

RECOMMANDATIONS POUR LA MISE EN PLACE DE CLOISONNEMENT SYSTÈME

GUIDE ANSSI

PUBLIC VISÉ :

Développeur

Administrateur

RSSI

DSI

Utilisateur



Informations



Attention

Ce document rédigé par l'ANSSI présente les « **Recommandations pour la mise en place de cloisonnement système** ». Il est téléchargeable sur le site www.ssi.gouv.fr. Il constitue une production originale de l'ANSSI. Il est à ce titre placé sous le régime de la « Licence ouverte » publiée par la mission Etalab (www.etalab.gouv.fr). Il est par conséquent diffusable sans restriction.

Ces recommandations n'ont pas de caractère normatif, elles sont livrées en l'état et adaptées aux menaces au jour de leur publication. Au regard de la diversité des systèmes d'information, l'ANSSI ne peut garantir que ces informations puissent être reprises sans adaptation sur les systèmes d'information cibles. Dans tous les cas, la pertinence de l'implémentation des éléments proposés par l'ANSSI doit être soumise, au préalable, à la validation de l'administrateur du système et/ou des personnes en charge de la sécurité des systèmes d'information.

Évolutions du document :

VERSION	DATE	NATURE DES MODIFICATIONS
1.0	14/12/2017	Version initiale du document

Table des matières

1	Préambule	3
2	Le cloisonnement, qu'est-ce que c'est? Quel en est l'intérêt?	4
2.1	Une implémentation du principe de moindre privilège	4
2.1.1	Rappels sur le principe de moindre privilège	5
2.1.2	Aperçu de la mise en place de cloisonnement	6
2.2	Formalisation en tant que fonction de sécurité	7
2.3	Application des définitions sur quelques exemples	9
2.3.1	Systèmes d'hypervision	9
2.3.2	Utilisateurs distincts dans un système d'exploitation	10
2.3.3	Conteneurs et bacs à sable	11
3	Identifier ses besoins en cloisonnement	13
3.1	Généralités sur la sécurité du composant	14
3.1.1	Biens sensibles à protéger par le composant	14
3.1.2	Composants de confiance	15
3.1.3	Périmètre du composant	16
3.1.4	Formaliser la sécurité attendue du composant	18
3.2	Définition des usages du composant	18
3.3	Objectifs de la mise en place de cloisonnement	20
4	Analyser la sécurité apportée par le cloisonnement mis en place	22
4.1	Analyse d'un mécanisme de cloisonnement	22
4.1.1	Le moniteur de référence parmi les composants de confiance	22
4.1.2	Composants de confiance développés	23
4.1.3	Recommandations portant sur tous les composants de confiance	24
4.1.4	Recommandations spécifiques à un moniteur de référence	25
4.1.5	Évaluer un mécanisme de cloisonnement	27
4.2	Analyse de la mise en place du cloisonnement	27
4.2.1	À niveau d'abstraction donné	28
4.2.2	Raffinement du cloisonnement à l'intérieur du composant	28
5	Éléments d'analyse d'une architecture de sondes de détection réseau	30
5.1	Première proposition d'architecture pour exIDS	30
5.2	Architecture retenue pour exIDS	32
	Bibliographie	34

1

Préambule

La fonction de sécurité de cloisonnement bénéficie d'une popularité bien moindre que celles de confidentialité et d'intégrité. Un mécanisme de cloisonnement permet de compartimenter un environnement d'exécution en plusieurs parties ne comportant pas les mêmes éléments et ne bénéficiant ni des mêmes droits ni des mêmes ressources. Intuitivement, il s'agit de découper un environnement monolithique à la manière d'un puzzle, sans impact sur le service rendu. L'avantage d'une telle démarche tient alors dans la possibilité de restreindre chaque partie de l'environnement aux actions dont elle a besoin. En d'autres termes, l'intérêt du découpage découle de l'application du principe de moindre privilège sur chaque sous-partie de l'environnement. Une fois ceci mis en œuvre, la compromission d'une sous-partie devient plus difficile car sa surface d'attaque est réduite. De plus, une corruption ne peut avoir que des conséquences limitées.

Cette démarche peut être appliquée à tout niveau, à l'échelle d'un système d'information entier comme à l'intérieur d'un processeur matériel dédié à des traitements spécifiques. Dans tous les cas, le même objectif est poursuivi : effectuer un découpage pertinent et choisir des mécanismes adaptés à la restriction des actions possibles pour chaque pièce du puzzle. Les mécanismes de cloisonnement se répartissent en trois grandes catégories qui sont complémentaires : le cloisonnement réseau, le cloisonnement cryptographique et le cloisonnement système. Seul le cloisonnement système est traité ici, bien qu'une grande majorité du document s'applique indifféremment aux trois catégories.

L'ambition de ce document est d'aborder le cloisonnement système de manière générique, en présentant l'intérêt et ses objectifs. En effet, il n'existe pas de méthode universelle de mise en place du cloisonnement. Le lecteur est donc invité à s'approprier une démarche et à développer un esprit critique sur des choix de découpage et de mécanismes. Des définitions et des critères de comparaison sont proposés au fil du document.

Le document débute, en chapitre 2, par la présentation générale de ce en quoi consiste le cloisonnement, de manière à en présenter son intérêt au lecteur. Cette première partie introduit de nombreuses définitions. Formelles, elles sont cependant indispensables pour clarifier les propos développés. De multiples exemples viennent agrémenter ce chapitre pour concrétiser les notions abordées. La détermination des besoins en cloisonnement est détaillée dans le chapitre 3. Il s'agit d'évaluer la pertinence du découpage en fonction des objectifs visés. Après la lecture des chapitres 2 et 3, le lecteur est familier avec les enjeux liés à la mise en place de cloisonnement. Il reste à lui fournir des outils concrets d'analyse de la sécurité apportée par un choix d'architecture donné. Le chapitre 4 propose ainsi une série de critères d'évaluation de mises en œuvre de cloisonnement. Enfin, l'approche du document dans sa globalité est illustrée dans le dernier chapitre par une brève analyse d'architecture de sonde réseau de détection des incidents de sécurité.

2

Le cloisonnement, qu'est-ce que c'est ? Quel en est l'intérêt ?



Composant, système

Pour éviter les confusions, le terme *composant* désigne dans la suite ce qui est développé ou évalué, par opposition au mot *système*, employé pour désigner l'écosystème (machine, système d'information, etc.) dans lequel le composant va être utilisé.

L'exception notable à cette règle concerne l'expression « composants de confiance » définie ci-après, qui ne coïncident pas avec le composant développé ou évalué.

Exemples. Un composant, au sens du document, recouvre donc des réalités aussi variées qu'un chiffreur, un composant cryptographique embarqué, un navigateur Internet, ou un hyperviseur.

2.1 Une implémentation du principe de moindre privilège

De l'utilisation d'un composant logiciel ou matériel résulte un certain nombre d'effets de bord sur son environnement. Idéalement, l'environnement d'exécution du composant ne doit jamais être mis en défaut, ni du point de vue fonctionnel ni du point de vue de la sécurité.

Deux conditions doivent être remplies pour fournir de telles garanties. Premièrement, être en mesure de spécifier parfaitement tous les comportements possibles du composant sans abstraire aucun détail, et en toutes circonstances. Deuxièmement, garantir que le composant se conforme toujours à ce qui est spécifié. Si ces deux critères sont remplis, rien de fâcheux ne peut se produire : tout est prévu. Néanmoins, une telle situation n'existe évidemment pas en pratique.



Objectif

Le cloisonnement est implémenté pour *restreindre les conséquences possibles de comportements inattendus* d'un composant, qu'il s'agisse d'un bogue ou de son détournement par un attaquant.

Pour ce faire, il est habituel de restreindre l'environnement d'exécution du composant aux ressources strictement nécessaires à ses besoins. Ce principe classique est connu sous le nom de principe de moindre privilège.

2.1.1 Rappels sur le principe de moindre privilège

Ce principe constitue l'un des fondements du développement sécurisé. Pour être complet, ce document en fournit un rappel en définissant les termes utilisés.

Les concepts utilisés ici sont issus de la littérature académique portant sur le contrôle d'accès. Ces travaux utilisent généralement la terminologie suivante : des *sujets* peuvent être autorisés à effectuer des *actions* sur des *objets* ou d'autres sujets. Dans ce document, les sujets sont des *tâches* et les objets des *ressources*.



Tâche

Une tâche est un ensemble d'instructions chargées en mémoire au fur et à mesure pour être exécutées.

Exemples. L'exemple le plus classique de tâche est celui d'un processus s'exécutant en espace utilisateur au sein d'un système d'exploitation implémentant une séparation entre espace utilisateur et espace noyau. Une machine virtuelle dans un hyperviseur est un autre exemple.



Ressource

La notion de ressource correspond ici à une information, et par extension à l'objet logiciel ou matériel qui la contient et permet de la manipuler.

Exemples. Un descripteur de fichier, une clé cryptographique, un fichier, une socket réseau sont autant d'exemples de ressources logicielles. La mémoire RAM, les registres du processeur, les caches du processeur, les unités de stockage de type disque dur ou encore les périphériques externes constituent des exemples de ressources matérielles.



Action (agir)

Toutes les formes d'accès, d'utilisation ou de transmission des ressources constituent des actions.

Exemples. Le concept d'action sur des ressources recouvre des réalités aussi diverses que l'ouverture de fichiers, l'émission et la réception de signaux, l'usage de sockets réseau ou la génération et le traitement d'interruptions.



Privilège

Un privilège permet à une tâche qui en dispose de mener légitimement à bien une action sur une ressource, autrement dit sur tout ou partie d'un composant ou des informations qu'il utilise.

Lorsqu'un attaquant élève illégitimement ses privilèges, il devient alors en capacité de mener à bien des actions normalement interdites par un mécanisme de sécurité.

Dans tout le document, sauf mention contraire, les termes tâche, ressource, action (ou le verbe agir) et privilège gardent leur sens très général défini ici.



