER* permet d'interdire aux membres du groupe

GRKERNSEC_SOCKET_SERVER_GID tout appel *bind()* sur une *socket* de type autre que *AF_UNIX / AF_LOCAL*.

Ces mécanismes ne sont pas utilisés dans CLIP dans la mesure où le LSM CLIP (cf. [CLIP_1201]) fournit des mesures comparables de contrôle d'accès au réseau, basées non pas sur les groupes mais sur les empreintes d'exécutables.

Sysctl Grsecurity

L'option *GRKERNSEC_SYSCTL* permet de créer plusieurs variables *sysctl kernel.grsecurity.**, permettant d'activer ou de désactiver dynamiquement plusieurs mesures restrictives apportées par Grsecurity. Sont ainsi configurables notamment :

- les restrictions TPE (cf. 2.2.1) et le groupe associé
- les différentes options de durcissement des prisons *chroot* (2.3)
- les différentes options de journalisation (2.4)
- les restrictions sur les *sockets* décrites plus-haut

L'option *GRKERNSEC_SYSCTL_ON* permet de définir si ces différents mécanismes sont activés par défaut au démarrage ou non. Une variable *sysctl* spécifique, *kernel.grsecurity.grsec_lock*, permet de verrouiller entièrement l'interface *sysctl* et d'interdire toute modification sans redémarrage.

L'interface *sysctl* n'est pas activée dans les noyaux CLIP, car elle n'a aucune utilité dans ce cadre et ouvrirait de potentielles vulnérabilités. La seule variable *sysctl* Grsecurity supportée par les noyaux CLIP est ainsi *kernel.grsecurity.disable_modules*, définie uniquement dans les configurations modulaires (cf. 2.1.2).

Restrictions sur l'accès au journal noyau

L'option *GRKERNSEC_DMESG* permet de restreindre l'accès en lecture – par *sys_syslog()* ou */proc/klog* – au tampon de messages noyau (*printk*) aux seuls processus disposant de la capacité effective *CAP_SYS_ADMIN*. Cette option n'est pas activée dans CLIP, car elle interdirait l'affichage de ces messages noyau au rôle d'audit.

Contrôles des rlimits lors d'un appel *exec()*

L'option *GRKERNSEC_EXECVE* ajoute une vérification de la *rlimit RLIMIT_NPROC* lors de chaque appel *execve()* et non plus seulement lors d'un *fork()*. Cette option n'est pas employée dans CLIP dans la mesure où les *rlimits* ne sont généralement pas mises en oeuvre à ce stade.

Affichage de l'adresse IP dans */proc*

L'option *GRKERNSEC_PROC_IPADDR* active un affichage de l'adresse IP de connexion (cf. 2.4) associée à chaque tâche, dans un fichier */proc/<pid>/ipaddr*. Cette option n'est pas mise en oeuvre dans CLIP, en l'absence de serveurs TCP en écoute sur le réseau, et dans la mesure où l'auditeur, à qui cette information pourrait profiter, n'a dans tous les cas pas accès à ce jour aux répertoires */proc* de la plupart des tâches du système (mais uniquement de celles de la cages $AUDIT_{clip}$).

Annexe A Références

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

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

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

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

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

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

[PAX] *PaX, http://pax.grsecurity.net*

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

[CLOSURE] *Thomas M. Breuel, Lexical Closures for C++, http://people.debian.org/aaronl/Usenix88-lexic.pdf*

[NONSELSEC] *Ulrich Drepper, Security Enhancements in Red Hat Enterprise Linux, http://people.redhat.com/drepper/nonselsec.pdf*

[SMM] Loïc Duflot, Olivier Grumelard, Daniel Etiemble, *Utiliser les fonctionnalités des cartes mères ou des processeurs pour contourner les mécanismes de sécurité des systèmes d'exploitation, http://www.ssi.gouv.fr/fr/sciences/fichiers/lti/sstic2006-duflot-papier.pdf*

[TOOLCHAIN] *Gentoo Linux Documentation – The Gentoo Hardened Toolchain, http://www.gentoo.org/proj/en/hardened/hardened-toolchain.xml*

Annexe B Liste des figures

Annexe C Liste des tableaux

Annexe D Liste des remarques