Ecole d'Ingénieurs du Canton de Vaud

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

VoIP & Security: IPS

Lalaina KUHN

Informatique Technique

Professeur: Stefano VENTURA

"Misaora an'i Jehovah, ry Fanahiko, ary aza misy hadinoinao ny fitahiany rehetra" (Sal. 103.2)

Cahier des charges

Ce travail de diplôme comprend les activités suivantes:

I. Tutorial présentant les problèmes liés à la traversée du NAT (*Network Address Translation*) et des firewalls lors du déploiement de services VoIP au sein de l'entreprise.(voir aussi travail de semestre)

II. Réalisation d'un firewall/ALG (*Application Level Gateway*) basé sur le produit *Open Source* SIPROXD (proxy SIP) permettant ainsi de réaliser un banc de test et laboratoire pouvant démontrer que le déploiement de la VoIP n'entame pas les défenses des réseaux d'entreprise. Cette étape fournira les spécifications des dispositifs firewall/ALG. (voir aussi travail de semestre)

III. Etude de la problématique de l'IPS (*Intrusion Prevention System*) et présentation de l'état de l'art. Cette étape fournira les spécifications des dispositifs IPS. Ce document doit indiquer dans quel type de déploiement de la VoIP en entreprise, une solution IPS est adaptée. Cette étude contiendra aussi:

   i. une description d'une fonction IPS SIP

   ii. une étude de produits IPS se trouvant sur le marché (s'ils existent)

IV. Réalisation d'une solution IPS *Open Source*.

   i. Cette phase prévoit la réalisation d'un IPS VoIP/SIP basé sur un IDS existant (SNORT, Prelude etc)

Table des matières

1 Résumé

Un des derniers grands avantages de la téléphonie traditionnelle par rapport à sa rivale la téléphonie sur Internet reste son immunité à tous les virus et attaques qui hantent le monde informatique. Cet avantage n'est pas des moindres et encore maintenant malgré le grand engouement pour la VoIP, beaucoup d'entreprises restent, pour des raisons de sécurité, attachées aux PBX traditionnels. Le téléphone restant le nerf de la guerre des entreprises, il est clair que les solutions de téléphonie sur IP doivent apporter des garanties de sécurité acceptables contre toutes les sortes d'attaques efficaces et évolutives du monde IP.

L'introduction de la VoIP sur IP implique la solution de trois grands problèmes suivants:

- La détection et prévention des intrusions,
- La résolution des problèmes du NAT,
- L'adaptation des firewalls déjà déployés en entreprise avec les exigences des protocoles VoIP.

Ce travail de diplôme propose d'étudier les points cités ci-dessus à savoir et en particulier d'approfondir les principales vulnérabilités de la VoIP par rapport aux attaques externes et de proposer des solutions de détections de manière à prévenir ces attaques.

2 Introduction

Nombreuses entreprises ont déjà adopté la voix sur IP. C'est une méthode de transmission de la voix utilisant le réseau IP. Cette nouvelle technologie devenue de plus en plus répandue et utilisée dans le monde est comme toute nouvelle technologie, les développeurs se sont préoccupé davantage des fonctionnalités et interopérabilités plutôt que de l'aspect de la sécurité. Les attaques contre de tels systèmes couvrent plusieurs catégories allant de déni du service jusqu'à la fraude de facturation.

Sécuriser les systèmes d'information et les réseaux est devenu une préoccupation majeure des entreprises. Beaucoup de méthodes ont été développées pour sécuriser les infrastructures réseaux et la communication sur Internet, parmi lesquelles l'utilisation des firewalls, le cryptage et les réseaux privés virtuels (VPN). La détection d'intrusion est une addition à ces techniques. On peut résumer en disant qu'un système de sécurité se compose d'outils multiples, incluant:

- Un firewall qui est utilisé pour bloquer le trafic entrant et sortant non désiré,

- Un système de détection d'intrusion (IDS) qui est utilisé pour savoir si un intrus est entré ou essaie d'entrer dans le réseau.

- Des outils d'évaluation de vulnérabilité, utilisés pour repérer d'éventuelles failles de sécurité existant sur le réseau.

Le système de prévention d'intrusion (IPS) est un outil de sécurité relativement nouveau pour compléter cette liste.

Ce document propose une première approche de système de prévention d'intrusion destiné à la VoIP. Cet IPS est organisé pour détecter les attaques basées sur les vulnérabilités du protocole SIP. Quelques scénarios d'attaque seront simulés et l'efficacité du système est évaluée avec des règles censées empêcher ces attaques.

3 Qu'est-ce qu'un système de détection d'intrusion?

On appelle IDS (*Intrusion Detection System*) un mécanisme écoutant le trafic réseau de manière furtive afin de repérer des activités anormales ou suspectes et permettant ainsi d'avoir une action de prévention sur les risques d'intrusion.

Les quatre modules importants d'un IDS sont: la journalisation, l'analyse, l'action et la gestion.

Journalisation: Enregistrement des évènements dans un fichier, comme l'arrivée d'un paquet ou une tentative de connexion.

Analyse: Analyse des journaux afin d'identifier des motifs (*patterns*) parmi la quantité importante de données que l'IDS a recueillie. Il existe deux principales méthodes d'analyse: sur la base de signatures d'attaques et par la détection d'anomalie.

Action: Prévenir le personnel lorsqu'une situation dangereuse est détectée par l'activation d'une alarme.

Gestion: Les IDS doivent être configurés, mis à jour et gérés activement de différentes manières.

On peut comparer un IDS à une caméra de sécurité dans un bâtiment. Il détecte des intrusions en temps réel et enregistre des bandes qui peuvent être étudiées après un incident. Mais il est insuffisant pour garantir à lui seul la sécurité du réseau. C'est un élément, parmi d'autres, de l'architecture de sécurité.

3.1 Les types d'IDS

Il existe deux grandes familles distinctes d'IDS:

Les **NIDS** (*Network Based Intrusion Detection System*), ils assurent la sécurité au niveau du réseau.

Les **HIDS** (*Host Based Intrusion Detection System*), ils assurent la sécurité au niveau des hôtes.

3.1.1 Network IDS

Un NIDS nécessite un matériel dédié et constitue un système capable de contrôler les paquets circulant sur un ou plusieurs lien(s) réseau dans le but de découvrir si un acte malveillant ou anormal a lieu. Le NIDS place une ou plusieurs cartes d'interface réseau du système dédié en mode espion[1] (*promiscuous mode*), ceci garantit un fonctionnement en mode furtif qui lui permet de lire et analyser tous les paquets qui passent par ce lien.

La force des NIDS est qu'ils voient tous les paquets passant par un point donné du réseau. Ces paquets sont souvent le meilleur diagnostic des attaques. Cependant, les NIDS possèdent certaines vulnérabilités inquiétantes.

Angles morts: Suivant où sont situés les NIDS dans le réseau, certaines situations induisent des angles morts où ils ne peuvent pas «voir» les paquets. Par exemple, si seuls les NIDS situés entre le firewall et l'Internet sont utilisés, le réseau interne entier est un gigantesque angle mort.

Données chiffrées: De même que les firewalls, les NIDS ne peuvent pas lire les données chiffrées. Ils peuvent scanner les parties non chiffrées (en général, entête IP ajouté) d'un paquet chiffré, mais cela ne fournit que des informations limitées.

3.1.2 Host IDS

Le HIDS réside sur un hôte particulier et est installé comme un agent. Il se comporte comme un démon (*daemon*) ou un service standard sur un système hôte. Traditionnellement, le HIDS analyse des informations particulières dans les fichiers de log pour détecter n'importe quelle activité d'intrus. Ce système de détection capture aussi les paquets réseaux entrant/sortant de l'hôte pour y déceler des signes d'intrusions et alerte quand quelque chose s'est produit.

Les HIDS sont intéressants car même si l'ordinateur est situé dans un angle mort du réseau, ils peuvent recueillir les données le concernant.

[1] Une carte réseau peut être mise en **mode espion** pour recevoir tout le trafic qui est envoyé sur le segment de réseau sur laquelle elle se trouve. Cela permet de "renifler" (*sniff*) ce qui se passe sur le réseau. Si la carte réseau n'est pas en mode espion, elle rejette tout le trafic qui ne lui est pas destiné.

De plus, seules les données relatives à l'ordinateur sur lequel le HIDS est installé sont recueillies, ce qui simplifie l'analyse.

Les HIDS ont deux faiblesses principales. En premier lieu, ils ont une vision limitée de ce qui se passe sur le réseau. Cette vision limitée qui leur permet d'être spécifiques, les empêche d'avoir une vision d'ensemble. En second lieu, les HIDS peuvent être victimes d'attaques. S'ils sont situés sur l'ordinateur compromis par l'attaquant, leurs fichiers peuvent être effacés ou modifiés.

3.2 Les techniques de détection

Les systèmes de détection d'intrusion se divisent en deux principales catégories : les systèmes de détection basés sur la signature et les systèmes de détection d'anomalie.

En effet, les intrus ont des signatures, comme les virus, qui peuvent être détectées par des logiciels adéquats. Il faut essayer de trouver tous les paquets de données qui contiennent les signatures ou anomalies connues liées aux intrusions ou au protocole Internet. Basé sur un ensemble de signatures et de règles, le système de détection peut trouver et loguer l'activité suspecte et produire des alertes. Les IDS nécessitent des mises à jours de signatures pour détecter les nouveaux types d'attaques.

La détection d'intrusion basée sur la détection d'anomalie consiste à comparer les schémas d'événements en cours pris dans leur ensemble aux schémas d'événements habituels pris dans leur ensemble. Il est par exemple utiles d' analyser de près des situations comme multiples échecs de connexion, utilisateur accédant à des fichiers systèmes critiques, modification des fichiers exécutables. Cette méthode est puissante car elle permet d'identifier des attaques inhabituelles. Cependant, il est difficile de distinguer ce qui est normal de ce qui ne l'est pas, car les schémas d'activités varient largement d'un système réel à un autre.

Habituellement un système de détection d'intrusion capture des données du réseau et applique ses règles à ces données ou détecte des anomalies.

3.3 Les différentes actions des IDS

Les IDS peuvent déclencher des alarmes automatiques et proposer un dispositif d'analyse interactive afin d'aider l'administrateur de la sécurité à identifier des motifs dans les journaux.

Les principales méthodes utilisées pour signaler les intrusions sont les suivantes:

- *Envoi d'un e-mail à un ou plusieurs utilisateurs*: pour notifier d'une intrusion sérieuse.

- *Journalisation (log) de l'attaque* : sauvegarde des détails de l'alerte dans une base de données centrale, comme par exemple des informations telles que adresse IP de l'intrus, adresse IP de la cible, protocole utilisé.

- *Sauvegarde des paquets suspects*: sauvegarde de l'ensemble des paquets réseaux capturés et/ou uniquement les paquets qui ont déclenché une alerte.

- *Démarrage d'une application*: lancement d'un programme extérieur pour exécuter une action spécifique (envoi d'un message sms, émission d'une alerte auditive…).

- *Notification visuelle de l'alerte*: affichage de l'alerte dans une ou plusieurs console(s) de management.