Principe de moindre privilège

Le principe de moindre privilège stipule qu'une tâche ne doit bénéficier que des privilèges strictement nécessaires à l'exécution du code menant à bien ses fonctionnalités.

En d'autres termes, une tâche ne devrait avoir la possibilité de mener à bien que les actions dont l'utilité fonctionnelle est avérée.

En général, on ne peut pas appliquer des restrictions à un tel niveau de granularité, mais l'idée est bien de s'en rapprocher le plus possible.

R1

Appliquer le principe de moindre privilège dès la conception

Interdire par défaut toute action et procéder à l'autorisation exclusive de ce qui est nécessaire aux tâches constitue la stratégie la plus efficace de mise en œuvre du principe de moindre privilège. Il convient de s'y conformer autant que possible dès la phase de conception du composant.

Le principe de moindre privilège va de pair avec l'idée de séparation des privilèges. Comme présenté en préambule du document, réduire les privilèges d'un composant monolithique est toujours profitable, mais souvent insuffisant. Les stratégies de découpage, qui relèvent plus de la séparation des privilèges que de l'application du principe de moindre privilège, sont discutées plus loin, en chapitre 3.

Cette manière de faire n'est pas toujours possible - en particulier lorsque l'on intègre des produits sur étagère. Cependant, s'y prendre ainsi permet d'éviter d'oublier d'interdire des accès.

2.1.2 Aperçu de la mise en place de cloisonnement

Le cloisonnement peut être appréhendé comme la mise en œuvre du principe de moindre privilège au sein d'un composant. Pour l'implémenter, il faut suivre les étapes suivantes :

- *identifier des privilèges nécessaires à un composant.* C'est identifier une liste d'actions qu'il doit être capable de mener à bien dans son environnement d'exécution ;
- *créer un environnement d'exécution réduit à cette liste d'actions (ou s'en rapprocher le plus possible).* Des mécanismes de cloisonnement, disponibles au niveau de l'environnement d'exécution, sont mis en œuvre dans ce but.
Beaucoup d'exemples peuvent être cités : dispositifs de contrôle d'accès (dans le système de fichiers, pour les accès réseau, etc.), utilisation de modes du processeur pour différencier les pages accessibles en mode privilégié des pages accessibles en mode utilisateur, usage de machines virtuelles différentes au sein d'un hyperviseur, etc. ;
- *itérer cette pratique sur chaque composant.* Scinder l'environnement d'exécution global d'un ensemble de composants en une série de sous-environnements appauvris.

Il existe en général plusieurs manières de parvenir à un même résultat, mais qui ne sont pas équivalentes du point de vue de la sécurité. Ce document vise à mettre en exergue une démarche de

conception qui respecte la philosophie dictée par les principes de moindre privilège et de défense en profondeur. Appliquer ces principes influe profondément sur l'architecture choisie.

R2

Tenir compte de ses besoins en cloisonnement dès l'initiation d'un projet

Les besoins en cloisonnement d'un composant doivent faire l'objet d'un diagnostic et être considérés comme des besoins au même titre que les besoins fonctionnels, et ce dès le début du projet.



Attention

Les conséquences sur l'architecture du composant peuvent être importantes et engendrer des surcoûts imprévus en cas de prise en compte tardive de besoins de sécurité.

Dans une démarche d'intégration, la qualité de la solution globale dépend également de la prise en compte des bonnes pratiques au niveau de chaque brique élémentaire. Plus particulièrement en ce qui concerne les besoins en cloisonnement, utiliser des briques nécessitant des privilèges trop élevés par rapport à leur fonction va dégrader la qualité de la solution globale.

R3

Préférer des composants implémentant un cloisonnement pertinent

Lors d'un choix entre différentes solutions, celles qui démontrent la meilleure prise en compte du principe du moindre privilège devront être préférées.

2.2 Formalisation en tant que fonction de sécurité

Cette section présente une approche générale du cloisonnement, certes un peu abstraite, mais qui permet de fixer le vocabulaire qui sera utilisé dans la suite du document.



Domaine

Un domaine est l'environnement d'exécution d'une tâche. Il est défini par l'ensemble des ressources (logicielles et matérielles) sur lesquelles la tâche peut effectuer des actions. Par opposition, une ressource sur laquelle la tâche ne peut pas agir ne fait pas partie de son domaine.



Information

Il est possible que plusieurs tâches cohabitent dans un domaine. Quelle que soit la réalité désignée, par abus de langage, ce document utilise systématiquement « tâche » au singulier. Ainsi, l'entité logique vis-à-vis du cloisonnement est une tâche.

Exemples. En considérant comme exemple de tâche un processus s'exécutant en espace utilisateur, un domaine est formé par toutes les ressources que le processus peut utiliser : tous les fichiers accessibles, les appels système possibles, les périphériques utilisables, etc.



Politique de sécurité

Cloisonner des tâches consiste à définir pour chacune son domaine d'exécution, ce qui a vocation à séparer les ressources en deux catégories :

- les *ressources partagées* entre plusieurs domaines, i.e. sur lesquelles plusieurs tâches peuvent légitimement agir ;
- les *ressources propres* à un domaine donné, i.e. sur lesquelles seule la tâche s'exécutant dans ce domaine doit être en capacité d'agir.

Cette spécification précise des domaines constitue une *politique de sécurité*.



Fonction de sécurité de cloisonnement (ou confinement)

Le cloisonnement est la fonction de sécurité garantissant qu'une tâche ne peut effectuer que les actions spécifiées par la politique de sécurité sur les ressources.

Exemples. Dans le cas des processus, les ressources partagées sont toutes celles auxquelles plusieurs processus peuvent accéder, parmi lesquelles le noyau du système d'exploitation, le matériel physique, etc. En l'absence de mesure particulière, les processus lancés par un utilisateur ont accès à tous les fichiers de l'utilisateur¹ : ces derniers constituent des ressources partagées.

Par défaut, un processus a pour ressources propres, l'espace mémoire utilisateur que le noyau lui a fourni, ainsi que le contexte d'exécution qui lui est associé. Ce contexte contient tout ce qui est mis en place par le noyau à chaque ordonnancement du processus : table de pages, pointeurs de pile, valeurs des registres du processeur, etc. Comme mentionné plus haut, le code du noyau est par contre une ressource partagée entre processus, et non une ressource propre : il n'est en effet pas dupliqué.



Attaquant de la fonction de sécurité de cloisonnement

L'attaquant d'une solution de cloisonnement est supposé être en contrôle d'un ou plusieurs domaines, et donc des tâches qui s'y exécutent. Son but est d'effectuer une action sur une ressource, interdite par la politique de sécurité.

Exemples. Un attaquant contrôlant un processus a réussi une attaque lorsque, par exemple, il peut écrire ou lire dans l'espace mémoire réservé à un autre processus, disons la valeur de secrets. Attention, si ces secrets sont en fait stockés dans un fichier auquel le processus compromis a accès par défaut selon la politique de sécurité, ce n'est pas le cloisonnement entre processus qui a été mis en défaut ! C'est la politique de sécurité qui n'est pas définie de manière satisfaisante.



Information

Même si le cloisonnement s'entend généralement comme existant entre au moins deux entités, il reste fréquent de parler de cloisonner ou confiner une tâche vis-à-vis du reste du système. Le cas échéant, deux domaines existent : celui de la tâche à confiner, et celui de toutes les autres tâches sur le système.

Mettre en place du cloisonnement implique d'implémenter concrètement les domaines, c'est-à-dire d'exercer un contrôle sur les actions effectuées par les tâches, de manière à interdire ce qui

1. On parle du cas par défaut sous Linux et Windows. Le positionnement de droits particuliers est abordé dans les exemples en section 2.3.

doit l'être. Sans un arbitre dont le rôle est de prendre la décision pour chaque action de chaque tâche de l'autoriser ou non, la politique de sécurité serait définie à titre indicatif et n'aurait aucun effet pour se protéger de comportements malveillants.



Moniteur de référence

Le moniteur de référence est l'entité qui implémente le mécanisme de contrôle d'accès et qui prend la décision d'autoriser ou d'interdire une action d'une tâche sur une ressource. C'est donc le moniteur de référence qui met en œuvre la politique de sécurité pour chaque domaine.

Il peut être complexe d'isoler ce code concrètement, auquel cas identifier un composant l'englobant est satisfaisant.

Exemples. En poursuivant sur l'exemple des processus utilisateurs, le moniteur de référence est constitué par le code du noyau qui gère les changements entre processus et les accès à la mémoire de chaque processus. C'est parce qu'un contexte d'exécution différent, propre à chaque processus, existe et est mis à jour par le noyau à chaque ordonnancement d'un processus, qu'il y a une isolation entre les espaces mémoire des processus. Le moniteur de référence est dans ce cas le noyau partagé par les processus ; même si tout le code du noyau n'est pas concerné, cette approximation est considérée comme valide.

Tâche	Processus utilisateur
Ressources propres	Contexte d'exécution du processus Mémoire en espace utilisateur
Ressources partagées	Objets système visibles par d'autres processus (fichiers, périphériques, etc.) Noyau, matériel physique
Moniteur de référence	Noyau du système d'exploitation

TAB. 2.1 – Exemple des processus utilisateurs

2.3 Application des définitions sur quelques exemples

Cette partie développe d'autres exemples de mécanismes de cloisonnement de manière à illustrer les concepts assez abstraits introduits précédemment.

2.3.1 Systèmes d'hypervision

Le premier exemple est celui d'un hyperviseur exécutant plusieurs machines virtuelles. Il existe deux types d'hyperviseurs, dits de type 1 et 2. Les hyperviseurs de type 1 s'exécutent nativement sur le matériel, contrairement aux hyperviseurs de type 2, qui s'exécutent au sein d'un système d'exploitation pour hyperviser des systèmes invités. Pour simplifier le propos, seuls les hyperviseurs de type 1 sont traités ici. Concrètement, Xen, KVM, ESXi ou encore Hyper-V sont des hyperviseurs de type 1, tandis que Qemu (utilisé seul) et VirtualBox sont de type 2.

