Architecture système sécurisée de sonde IDS réseau

Pierre Chifflier and Arnaud Fontaine

Agence Nationale de la Sécurité des Systèmes d'Information
{pierre.chifflier,arnaud.fontaine}@ssi.gouv.fr

Résumé Les systèmes de détection d'intrusion réseau (NIDS [1]) sont largement utilisés pour effectuer la supervision de sécurité. Ces systèmes sont cependant eux-mêmes peu sécurisés, alors qu'ils sont par nature très exposés aux attaques.

Nous proposons une architecture de référence pour les sondes hébergeant des logiciels IDS [2]. Cette architecture renforce la sécurité des logiciels IDS et permet de limiter les conséquences d'une attaque sur la sonde. Elle permet de définir des nouveaux flux d'information pour traiter les alertes collectées.

Un prototype d'implémentation, adapté à la détection sur des réseaux industriels, est réalisé pour démontrer la faisabilité et analyser l'efficacité des protections et leur impact sur les performances.

Keywords : architecture, IDS, sécurité, réseau, détection

1 Introduction

En complément des mesures de protection appliquées sur un réseau, il est classique de déployer un système de supervision. Ce système a pour but d'observer en temps réel les communications afin de détecter des tentatives de compromission des équipements connectés au réseau supervisé, mais également les compromissions réussies de ces équipements.

De part leur fonction et leur exposition, les sondes IDS réseau constituent une cible potentielle pour les attaquants : elles embarquent une grande quantité de décodeurs protocolaires, sont présentes sur la plupart des réseaux, et perçoivent l'ensemble du trafic.

Les décodeurs protocolaires au cœur des logiciels IDS ont un rôle très sensible en sécurité car leurs implémentations sont fréquemment sujettes à l'introduction de vulnérabilités [2,3]. En effet, les protocoles à décoder sont d'une part de plus en plus complexes, et d'autre part les spécifications qui les accompagnent sont parfois incomplètes ou erronées, lorsqu'elles existent. Le plus souvent écrits dans des langages bas niveaux tels que le C pour des raisons de performances, ces décodeurs sont alors potentiellement vulnérables à de nombreuses classes de problèmes : débordements mémoire, injection de code, corruption mémoire, double

1. *Network Intrusion Detection System*
2. *Intrusion Detection System*

libération, *etc.* Bien que ces problèmes soient connus et récurrents, aucun changement notable n'est constaté sur le choix des langages [10] employés par les logiciels IDS.

En ciblant directement les sondes, un attaquant peut rendre aveugle la supervision de sécurité d'un réseau. Au sein d'un même système de supervision, les sondes sont généralement identiques : une vulnérabilité exploitable sur l'une d'entre elles est *de facto* reproductible sur toutes les autres, ce qui peut conduire à l'indisponibilité (ou la compromission) de l'ensemble du système de supervision de sécurité.

Lorsque le logiciel IDS d'une sonde s'exécute avec des privilèges importants, un attaquant qui réussit à exploiter une vulnérabilité avec succès peut prendre le contrôle de la sonde. Si les sondes sont inter-connectées et gérées de manière centralisées, l'attaquant peut alors remonter de la sonde vers la machine d'administration, puis rebondir vers d'autres machines. Le réseau de supervision de sécurité se retrouve alors dans la situation singulière où il risque de devenir le réseau permettant de compromettre l'ensemble des machines supervisées.

Enfin, les alertes correspondant aux événements détectés sont souvent remontées sur le même réseau que le réseau surveillé. La confidentialité et l'intégrité des événements ne sont donc pas toujours assurées.

Face à ces constatations, il nous est apparu essentiel d'assurer la sécurité des éléments de détection réseau, condition indispensable pour avoir confiance dans le système de supervision de sécurité. L'objectif de cet article est de proposer une architecture de sonde IDS réseau, en s'appuyant sur un matériel muni de fonctions de sécurité, permettant d'en améliorer la sécurité intrinsèque, indépendamment du logiciel IDS utilisé. En effet, la modification du logiciel IDS peut s'avérer complexe à réaliser, en particulier sans introduire de vulnérabilité, mais rend surtout difficile sa maintenance en conditions opérationnelle et de sécurité.