De nombreux IDS peuvent effectuer des actions limitées dans le cas de certains schémas d'événements, sans intervention humaine. Le plus souvent, les NIDS utilisent cette capacité pour déjouer des attaques de déni de service. Si l'attaque provient d'une adresse IP unique, par exemple, il est possible de bloquer cette adresse automatiquement. De leur côté, les HIDS peuvent empêcher les utilisateurs d'accomplir certaines actions.

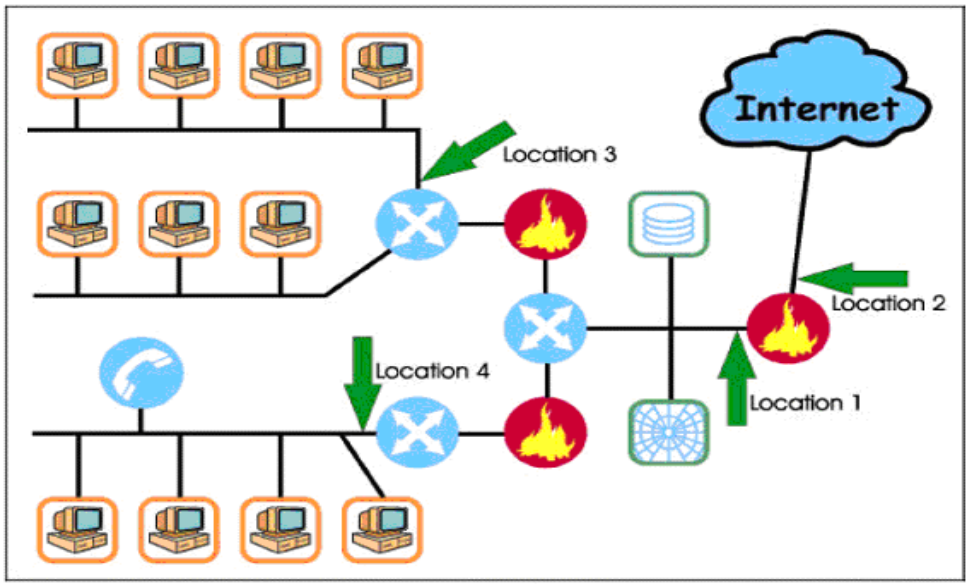
Cependant, les actions automatisées doivent être extrêmement limitées car la plupart des schémas indiquent uniquement des activités suspectes, et non des actions assurément malveillantes.

3.4 Où placer l'IDS dans une topologie réseau?

Selon la topologie réseau, on peut placer des systèmes de détection d'intrusion à un ou plusieurs endroits. Ceci dépendent également de quels types d'activités d'intrusion on souhaite détecter : interne, externe ou les deux. Par exemple, si l'entreprise veut uniquement détecter les activités d'intrusion externes, et qu'il a seulement un routeur relié à Internet, le meilleur endroit pour un système de détection d'intrusion peut être juste après le routeur ou le firewall. S'il a plusieurs connections à Internet, on peut placer un IDS à chaque point d'entrée.

Cependant si on veut également détecter les menaces internes, on peut placer un IDS dans chaque segment de réseau.

Dans de nombreux cas, il n'est pas nécessaire d'avoir un IDS dans tous les segments de réseau et on peut se limiter seulement aux secteurs sensibles du réseau. Notez que plus il y a de systèmes de détection d'intrusion, plus il y a de travail et plus il y a de coûts d'entretien. La figure suivante montre les endroits typiques où on peut placer un système de détection d'intrusion.



Source: NIST, special publication on intrusion detection system

Figure 3-1-Endroits typiques pour un IDS réseau (NIDS)

- Location 1: dans la zone démilitarisée (DMZ). Voir les attaques qui visent surtout les serveurs de services de l'entreprise.
- Location 2: entre le firewall externe et l'Internet. Permet d'analyser au mieux les attaques lancées contre l'entreprise.
- Location 3: à chaque segment du réseau. L'IDS situé à cet endroit se concentre plus sur les charges intrusives plus dangereuse, qui passent à travers le firewall.
- Location 4: plus près des sous-réseau sensibles, permet de détecter les attaques visant des ressources ou des systèmes critiques. Par exemple des serveurs de comptes sensibles.

Une nouvelle famille d'outils de sécurité est apparue sur le réseau: les systèmes de prévention des intrusions (*IPS Intrusion Prevention System*).

4 Qu'est-ce qu'un système de prévention d'intrusion?

Un système de prévention d'intrusion est un dispositif (matériel ou logiciel) capable de détecter des attaques, connues et inconnues, et de les empêcher d'être réussies.

Un IPS n'est pas un observateur: il fait partie intégrante du réseau. Il est placé en ligne et examine tous les paquets entrants ou sortants. L'IPS a été développé de manière à prendre les mesures nécessaires pour contrer les intrusions détectables avec précision. Il surveille le trafic et intervient activement par limitation ou suppression du trafic jugé hostile, par interruption des sessions suspectes, ou par d'autres mesures en réaction à une attaque. Il réalise un ensemble d'analyses de détection, non seulement sur chaque paquet individuel, mais également sur les motifs du réseau, en visualisant chaque transaction dans le contexte de celles qui précédent ou qui suivent.

En général, les quatre modules principaux qui composent un IPS sont:

- Normalisation du trafic (*Traffic normalizer*),
- Scanner de service (*Service scanner*),
- Moteur de détection (*Detection engine*),
- Traffic shaper.

Le *traffic normalizer* surveille le trafic, analyse les paquets et exécute les mesures de base contre les attaques, comme le blocage d'adresse IP. Le trafic analysé est ensuite passé au moteur de détection et au scanner de service. Ce dernier construit une table de référence classifiant les informations afin d'aider le *traffic shaper* à mieux gérer le flux de ces informations. Le moteur de détection compare les schémas d'événements en cours avec ceux contenus dans la table de référence.

4.1 Les Types d'IPS

Il existe actuellement deux catégories de produit IPS:

Les **NIPS** (*Network Intrusion Prevention System*), un logiciel ou un matériel dédié connecté directement à un segment du réseau et qui protège tous les systèmes rattachés à ce segment.

Les **HIPS** (*Host Intrusion Prevention System*), un programme système installé directement sur l'ordinateur à protéger.

4.1.1 Network IPS

Le Network IPS (NIPS) combine les caractéristiques d'un IDS standard et d'un firewall. Il est parfois qualifié comme étant la prochaine génération de firewall, le firewall à «inspection en profondeur (*deep inspection*)».

Comme avec un firewall, le NIPS a au moins deux interfaces réseau, une interne et une externe. Les paquets arrivent à l'une ou l'autre des deux interfaces, puis sont passés au moteur de détection. Jusqu'ici, l'IPS fonctionne comme un IDS, c'est-à-dire qu'il déterminer si oui ou non le paquet est malveillant. Cependant, s'il en détecte, en plus de déclencher une alerte, il rejettera le paquet et marquera cette session «suspecte».

Quand les paquets restants qui composent cette session particulière arriveront à l'IPS, ils seront jetés immédiatement. Les paquets propres sont passés à travers la deuxième interface pour atteindre leur destination prévue.

Les NIPS sont déployés **en ligne** avec le segment du réseau à protéger. De ce fait, toutes les données qui circulent entre le segment protégé et le reste du réseau doivent passer par le NIPS. Dès qu'un paquet suspect est détecté - et avant qu'il soit passé à l'interface interne et au réseau protégé - il peut être rejeté. En plus de cela, au moment où la session a été marquée comme «suspecte», tous les paquets suivants qui font partie de cette session peuvent être également rejeté avec un éventuel traitement additionnel. Quelques produits envoient, par exemple, un *Reset TCP* ou un message *ICMP Unreachable* à la machine attaquante.

Comme le trafic traverse l'IPS, il est inspecté pour vérifier la présence d'une attaque. Le NIPS peut employer l'approche de détection basée sur les signatures d'attaque. Une différence principale cependant est qu'il emploie des signatures qui sont basées sur des vulnérabilités, et non pas sur les exploits[2]. En utilisant des signatures sur des vulnérabilités, le NIPS peut ajouter une protection (promettante) avant que des exploits réels ne soient libérés dans la nature.

[2] Un exploit tire profit d'une faiblesse dans un système afin de l'entailler. Les exploits sont la racine de la culture d'intrus.

Un NIPS déclenche, tout comme un NIDS, des alarmes face aux attaques, mais le contenu de l'alarme est différent. C'est plutôt une notification indiquant «*tel ou tel trafic a été détecté en train d'essayer d'attaquer ce système et a été bloqué*»

Est-ce qu'avec des alarmes de ce type, une intervention humaine est nécessaire?

Dans un environnement où la sécurité est primordiale, la réponse est plutôt oui. Une intervention humaine est nécessaire pour vérifier l'intervention automatisée, et aussi pour recueillir les indices de sorte qu'ils puissent être employés pour se protéger contre de futures attaques, ou pour identifier une plus grande menace.

Dans la plupart des environnements où la sécurité est un souci, et non pas une priorité, aucune intervention humaine n'est vraiment nécessaire.

Les **avantages** d'un network IPS:

- Un seul point de contrôle pour le trafic peut protéger des milliers de systèmes situés "en aval" du dispositif. Ceci permet à une organisation de mesurer sa solution rapidement et de fournir la flexibilité requise pour répondre aux changements constants de l'architecture réseau.

- Le déploiement devient facile car un seul dispositif IPS peut protéger des centaines de systèmes.

- Il fournit une vue plus large de l'environnement de menaces, telles que les balayages, sondes… Avoir une vision stratégique de l'environnement de menaces permet à la gestion de sécurité de s'adapter à un changement de perspective.

- Il protège les autres dispositifs du réseau: toutes les attaques ne sont pas dirigées que contre les systèmes dotés d'un système d'exploitation. Par exemple les routeurs, firewalls, concentrateurs VPN, serveurs d'imprimantes etc., sont tous vulnérables et exigent une protection.

- Il protège des attaques réseaux: DoS, DDos, SYN flood etc. Travailler au niveau du réseau permet à un NIPS de protéger contre ces types d'attaques.

4.1.2 Host IPS

Le HIPS est un programme qui réside sur un système tel que serveurs, postes de travail ou portables.

Le trafic venant ou sortant de ce système particulier est inspecté et les activités au niveau des applications et du système d'exploitation peuvent être examinées afin de trouver des signes d'une attaque. Un HIPS peut détecter les attaques qui visent la machine, et arrêter le processus malveillant avant qu'il ne s'exécute. On peut donc considérer le HIPS comme étant plus un moniteur de système d'exploitation ou moniteur d'application.

Le HIPS n'exige plus qu'un service génère une notation d'événement (*event log*) ou des changements dans des fichiers systèmes avant d'agir. C'est la nouveauté de cet outil. Quand une attaque est détectée, le logiciel bloque l'attaque soit au niveau de l'interface réseau, soit en envoyant des commandes à l'application ou au système d'exploitation d'arrêter l'action lancée par l'attaque.

La méthode de détection varie selon le fabricant, mais la plus utilisée est l'approche basée sur des règles.