Dans ce cas de figure, une machine virtuelle, dite aussi invitée, constitue une tâche. Les ressources propres à une machine virtuelle regroupent toutes les ressources virtuelles telles que vues par le

système invité², son espace mémoire, fourni par l'hyperviseur, ainsi que toutes les informations de contexte stockées par l'hyperviseur entre deux ordonnancements. Les ressources partagées comprennent les composants matériels communs à plusieurs machines virtuelles, l'hyperviseur qui les exécute, ainsi que les éventuels partages mis en place entre machines virtuelles. Le domaine d'une machine virtuelle regroupe donc toutes ses ressources propres et ce qu'elle partage avec d'autres machines virtuelles.

Dans les faits, un système d'hypervision est rarement réduit à un hyperviseur seul. En effet, les rôles d'analyse des actions effectuées par les machines virtuelles et d'émulation des périphériques réels sont rarement effectués par l'hyperviseur lui-même. En pratique, c'est classiquement un noyau ou une machine invitée ayant un statut un peu particulier qui remplit ces rôles, tout en s'exécutant avec moins de privilèges que l'hyperviseur lui-même. Concrètement, c'est le cas du dom0 de Xen, du noyau Linux hôte de KVM, ou de VMkernel dans ESXi. Il convient de considérer cette ressource particulière, utilisée par toutes les autres machines virtuelles, comme une ressource partagée. Dans la suite, pour éviter les confusions, les tâches désignent les machines virtuelles sans rôle particulier.

En ce qui concerne Windows 10, le parti pris de ce document est de considérer Hyper-V comme seul élément du système d'hypervision, dont la vocation est de fournir du cloisonnement mémoire entre la machine invitée « Virtual Secure Mode » et celle contenant l'environnement Windows standard.

Dans ces différents cas, le moniteur de référence est le système d'hypervision : sollicité lors d'accès par une machine virtuelle au matériel partagé, en particulier la mémoire physique, il est garant du respect de l'isolation entre les tâches. D'ailleurs, la terminologie VMM (pour Virtual Machine Monitor) est aussi employée pour parler d'hyperviseurs, ce qui illustre bien cette idée de moniteur.

Tâche	Machine virtuelle
Ressources propres	Contexte d'exécution Mémoire occupée par la machine virtuelle Ressources virtuelles dans le système invité
Ressources partagées	Partages configurés Système d'hypervision, matériel physique
Moniteur de référence	Système d'hypervision

TAB. 2.2 – Exemple des hyperviseurs

2.3.2 Utilisateurs distincts dans un système d'exploitation

Au sein des systèmes d'exploitation classiques comme Windows, ou les systèmes type UNIX, il existe une notion d'utilisateur et de privilèges dont bénéficie cet utilisateur. Typiquement, un contrôle d'accès est exercé par le noyau du système d'exploitation pour décider si un processus lancé par un utilisateur a le droit d'accéder à une ressource du système. Différents modèles de contrôle d'accès existent. Dans le cadre d'un contrôle d'accès discrétionnaire (dit DAC pour *Discretionary Access Control* en anglais), le propriétaire d'une ressource peut configurer les permissions d'accès à celle-ci. Ce genre de contrôle d'accès est souvent complété d'un contrôle d'accès obligatoire (dit MAC pour *Mandatory Access Control*), qui consiste en une politique de sécurité imposée sur toute ressource du

². et également, s'il y en a, l'ensemble des composants matériels exclusivement accessibles à la machine virtuelle (dits périphériques délégués).

système indépendamment de son propriétaire. Parmi les implémentations classiques de contrôles d'accès obligatoires figurent le mécanisme MIC (*Mandatory Integrity Control*) dans les systèmes Windows, ainsi que SELinux ou encore AppArmor dans les systèmes Linux.

Dans ce contexte, un domaine est constitué de tout ce qui est utilisable par un utilisateur donné. Une tâche est formée de tous les processus d'un utilisateur donné. Elle regroupe donc autant de processus qu'il s'en exécute sous l'identité à laquelle elle correspond, au contraire de l'exemple détaillé en 2.2, selon lequel une tâche est un processus donné. En effet, ici on s'attache à instancier les définitions dans le cadre du cloisonnement entre utilisateurs, et non entre processus.

Les ressources propres à une tâche sont celles accessibles exclusivement à l'utilisateur auquel la tâche correspond (certains fichiers, données d'authentification, etc.). Les ressources partagées regroupent tous les objets système partagés par configuration, ainsi que l'intégralité du code noyau et l'ensemble des composants matériels.

Enfin, l'entité qui gère le contrôle d'accès dans le noyau est ici le moniteur de référence. Plus généralement, le noyau peut être considéré comme moniteur de référence garant du respect des permissions sur les objets système, qu'il met à disposition des utilisateurs.

Tâche	Tout ce qui s'exécute sous l'identité de l'utilisateur
Ressources propres	Objets systèmes accessibles exclusivement à l'utilisateur
Ressources partagées	Objets systèmes accessibles à d'autres utilisateurs Noyau, matériel physique
Moniteur de référence	Noyau du système d'exploitation

TAB. 2.3 – Exemple des utilisateurs d'un système d'exploitation

2.3.3 Conteneurs et bacs à sable

Il existe de multiples solutions de conteneurisation et autres bacs à sable, parmi lesquelles LXC, CoreOS Rkt, Docker ou Kubernetes. Ces solutions ne sont pas équivalentes d'un point de vue de la sécurité apportée. Pour comprendre en détail les enjeux des conteneurs Linux, les documents rédigés par NCC Group ([7] et [8]) constituent un bon point de départ. Il s'agit seulement ici d'établir ce à quoi correspondent les définitions de manière assez générique.

Les conteneurs ou bacs à sable créés par une solution sont nommés *cages* dans la suite. Les cages constituent des domaines, et tout ce qui s'exécute à l'intérieur d'une cage forme une tâche.

Suivant les solutions, les ressources propres et partagées peuvent varier ; souvent les cages ont leur propre instance de système de fichiers. Un fichier de configuration par cage permet en général de mettre en place les partages désirés.

Les cages sont contrôlées par un ou plusieurs programmes qui les initialisent et permettent de les gérer. Ceux-ci peuvent être intégrés ou non, suivant les implémentations, au noyau du système d'exploitation qui supporte la solution. Ils s'appuient sur le noyau du système pour isoler les cages entre elles (positionnement de permissions dédié, usage de solutions de contrôle d'accès intégrées au noyau, etc.). Ces programmes gestionnaires et le noyau du système d'exploitation constituent le moniteur de référence pour cet exemple.

Tâche	Tout ce qui s'exécute dans une cage
Ressources propres	Configuration de la cage Ressources auxquelles seule la cage peut accéder (par choix d'implémentation ou par configuration)
Ressources partagées	Programme de gestion des cages Noyau du système d'exploitation Matériel physique Autres, suivant les implémentations
Moniteur de référence	Noyau et programmes de gestion des cages

TAB. 2.4 – Exemple des cages (conteneurs ou bacs à sable)

3

Identifier ses besoins en cloisonnement

Ce chapitre a pour but de dégager une manière de dresser un inventaire des besoins d'un composant en matière de cloisonnement. L'identification de ces besoins permet d'orienter son développement, ou encore de choisir entre plusieurs composants déjà disponibles.

Le cahier des charges fonctionnel définissant le composant est supposé établi. Le composant est, à ce stade, encore appréhendé de manière imprécise et monolithique.

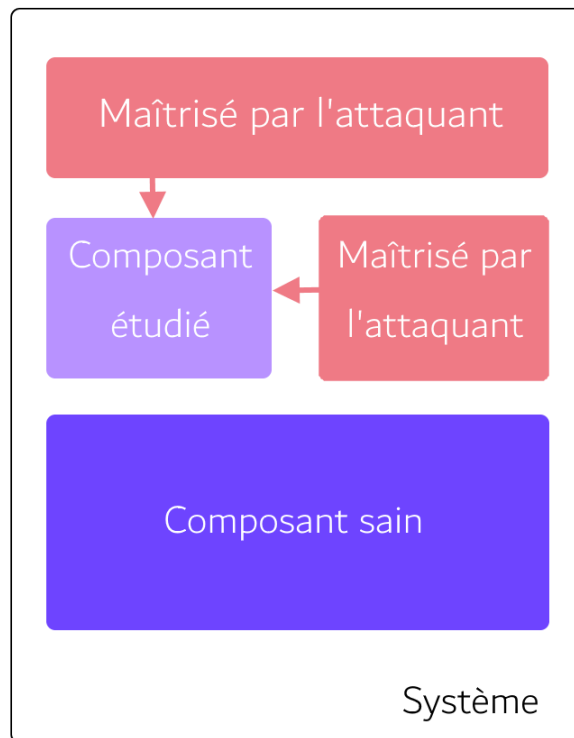


FIG. 3.1 – Perception initiale du composant au sein du système qui l'accueille

Il n'existe pas de méthode absolue à recommander inconditionnellement. Par conséquent, ce document aspire à expliquer précisément *comment* la mise en place d'un cloisonnement pertinent permet d'augmenter la sécurité d'un composant. Le lecteur du document est encouragé à s'appropriier les raisonnements présentés pour les décliner dans son propre contexte.

Dans un premier temps, ce document rappelle des fondamentaux de la sécurité du composant dans sa globalité, avant d'aborder l'introduction du cloisonnement pour augmenter le niveau de sécurité du composant.

Exemples. Pour illustrer les concepts présentés, l'exemple choisi est celui du développement d'une application métier sur un modèle client-serveur, permettant à des utilisateurs de travailler sur des projets en commun. La partie serveur de l'application, vouée à s'exécuter en espace utilisateur sur un serveur accessible par plusieurs machines distantes, est plus précisément celle évoquée.



Attention

Cet exemple n'a en aucun cas valeur d'architecture recommandée : il s'agit d'un outil pédagogique permettant d'illustrer les définitions et les raisonnements utiles à l'évaluation d'une architecture en matière de sécurité.