Cet article comporte trois parties. Nous détaillons tout d'abord dans la section 2 les bases systèmes nécessaires à la conception d'une architecture système sécurisée de sonde IDS réseau. Dans la section 3, nous expliquons ensuite comment ces protections permettent de définir une architecture robuste, basée sur l'isolation des rôles et sur des flux d'informations unidirectionnels. Enfin, nous expliquons dans la section 4 comment nous avons conçu un prototype de sonde légère mettant en œuvre cette architecture et quelques résultats expérimentaux permettant de valider cette architecture.

2 Protections système

La fonction principale d'un IDS étant d'analyser du trafic réseau, on peut considérer que tout le trafic qu'il perçoit est potentiellement malveillant. L'objectif des protections présentées dans cette section est de s'assurer que l'attaquant ne puisse pas désactiver la fonction IDS, corrompre le journal d'alertes, prendre le contrôle de la machine hébergeant la fonction IDS, voire remonter dans le réseau de supervision. Il ne doit pas non plus pouvoir faire fuir de l'information depuis l'IDS, par exemple les règles de l'IDS qui pourraient être sensibles.

L'idée générale est d'appliquer au maximum les principes de défense en profondeur et de moindre privilèges. Les protections apportées peuvent être classées en deux catégories : celles qui renforcent la sécurité des logiciels s'exécutant sur la sonde présentées dans la section 2.1, et celles qui renforcent l'isolation des composants du système présentées dans la section 2.2. La section 2.3 présente les éléments matériels nécessaires au bon fonctionnement de l'architecture proposée, notamment au regard de l'impact sur les performances.

2.1 Système d'exploitation

Afin de protéger n'importe quel logiciel IDS qui s'exécute sur la sonde, la sécurité de l'architecture proposée s'articule autour de deux axes : la configuration du système, et le durcissement du noyau.

Le travail initial consiste à appliquer les principes essentiels de sécurisation [5] : réduire la surface d'attaque, supprimer les applications et services inutiles, réduire les permissions et les privilèges des processus importants, et recompiler les applications et services avec les options de durcissement de `gcc`. Pour empêcher la persistance d'une attaque réussie, et assurer que chaque démarrage se fait dans un état « propre », l'environnement d'exécution des processus est figé : toute modification (configuration, fichiers exécutés ou installés, accès aux périphériques) est interdite en montant toutes les partitions en lecture seule, à l'exception de celle contenant des données variables (telles que les journaux) qui doivent persister entre les redémarrages.

Bien qu'essentiels, ces éléments de configuration ne sont pas suffisants. Des modifications plus profondes s'avère en effet nécessaires, comme l'application des *patches* PaX et grsecurity [7] au noyau. Sans entrer dans le détail des bénéfices apportés par ces modifications [4], il convient toutefois de citer les propriétés les plus pertinentes dans le cadre de l'élaboration d'une sonde durcie :

- espace d'adressage aléatoire (ASLR [3]) ;
- vérification de débordement lors de copies de données ;
- interdiction d'exécuter du code situé en espace utilisateur depuis l'espace noyau ;
- restrictions sur les programmes utilisateurs : suppression de la visibilité des programmes des autres utilisateurs, interdiction de créer des pages en écriture et exécution, allocations de mémoire forcée à des adresses aléatoires, interdiction du *debugging* d'un processus ;
- protections additionnelles des conteneurs : réduction systématique des *capabilities*, restrictions additionnelles appliquées aux environnements confinés (*chroots*) ;
- montage définitif d'une partition en lecture seule (sans possibilité de refaire un montage avec écriture) ;
- *Trusted Path Execution*, qui permet de s'assurer qu'un utilisateur ne peut pas exécuter de programmes situés dans un répertoire qui n'appartient pas à *root*.

3. *Address Space Layout Randomization*

L'application de ces *patches* est complétée par une configuration minimale du noyau, qui supprime les éléments inutiles (USB, accélération graphique, *etc.*) voire dangereux (chargement des modules, accès direct à la mémoire, *etc.*).

Il est important de noter que ces mesures ne modifient pas le comportement de l'IDS, ou ses vulnérabilités intrinsèques. Si les techniques classiques telles que l'exécution sur la pile seront détectées et bloquées, certaines autres restent applicables, comme le ROP [4] [12] par exemple.

Afin de diminuer autant que possible les conséquences d'une compromission, nous proposons d'utiliser, en plus des protections décrites ci-dessus, un mécanisme de cloisonnement.