Un HIPS possède une liste prédéfinie de règle, établie par le fabriquant et livrée avec le produit. Ces règles savent comment un système d'exploitation ou une application devrait se «comporter». Si l'application initie une action suspecte, une des règles est déclenchée et le processus est tué avant même qu'il ait pu nuire au système.

Par exemple, les attaques par *buffer overflow* peuvent être empêchées en interdisant l'exécution du programme malveillant inséré dans l'espace adresse exploité par l'attaque. Ou encore, les tentatives d'installation de *back door* via une application comme Internet Explorer sont bloqués en arrêtant et en refusant la commande *write file*, commande provenant de l'IE.

D'autres systèmes HIPS emploient l'approche "surveillance" ou "observation". Un agent HIPS tourne sur la machine et se concentre sur les événements d'un système d'exploitation en observant tous les appels système, les entrées de la base de registre et les services des communications.

Enfin, l'approche hybride est aussi utilisée. Un HIPS hybride peut employer une combinaison de: règles, contrôle de l'activité des applications, et les signatures pour détecter une attaque. L'avantage principal de ce type de systèmes est la capacité d'identifier absolument les attaques connues.

Les systèmes basés sur les règles ou ceux qui adoptent l'approche «surveillance» peuvent ne découvrir que les actions suspectes, actions qui sont arrêtées bien-sûr.

Les **avantages** d'un Host IPS:

Un logiciel installé directement sur le système le protège contre une attaque, mais en plus, le protège des résultats d'une attaque en empêchant un programme d'écrire dans un fichier, ou un utilisateur d'augmenter son privilège, etc…

- *Protège les systèmes mobiles* contre l'attaque quand ils sont hors du réseau protégé. Les ordinateurs portables sont le vecteur principal de vers dans un réseau protégé. Installer un NIPS avec un système mobile n'est pas une solution pratique.

- *Protège des attaques locales*: le personnel avec un accès physique à un système peut lancer des attaques locales en exécutant des programmes introduits par une disquette ou un CD etc... Ces attaques se concentrent souvent sur l'augmentation des privilèges de l'utilisateur en *root* ou *administrator* afin de compromettre facilement d'autres systèmes dans le réseau.

- *Garantir une «dernière ligne de défense»* contre les attaques qui ont pu se soustraire à d'autres outils de sécurité. Un potentiel système victime lui-même est un dernier point de défense disponible pour l'administrateur de la sécurité afin de se protéger des violations du système.

- *Empêche l'attaque interne ou l'abus* des dispositifs situés sur le même segment du réseau: le NIPS assure uniquement une protection des données qui se déplacent entre différents segments. Les attaques lancées entre les systèmes situés sur le même segment ne peuvent être contrées qu'avec le HIPS.

- *Protège des attaques chiffrées* quand le flux de données chiffrées s'arrête au système protégé. Le HIPS examine ces données et/ou le comportement après que ces données chiffrées ont été déchiffrées sur le système.

- *Indépendant* de l'architecture réseau.

Comme le HIPS intercepte toutes les requêtes destinées au système qu'il protège, il doit respecter une certaine condition préalable il doit être très fiable, ne doit pas avoir un impact négatif sur la performance du système, et ne doit pas bloquer le trafic reconnu comme légitime.

4.2 Les techniques de détection

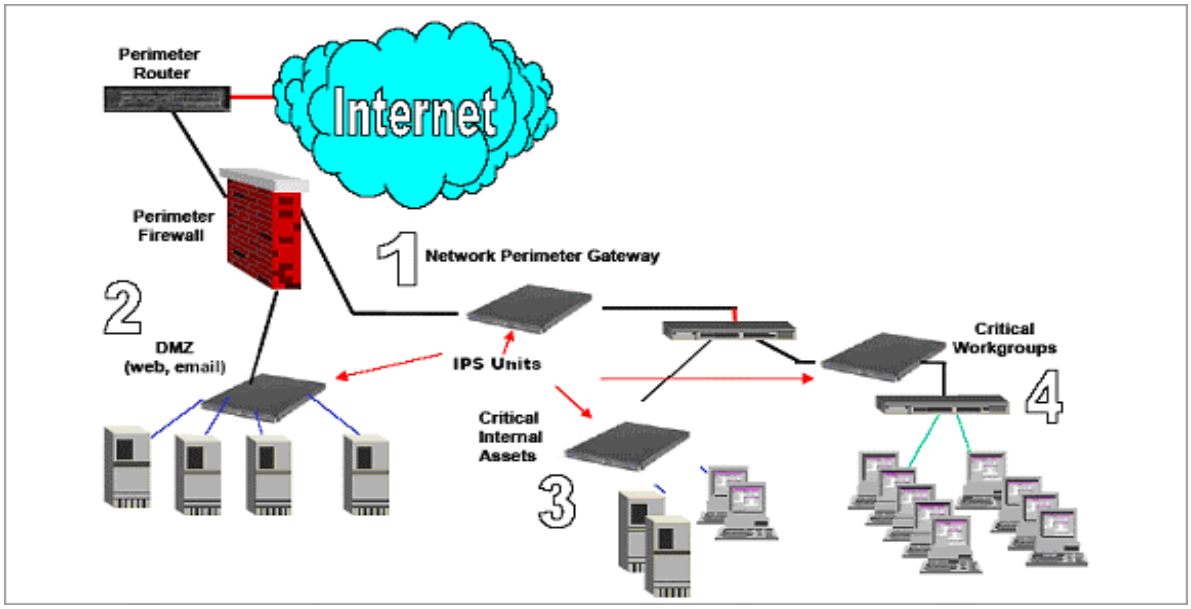
La première étape pour arrêter les attaques et éviter les intrusions consiste à les détecter. Plusieurs méthodes de détection sont applicables:

1) **L'analyse de protocole**: c'est une détection d'anomalie de protocole. L'IPS contrôle la conformité à la norme du protocole donné. Avec cette méthode, la recherche d'une activité d'intrusion est plus ciblée et donc plus efficace.

2) **Correspondance de motifs** (*pattern matching*): le trafic est scruté pour déterminer les signatures des motifs d'attaque connus. Le système analyse chaque paquet. Cette méthode peut être *stateful*, c'est-à-dire que le système examine le contexte et l'emplacement de la signature. L'IPS conserve la trace de l'état de connexion avec l'entité extérieure et évaluera le contexte plus large de toutes les transactions établies au cours de la connexion. Cette méthode ne permet de détecter que les attaques connues, et seulement si la liste des signatures d'attaques est régulièrement actualisée.

3) **Comportementale**: détection basée sur le comportement (*behaviour based detection*). L'IPS tente de découvrir des activités «anormales» en observant le comportement du système et des utilisateurs. Après une phase d'apprentissage, l'IPS dispose d'un modèle du «comportement normal» du système et de ses utilisateurs. SI l'IPS constate que l'activité courante s'éloigne trop du comportement normal, il prenne une mesure. Cette méthode a l'avantage de pouvoir détecter des tentatives d'exploitation de vulnérabilités jusque là inconnues.

4.3 Où placer l'IPS dans une topologie réseau?



Source: NITRO Data System, Intrusion Prevention White Paper

Figure 4-1 – Endroits typiques pour un IPS réseau (NIPS)

1. Derrière le firewall périmètre de réseau et la fin de tout réseau VPN,

2. Entre le firewall périmètre de réseau et les serveurs placés dans la DMZ tels que le serveur web et les serveurs d'e-mail,…

3. Devant les serveurs internes critiques tels que les serveurs d'application,

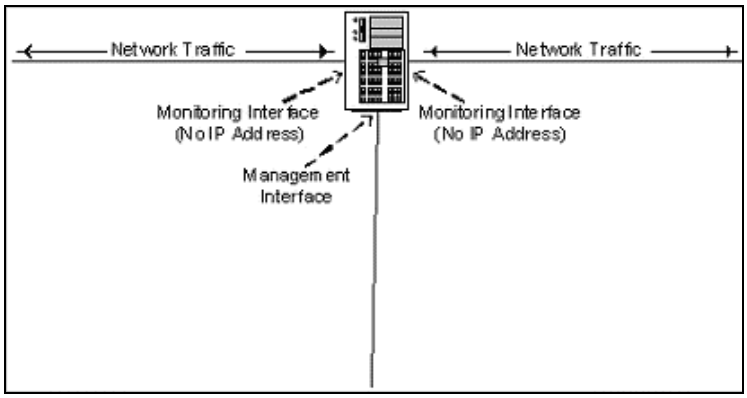
4. Devant les principaux *workgroups* d'un département.

4.4 Les différentes façons de réaliser un IPS

Il existe 5 différentes façons de réaliser un système de prévention d'intrusion, à savoir: le NIDS en ligne (*inline NIDS*), les firewalls/IDS applicatifs (*application-based firewalls/IDS*), les commutateurs de la couche 7 (*layer 7 switches*), les IDS réseaux applicatifs (*network-based application IDSs)* et les applications trompeuses (*deceptive applications*).

4.4.1 Le NIDS en ligne (*Inline NIDS*)

Le *NIDS inline* fonctionne comme un pont de niveau 2. Il se trouve entre le système à protéger et le reste du réseau comme le montre la figure:



Source: N.Deseau, IPS: the next step in the evolution of IDS

Figure 4-2 - Inline IDS

Tout le trafic passe par le NIDS en ligne, mais à la différence d'un dispositif de pont régulier, le NIDS en ligne inspecte chaque paquet. Si l'un d'eux contient une information qui déclenche une signature, il sera rejeté et logué. Un NIDS en ligne offre les capacités d'un NIDS régulier, avec en plus la possibilité de «blocage» d'un firewall. Comme avec la plupart des NIDS, l'utilisateur peut surveiller, et dans ce cas-ci, protéger plusieurs serveurs ou réseaux avec un seul dispositif.

Puisque ces IPS sont des variantes des NIDS existants, l'écriture des règles est assez facile. Pour bloquer les attaques inconnues avec un NIDS en ligne basé sur des signatures, on doit avoir quelques règles génériques. Ceci, bien évidemment n'arrête pas toutes les nouvelles attaques.

Un NIDS basé sur la détection d'anomalie de protocole peut arrêter des attaques inconnues basées sur les protocoles qu'il peut décoder.

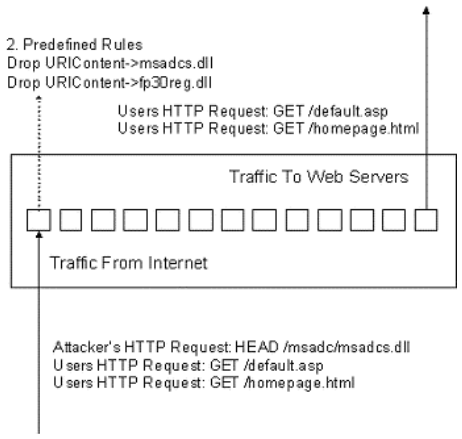
Ces deux systèmes ont l'inconvénient de n'offrir aucune protection contre la mauvaise programmation ou l'erreur de configuration.

4.4.2 Commutateur de la couche 7 (*layer 7 switches*)

Traditionnellement, les commutateurs étaient des dispositifs de la couche 2. Mais avec l'évolution des réseaux, les commutateurs de la couche 7 sont de plus en plus utilisés. On emploie la plupart du temps ces commutateurs pour faire un *load-balance* (équilibre de la charge) d'une application. Ils peuvent inspecter l'information de la couche 7 (comme HTTP, DNS, smtp …) pour prendre les décisions de routage (*routing decision*) ou de commutation (*switching*).