3.1 Généralités sur la sécurité du composant

Spécifier la sécurité attendue d'un composant consiste à identifier quatre éléments fondamentaux :

- les biens sensibles que le composant doit protéger ;
- les capacités de l'attaquant duquel le composant se protège ;
- le périmètre exact du composant ;
- les fonctions de sécurité à remplir par le composant.

Ces quatre éléments sont interdépendants.

D'autres documents de référence peuvent aider à formuler les attentes en matière de sécurité à divers niveaux de précision et d'abstraction (par exemple [5] ou [6]). Le présent document ne présente pas détailler l'élaboration de l'analyse de la sécurité attendue. Seules les définitions et les précisions qui sont utiles pour la suite sont explicitées ici.

3.1.1 Biens sensibles à protéger par le composant



Biens sensibles

Les biens à protéger sont constitués des informations sensibles manipulées par le composant. Bien souvent, il s'agit directement d'informations métier utilisées par le composant. D'une façon plus générale, cela inclut aussi les informations dont l'obtention permet celle d'informations métier utiles.

C'est en particulier classiquement le cas des clés cryptographiques et de secrets d'authentification. Souvent, ces informations ne sont pas les objets protégés in fine. Cependant, il faut les protéger d'un attaquant au même titre que les données métier car elles peuvent permettre à l'attaquant d'élargir son périmètre d'influence.

Exemples. Dans le cas de l'application serveur métier, les biens à protéger sont les données métier manipulées par les utilisateurs. Un contrôle d'accès sur ces données doit être imposé, permettant que seuls les utilisateurs autorisés puissent lire et/ou modifier des données. Ceci va engendrer la nécessité d'authentifier les utilisateurs auprès de l'application, et les secrets cryptographiques utilisés pour cette authentification seront alors ajoutés à la liste des biens sensibles.

3.1.2 Composants de confiance

Lors de l'analyse de sécurité d'un composant donné, d'autres composants sont considérés comme *de confiance*. Ceci ne signifie **pas** qu'ils ne soient pas corruptibles dans l'absolu ; il s'agit d'une hypothèse de travail. Prendre des mesures assurant leur intégrité n'est pas superflu, bien au contraire.



Composants de confiance

Les composants de confiance sont les composants logiciels et matériels supposés parfaits dans l'analyse de la sécurité attendue d'un composant. En d'autres termes, l'analyse de sécurité repose sur l'hypothèse que les composants de confiance ne peuvent pas être corrompus par un attaquant tel que défini dans cette analyse.



Attention

L'analyse de sécurité d'un composant ne peut pas être construite sur l'hypothèse que le composant entier est lui-même de confiance.

Il serait faux d'affirmer que le composant développé ne peut pas contenir de composants de confiance. Typiquement, c'est le cas s'il dispose de l'accès au matériel au plus bas niveau existant dans le contexte d'utilisation envisagé, sans contrôle possible d'une autre entité système. Cette éventualité se présente si le composant comprend un noyau de système d'exploitation non hypervisé ou un hyperviseur. Dans ce cas de figure, il convient en réalité d'effectuer une analyse de sécurité par niveau d'abstraction couvert. Par exemple, la sécurité est d'abord étudiée en supposant que le système d'hypervision et les noyaux des systèmes hypervisés sont de confiance, au contraire des espaces utilisateurs des systèmes hypervisés qui peuvent être compromis. Dans un second temps, appliquant une démarche de défense en profondeur, la sécurité est examinée en prenant pour hypothèse que seul le système d'hypervision est de confiance, et que l'attaquant est capable de compromettre les systèmes hypervisés complets.

Dans ce chapitre, le propos tenu s'applique à un niveau d'abstraction donné, supposé pour simplifier contenir tout le composant étudié. Ce dernier s'appuie donc sur des composants de confiance et un moniteur de référence externes.



Attention

Le raisonnement justifiant la sécurité du composant étudié repose sur l'hypothèse de l'intégrité des composants de confiance. Il faut donc faire en sorte que dans la réalité, cette hypothèse soit vérifiée.

R4

Garantir l'intégrité des composants de confiance

Il est impératif de garantir concrètement l'intégrité des composants de confiance pour que la sécurité de l'ensemble du composant soit assurée. Des recommandations plus précises sont fournies en 4.1.

Exemples. Dans le cas de l'application serveur, le noyau du système d'exploitation du serveur physique (et les couches d'hypervision éventuelles qui l'exécutent), ainsi que le matériel sur lequel ceci est installé font partie des composants de confiance. Les périphériques matériels (et leurs micrologiciels) utilisés par le serveur physique font également partie des composants de confiance.

3.1.3 Périmètre du composant

Dans le système au sein duquel il sera utilisé, le composant va partager des *interfaces* avec d'autres composants, ce qui peut mettre en péril les fonctions de sécurité visées. Décrire le périmètre exact du composant à sécuriser, c'est notamment dresser la liste de ses *interfaces externes*.



Interfaces du composant

Une interface de programmation définit la manière d'échanger de l'information entre deux composants.

Les interfaces externes sont des interfaces entre le composant pris dans sa globalité et son environnement d'exécution. Elles contiennent au moins toutes les interfaces exposées par le composant à ses utilisateurs.

Exemples. Dans l'exemple du chapitre, les API (Application Programming Interface) mises à disposition par l'application serveur pour ses clients sont des interfaces externes exposées. Les appels système au système d'exploitation au-dessus duquel s'exécute l'application et le système de fichiers utilisé par l'application pour stocker les données métier constituent des exemples d'interfaces externes utilisées par le composant.

Une interface externe ne sépare pas forcément deux éléments logiciels. L'interface entre un pilote de périphérique d'une part et le matériel qu'il contrôle d'autre part est mixte.



Surface d'attaque et surface de friction

La *surface d'attaque* d'un composant est constituée de toutes les interfaces externes du composant lui permettant de communiquer avec un environnement contrôlé par l'attaquant. Dans l'immense majorité des cas, toutes les interfaces exposées par le composant à ses utilisateurs en font partie.

Les autres interfaces externes du composant forment la *surface de friction* du composant avec le reste du système. La surface de friction comprend entre autres les interfaces du composant avec les composants de confiance.

Ces concepts sont illustrés sur la figure 3.2. Identifier la surface d'attaque et la surface de friction du composant avec le système global est essentiel pour permettre une prise en compte de tous les besoins en sécurité du composant. Bien définir la surface de friction permet aussi de garantir l'innocuité du composant pour le système global.

Exemples. Dans le cas de l'application serveur, la surface d'attaque est constituée de l'interface utilisateur de l'application. La surface de friction comprend tous les appels système.

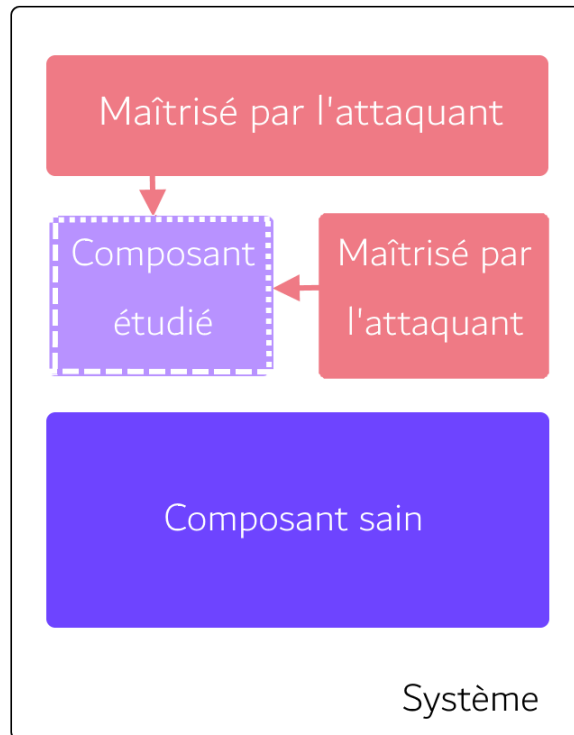


FIG. 3.2 – Interfaces externes du composant

Les appels système forment une interface entre le noyau du système d'exploitation, composant de confiance dans l'exemple du chapitre, et l'application métier. Évidemment, le noyau n'est pas « dangereux » pour l'application. Il est de confiance, et donc réputé non corrompible par un attaquant du composant.

Cependant, les appels système ne sont pas tous équivalents du point de vue de l'application développée. Ceux qui permettent d'utiliser le système de fichiers ont beau s'exécuter exactement comme prévu, ils peuvent permettre l'accès à des données métier gérées par cette application via d'autres applications s'exécutant sur le serveur, sans que l'attaquant ait corrompu l'application développée. Inversement, la compromission dudit composant ne devrait idéalement pas permettre d'accéder à toutes les données du serveur.

3.1.4 Formaliser la sécurité attendue du composant

Pour définir les besoins de sécurité d'un composant, il sera donc nécessaire de réaliser une analyse de la sécurité attendue, qui présentera le bilan de tous les éléments évoqués dans les paragraphes précédents.

R5

Rédiger l'analyse de la sécurité attendue du composant

Une analyse de la sécurité attendue du composant doit faire partie des documents de conception ou intégration mis à disposition.

Elle doit comporter les définitions des quatre éléments cités ci-dessus :

- la liste des biens sensibles à protéger ;
- le modèle d'attaquant pris en compte (duquel découle en particulier l'identification des composants de confiance) ;
- le périmètre du composant, c'est-à-dire sa surface d'attaque et sa surface de friction avec le système global ;
- les fonctions de sécurité attendues.

3.2 Définition des usages du composant

Cette section et la suivante ont pour but d'explicitier le principe global d'architecture consistant à mettre en oeuvre, au sein d'un composant, du cloisonnement entre chacun de ses usages. Mais comment déterminer ce qui relève d'un même usage ? Quand deux contextes d'utilisation font appel à des ressources de sensibilité différente, il est pertinent de séparer ceux-ci en des usages distincts. De même, la disparité des risques de compromission provenant de la manière d'interagir avec le reste du système constitue un critère de séparation. L'exemple typique est l'administration d'un composant, qu'il convient d'appréhender comme un usage particulier.