2.2 Cloisonnement

Le cloisonnement permet d'ajouter une isolation supplémentaire entre les différentes fonctions à accomplir dans la sonde, et ainsi de maîtriser plus finement les communications entre ces différents composants logiciels. De cette manière, chaque processus isolé ne voit que l'environnement matériel et logiciel qui lui est strictement nécessaire, et voit ses possibilités de modification du système réduites.

Le cloisonnement est utilisé pour créer un flux de traitement des données, et des séparations entre les rôles des utilisateurs et des processus. Ces aspects seront décrits en détail dans la section 3.

Il existe différentes méthodes pour mettre ce mécanisme en œuvre : la virtualisation, ou le contrôle d'accès obligatoire.

La virtualisation permet de créer une isolation, soit par l'utilisation de fonctions matérielles (virtualisation assistée), soit logicielles (traduction d'instructions ou interprétation). Elle peut être complète, par la virtualisation d'un systèmes d'exploitation et des processus (Qemu ou Xen par exemple), ou légère, en partageant le noyau mais en créant des conteneurs pour isoler des processus. Parmi les mécanismes de virtualisation légère, on trouve trouve notamment LXC [5], VServer, OpenVZ, les « jails » de FreeBSD et les « zones » de Solaris.

Le contrôle d'accès obligatoire, ou MAC [6], est une autre possibilité pour restreindre et isoler chaque processus. En utilisant un mécanisme tel que SELinux [6], RBAC [7] (intégré à grsecurity), ou encore AppArmor [1], il est possible de contrôler finement les actions de chaque processus, et d'appliquer une politique de sécurité dédiée. Les processus ainsi isolés ont une vision limitée du système, et ne peuvent pas appeler des méthodes ou du code qui n'ont pas été explicitement autorisés.

Chacun de ces mécanismes permet d'aboutir au même résultat fonctionnel. Les différences se font surtout sur les performances, la simplicité, et sur les ressources nécessaires : la virtualisation matérielle demande plus de ressources que

4. *Return Oriented Programming*
5. *LinuX Containers*
6. *Mandatory Access Control*
7. *Role Based Access Control*

la virtualisation légère et les MAC, mais permet d'isoler les noyaux du socle et des conteneurs. En contrepartie, la virtualisation légère et les MAC s'avèrent globalement plus performants que la virtualisation matérielle.

2.3 Impact sur le matériel

L'attaquant pouvant manipuler les données du réseau surveillé, il apparaît comme évident de disposer au minimum de deux réseaux physiques distincts : l'un dédié à l'administration et à la remontée d'alertes, l'autre dédié à l'acquisition du trafic du réseau surveillé. Un troisième réseau serait idéal pour pouvoir séparer l'administration et la remontée d'alertes, mais en pratique, ces deux réseaux peuvent coexister sur un même réseau physique si ceux-ci sont isolés de manière logique, par exemple, par des VLAN ou des connexions chiffrées différentes. Deux interfaces réseau au moins sont donc nécessaires.

Certaines plates-formes apportent des fonctions matérielles qui peuvent être exploitées pour améliorer la sécurité du système d'exploitation et des applications exécutées. C'est le cas par exemple du bit NX [8], appelé XN sur ARM, qui permet de s'assurer qu'une page mémoire ne peut pas être projetée avec à la fois les droits d'écriture et d'exécution.

D'autres éléments peuvent être ajoutés, tels qu'une source d'entropie matérielle, élément utile pour l'utilisation de la cryptographie, ou encore un TPM [9], qui permettra d'assurer l'intégrité du code exécuté sur la plate-forme depuis le démarrage. La fonction de génération d'aléa d'un TPM peut également être avantageusement utilisée pour fournir une source d'aléa supplémentaire au système d'exploitation.

3 Flux d'information entre conteneurs

Par défaut, un IDS réseau est un processus qui demande des privilèges élevés pour pouvoir lire tous les paquets depuis le réseau. Sur un système classique, ce processus s'exécute avec les mêmes privilèges que l'administrateur du système, ce qui n'est pas souhaitable.

Nous définissons des rôles distincts :
- l'*administrateur du système* met à jour les logiciels ;
- l'*administrateur de l'IDS* met à jour les règles de l'IDS ;
- l'*auditeur* accède aux alertes remontées par l'IDS.

Nous nous appuyons sur l'architecture présentée précédemment pour créer des conteneurs correspondants à chacun de ces rôles. Chaque conteneur n'a qu'une vue en lecture seule sur le système de fichiers, à l'exception des répertoires où il doit pouvoir écrire des données. Ces répertoires peuvent être utilisés pour transférer des fichiers entre un conteneur et le socle (ou un autre conteneur) : le conteneur source écrit le fichier, et le destinataire surveille le répertoire (périodiquement, ou par notifications) pour pouvoir traiter le fichier.