Les constructeurs ont maintenant commencé à ajouter des dispositifs de sécurité à leurs produits, comme la protection contre les attaques DOS et DDoS.

Ces commutateurs sont des matériels afin de fournir une meilleure performance. Ils fonctionnent de la même manière qu'un NIDS en ligne basé sur des signatures quand il doit arrêter des attaques.



Source: N.Deseau, IPS : the next step in the evolution of IDS

Figure 4-3 - Illustration du fonctionnement d'un layer 7 switch

Quand on place un de ces dispositifs devant les firewalls, il fournit une protection pour le réseau entier.

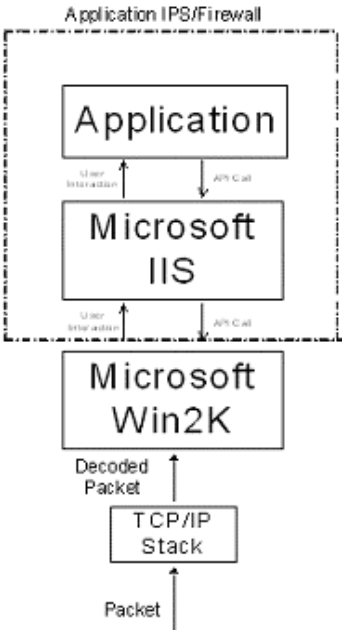
Les commutateurs de la couche 7 sont également configurables pour la redondance. Ils peuvent être également configurés en mode secours automatique ou en mode *load-balance*. Cette caractéristique n'est présente dans aucun des autres IPS.

Cela dit, les inconvénients sont semblables au NIDS en ligne. Ils peuvent uniquement arrêter les attaques qu'ils connaissent. Ce type d'IPS a l'avantage de pouvoir atténuer les attaques DoS sans affecter le reste du réseau.

4.4.3 Application based firewall/IDS

Ces IPS sont chargés sur chaque serveur qui doit être protégé. Ils peuvent être personnalisés pour chaque application.

Ils ne font pas attention à l'information au niveau des paquets, mais observent plutôt les appels API (*Application Program Interface*), la gestion de la mémoire, comment l'application interagit avec le système d'exploitation, et comment l'utilisateur interagit avec l'application. Ceci est fait dans le but de se protéger des attaques inconnues.



Source: N.Deseau, IPS : the next step in the evolution of IDS

Figure 4-4 - Fonctionnement d'un application based firewall

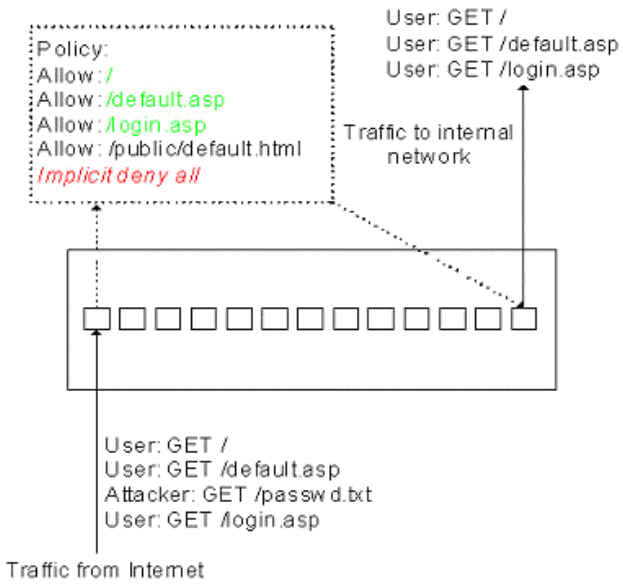
Les IPS de la couche application peuvent dresser un profil du système avant de le protéger. Pendant cette phase, l'IPS observe l'action de l'utilisateur avec l'application et l'action des applications avec le système d'exploitation afin de déterminer à quoi ressemble une action légitime. Après que l'IPS a créé un profil de l'application, il se met au travail. À la différence du NIDS en ligne ou des commutateurs de la couche 7, l'IPS de la couche application est un système du type «*fail close*[3]», ce qui signifie que si une certaine action qui n'est pas prédéfinie est tentée, l'IPS empêche l'action d'avoir lieu.

L'inconvénient de ce type de système est que pendant la phase de dressage d'un profil, l'utilisateur doit s'assurer que chaque aspect de l'application est employé de sorte que l'IPS puisse voir l'action et écrire une règle. Si un essai complet de l'application n'est pas effectué, alors quelques parties de l'application peuvent ne pas fonctionner. L'autre inconvénient est que quand l'application est mise à jour, il est possible qu'elle doive être à nouveau profilée pour s'assurer que la politique ne bloque pas l'utilisation légitime.

[3] Interdit entièrement toute communication ou instaure des règles de base sur le trafic lorsque le programme est interrompu involontairement.

4.4.4 Hybrid switches

Ce type de technologie est un hybride entre le *host based application firewall/IDS* et le commutateur de la couche 7. Ces systèmes sont des solutions matérielles qu'on place devant un serveur, comme le commutateur de la couche 7. Mais au lieu d'employer un ensemble de règle d'un NIDS normal, les commutateurs hybrides emploient une politique semblable à l'IDS/firewall applicatif.



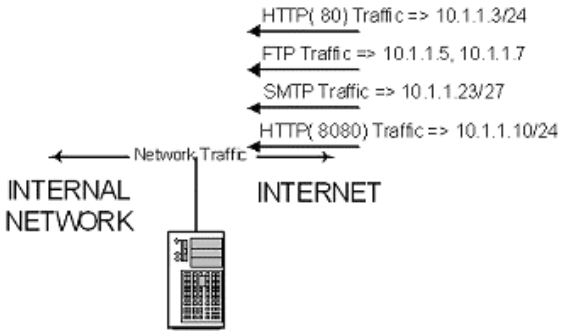
Source: N.Deseau, IPS : the next step in the evolution of IDS

Figure 4-5 - Mécanisme employé par les commutateurs hybrides

Ils inspectent le trafic spécifique pour s'assurer qu'il n'y a aucun contenu malveillant. Certains constructeurs de ce type d'IPS offrent des produits d'évaluation de vulnérabilité de couche application qui complètent leur IPS. Une application peut être balayée avec leur produit d'évaluation de vulnérabilité et l'information de ce balayage peut être importée dans leur IPS comme politique.

4.4.5 Deceptive application

Ce type de technologie emploie quelques pratiques trompeuses. D'abord, il observe tout le trafic réseau pour comprendre ce qu'est un trafic légitime.

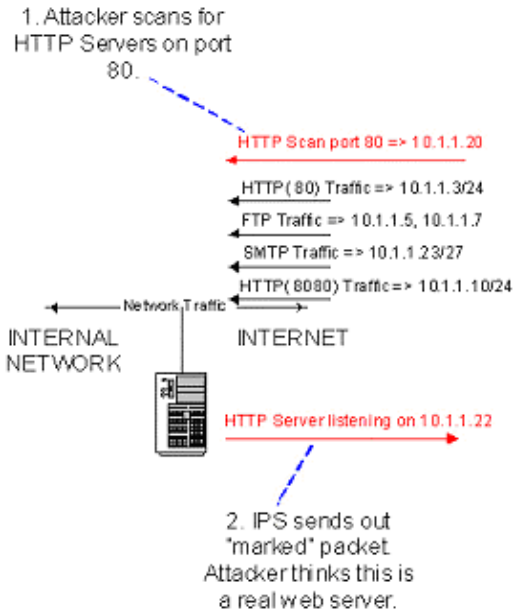


Source: N.Deseau, IPS : the next step in the evolution of IDS

Figure 4-6 - Phase d'observation du trafic

Ceci est semblable à la phase de profilage du firewall/IDS applicatif.

Puis, quand il voit des tentatives de connexion aux services qui n'existent pas sur ce serveur, il envoie une réponse à l'attaquant.



Source: N.Deseau, IPS : the next step in the evolution of IDS

Figure 4-7 - Envoi d'une réponse à l'attaquant

La réponse sera "marquée" avec quelques fausses données, de sorte que quand l'attaquant revient et essaie d'exploiter le serveur, l'IPS voit les données "marquées" et arrête tout le trafic venant de l'attaquant.

5 Détection vs Prévention

Un IDS (réseau) et un IPS (réseau) se diffèrent principalement par 2 caractéristiques:

- Le positionnement sur le réseau de l'IPS (traditionnellement l'IDS est positionné comme un sniffer).

- La possibilité de bloquer immédiatement les intrusions et ce quel que soit le type de protocole de transport utilisé, sans reconfiguration d'un équipement tiers. L'IPS est constitué d'une technique de filtrage de paquets et de moyens de blocage (*drop connection, drop offending packets, block intruder*, …).

Mais on peut également ajouter le déterminisme. Les IDS peuvent (et devraient) utiliser des méthodes non déterministes pour «prédire» toute sorte de menace ou une menace potentielle du trafic existant. Ces méthodes se composent de l'analyse statistique du volume de trafic, des modèles de trafic, et des activités anormales. Mais tout ceci n'est destiné qu'à celui qui veut savoir ce qui se passe sur son réseau et rien de plus. En revanche, un IPS doit être déterministe – juste, exact - dans toutes ses décisions, afin d'exécuter sa fonction de «nettoyeur» de trafic. Un dispositif IPS n'est pas censé prendre des risques ou réagir avec une certaine technique d'«intuition». Il est supposé fonctionner tout le temps,faire un contrôle d'accès du réseau et prendre des décisions. Les firewalls ont fourni la première approche déterministe du contrôle d'accès du réseau.

IDS

Les systèmes de détection d'intrusion ont été développés pour identifier le trafic hostile et envoyer des alertes. Ils ne font rien pour arrêter une attaque. Ils sont déployés hors ligne, et possèdent l'avantage de ne pas pouvoir causer eux-mêmes de panne de réseau.

La nature passive d'un IDS le positionne bien pou identifier:

- Les attaques connues par l'intermédiaire des signatures et des règles.

- Les variations de volume et de direction de trafic en utilisant des règles complexes et une analyse statistique.

- Les variations de modèle de trafic de communication en utilisant l'analyse des flux.

- La détection d'une activité suspecte en utilisant l'analyse de flux, les techniques statistiques, et la détection d'anomalie.

Mais cette passivité ne donne pas que des avantages. On reproche également à un IDS:

Détection uniquement, sans prévention. Comme ils sont placés en dehors du réseau, les NIDS peuvent difficilement avoir une incidence sur le trafic. Les NIDS sont généralement impuissants pour arrêter ou même contrer une attaque en cours.

Durée de récupération du système importante. La nécessité d'une intervention manuelle, de même que la quantité considérable de données à scruter, entraîne une réaction lente face à une attaque ou une intrusion.