Usages d'un composant

Des scénarios d'utilisation du composant dans lesquels les ensembles de ressources utilisées diffèrent notablement constituent des usages distincts du composant. Pour mesurer cette différence, il convient d'examiner entre autres :

- les formes et sources d'interaction du composant avec l'extérieur, qui représentent autant de vecteurs de compromission potentiels : par exemple, une différence de connectivité réseau (un usage demande une connexion Internet et l'autre une connexion Intranet), ou encore l'exécution de traitements compliqués de données provenant de l'extérieur (du *parsing* par exemple) pour lesquels la présence d'une vulnérabilité ne peut être écartée ;
- la criticité et la sensibilité des ressources utilisées ;
- la périodicité de l'exécution des actions. Par exemple, certaines actions uniquement effectuées lors d'une initialisation sont à distinguer des actions utiles par la suite ;
- s'il s'agit de fonctionnalités liées uniquement au cycle de vie du composant, par opposition au service qu'il rend : administration, mise à jour, journalisation forment autant d'exemples d'usages.

R6

Caractériser des usages du composant

Réaliser une liste des usages du composant à partir de ses fonctionnalités et de la liste des ressources auxquelles il accède.

Une fois les usages identifiés, le composant peut mettre en oeuvre un cloisonnement entre usages en s'appuyant sur un moniteur de référence. L'architecture qui en résulte est représentée dans la figure 3.3 .

Exemples. Quelques exemples génériques ont déjà été mentionnés, tels que les usages d'administration, de journalisation, ou de mise à jour.

Des usages liés aux particularités métier du composant seront évidemment dépendants du service rendu. Dans l'exemple de ce chapitre, il y a un usage par utilisateur de l'application. En effet, d'une part, chaque utilisateur de l'application devrait avoir exclusivement accès aux fichiers qui le concernent. D'autre part, les postes que les utilisateurs de l'application utilisent pour se connecter sont a priori distincts et non-homogènes. Avant authentification d'un utilisateur ou d'un administrateur de l'application, aucun des usages ci-dessus n'est identifié. Cependant, il faut rester à l'écoute de nouvelles connexions, il y a donc un usage d'écoute.

Il est difficile de définir de manière générique ce qui constitue un même usage pour un composant, car l'analyse peut être réitérée à des niveaux de granularité différents et à des niveaux d'abstraction différents. Cela suppose également de placer le composant dans son contexte réel d'utilisation pour s'assurer qu'il ne viole pas le cloisonnement global mis en place au sein du système d'information qui l'accueillerait.

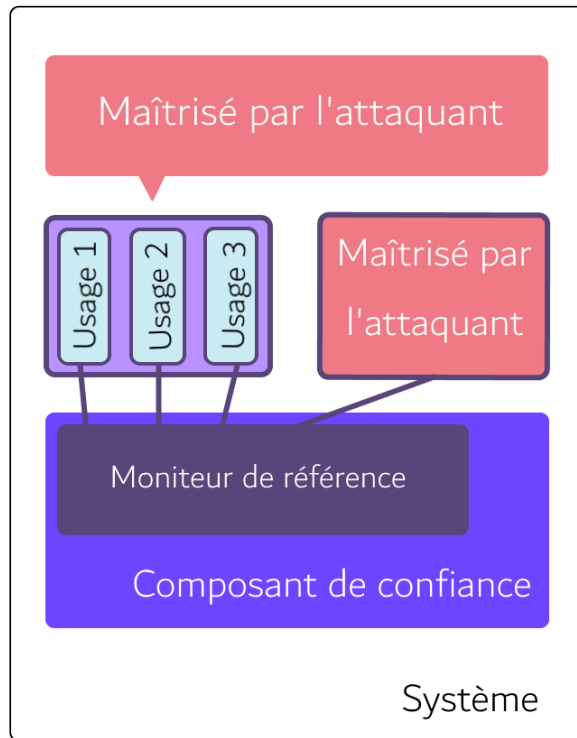



FIG. 3.3 – Nouvelle perception du composant au sein du système qui l'accueille

3.3 Objectifs de la mise en place de cloisonnement

Tous les concepts nécessaires ayant été définis et illustrés précédemment, les objectifs recherchés par la mise en place de cloisonnement peuvent être formulés. Une mise en oeuvre pertinente du cloisonnement doit remplir tous ces objectifs.

R7 | **Minimiser de la surface d'attaque**
 Réduire systématiquement la surface d'attaque pour chaque usage, de manière à n'exposer que les interfaces externes utiles pour l'usage considéré.

Exemples. Dans l'exemple, l'API exposée aux clients sera divisée en trois parties : celle dédiée à l'administration de l'application, celle dédiée à l'écoute et celle dédiée aux utilisateurs authentifiés. Chaque domaine exposera une et une seule de ces API, suivant l'usage qui lui correspond.

 | **Objectif**
 Limiter les possibilités offertes à l'attaquant de prendre le contrôle du composant pendant son utilisation.

R8 | **Cloisonner les usages entre eux**
 Mettre en place un moniteur de référence en s'appuyant sur les composants de confiance pour faire en sorte que chaque usage soit confiné dans un domaine.

Minimiser la surface de friction

Réduire systématiquement les actions possibles pour chaque domaine aux seuls besoins liés à l'usage, c'est-à-dire réduire la surface de friction avec le système global.

Exemples. Dans le cas d'étude du chapitre, il est fait en sorte que l'application serveur, quel que soit l'usage considéré, s'exécute sans privilège particulier (car elle n'en a pas besoin) : possibilité d'effectuer uniquement les appels systèmes qui lui sont utiles (gestion des fichiers, sockets, etc.), impossibilité d'utiliser d'autres fichiers que ceux des utilisateurs légitimes et ses propres fichiers (configuration, etc.). Cela peut par exemple être mis en place au moyen de techniques de bacs à sable ou conteneurs.



Objectif

Minimiser les conséquences d'une compromission du composant dans un usage donné :

1. minimiser les conséquences d'une compromission sur les fonctions de sécurité assurées par le composant ;
2. minimiser les conséquences d'une compromission sur le système global.

Exemples. Si un attaquant contrôle une session utilisateur, seuls les fichiers de ce dernier sont à disposition de l'attaquant. Le cloisonnement mis en place garantit que contrôler une session utilisateur ne permet pas de s'arroger les privilèges d'administration de l'application ou d'interférer avec les sessions des autres utilisateurs. La compromission est alors circonscrite au domaine de l'utilisateur compromis.

Concernant le deuxième point, l'absence de privilège particulier requis ou le contrôle d'accès sur les fichiers des utilisateurs empêchent une progression de l'attaquant en cas de compromission du serveur applicatif. En l'absence de cloisonnement, tous les fichiers ou appels systèmes seraient librement accessibles à l'attaquant qui contrôle les domaines du serveur. De même, l'exécution du serveur avec des privilèges inutiles les octroie systématiquement à l'attaquant.



Objectif

Protéger le composant d'une compromission par d'autres composants (non de confiance) appartenant au système global.

Exemples. Les domaines utilisateur du composant étudié sont les seuls environnements applicatifs capables d'accéder aux fichiers métier gérés dans le composant. Par conséquent, la compromission d'une application tierce s'exécutant au sein du même système d'exploitation n'entraîne pas la fuite de ces données métier.

Les usages définis doivent être cohérents pour que le système reste utilisable, mais ils doivent également se prêter de manière efficace à l'application du principe de moindre privilège. Un usage sera considéré trop vaste si la proportion d'actions possibles dans son domaine est grande par rapport à la totalité des actions disponibles en l'absence de cloisonnement.

4

Analyser la sécurité apportée par le cloisonnement mis en place

Ce chapitre énonce les critères d'évaluation du cloisonnement mis en place dans un composant. Tout d'abord, la démarche d'analyse d'un mécanisme de cloisonnement donné est explicitée, avant d'aborder les éléments permettant de se prononcer sur la sécurité de sa mise en oeuvre.

4.1 Analyse d'un mécanisme de cloisonnement

Cette section établit des éléments génériques d'analyse de la sécurité d'un mécanisme de cloisonnement donné. L'évaluation porte sur le cloisonnement en tant que fonction de sécurité, comme défini précédemment en 2.2.

La sécurité d'un composant dépend directement de l'absence de vulnérabilités dans les composants de confiance qu'il utilise. Après avoir détaillé ce qu'il est nécessaire de considérer de manière générique comme composant de confiance, le document aborde les recommandations qu'un tel composant doit respecter. Il convient de noter que certaines recommandations ne sont pas possibles à respecter en intégrant des composants sur étagère ; aussi le document distingue-t-il ce qui doit être appliqué aux composants développés de ce qui doit être appliqué systématiquement.

4.1.1 Le moniteur de référence parmi les composants de confiance

Lorsque l'environnement d'exécution d'un composant est scindé en domaines cloisonnés entre eux, le moniteur de référence est garant du respect de la politique de sécurité dans chaque domaine. C'est parce qu'il y a un moniteur que les domaines existent : c'est le moniteur qui autorise ou empêche chaque action d'une tâche sur une ressource.

R10

Considérer le moniteur comme un composant de confiance

Vis-à-vis d'un attaquant de la fonction de sécurité de cloisonnement, le moniteur de référence fait partie des composants de confiance. Ainsi, il doit respecter les recommandations de cette section.

R11

Identifier les composants de confiance

Tout composant du système qui dispose d'un privilège lui permettant de porter atteinte à l'intégrité du moniteur, ou des politiques de sécurité qu'il met en place, est à considérer comme faisant partie des composants de confiance vis-à-vis du composant analysé.

Tous ces composants doivent donc vérifier les exigences propres aux composants de confiance.