8. *No eXecute*
9. *Trusted Platform Module*

3.1 Détection

Nous décrivons ici le cas où le socle doit héberger soit plusieurs logiciels IDS tels que Suricata [11], Bro [8] ou encore Snort [13], soit des instances différentes du même IDS, avec des paramétrages distincts.

L'architecture proposée est décrite dans la figure 1. Chaque instance d'IDS est isolée dans un conteneur dédié. La sécurité de ces conteneurs repose sur les mécanismes suivants :

1. tous les systèmes de fichiers des conteneurs sont montés en lecture seule. Si le conteneur a besoin d'écrire des données temporaires, un point de montage en mémoire uniquement (*tmpfs*) est utilisé (contenu non exécutable, effacé à chaque démarrage du conteneur) ;
2. les conteneurs n'ont accès à aucun périphérique de l'hôte, en particulier les cartes réseau ou l'affichage ;
3. les protections systèmes décrites dans la section 2 doivent également être appliquées dans les conteneurs ;
4. les données de configuration et les règles des IDS sont exposées depuis le socle par un mécanisme de type montage par recouvrement (*bind-mount*), qui s'assure que le conteneur en charge d'exécuter un IDS ne peut en aucun cas écrire dans ces données ;
5. les protections apportées sont complémentaires des mesures qui peuvent être proposées dans un IDS, telles que la réduction de privilèges (ou de *capabilities*), l'utilisation d'un utilisateur non privilégié, ou encore la limitation d'utilisation de ressources système.

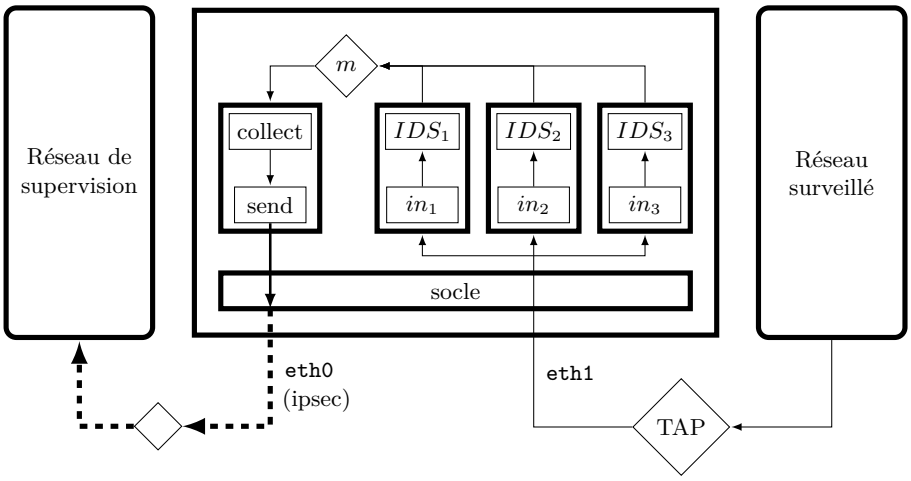

Figure 1. Architecture du système de détection.

Pour pouvoir être perçues depuis chaque conteneur qui héberge une instance de logiciel IDS, les données du réseau doivent être collectées par le socle, puis copiées dans chacun des conteneurs i, en direction de leurs mécanismes d'acquisition respectifs in_i. Plusieurs options sont possibles, en fonction du type de cloisonnement, et des capacités d'acquisition de chaque IDS_i : *socket unix* exposée *via* un autre *bind-mount*, connexion réseau *via* interface virtuelle privée, utilisation d'une file *NFLOG*, répertoire de données, *etc.*

Ce mécanisme doit respecter, au sens fonctionnel, les propriétés d'une diode : les communications doivent être unidirectionnelles depuis le socle vers le(s) conteneur(s), et ne doivent pas exposer d'élément du socle qui pourrait être compromis depuis ce(s) conteneur(s).

L'impossibilité d'envoyer des données sur le réseau depuis les conteneurs permet de limiter les éventuelles fuites d'information mais aussi la compromission d'autres équipements connectés au réseau surveillé depuis la sonde elle-même.