Fausses alarmes. Les NIDS confondent parfois un trafic inoffensif avec une attaque: ce sont les «faux positifs». Ces erreurs demandent souvent un temps considérable pour contrôler les paquets en question et déterminer que le système NIDS s'est trompé. Dans d'autres circonstances, le NIDS identifie correctement un motif de trafic particulier qui correspond à une signature, mais la structure s'avère en fait normale pour l'organisation en question.

Données des journaux trop importantes. Un NIDS doit générer des journaux de compte-rendus d'activité anormale ou douteuse sur le réseau. Les responsables de la sécurité doivent passer du temps à examiner ces journaux, pour déterminer ce qui s'est passé et les actions à entreprendre.

Difficultés de déploiement. Les solutions NIDS doivent être placées à toutes les entrées des systèmes et serveurs de l'entreprise, être correctement configurées et régulièrement mises à jour. Mais plus le niveau des menaces augmente plus le déploiement des IDS prend de l'empleur, et on constate que la quantité d'heure nécessaire pour analyser les logs et pour répondre aux incidents (gérer et réparer) devient trop importante.

Les solutions traditionnelles comme le firewall et les anti-virus ne peuvent pas faire face à la nouvelle génération de menaces. Une solution qui protège les informations au moment opportun sans attendre une nouvelle création et distribution de signatures est nécessaire.

IPS

Les solutions de prévention d'intrusion sont mises en place pour traiter:

- Les applications non désirées, les attaques cheval de Troie contre les réseaux privés et les applications en utilisant des règles déterministes et des listes de contrôle d'accès.

- Les attaques DoS ou DDoS comme les SYN et ICMP Flooding en utilisant un algorithme de filtrage à seuil.

- Les abus d'application et les manipulations de protocole -attaques connues et inconnues contre HTTP, ftp, DNS, smtp etc.- en utilisant des règles de protocole d'application et des signatures.

- Les attaques de surcharge ou d'abus d'application en utilisant des limites de consommation de ressource basées sur des seuils.

Ces systèmes possèdent les **avantages** suivants:

Blocage rapide des intrusions. Un événement d'intrusion est le début d'un processus d'atteinte aux ressources informatiques d'une entreprise, sans parler des responsabilités juridiques potentielles. En intervenant dès la détection, un IPS bloque rapidement l'intrusion et minimise la durée totale avant que le réseau ne revienne à la normale.

Détection précise et fiable. A l'aide de plusieurs méthodes de détection, et en tirant parti de sa position en ligne, l'IPS peut détecter les attaques et intrusions avec une précision et une fiabilité supérieures.

Moins dépendant des signatures et davantage des méthodes intelligentes de détection, l'IPS génère beaucoup moins de fausses alarmes. Ainsi le temps et les efforts de l'entreprise sont exclusivement concentrés sur les véritables menaces.

Prévention active. Alors qu'un NIDS avertit simplement de la présence d'un trafic suspect ou anormal, un IPS peut lancer divers mécanismes de réaction.

Le tableau suivant résume les différentes caractéristiques d'un IPS et d'un IDS:

Détection d'intrusion	Prévention d'intrusion
HIDS	**HIPS**
Pour: - Peut détecter l'utilisation d'un système qui viole la politique de sécurité de l'entreprise. - Peut alerter sur des changements au niveau du système tels qu'une importante modification de fichier.	**Pour:** - Assure une protection contre les attaques inconnues. - Exige peu ou aucune mise à jour de sécurité dans une période annuelle. - Empêche les attaques de s'exécuter sur une machine au niveau noyau plutôt que de détecter les résultats d'une attaque réussie.
Contre: - Le coût du déploiement et de la gestion est élevé puisque chaque hôte doit être équipé et une politique doit être développée. - La plupart des fabricants de HIDS ne construisent pas un produit destiné aux postes de travail, ainsi seuls les serveurs sont protégés. - La détection est généralement "a posteriori" dans la courbe de réponse. Une détection réussie vient d'une tentative réussie d'attaque.	**Contre:** - Le coût de tout le système peut être élevé puisqu'un agent est nécessaire pour chaque serveur et/ou poste de travail critique. - Le temps de déploiement afin d'équiper chaque serveur/poste de travail peut être long. - Le produit nécessite un ajustement après l'installation initiale pour être un outil de sécurité parfaitement fonctionnel. - Peut arrêter des applications légitimes en cas de mauvais ajustement. De nouvelles applications ont peut-être besoin d'être examinées par les HIPS avant qu'elles soient déployées.

NIDS	**NIPS**
Pour: - Les systèmes basés sur la détection d'anomalie peuvent détecter une attaque même sur les systèmes qui emploient le cryptage (ils ne voient pas les exploits, ils voient la circulation anormale résultant d'une attaque réussie) - L'observation du trafic avec un système basé sur des règles peut aider à imposer une utilisation du réseau en respectant la politique de l'entreprise.	**Pour:** - Peut arrêter la propagation des vers si déployé correctement sans arrêter le trafic. - Protège contre de nouvelles attaques avant que le code d'exploit soit sorti (dans la plupart des cas). - Réduira le coût de la réponse aux incidents (puisque la plupart des réponses aux incidents sont automatiques).
Contre: - Le coût du facteur "humain" est élevé pour surveiller les événements et pour répondre aux incidents. - À moins qu'un plan de réponse ne soit conçu et mis en place, l'IDS fournit peu ou aucune sécurité. - Un déploiement réussi demande un important ajustement de l'IDS pour réduire au minimum les faux positifs.	**Contre:** - Le coût du déploiement NIPS au sein d'un réseau peut être important. - Puisque les NIPS sont un dispositif intégré au réseau, ils créent un point de défaillance bien qu'il y ait des méthodes pour traiter ce problème. L'approche la plus commune est d'ajouter des éléments redondants, sachant que tout le trafic de réseau traverse le NIPS. - Un NIPS nécessite toujours des mises à jour de sécurité pour être vraiment efficace.

6 Les vulnérabilités de la VoIP

Le système VoIP utilise l'Internet, de ce fait il hérite des bons, mais aussi des mauvais côtés de celui-ci.

Deux types d'attaque sont possibles sur le réseau: l'attaque externe et l'attaque interne. Dans un réseau VoIP, par exemple, les menaces externes sont des attaques lancées par une personne autre que celle qui participe à un appel basé sur la VoIP. Les menaces externes se produisent généralement quand les paquets de la voix traversent un réseau peu fiable et/ou l'appel traverse un réseau tiers durant le transfert de message.

Il existe deux principales vulnérabilités sur un environnement VoIP. La première dépend des protocoles utilisés (SIP, H.323 …) et la deuxième est reliée aux systèmes d'exploitation sur lesquels les éléments VoIP sont implémentés. Chaque protocole ou service a ses propres vulnérabilités. Ce qui suit ne concerne que le protocole SIP.

6.1 Les vulnérabilités du protocole

Un appel téléphonique VoIP est constitué de deux parties: la signalisation, qui instaure l'appel, et les flux de médias, qui transportent la voix.

La signalisation, en particulier SIP (*Session Initiation Protocol* RFC 3261), transmet les entêtes et la charge utile (*Payload*) du paquet en texte clair, ce qui permet à un attaquant de falsifier facilement les paquets. Elle est donc vulnérable aux faussaires qui essaient de voler ou perturber le service téléphonique et à l'écoute clandestine qui recherche des informations sur un compte utilisateur valide, pour passer des appels gratuits par exemple. La signalisation utilise, en général, le port par défaut UDP/TCP 5060. Le firewall doit être capable d'inspecter les paquets de signalisation et ouvre ce port afin de leur autoriser l'accès au réseau. Un firewall qui n'est pas compatible aux protocoles de la VoIP doit être configuré manuellement pour laisser le port 5060 ouvert, créant un trou pour des attaques contre les éléments qui écoutent l'activité sur ce port.

Le protocole RTP (*Real time Transport Protocol*) utilisé pour le transport des flux de média présente également plusieurs vulnérabilités dues à l'absence d'authentification et de chiffrage. Chaque entête d'un paquet RTP contient un numéro de séquence qui permet au destinataire de reconstituer les paquets de la voix dans l'ordre approprié.

Cependant, un attaquant peut facilement injecter des paquets artificiels avec un numéro de séquence plus élevé. En conséquence, ces paquets seront diffusés à la place des vrais paquets.

Généralement, les flux de médias contournent les serveurs proxy et circulent directement entre les points finaux. Les menaces habituelles contre le flux de la voix sont l'interruption de transport et l'écoute clandestine.

Les protocoles de la VoIP utilisent TCP et UDP comme moyen de transport et par conséquent sont aussi vulnérables à toutes les attaques de bas niveau contre ces protocoles, telles que le détournement de la session (TCP) (*session Hijacking*), la mystification (UDP) (*spoofing)*, etc.

Les types d'attaques les plus fréquents contre un système VoIP sont:

6.1.1 Déni de Service (*DoS: Denial of Service*)

Il existe deux sortes d'attaque de déni de service:

- Ceux qui exploitent les erreurs de programmation pour «faire tomber» routeurs et serveurs (ex: attaque par *buffer overflow*),
- Ceux causés par une attaque par saturation.

L'attaque par saturation a pour but de rendre un élément particulier du réseau indisponible, généralement en dirigeant une quantité excessive du trafic réseau à ses interfaces. Cette attaque peut être distribuée (DDos) en impliquant la collaboration de plusieurs ordinateurs. Dans ce cas, l'attaque comporte 2 phases:

Phase1: L'installation des logiciels maîtres et zombies. Les logiciels zombie livrent concrètement l'assaut contre la victime, tandis que les logiciels maîtres, installés sur d'autres ordinateurs, déclenchent l'attaque.

Phase2: Lancement de l'offensive.

Dans le cas de SIP, une attaque DoS peut être directement dirigée contre les utilisateurs finaux ou les dispositifs tels que téléphones IP, routeurs, proxy SIP etc … ou contre les serveurs concernés par le processus, en utilisant le mécanisme du protocole SIP ou d'autres techniques traditionnelles de DoS.

Voyons maintenant en détail les différentes formes d'attaque DoS.

6.1.1.1 CANCEL

C'est un type de déni de service lancé contre l'utilisateur. L'attaquant surveille l'activité du proxy SIP et attend qu'un appel arrive pour un utilisateur spécifique. Une fois que le dispositif de l'utilisateur reçoit la requête INVITE, l'attaquant envoie immédiatement une requête CANCEL. Cette requête produit une erreur sur le dispositif de l'appelé et termine l'appel. Ce type d'attaque est employé pour interrompre la communication.