Exemples. Dans le cas du contrôle d'accès obligatoire implémenté dans le noyau d'un système d'exploitation classique, le moniteur de référence est le code gérant le contrôle d'accès obligatoire. En pratique, le code du noyau responsable d'autres choses, comme l'ordonnancement ou la gestion des périphériques, ne fait pas partie du moniteur de référence utilisé pour ce type de cloisonnement. Cependant, tout ce code s'exécute avec des privilèges suffisants pour modifier le moniteur de référence ou la politique qu'il applique. Il faut donc avoir le même niveau de confiance dans tout ce code que dans celui du moniteur à proprement parler.

4.1.2 Composants de confiance développés

Les recommandations fournies pour les composants de confiance développés sont le cahier des charges d'un composant de confiance idéal. Même s'il existe très peu de cas réels dans lesquels toutes ces recommandations sont respectées, il est important de disposer de cette liste de recommandations pour y revenir et se forger un avis sur la sécurité et la robustesse d'un composant de confiance donné.

R12

Concevoir des composants simples et concis

Les composants de confiance doivent être conçus en suivant des spécifications claires et complètes, permettant de définir et combiner des éléments simples (suivant le principe « Keep It Short and Simple ») qui facilitent développement et validation.

L'utilisation de langages fortement typés (tels Ada, OCaml, Rust, Go, etc.) permet de réduire notablement le nombre de vulnérabilités exploitables dans un composant.

R13

Choisir un langage approprié

Utiliser un langage de programmation fortement typé, qui entre autres prévient les débordements de tableau, d'entier, ou l'utilisation de pointeurs invalides, est fortement souhaitable pour le développement des composants de confiance.

R14

Développer selon un référentiel de sécurité

Un référentiel de développement sécurisé doit être utilisé systématiquement lors du développement de composants de confiance.

La vérification du respect du référentiel de codage doit être assurée.

Le respect de cette recommandation est critique dans le cas du choix d'un langage ne respectant pas les critères conseillés ci-dessus.

Des exemples de tels référentiels incluent les standards SEI CERT Coding Standard ([2]) et le guide mis à disposition par MISRA ([1]).

R15

Auditer le code de l'implémentation des composants de confiance

L'intégralité du code des composants de confiance devra faire l'objet d'un audit de code par des personnels qui n'en sont pas développeurs, de préférence indépendants du projet concerné.

R16

Valider l'implémentation des composants de confiance

Des tests fonctionnels complets devront permettre de valider le bon fonctionnement des fonctions de sécurité implémentées. Une attention particulière sera portée à tester aussi bien les opérations qui doivent être refusées que celles qui doivent être menées à bien avec succès.

R17

Prouver l'implémentation des composants de confiance

Il est fortement recommandé de compléter les tests par une preuve formelle de l'absence d'erreur à l'exécution d'un composant de confiance, par exemple à l'aide d'outils d'analyse (statique ou dynamique) pour prévenir certains types de vulnérabilités.

4.1.3 Recommandations portant sur tous les composants de confiance

Les recommandations précédentes sont très exigeantes, et sont très rarement applicables aux composants sur étagère intégrés. Qu'un composant de confiance respecte ou non le cahier des charges établi par ces recommandations, il doit respecter les recommandations suivantes.

R18

Supprimer toute partie inutilisée d'un composant de confiance

Un composant de confiance doit être minimal. Dès que possible, la suppression du code inutilisé sera privilégiée. A défaut, sa désactivation par configuration sera effectuée.

R19

Appliquer des techniques de durcissement aux composants de confiance

Les composants de confiance doivent se voir appliquer des techniques de durcissement à l'état de l'art afin de compliquer l'exploitation d'une vulnérabilité et le détournement du flot de contrôle, comme par exemple :

- présence de motifs d'intégrité de la pile et du tas (canaris) ;
- principe $W \oplus X$: au cours de toute son utilisation par le système, une zone mémoire donnée ne doit pas être inscriptible et exécutable, que ce soit simultanément ou non ;
- répartition aléatoire de l'espace d'adressage (ou Address Space Layout Randomization (ASLR)).

Exemples. Dans le cas d'un système Linux, le respect de ces deux recommandations peut impliquer une recompilation d'un noyau minimal avec les options satisfaisantes. Consulter le document rédigé sur la sécurité de ces systèmes [4] est recommandé.

4.1.4 Recommandations spécifiques à un moniteur de référence

Dans son ouvrage sur la sécurité des systèmes d'exploitation [9], Trent Jaeger définit et présente en détails chacune des propriétés exigées ci-dessous. Comme plus haut, il s'agit d'un idéal vers lequel tendre, en particulier lors de l'intégration de solutions sur étagère. Tout ce qui est détaillé ici peut fournir des critères de choix entre solutions ou dans leurs configurations. Tout moniteur de référence développé devrait suivre ces recommandations.

R20

Justifier le respect des propriétés fondamentales du moniteur de référence

Le développement, l'intégration et la validation du moniteur de référence doivent permettre de garantir qu'il vérifie les propriétés suivantes.

1. *Complétude du contrôle exercé* : le moniteur est impliqué pour chaque tentative d'accès à une ressource, et ce au moment opportun (i.e. absence d'attaques TOCTTOU (Time Of Check To Time Of Use), liées à une évolution des propriétés de l'objet entre le moment de la requête au moniteur et l'action effective).
2. *Maintien de l'intégrité du moniteur.*
3. *Validation et audit de la politique de sécurité.* La possibilité de consulter la politique de sécurité implémentée par le moniteur à un instant donné permettra l'audit de celle-ci.

Des justifications construites confirmant le respect de ces propriétés doivent figurer dans les documents de conception du composant utilisant le moniteur de référence.

Comme vu en section 2.2, l'attaquant de la fonction de sécurité de cloisonnement mise en place par le moniteur est supposé contrôler une ou plusieurs tâches. Le but est de s'assurer, en présence de cet attaquant, que les tâches cloisonnées soient dans l'incapacité de s'arroger des permissions supplémentaires en modifiant l'environnement d'exécution du moniteur. Pour protéger le moniteur, il faut donc garantir qu'un attaquant contrôlant une tâche **ne dispose pas** des privilèges nécessaires à une modification en sa faveur.

R21

Assurer au moniteur de référence un niveau de privilège supérieur à celui des tâches cloisonnées

De manière à préserver l'intégrité du moniteur et des politiques de sécurité qu'il met en application, les tâches cloisonnées ne doivent pas disposer des privilèges nécessaires à la relaxe de la politique de sécurité appliquée.

Une façon de garantir ceci est d'assurer que les politiques sont non-modifiables par les tâches cloisonnées et que le moniteur s'exécute à un niveau de privilège supérieur à celui des tâches qu'il cloisonne.

Exemples. Effectuer une comparaison de deux choix architecturaux sur l'exemple du serveur applicatif présenté dans le chapitre précédent permet d'illustrer cette recommandation. Dans les

deux cas, une session utilisateur correspond à un processus. Ce n'est pas sur cet aspect que porte la comparaison, mais sur la mise en place du contrôle d'accès sur les fichiers utilisateur.

Le premier choix consiste à écrire un module de code dédié à la prise de décision, qui s'exécutera au sein du processus gérant un utilisateur du serveur. A chaque fois qu'un utilisateur veut accéder à un fichier, un appel est effectué aux fonctions du module, qui retourne un refus ou un descripteur de fichier vers le fichier demandé.

Le second choix consiste à exécuter chaque session utilisateur dans un processus distinct sous une identité (uid) qui lui est propre, et à s'appuyer sur le noyau pour implémenter le contrôle d'accès sur les fichiers.

Contrairement à la seconde solution, la première alternative **ne cloisonne pas** les sessions utilisateur entre elles. En effet, l'attaquant est supposé contrôler un ou plusieurs domaines cloisonnés. Ici, rien n'isole le module dédié au contrôle d'accès du reste du code des sessions utilisateur. Il est donc impossible de garantir que les ouvertures de fichiers passent par le module, et il est impossible de garantir son intégrité. Un attaquant en capacité d'exécuter du code arbitraire au sein de l'environnement d'une session utilisateur a le contrôle du processus, et donc peut modifier le code exécuté à sa guise.

En terme de sécurité, un moniteur de référence bénéficiera à l'usage de configurations par défaut le moins permissives possible. Même dans le cas de l'utilisation d'un moniteur de référence sur étagère, ces recommandations peuvent être appliquées.

R22

Appliquer le principe d'interdiction par défaut

Un moniteur de référence doit être configurable pour que toute action soit interdite à une tâche, sauf à lui être explicitement autorisée.

L'application de ce précepte prétend éviter l'oubli d'interdiction d'actions que le développeur ou l'utilisateur du composant n'aurait pas envisagées, en obligeant à dresser la liste des actions qu'un composant doit pouvoir effectuer.

Pouvoir appliquer une telle configuration au moniteur de référence est salutaire, surtout lorsque la politique de sécurité est difficilement auditable. Pourtant, les mécanismes de cloisonnement ne respectent pas tous ce principe. Par exemple, dans le cadre du cloisonnement entre processus présenté précédemment, il est quasiment impossible sans s'appuyer sur d'autres mécanismes de cloisonnement supplémentaires de s'assurer que deux processus ne communiquent pas entre eux.

R23

Minimiser le nombre et l'impact des configurations possibles

Dans une démarche de durcissement du moniteur, les options de configuration laissées au choix de l'utilisateur (même privilégié) en production doivent être restreintes au strict minimum, de manière à réduire l'impact possible d'une erreur sur la sécurité globale du système. Pour les choix laissés à l'utilisateur, les messages d'avertissement sur les conséquences en terme de sécurité doivent être suffisamment explicites.

Exemples. Ne pas permettre de désactiver les alertes, la journalisation, ou d'activer les fonctions de débogage dans une version de production est pertinent. Demander une confirmation lors d'un changement de configuration, documenter les options de configuration de manière extensive pour qu'elles puissent être maîtrisées par les administrateurs du produit sont également de bonnes pratiques.