Après le traitement des données réseau, les alertes ou informations émises par chaque IDS_i doivent être collectées/agrégées avant de pouvoir être remontées vers un point de collecte sur le réseau de supervision. Pour cela, on propose également la mise en place d'une « diode » exposée par le socle dans chaque conteneur hébergeant un IDS, afin qu'il puisse y écrire ses journaux d'alertes et d'informations. Généralement, un fichier de journalisation pour chaque conteneur i suffit, ce qui présente l'avantage de pouvoir utiliser les attributs de fichiers (commande `chattr`) pour le marquer en mode ajout uniquement, ce qui garantit l'interdiction de modification ou de suppression.

Le socle doit également disposer d'un mécanisme de collecte m, dont le but est de collecter et d'agréger les informations remontées par tous les IDS. Ces données seront ensuite envoyées à un autre conteneur, dont le but est de gérer la remontée d'alertes. Cette transmission doit impliquer le moins de manipulations possibles pour s'assurer que le socle ne puisse pas être compromis ; idéalement, une simple recopie est donc préférable.

3.2 Remontée d'alertes

Une fois les informations collectées par le socle, elles doivent être mises en forme et envoyées vers le réseau de supervision de sécurité. La mise en forme implique la manipulation des données, c'est pourquoi elle est isolée dans un conteneur spécifique soumis aux mêmes mécanismes de durcissement que les conteneurs hébergeant les IDS.

L'exfiltration des événements depuis la sonde doit respecter des contraintes de confidentialité et d'intégrité. Il faut donc s'assurer qu'aucune donnée ne puisse être envoyée en clair, ou être modifiée pendant l'envoi sur le réseau de supervision. Pour cela, il est nécessaire de chiffrer puis signer les données à transmettre. La solution la plus efficace pour transmettre en temps réel les alertes est de mettre en place un réseau privé virtuel (VPN), par exemple de type *IPsec*, entre le socle et le réseau de supervision. Une communication doit également être mise en place entre le conteneur de collecte et le socle. Pour cela, un moyen simple est d'utiliser une interface privée virtuelle entre les deux. Le conteneur envoie

simplement ses données vers le réseau de supervision sans avoir à se soucier de leur protection ; la politique de routage du socle s'assure alors que toute donnée sortante est protégée.

La confidentialité et l'intégrité des données sont assurées par le VPN. La séparation en conteneurs permet de faire en sorte que seul le socle ait accès aux éléments secrets nécessaires à l'établissement du VPN, garantissant ainsi la protection des éléments secrets nécessaires l'établissement du VPN.

3.3 Mises à jour

La mise à jour des logiciels est un processus crucial pour le maintien en condition de sécurité. Cependant, comme le système (socle et conteneurs) n'est accessible qu'en lecture seule, il est impossible de modifier les fichiers. Nous proposons un mécanisme de démarrage spécifique qui permet de résoudre ce problème.

Le mécanisme de mise à jour sécurisée est le suivant : l'administrateur du système dépose les paquets de mise à jour dans le répertoire qui lui est accessible dans son conteneur dédié. Le socle copie ces fichiers dans une zone qui lui est réservée, soit par une tâche planifiée, soit par une action déclenchée. Il faut ensuite attendre le prochain redémarrage.

Le plus tôt possible au démarrage, avant que les différents conteneurs ne soient démarrés, et que les interfaces réseau ne soient configurées et opérationnelles, il existe une période pendant laquelle le système peut sans risque ne pas être verrouillé en écriture. Durant cette période, le système vérifie si il existe des paquets à mettre à jour dans la zone réservée. Si c'est le cas, il vérifie leur signature cryptographique par rapport à une chaîne de confiance. Si l'ensemble des paquets présents sont correctement authentifiés, alors les mises à jour sont appliquées. Lorsque les mises à jour ont été appliquées, le démarrage « normal » du système peut reprendre : les systèmes de fichiers sont verrouillés en écriture (de manière irréversible, *cf* section 2.1), les autres actions de durcissement sont mises en place, les interfaces réseau sont configurées et rendues opérationnelles, et enfin les différents conteneurs peuvent être instanciés.

4 Preuve de concept d'une sonde IDS légère

Généralement, les besoins de surveillance réseau peuvent être classés en deux grandes catégories :

- les réseaux à fort trafic, de type bureautique par exemple, pour lesquels les sondes déployées doivent disposer de ressources conséquentes, mais pour lesquelles le déploiement se fait sans trop de contraintes ;
- les réseaux répartis, qui nécessitent de multiples points d'analyse et disposent de fortes contraintes sur l'encombrement (impossible de déployer une baie de serveurs), la consommation électrique, et le nombre important d'éléments à déployer. C'est le cas des réseaux de systèmes industriels.