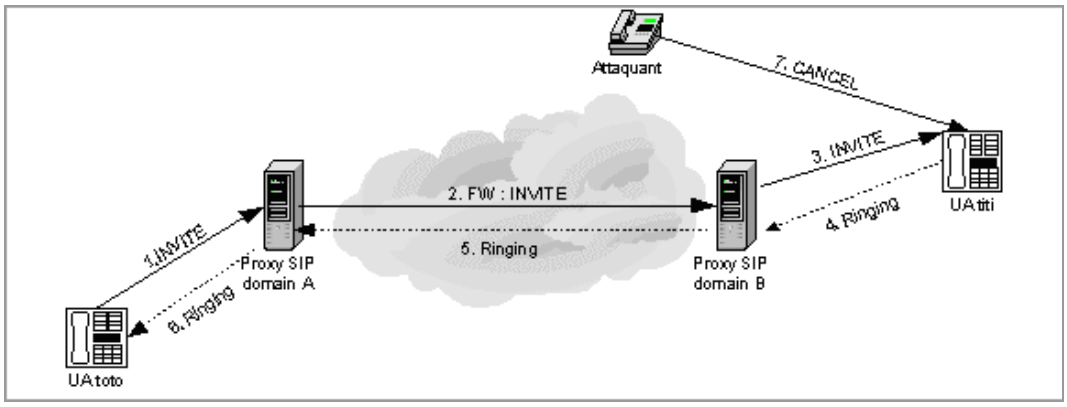


Figure 6-1 - Attaque DoS via une requête CANCEL

L'utilisateur toto initie l'appel, envoie une invitation (1) au proxy auquel il est rattaché. Le proxy du domaine A achemine la requête (2) au proxy qui est responsable de l'utilisateur titi. Ensuite c'est le proxy du domaine B qui prend le relais et achemine la requête INVITE (3) qui arrive enfin à destination. Le dispositif de titi, quand il reçoit l'invitation, sonne (4). Cette information est réacheminée jusqu'au dispositif de toto. L'attaquant qui surveille l'activité du proxy SIP du domaine B envoie une requête CANCEL (7) avant que titi n'ait pu envoyer la réponse OK qui accepte l'appel. Cette requête annulera la requête en attente -l'INVITE-, l'appel n'a pas lieu, c'est un déni de service.

6.1.1.2 BYE

Un autre type d'attaque lancée contre les utilisateurs est le déni de service par requête BYE. Cette dernière est envoyée soit à l'appelant, soit à l'appelé.

Ce type d'attaque peut être utilisé pour perturber l'appel à n'importe quel moment de la communication.

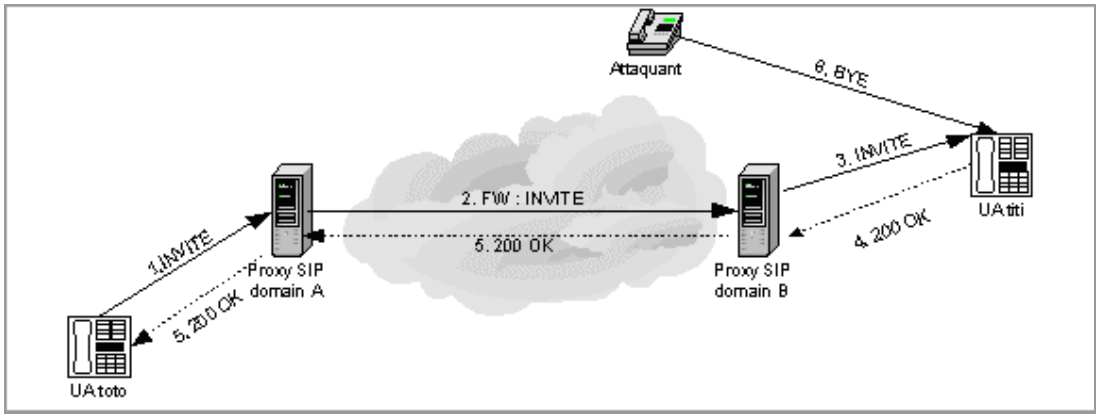


Figure 6-2 - Attaque DoS via une requête BYE

C'est exactement le même scénario que sur la Figure 6-1 ,sauf que dans ce cas-ci, l'attaquant attend qu'une réponse positive acceptant l'appel (4) soit envoyée par titi pour lancer son attaque. Dès que la 200 OK est envoyée, l'attaquant envoie une requête BYE à l'un des participants ou même aux deux, ce qui terminera l'appel sans que les communiquants n'y puissent rien.

6.1.1.3 REGISTER

Le serveur d'enregistrement lui-même est une source potentielle de déni de service pour les utilisateurs. En effet ce serveur peut accepter des enregistrements de tous les dispositifs. Un nouvel enregistrement avec une «*» dans l'entête remplacera tous les précédents enregistrements pour ce dispositif. Les attaquants, de cette façon, peuvent supprimer l'enregistrement de quelques-uns des utilisateurs, ou tous, dans un domaine, empêchant ainsi ces utilisateurs d'être invités à de nouvelles sessions.

Notez que cette fonction de suppression d'enregistrement d'un dispositif au profit d'un autre est un comportement voulu en SIP afin de permettre le transfert d'appel. Le dispositif de l'utilisateur doit pouvoir devenir le dispositif principal quand il vient en ligne. C'est un mécanisme très pratique pour les utilisateurs mais également pour les pirates.

6.1.1.4 Abus des codes de réponses

Le client doit se méfier de certaines réponses. En effet il ne peut pas faire confiance à une réponse non-authentifiée terminant, redirigeant ou interférant l'appel. Des espions peuvent avoir écouté la requête de ce client et générer une réponse non-authentifiée. Le client doit plus particulièrement être prudent avec les réponses de redirection (code 3xx) et d'erreur (codes 4xx à 6xx).

Exemple: l'attaque de déni de service en utilisant un serveur proxy intrus. Ce dernier renvoie une réponse de code 6xx au client. Si le client n'ignore pas cette réponse et envoie une requête vers le serveur "régulier" auquel il était relié avant la réception de la réponse du serveur "intrus", la requête aura de fortes chances d'atteindre le serveur intrus et non son vrai destinataire.

6.1.1.5 SIP INVITE flood

Un déni de service plus traditionnel et des attaques de déni de service distribué sont possibles en utilisant les caractéristiques du protocole SIP. Envoyer simultanément une requête INVITE à un grand nombre d'utilisateurs en falsifiant l'adresse source est un exemple. En Conséquence, tous les dispositifs répondent simultanément au même dispositif créant une situation de déni de service.

Le protocole SIP permet l'envoi de messages INVITE multiples entre les terminaux afin d'implémenter des services comme l'appel en attente (*call hold*) ou la mise en attente d'un correspondant (*call park*).

6.1.1.6 Requêtes manipulées

Dans de nombreuses architectures, les proxies SIP sont placés face à l'Internet afin d'accepter les requêtes depuis le monde entier. Cette situation fournit un certain nombre d'opportunités potentielles en faveur des attaques de déni de service.

Les attaquants peuvent créer des fausses requêtes qui contiennent une adresse IP source falsifiée et un entête Via qui identifient l'hôte cible. Ils envoient cette requête à un grand nombre d'éléments du réseau SIP. De cette façon, un User Agent SIP ou un proxy SIP est utilisé pour produire du trafic DoS visant la cible.

Les attaquants peuvent également essayer d'épuiser la mémoire disponible et les ressources du disque d'un *registrar* en enregistrant un énorme nombre de liaisons (*binding*).

Notez aussi qu'une attaque DoS peut priver le réseau d'adresses IP en épuisant le *pool* d'adresses IP d'un serveur DHCP dans un réseau VoIP. Cette situation est plus connue sous le nom d'épuisement de ressources.

On constate que le déni de service peut être accompli par divers moyens. Le DoS plus traditionnel et les attaques DDoS demandent en général l'inondation d'un hôte par un grand nombre de requêtes de service pour qu'aucune requête légitime ne puisse être traitée.

6.1.2 Ecoute clandestine (*Eavesdropping*)

L'*eavesdropping* est l'écoute clandestine d'une conversation téléphonique. Un attaquant avec un accès au réseau VoIP peut *sniffer* le trafic et décoder la conversation vocale. Un outil nommé VOMIT (*Voice Over Misconfigured Internet Telephones*) est téléchargeable facilement sur le web pour réaliser cette attaque. VOMIT convertit les paquets sniffés en fichier .wav qui peut être réécouté avec n'importe quel lecteur de fichiers son.

6.1.3 Call hijacking

Le *Call hijacking* consiste à détourner un appel. Plusieurs fournisseurs de service VoIP utilisent le web comme interface permettant à l'utilisateur d'accéder à leur système téléphonique. Un utilisateur authentifié peut changer les paramètres de ses transferts d'appel à travers cette interface web. C'est peut être pratique, mais un utilisateur malveillant peut utiliser le même moyen pour mener une attaque.

Exemple: quand un agent SIP envoie un message INVITE pour initier un appel, l'attaquant envoie un message de redirection 3xx indiquant que l'appelé s'est déplacé et par la même occasion donne sa propre adresse comme adresse de renvoi. A partir de ce moment, tous les appels destinés à l'utilisateur sont transférés et c'est l'attaquant qui les reçoit.

Un appel détourné en lui-même est un problème, mais c'est encore plus grave quand il est porteur d'informations sensibles et confidentielles.

6.1.4 Sniffing

Un reniflage (*sniffing*) peut avoir comme conséquence un vol d'identité et la révélation d'informations confidentielles. Il permet également aux utilisateurs malveillants perfectionnés de rassembler des informations sur les systèmes VoIP. Des informations qui peuvent par exemple être employées pour mettre en place une attaque contre d'autres systèmes ou données.

Tous les outils requis pour renifler, y compris pour le protocole H.323 et des plugins SIP pour les renifleurs de paquet (*packet sniffer*) existants, sont disponibles en *open source*.

6.1.5 Interruption du trafic (*Traffic flow disruption*)

Les paquets de données ne circulent pas sur une connexion dédiée pour la durée d'une session,

par conséquent, un attaquant peut manipuler le routage des paquets et causer un délai dans certains chemins, obligeant ainsi ces paquets à prendre un autre chemin choisi par l'attaquant. Ceci a comme conséquence:

- Augmentation de la vulnérabilité de reniflage parce que l'attaquant peut très bien prévoir l'endroit idéal pour placer un dispositif de sniffing.

- Augmentation de la vulnérabilité face à l'attaque DoS.

6.1.6 Replay

Cette attaque consiste à retransmettre de la véritable session de sorte que le dispositif qui la reçoit retraite l'information.

6.1.7 Intégrité des messages

L'attaquant peut diriger une attaque «*man-in-the-middle*» et modifier la communication originale entre les deux parties.

6.1.8 Fraude de facturation

Un attaquant peut emprunter l'identité d'un véritable utilisateur ou même d'un téléphone IP et l'utiliser pour faire des appels longue distance gratuits sur le réseau VoIP.

Une autre forme de cette attaque est, par exemple, des messages SIP BYE truqués et OK échangés qui semblent terminer un appel. La facturation est arrêtée alors que le chemin des médias est resté ouvert. Si ces attaques ne sont pas décelées, d'énormes revenus peuvent être dérobés à une entreprise.

6.1.9 Spam

Trois formes principales de spams sont jusqu'à maintenant identifiés dans SIP:

- Call Spam: Ce type de spam est défini comme une masse de tentatives d'initiation de session (des requêtes INVITE) non sollicitées.

Généralement c'est un UAC (*User Agent Client*: ex Sip Softphone, téléphones IP …) qui lance, en parallèle, un grand nombre d'appels. Si l'appel est établi, l'application spammeuse génère un ACK, joue une annonce pré-enregistrée, et ensuite termine l'appel.

- IM (*Instant Message*) **Spam**: Ce type de spam est semblable à celui de l'e-mail.