4.1.5 Évaluer un mécanisme de cloisonnement

Évaluer un mécanisme de cloisonnement, c'est mesurer l'effort nécessaire à un attaquant pour le mettre en défaut. Pour quantifier la difficulté de contourner un mécanisme, il est recommandé d'appliquer la démarche suivante.

1. Identifier ce à quoi correspondent les définitions de tâches, ressources propres et partagées et moniteur de référence dans le contexte du mécanisme de cloisonnement étudié.
2. Dédire du contexte d'emploi (générique ou souhaité) l'ensemble des composants de confiance.
3. Mesurer l'écart entre les caractéristiques réelles de l'implémentation du moniteur de référence et les recommandations fournies.
4. Mesurer l'écart entre les caractéristiques réelles des composants de confiance et les recommandations à respecter listées dans ce document. Des critères de mesure de cet écart peuvent aussi comprendre la mesure de la maturité du produit, le niveau d'assurance quant à son maintien en conditions opérationnelle et de sécurité, ou encore son historique en termes de parution de vulnérabilités, etc.
5. Evaluer la possibilité de réduire les écarts identifiés en appliquant des techniques de durcissement, comme la minimisation de l'ensemble des composants de confiance par désinstallation de programmes inutiles, l'usage de versions durcies des composants de confiance... Le combinaison de plusieurs mécanismes traitant divers types de ressources peut également être envisagée³.

4.2 Analyse de la mise en place du cloisonnement

Tous les outils permettant de mener une analyse de la sécurité apportée par l'implémentation de cloisonnement sont maintenant à disposition du lecteur.

3. à la manière des conteneurs Linux combinant espaces de noms et cgroups pour aboutir à une solution.

4.2.1 À niveau d'abstraction donné

Une telle analyse débutera par une spécification claire de ce qui est mis en oeuvre, avant de procéder à la vérification du respect de tous les points recommandés précédemment.

R24

Spécifier le cloisonnement proposé

Pour présenter le cloisonnement mis en place, il est recommandé de procéder de la manière suivante.

1. Expliciter les usages identifiés.
2. Décrire le mécanisme de cloisonnement mis en oeuvre en explicitant ce à quoi correspondent les définitions de tâches, ressources propres et partagées et moniteur de référence.
3. Expliciter la politique de sécurité mise en place pour chaque usage. Pour chaque domaine, il faut reprendre la liste des interfaces externes dressées précédemment et caractériser les actions autorisées.

Pour structurer l'évaluation de la sécurité apportée par le cloisonnement mis en place au sein d'un composant, la liste de critères suivante peut être suivie :

1. adéquation entre les scénarios vraisemblables d'utilisation du composant et les usages définis ;
2. pertinence du choix du mécanisme de cloisonnement. Vérifier la compatibilité de l'ensemble des composants de confiance issus de la définition du mécanisme de cloisonnement avec la définition des composants de confiance telle que donnée par l'analyse de sécurité. Vérifier que la difficulté de la mise en défaut du mécanisme utilisé correspond au moins au niveau estimé de l'attaquant duquel le système doit être protégé ;
3. vérification que la politique de sécurité des domaines implémente bien le principe de moindre privilège (minimisation de la surface d'attaque et de la surface de friction) ;
4. vérification de la complétude du cloisonnement : garantie de la couverture de toutes les interfaces externes du composant ;
5. évaluation de l'innocuité du composant pour le système qui l'exécute, par l'évaluation des conséquences de sa compromission.

R25

Assurer l'innocuité du composant pour le système qui l'accueille

Un composant ne doit pas dégrader la sécurité globale du système, notamment en limitant la mise en place du cloisonnement par d'autres composants utilisés sur le même système.

En particulier, un composant ne doit pas exiger pour fonctionner d'abaisser le niveau de sécurité du système qui l'héberge : le composant doit être compatible avec un niveau de durcissement à l'état de l'art au moment de sa mise en production.

4.2.2 Raffinement du cloisonnement à l'intérieur du composant

Dans une démarche de durcissement et défense en profondeur, il est pertinent de mettre en place du cloisonnement interne dans les différents domaines. Ceci est d'autant plus important que les

domaines sont de taille importante, par exemple s'il s'agit de machines virtuelles ou de conteneurs. Il suffit de se placer dans un domaine donné et d'itérer l'analyse de sécurité présentée ci-dessus à l'intérieur du domaine.

Par exemple, à l'intérieur d'une machine virtuelle ayant pour fonction d'héberger un serveur Web, il convient d'isoler dans un conteneur ce qui relève des fichiers et programmes nécessaires au serveur Web.

Le cloisonnement dépend du niveau d'abstraction envisagé, et sa mise en place peut être faite par divers moyens. Des solutions de cloisonnement emboîtées comme des poupées russes forment autant de barrières entre la compromission par un attaquant du maillon le plus faiblement sécurisé présent sur un système et le contrôle complet du système par cet attaquant.

5

Éléments d'analyse d'une architecture de sondes de détection réseau

Cette section a pour vocation d'illustrer, à partir d'un exemple concret, la démarche que ce document vise à transmettre. L'étude porte sur la conception hypothétique d'une sonde réseau de détection d'incidents de sécurité, nommée exIDS dans la suite.

La suite de cette section s'appuie fortement sur deux documents que le lecteur est invité à consulter. D'une part, l'article « Architecture système sécurisée de sonde IDS réseau » ([10]) fournit une description assez détaillée de choix architecturaux. D'autre part, la cible de sécurité générique pour ce type de produits ([3]) présente l'analyse de sécurité nécessaire à la bonne conception d'exIDS. L'article étant paru avant et indépendamment du travail de rédaction de cette cible de sécurité, il existe quelques détails à adapter pour que les deux documents soient parfaitement conciliables. L'exemple exIDS est le simple fruit de cette adaptation. Une première architecture est envisagée, puis modifiée pour satisfaire des contraintes de performance.

L'étude proposée ici n'a pas pour objet de recommander une architecture plutôt qu'une autre. Il s'agit plutôt d'un prétexte à un exercice d'architecture comparée. D'autres solutions satisfaisant la cible de sécurité pourraient être envisagées, par exemple plus raffinées dans le cloisonnement proposé, ou s'appuyant sur de la virtualisation plutôt que des conteneurs Linux.

5.1 Première proposition d'architecture pour exIDS

La cible générique prévoit quatre utilisateurs de la sonde : auditeur et administrateur local d'une part, opérateur et administrateur système d'autre part⁴. Chacun de ces utilisateurs correspond à un usage dans l'architecture proposée. La capture des flux réseau bruts entrants nécessite des privilèges inutiles aux autres usages ; elle est identifiée dans un premier temps comme un usage particulier. Par ailleurs, comme dans l'article cité, chaque logiciel IDS utilisé constitue un usage distinct. Enfin, la collecte des alertes relevées par les logiciels IDS et leur transmission à l'opérateur est également un usage. Les logiciels IDS effectuent des traitements complexes sur les données issues des captures, et sont donc potentiellement porteurs de vulnérabilités. Ainsi, il convient de les séparer du reste du composant, et de les séparer entre eux : ceci permet de circonscrire la compromission éventuelle d'un de ces logiciels au domaine de celui-ci. De surcroît, cela permet de n'exposer aucune interface réseau à ces domaines, puisque les logiciels IDS n'en ont pas besoin. En rendant inaccessible le réseau dans ces domaines, le risque de rebond depuis la sonde dans les réseaux qui lui sont reliés suite à une compromission des logiciels IDS est couvert. Enfin, comme

4. L'administration est ici divisée en usages distincts. L'administrateur système agit a priori à distance et ne peut que mettre à jour et redémarrer la sonde. L'administrateur local peut effectuer toutes les autres actions, tenant ainsi plus un rôle d'administrateur métier. Ce document suit la terminologie du document cible générique [3].

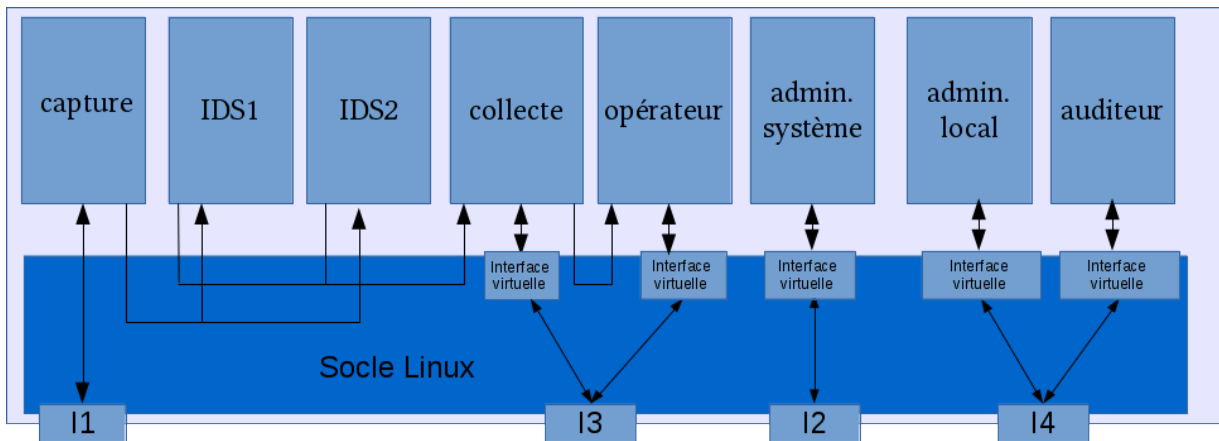


FIG. 5.1 – Première architecture envisagée pour la sonde exIDS

souligné dans l'article, la capacité à mettre à jour le composant est fondamentale ; elle n'est pas détaillée ici car aucun élément nouveau n'est intégré à exIDS.

À l'instar de ce qui est choisi dans l'article, exIDS présente un socle constitué d'un noyau Linux re-compilé avec des options de durcissement adéquates, les patches PaX et grsecurity et la suppression de tous les modules, fonctionnalités et options du noyau inutilisés. Les protections système décrites dans l'article sont appliquées. Chaque usage listé est associé à un conteneur ; les conteneurs vérifient les prescriptions de l'article également. Les flux autorisés entre usages sont limités à ceux spécifiés dans les documents référencés. L'architecture envisagée est présentée dans la figure 5.1.