Bien que les principes abordés dans les sections précédentes sont valables dans ces deux cas de figure, nous avons choisi de réaliser une sonde IDS réseau légère dont le but est de surveiller des communications au sein de réseaux industriels. L'objectif est de réaliser un prototype fonctionnel, en implémentant l'architecture présentée, sur du matériel léger, peu encombrant, et peu coûteux.

4.1 Choix matériel et logiciel

Nous avons choisi un matériel de type embarqué, pour lequel plusieurs solutions à base de processeur ARM sont disponibles sous forme de set-top box. Ces solutions conviennent en terme d'encombrement, et peuvent être facilement déployées. De plus, elles ont l'avantage de pouvoir n'utiliser aucun élément mécanique (disque dur ou ventilateur), ce qui augmente considérablement leur durée de vie et diminue leur consommation.

De nombreuses plates-formes de ce type existent sur le marché, telles que RaspberryPi, GuruPlug, cartes FreeScale (SabreLite, Nitrogen, ...) ou encore MiraBox. Étant données les contraintes exposées dans la section 2.3, nous avons retenu la MiraBox qui dispose d'un CPU ARMv7 cadencé à 1,2 GHz, d'un Go de RAM et de deux interfaces réseau gigabit.

Pour le socle, une Debian *unstable* avec un noyau Linux 3.16.3 modifié pour intégrer grsecurity a été installée. Pour cloisonner les conteneurs, nous avons retenu la virtualisation légère LXC intégrée au noyau Linux. Chaque conteneur est basé sur une Debian *unstable* réduite à son strict minimum.

Un seul conteneur IDS est instancié, exécutant la version 2.0.3 de Suricata [11], modifié pour utiliser les règles et le préprocesseur Modbus spécifiques aux systèmes industriels [9].

Les flux de remontée d'alertes et d'administration de la sonde se font sur la même interface physique, en les isolant dans des VPN IPsec séparés.

La seconde interface est dédiée à l'acquisition. Cette interface n'est pas directement exposée dans le conteneur IDS. Un *TAP* logiciel est mis en place à l'aide d'un *bridge* et d'une paire d'interfaces virtuelles *veth* dont une extrémité est présentée à l'IDS. Des règles *ebtables* sont ajoutées dans le socle pour garantir l'unidirectionnalité des flux.

Le mécanisme de diode utilisé pour remonter les alertes est un tube nommé de type *FIFO*[10] entre le conteneur IDS et le conteneur de collecte de journaux.

4.2 Performances

Les tests ont pour but de mesurer l'impact des mesures de durcissement sur les performances. À cet effet, nous nous sommes basés sur les travaux de [9], pour disposer d'une sonde témoin (non durcie) et d'une capture de 200 000 paquets de trafic réel d'un système industriel. Cependant, les règles de l'IDS déclenchent une alerte pour chaque transaction Modbus. En pratique, sur la capture utilisée, cela équivaut à une alerte tous les deux paquets : l'objectif n'est pas de représenter

10. First In First Out

un volume d'alertes réaliste, mais de se placer dans le cas le plus défavorable pour évaluer les performances.

Les figures 2 et 3 présentent les résultats obtenus en rejouant la capture à différents débits avec la sonde témoin et la sonde durcie, respectivement.

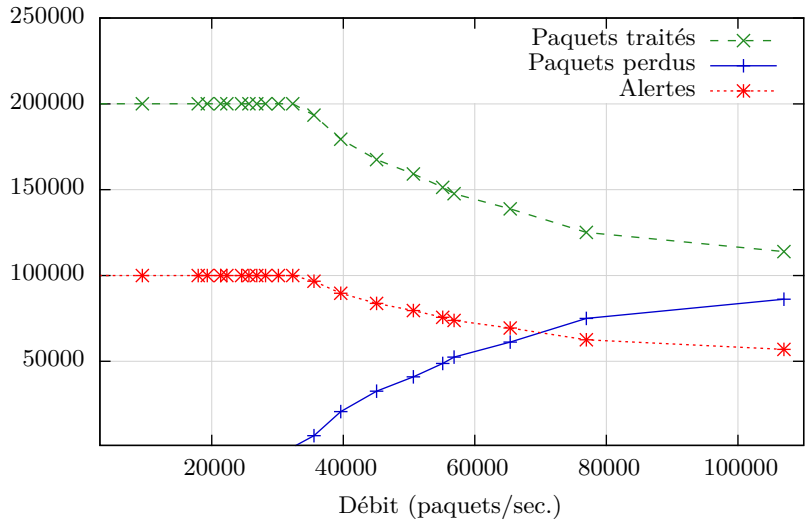

Figure 2. Résultats sur la sonde IDS témoin.