Il est défini comme une masse de messages instantanés non sollicitées. Les IM spams sont pour la plupart envoyés sous forme de requête SIP. Ce pourraient être des requêtes INVITE avec un entête `Subject` très grand, ou des requêtes INVITE avec un corps en format texte ou HTML. Bien-sûr, l'IM spam est beaucoup plus intrusif que le spam email, car dans les systèmes actuel, les IMs apparaissent automatiquement (*pop-up*) à l'utilisateur.

- Presence Spam: Ce type de spam est semblable à l'IM spam. Il est défini comme une masse de requêtes de présence (des requêtes SUBSCRIBE) non sollicitées. L'attaquant fait ceci dans le but d'appartenir à la "*white list*" d'un utilisateur afin de lui envoyer des messages instantanés ou d'initier avec lui d'autres formes de communications. L'*IM Spam* est différent du *Presence Spam* dans le fait que ce dernier ne transmet pas réellement de contenu dans les messages.

6.2 Les vulnérabilités de l'infrastructure

Une infrastructure VoIP est composée de téléphones IP, *gateway*, serveurs (*proxy, register*, *location*). Chaque élément, que ce soit un système embarqué ou un serveur standard tournant sur un système d'exploitation, est accessible via le réseau comme n'importe quel ordinateur.

Chacun comporte un processeur qui exécute des logiciels qui peuvent être attaqués ou employés en tant que points de lancement d'une attaque plus profonde.

6.2.1 Faiblesses dans la configuration des dispositifs de la VoIP

- Plusieurs dispositifs de la VoIP, dans leur configuration par défaut, peuvent avoir une variété de ports TCP et UDP ouverts. Les services fonctionnant sur ces ports peuvent être vulnérables aux attaques DoS ou *buffer overflows*.

- Plusieurs dispositifs de la VoIP exécutent également un serveur WEB pour la gestion à distance qui peut être vulnérable aux attaques *buffer overflows* et à la divulgation d'informations.

- Si les services accessibles ne sont pas configurés avec un mot de passe, un attaquant peut acquérir un accès non autorisé à ce dispositif.

- Les services SNMP (*Simple Network Management Protocol*) offerts par ces dispositifs peuvent être vulnérables aux attaques de reconnaissance ou aux *buffer overflows*.

- Plusieurs dispositifs de la VoIP sont configurés pour télécharger périodiquement un fichier de configuration depuis un serveur par TFTP ou d'autres mécanismes. Un attaquant peut potentiellement détourner ou mystifier cette connexion et tromper le dispositif qui va télécharger un fichier de configuration malveillant à la place du véritable fichier.

6.2.2 Les téléphones IP

Un pirate peut compromettre un dispositif de téléphonie sur IP, par exemple un téléphone IP, un *softphone*, ou d'autres programmes ou matériels clients. Généralement, il obtient les privilèges qui lui permettent de commander complètement la fonctionnalité du dispositif.

Compromettre un point final (téléphone IP) peut être fait à distance ou par un accès physique au dispositif. Le pirate pourrait modifier les aspects opérationnels d'un tel dispositif:

- La pile du système d'exploitation peut être changée. Ainsi la présence de l'attaquant ne sera pas remarquée.

- Un *firmware* modifié de manière malveillante peut avoir été téléchargé et installé. Les modifications faites à la configuration des logiciels de téléphonie IP peuvent permettre:

  - Aux appels entrants d'être réorientés vers un autre point final sans que l'utilisateur soit au courant.

  - Aux appels d'être surveillés.

  - A l'information de la signalisation et/ou les paquets contenant de la voix d'être routés vers un autre dispositif et également d'être enregistrés et/ou modifiés.

  - De compromettre la disponibilité du point final. Par exemple, ce dernier peut rejeter automatiquement toutes les requêtes d'appel, ou encore, éliminer tout déclenchement de notification tel qu'un son, une notification visuelle à l'arrivée d'un appel. Les appels peuvent également être interrompus à l'improviste (quelques téléphones IP permettent ceci via une interface web).

- D'autres conséquences possibles sont:

  - Des *backdoors* pourraient été installés.

  - Toutes les informations concernant l'utilisateur qui sont stockées sur le dispositif pourraient été extraites.

L'acquisition d'un accès non autorisé sur un dispositif de téléphonie IP peut être le résultat d'un autre élément compromis sur le réseau IP, ou de l'information récoltée sur le réseau.

Les *softphones* ne réagissent pas de la même façon aux attaques comparés à leur homologues téléphones IP. Ils sont plus susceptibles aux attaques dues au nombre de vecteurs inclus dans le système, à savoir les vulnérabilités du système d'exploitation, les vulnérabilités de l'application, les vulnérabilités du service, des vers, des virus, etc... En plus, le *softphone* demeure sur le segment de données, est ainsi sensible aux attaques lancées contre ce segment et pas simplement contre l'hôte qui héberge l'application *softphone*.

Les téléphones IP exécutent quant à eux leurs propres systèmes d'exploitation avec un nombre limité de services supportés et possèdent donc moins de vulnérabilités.

6.2.3 Les serveurs

Un pirate peut viser les serveurs qui fournissent le réseau de téléphonie sur IP. Compromettre une telle entité mettra généralement en péril tout le réseau de téléphonie dont le serveur fait partie.

Par exemple, si un serveur de signalisation est compromis, un attaquant peut contrôler totalement l'information de signalisation pour différents appels. Ces informations sont routées à travers le serveur compromis. Avoir le contrôle de l'information de signalisation permet à un attaquant de changer n'importe quel paramètre relatif à l'appel.

Si un serveur de téléphonie IP est installé sur un système d'exploitation, il peut être une cible pour les virus, les vers, ou n'importe quel code malveillant.

6.3 Les vulnérabilités du système d'exploitation

Une des principales vulnérabilités des systèmes d'exploitation est le *buffer overflow*. Il permet à un attaquant de prendre le contrôle partiel ou complet de la machine. Bien-sûr, ce n'est pas la seule vulnérabilité et elles varient selon le fabricant et la version.

Elles sont pour la plupart relatives au manque de sécurité lors de la phase initiale de développement du système d'exploitation et ne sont découvertes qu'après le lancement du produit.

Les dispositifs de la VoIP tels que les téléphones IP, *Call Managers*, *Gateways* et les serveurs proxy, … héritent les mêmes vulnérabilités du système d'exploitation ou du *firmware* sur lequel ils tournent.

Il existe une centaine de vulnérabilités exploitables à distance sur Windows et même sur Linux. Un grand nombre de ces exploits sont disponibles librement et prêts à être téléchargés sur l'Internet.

Peu importe comment une application de la VoIP s'avère être sûre, ceci devient discutable si le système d'exploitation sur lequel elle tourne est compromis.

7 La sécurité de la VoIP

Pour se protéger contre ces catégories d'attaques, différents mécanismes de sécurité existent. Ci-après, quelques exemples pour le protocole SIP.

7.1 Exemples de mécanismes de sécurité

Le cryptage: les messages SIP peuvent contenir des données confidentielles. Pour les protéger, SIP possède 3 mécanismes de cryptage:

- Cryptage de bout en bout du corps du message SIP et de certains champs d'entête sensibles aux attaques,

- Cryptage au saut par saut (*hop by hop*),

- Cryptage au saut par saut du champ d'entête Via pour dissimuler la route qu'a empruntée la requête.

Toutes les requêtes et réponses SIP ne peuvent pas être cryptées de bout en bout car les champs d'entête To et Via doivent être lisibles par les proxies SIP (pour router correctement les requêtes et réponses). Par contre, elles peuvent toutes être intégralement cryptées au saut par saut.

Authentification: afin d'empêcher tout intrus de modifier et de retransmettre des requêtes ou réponses SIP, des mécanismes de contrôle d'intégrité et d'authentification des messages sont mis en place. Les mécanismes des couches réseau et transport sont utilisés pour l'authentification des messages transmis au saut par saut. Et pour des messages SIP transmis cryptés de bout en bout, des clés publiques et des signatures sont utilisées et stockées dans les champs d'entête *Authorization*.

Ces mesures aident à prévenir l'écoute clandestine. Leur application empêche les pirates de saisir et de mystifier l'information.

Il existe bien évidemment d'autres mécanismes, ce document ne s'étendra pas plus sur ce point. Le lecteur qui veut en savoir plus peut consulter la RFC 3261.

7.2 La protection par prévention

L'objectif principal de ce chapitre est de présenter un moyen de protection contre les attaques au niveau du protocole applicatif.

On distingue deux types d'attaques applicatives:

i. Les attaques qui ne respectent pas le protocole applicatif, qui enfreignent les règles liées à ce protocole ou qui se caractérisent par un usage «anormal» de celui-ci,

ii. Les attaques qui respectent le protocole applicatif et qui ne peuvent donc pas être détectées par une analyse protocolaire.

L'analyse protocolaire appelée aussi détection d'anomalie de protocole est justement la technique de prévention d'intrusion utilisée dans le cadre de ce travail. Il s'agit d'une analyse des paquets circulant sur le réseau ayant pour but de décoder le protocole applicatif utilisé et de contrôler la conformité de l'utilisation de la RFC. Le système de prévention effectue ce qu'on appelle une analyse applicative «temps réel». En effet, le flux est recomposé et analysé «à la volée». Cette analyse n'est plus faite dans une application qui s'exécute au-dessus du système d'exploitation (comme dans le cas des *reverses proxies*), mais directement dans le noyau du système d'exploitation.

Cette technique possède l'avantage d'être une technique *proactive*[4]. Elle permet de bloquer des attaques non connues, simplement parce qu'elles dérogent à la RFC. Mais c'est une technique non générique, qui n'est efficace que pour le protocole analysé.

La protection par l'IPS est une solution destinée à toutes les entreprises qui ont déployé un service de VoIP comme montre la figure suivante.

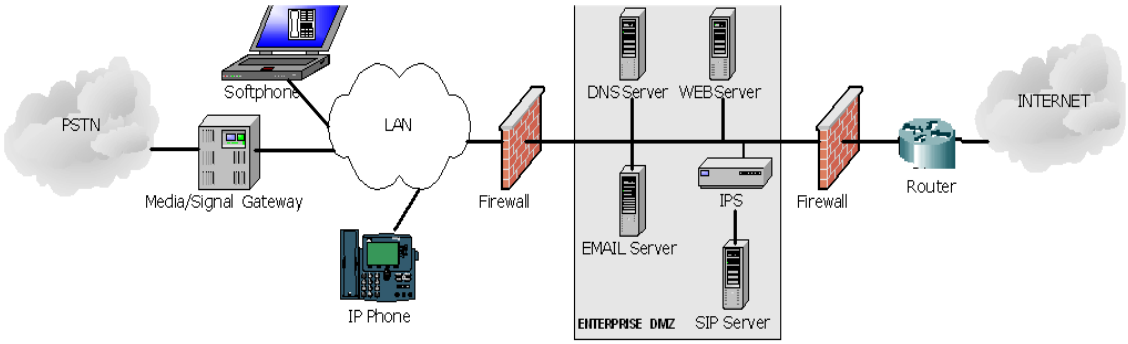


Figure 7-1 - Réseaux d'entreprise avec un service de téléphonie IP

[4] Qui fonctionne de manière dynamique, par anticipation.