Le cloisonnement entre usages repose donc dans exIDS sur des conteneurs. Le socle constitue le moniteur de référence. Les tâches sont les programmes s'exécutant dans les conteneurs, les ressources partagées comprennent les partages de fichiers documentés dans l'article, ainsi que le socle et le matériel en commun. Certains répertoires sont partagés entre domaines, mais il n'existe pas de partage de fichiers dans lequel deux domaines distincts peuvent écrire. Ceci permet de créer des communications unidirectionnelles entre domaines. Les interfaces réseau virtuelles éventuellement disponibles dans les conteneurs sont des ressources propres. Les conteneurs n'ont pas accès à des interfaces réseau physiques, à l'exception de l'usage de capture qui nécessite les privilèges adéquats à la récupération bas-niveau de paquets réseau. Un pare-feu local est mis en place au sein du socle de manière à restreindre aux stricts flux spécifiés les flux réseau possibles entre interfaces.

Disposer d'interfaces réseau distinctes comme exigé par la cible de sécurité permet de prolonger le cloisonnement au niveau matériel : seule l'interface utilisée par l'usage, lorsqu'elle existe, est accessible dans le conteneur adéquat. En adoptant le point de vue du cloisonnement réseau, la présence de plusieurs interfaces physiques et le cloisonnement entre les domaines qui les utilisent permet de ne pas créer de jonctions facilement exploitables entre réseaux physiques disjoints. Comme mentionné précédemment, le fait qu'aucune interface ne soit disponible dans les conteneurs liés aux usages IDS joue également en faveur de l'innocuité du composant pour le système. Pour finir le tour des définitions, les composants de confiance sont constitués du matériel, dans son intégralité cette fois, ainsi que du socle.

Concernant les interfaces externes, seules les interfaces entre domaines destinées à servir les flux

documentés dans l'article ou la cible seront à disposition dans chaque conteneur. En particulier, les fichiers exposés dans les conteneurs sont en lecture seule, sauf dans le cas particulier d'un besoin d'écriture dans un répertoire bien défini. Ceci sera utilement complété par l'interdiction des appels système inutilisés par les processus légitimes, la réduction des capacités disponibles aux programmes dans les conteneurs au strict nécessaire, ainsi que la limitation de l'utilisation de ressources système.

Dans une analyse d'un produit réellement développé, les détails d'implémentation devraient être explicités. Cependant, il convient d'insister sur le fait que l'analyse architecturale haut niveau présentée ici est réalisable avant tout développement de preuve de concept qui permettrait de valider la faisabilité et l'impact sur les performances des choix effectués. A contrario, introduire après une première phase de développement le cloisonnement impliquerait vraisemblablement des changements drastiques sur le produit final difficiles à maîtriser et coûteux.

5.2 Architecture retenue pour exIDS

Après avoir prototypé le projet exIDS, les développeurs concluent que l'architecture proposée initialement s'avère trop ambitieuse au niveau du cloisonnement pour que le produit puisse offrir des performances satisfaisantes. Ceci est dû à la manière dont fonctionnent la plupart des logiciels IDS. D'une part, pour fonctionner, un tel logiciel a besoin des données acquises par la carte. Recopier ces données pour les mettre à disposition prendrait un temps considérable non-compatible avec le niveau de performances attendu de plusieurs gigabits par seconde. D'autre part, le logiciel, au moins au début de son exécution, a en général besoin d'accès à des registres de configuration de la carte qu'il utilise. Régler ces deux problèmes en préservant l'architecture précédente et en satisfaisant les attentes en termes de performances, sans modifier de manière assez invasive les logiciels IDS utilisés est jugé irréaliste par les concepteurs d'exIDS.

Une solution envisagée pour régler les difficultés consiste à supprimer l'usage de capture (et donc le conteneur correspondant) et à mettre à disposition des logiciels IDS la carte réseau I1. Le reste de l'architecture peut être conservé ; la figure 5.2 présente le nouveau projet. Pour valider un tel changement architectural, il est impératif de vérifier que les deux risques que couvraient l'usage de capture sont traités par d'autres mesures. Premièrement, l'effort nécessaire à un attaquant exploitant une vulnérabilité dans un logiciel IDS pour prendre le contrôle de la sonde devrait être similaire. Deuxièmement, il doit rester impossible en toutes circonstances d'utiliser la sonde pour pénétrer le réseau qu'elle est supposée surveiller.

Concernant le premier risque, en s'appuyant sur les espaces de noms réseau et les conteneurs dédiés à chaque logiciel IDS, il est raisonnable de considérer que l'accès à l'interface I1 n'augmente pas significativement le risque de prise de contrôle d'autres conteneurs ou du socle de la sonde. Cependant, la surface de friction des conteneurs IDS est clairement plus importante que dans l'architecture initiale. Aussi, faire en sorte que les logiciels IDS n'aient que les capacités réellement utiles et perdent droits et accès dès qu'ils n'en ont plus besoin⁵ s'avère une précaution supplémentaire profitable.

Quant au second risque, il est impossible de garantir au sein du produit l'absence de flux de communication initiés par la sonde vers le réseau surveillé. Dans ce contexte, utiliser un autre produit

5. Typiquement, après avoir effectué les opérations de configuration du matériel requises à leur lancement.

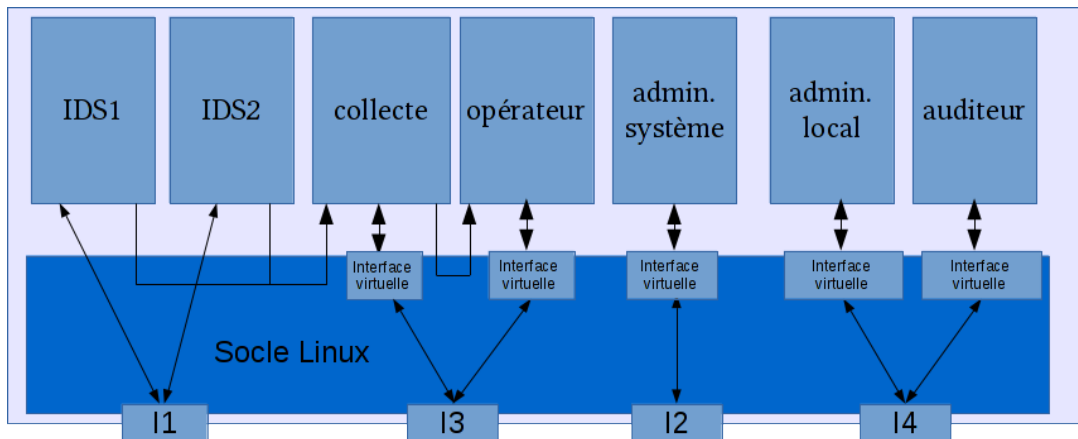


FIG. 5.2 – Architecture retenue pour la sonde exIDS

à même de garantir des communications unidirectionnelles pour relier l'interface I1 au réseau surveillé offre une solution. Cette mesure organisationnelle est d'ailleurs déjà connue du lecteur attentif de la cible de sécurité citée précédemment, où apparaît l'hypothèse sur l'environnement H2 de l'existence d'un TAP unidirectionnel qualifié.

En conclusion, les risques ouverts par la modification architecturale sont considérés couverts. La nouvelle architecture est retenue comme présentant une solution satisfaisante en terme de cloisonnement vis-à-vis du besoin spécifié.

Bibliographie

- [1] *MISRA Publications.*
<https://www.misra.org.uk/Publications/tabid/57/Default.aspx>.
- [2] *SEI CERT Standard.*
<https://www.securecoding.cert.org/confluence/x/BgE>.
- [3] *Sonde réseau de détection des incidents de sécurité.*
Cible de sécurité, ANSSI, mai 2017.
https://www.ssi.gouv.fr/uploads/2015/03/20170512-profil-de-protection-cspn-np_v1.41.pdf.
- [4] *Recommandations de sécurité relatives à un système GNU/Linux.*
Note technique DAT-NT-002/ANSSI/SDE/NP v1.1, ANSSI, juillet 2012.
<https://www.ssi.gouv.fr/reco-securite-systeme-linux>.
- [5] *Expression des besoins et identification des objectifs de sécurité.*
Guide Version 1.1, ANSSI, janvier 2010.
<https://www.ssi.gouv.fr/ebios/>.
- [6] *Instruction - Critères pour l'évaluation en vue d'une certification de sécurité de premier niveau.*
Référentiel ANSSI-CSPN-CER-I-02 v1.1, ANSSI, avril 2014.
https://www.ssi.gouv.fr/uploads/2015/01/ANSSI-CSPN-CER-I-02_Criteres_pour_evaluation_en_vue_d_une_CSPN_v1-1.pdf.
- [7] *Understanding and Hardening Linux Containers.*
Aaron Grattafiori.
Rapport technique, NCC Group, 2016.
https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-1-1.pdf.
- [8] *Abusing Privileged and Unprivileged Linux Containers.*
Jesse Hertz.
Rapport technique, NCC Group, 2016.
<https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/june/abusing-privileged-and-unprivileged-linux-containers.pdf>.
- [9] *Operating System Security.*
Trent Jaeger.
Ouvrage scientifique, 2008.
- [10] *Architecture système sécurisée de sonde IDS réseau.*
Arnaud Fontaine Pierre Chifflier.
Publication scientifique, ANSSI, novembre 2014.
<https://www.ssi.gouv.fr/publication/architecture-systeme-securisee-de-sonde-ids-reseau>.

ANSSI-PG-040
Version 1.0 - 14/12/2017
Licence ouverte/Open Licence (Étalab - v1)

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de La Tour-Maubourg, 75700 PARIS 07 SP
www.ssi.gov.fr / conseil.technique@ssi.gov.fr