En observant le nombre de paquets perdus, on constate que l'impact global des mesures de durcissement sur les performances est d'environ 33%. En effet, tous les paquets sont correctement traités jusqu'à un débit de 32 500 paquets par seconde pour la sonde témoin (générant 16 500 alertes par seconde), alors que la sonde durcie peut traiter jusqu'à environ 21 500 paquets par seconde (générant 11 250 alertes par seconde).

Des mesures supplémentaires (non présentées ici) ont permis de déterminer que la mesure de durcissement la plus coûteuse en performance est l'utilisation du *TAP* logiciel (environ 60% du coût total). Viennent ensuite la diode de collecte d'alertes (environ 15%), les modifications apportées par PaX dans le patch grsecurity (environ 13%) et enfin la virtualisation légère LXC (environ 12%).

4.3 Discussion

Durant les tests de performances, la principale limitation constatée est la saturation des ressources CPU, alors qu'une marge était encore disponible pour les ressources mémoire, disque et réseau. En dehors de quelques réglages, aucune optimisation particulière de l'IDS n'a été réalisée.

Le *TAP* logiciel a un impact important sur les performances. Il est cependant facile et peu coûteux de remplacer ce *TAP* logiciel par un *TAP* matériel qui offre

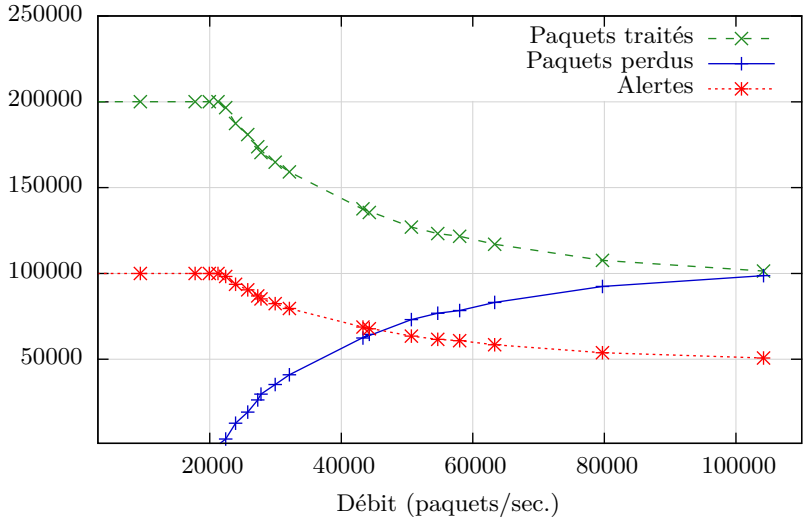

Figure 3. Résultats sur la sonde IDS durcie.

tout autant voire plus de garantie en terme de sécurité que le *TAP* logiciel, mais dont l'impact sur les performances est nul.

Le matériel choisi (MiraBox) ne dispose que d'un seul CPU, qui de plus est mono-cœur. Cette limitation est particulièrement importante ici, car l'IDS choisi (Suricata) et la virtualisation légère (LXC) sont des composants qui bénéficieraient particulièrement d'une augmentation des ressources CPU.

Dans l'ensemble, la preuve de concept a permis de valider qu'un matériel non spécialisé, peu coûteux (moins de 150€), supporte la mise en place d'une architecture sécurisée pour une sonde IDS. Si le nombre d'alertes générées dans notre expérience est artificiellement élevé, il permet toutefois de valider le fait que cette preuve de concept de sonde IDS durcie peut effectivement traiter un débit équivalent à plusieurs dizaines d'équipements [9] sans perdre aucune alerte.

5 Conclusion

Les sondes réseau, et plus généralement les logiciels IDS, sont les briques essentielles des systèmes de supervision de sécurité. La qualité de ces composants se résume traditionnellement à la sensibilité et à la précision de leur processus de détection de comportements inhabituels ou malveillants. Cependant, de part leur fonction et leur exposition, la sécurité intrinsèque de ces équipements s'avère cruciale. En effet, la compromission d'un IDS par un attaquant lui donnerait non seulement une position stratégique pour observer tout le trafic du réseau surveillé, mais lui permettrait également de compromettre d'autres équipements de ce réseau sans être détecté.

L'architecture logicielle de sonde IDS que nous proposons intègre les principes de défense en profondeur à l'état de l'art en matière de sécurité logicielle : durcissement du socle logiciel pour prévenir des exploitations « triviales », principe de moindre privilèges pour les utilisateurs et processus, différenciation des rôles d'administrateur système, d'administrateur IDS et d'auditeur, cloisonnement par virtualisation légère des différents composants logiciels (IDS, collecte, journalisation, mises à jour), flux unidirectionnels de collecte d'alertes, remontée des alertes sur un réseau de supervision dédié protégé en confidentialité et en intégrité.

Il faut cependant rappeler que l'architecture proposée ne modifie pas la qualité de la détection, ni les causes premières des vulnérabilités, qui restent liées à chacun des logiciels IDS utilisés. Sur ce point, le durcissement des logiciels IDS, voire l'écriture de parties critiques (en particulier les décodeurs protocolaires) dans des langages sécurisés restent des actions complémentaires indispensables.

Afin de montrer que l'architecture durcie que nous présentons dans ce papier est réalisable et viable, nous avons choisi de monter une preuve de concept de sonde IDS légère, particulièrement adaptée aux réseaux industriels. Nous avons conduit différents tests sur cette sonde durcie en utilisant une capture de trafic réel d'un système industriel dans le but de mesurer l'impact des mesures de durcissement. Les résultats obtenus montre un impact significatif des mesures de durcissement sur les performances. Néanmoins, nous nous sommes délibérement placés dans des conditions défavorables tant sur le nombre d'alertes générées que sur les choix d'architectures matérielle et logicielle. Nous pouvons donc raisonnablement statuer sur le fait que cette mesure constitue une borne maximale du coût engendré par les mesures de durcissement que nous préconisons dans notre architecture.

Bien qu'appliquée à la conception d'une sonde IDS légère, l'architecture que nous proposons reste entièrement valable pour la conception de sonde disposant de plus de ressources et capables de traiter des volumes de données dépassant le gigabit. De plus, sur du matériel de type serveur, la majorité des mécanismes de durcissement que nous proposons d'utiliser tirent partie de propriétés du matériel sous-jacent. L'impact sur les performances engendré par ces protections s'en trouvera alors fortement réduit puisque qu'ils ne sont plus implantés de manière logicielle, comme dans notre preuve de concept.

Références

1. AppArmor, Application Armor Linux security module. `http://wiki.apparmor.net/`.
2. Exemples de vulnérabilités critiques dans le logiciel Snort. `http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=snort`. Consulté en Juin 2014.
3. Exemples de vulnérabilités dans les décodeurs du logiciel WireShark. `http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=wireshark`. Consulté en Juin 2014.
4. PaX/grsecurity features. `http://grsecurity.net/features.php`.

5. Recommandations de sécurité relatives à un système GNU/Linux. http://www.ssi.gouv.fr/fr/guides-et-bonnes-pratiques/recommandations-et-guides/securite-du-poste-de-travail-et-des-serveurs/recommandations-de-securite-relatives-a-un-systeme-gnu-linux.html.
6. SELinux, Security-Enhanced Linux. http://selinuxproject.org/.
7. B. Spengler *et al.* PaX/grsecurity. http://grsecurity.net/.
8. Bro community. The Bro Network Security Monitor. http://www.bro.org/.
9. D. Diallo and M. Feuillet. Détection d'intrusion dans les systèmes industriels : Suricata et le cas de Modbus. In *C&ESAR*, 2014.
10. E. Jaeger and O. Levillain. Mind You Language(s). In *1st LangSec workshop of IEEE Security & Privacy*, May 2014.
11. Open Information Security Foundation (OISF). Suricata, Open Source IDS / IPS / NSM engine. http://suricata-ids.org/.
12. H. Shacham, E. Buchanan, R. Roemer, and S. Savage. Return-Oriented Programming : Exploits Without Code Injection. In *Black Hat USA*, 2008.
13. Sourcefire. Snort, Open Source network intrusion prevention and detection system. http://www.snort.org/.