Le réseau d'entreprise est en général constitué de deux sections: le réseau interne (LAN) et la zone démilitarisée (DMZ). La DMZ est connectée au réseau public à travers le firewall externe et comporte différents serveurs accessibles depuis l'extérieur, tels que le serveur web, e-mail, DNS (*Domain Name Server*), etc. Le réseau interne est connecté à la DMZ par un autre firewall.

Chaque dispositif de sécurité est utilisé dans le but de se protéger d'une menace particulière. De plus, ils possèdent tous un point faible. Ainsi, c'est seulement en combinant les différentes techniques qu'on arrive réellement à une protection optimale. Les firewalls sont la barrière de trafic, ils laissent passer ou non les trafics entrant selon leur politique. Les IDS sont le moniteur de trafic, ils observent l'activité du réseau et consignent ce qui est suspect. Ils peuvent détecter les menaces qui ont pu échapper au firewall et les notifient. Les IPS fournissent les mesures préventives et les commandes utiles pour combattre de nouvelles menaces.

7.3 L'IPS par Snort Inline

Ce travail implémente un NIDS en ligne (*Inline NIDS*), méthode choisie parmi les différentes façons de réaliser un IPS présentées à la section 4.4.

Qu'est-ce qui a motivé ce choix?

L'objectif était de réaliser un IPS pour le protocole SIP en respectant dans la mesure du possible les impératifs:

- Ne pas toucher la topologie du réseau existant.
- Intégrer une solution totalement transparente, c'est-à-dire ne rien toucher au système de protection déjà déployé.
- Avoir une solution la moins coûteuse possible

Le dernier point conduit tout de suite à une solution *open source*. Il existe dans le domaine des logiciels libres un IPS, Snort Inline, qui est une version modifiée du célèbre IDS Snort. Celui-ci constituera la base de l'IPS SIP.

Des fonctionnalités permettant d'inspecter le trafic SIP lui seront ajoutées.

7.4 Brève présentation de Snort Inline

Snort Inline est un système de prévention d'intrusion. Il exploite toutes les fonctionnalités d'origine de Snort telles que le décodage, le réassemblage des paquets… Il utilise également les mêmes composants que Snort, tels que les préprocesseurs de flux. Par contre, Snort Inline utilise **iptables** et **ip_queue**[5] pour l'acquisition et le transfert des paquets.

Iptables est configuré comme un firewall normal. Il prend tous les paquets entrants et les transmet à Snort Inline. Ce dernier a un moteur de détection d'intrusion intégré qui analyse les paquets reçus et transmet uniquement les paquets jugés non hostiles au réseau interne.

L'hôte Snort Inline doit résider entre la source et la destination des paquets, pour que tout le trafic passe par l'ordinateur qui héberge Snort Inline.

7.4.1 Les composants

En général, Snort Inline est divisé en plusieurs composants:

- Décodeur de paquet (*Packet decoder*)
- Préprocesseurs (*Preprocessors*)
- Moteur de détection (*Detection Engine*)
- Système d'alerte et d'enregistrement de log (*Logging and Alerting System*)

Et éventuellement des Modules de sortie (*Output Module*)

Ces composants travaillent ensemble pour détecter des attaques particulières. La figure suivante montre comment ces composants sont organisés.

[5] Ip_queue est un module du noyau responsable de passer les paquets depuis le target QUEUE dans l'espace noyau pour une application qui se trouve dans l'espace utilisateur.

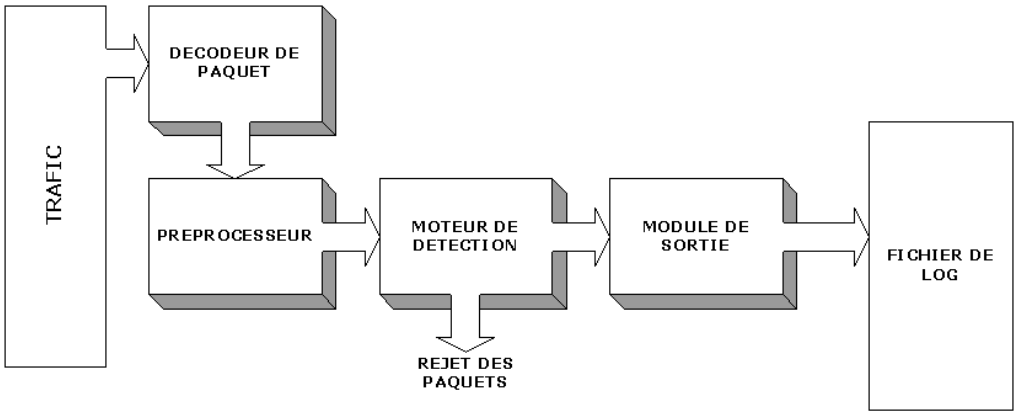


Figure 7-2 - Les différents composants de Snort Inline

Chaque paquet de données passe par le décodeur de paquet. Sur son chemin vers les modules de sortie, il est soit rejeté, soit logué.

7.4.1.1 Le décodeur de paquet

Le décodeur de paquet récupère les paquets des différentes interfaces réseaux et les prépare pour passer dans le préprocesseur ou pour être envoyés au moteur de détection. Les interfaces peuvent être Ethernet, SLIP (*Serial Line Internet Protocol*), PPP (*Point to Point Protocol*)…

Le décodeur de paquet se compose d'une série de décodeurs spécialisés chacun dans un élément de protocole spécifique.

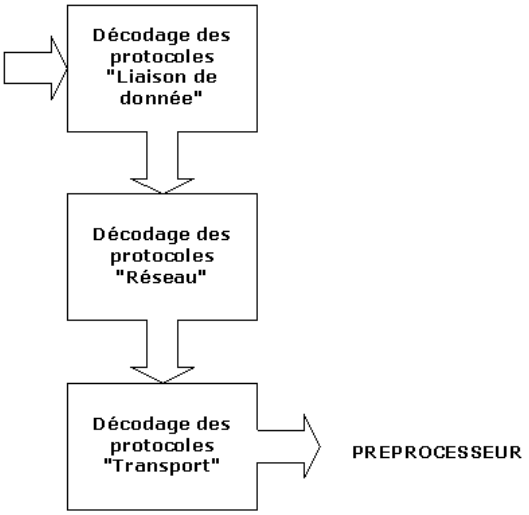


Figure 7-3 - Décodeur de paquet

Le décodeur de paquet transforme les éléments du protocole (par exemple: les entêtes LLC, Ethernet, IP, TCP, UDP …) spécifiques en une structure de données interne.

7.4.1.2 Les préprocesseurs

Les préprocesseurs sont des composants ou *plug-ins.* Ils peuvent être utilisés avec Snort Inline pour «arranger» ou «modifier» les paquets avant que le moteur de détection ne fasse ses opérations pour découvrir si le paquet est utilisé par un intrus. Certains préprocesseurs effectuent également une détection pour trouver des anomalies dans les entêtes de paquet et produisent des alertes ou des logs.

7.4.1.3 Le moteur de détection

Le moteur de détection est la partie la plus importante. Son rôle est de détecter s'il existe une activité d'intrusion possible dans un paquet. Le moteur de détection utilise des règles dans ce but. Les règles sont lues dans les structures ou les chaînes de données internes où elles sont associées avec chaque paquet. Si un paquet correspond à une des règles, une action appropriée est effectuée. Le paquet est par exemple rejeté et un log est noté.

7.4.2 Les modes

L'IPS peut être déployé en deux modes.

7.4.2.1 Mode NAT

En mode NAT, le réseau est divisé en deux parties: la partie publique et la partie privée. Le côté public du réseau est séparé du côté privé par l'IPS. L'IPS exécute la configuration NAT, il établit une correspondance de toutes les adresses publiques à son interface publique. Ensuite il les traduit à leurs adresses privées correspondantes.

7.4.2.2 Mode Bridge

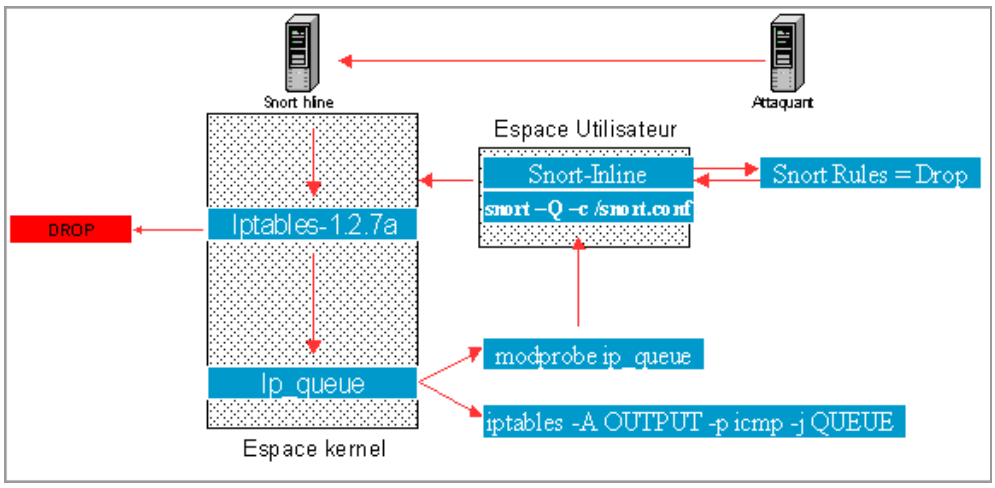
En mode bridge, l'IPS agit en tant que pont entre le réseau public et le réseau privé. Le mode pont est furtif. En effet, le dispositif devient non détectable puisqu'il ne possède pas d'adresse IP. Il est donc plus sûr, ce qui en fait le mode recommandé.

7.4.3 Les règles

Snort Inline est équipé de réponses de prévention des intrusions, qui prennent la forme d'actions de règles classiques:

7.4.3.1 Drop

La règle *drop* indiquera à iptables de rejeter le paquet et d'envoyer un *TCP reset* si le protocole est TCP ou un *ICMP port unreachable* si le protocole est UDP, et de loguer le paquet par l'intermédiaire des moyens habituels de Snort.



Source: South Florida Honeynet Project

Figure 7-4 - Snort Inline drop mode

Quand des paquets sont lus par Snort Inline dans le *target* QUEUE d'iptables, les meta-données et/ou la charge utile (*payload*) du paquet sont comparées à l'ensemble des règles de Snort Inline. Si le paquet a été identifié comme étant un paquet malveillant, contenant un certain type de signature d'exploit, Snort Inline détermine ce qu'il va devenir. Il se base sur les règles prédéfinies pour prendre sa décision. Si on a défini des règles drop, alors le paquet sera rejeté. Cette décision est passée à iptables qui va la mettre en œuvre.

7.4.3.2 Sdrop

La règle *sdrop* indiquera à iptables de rejeter le paquet, et d'envoyer un *reset* mais aucun log n'est écrit.

Il existe deux options sur la façon d'envoyer un *reset*:

- On peut toujours utiliser un *raw socket*[6]*,* c'est le comportement par défaut de Snort Inline. Dans ce cas, on doit avoir une interface qui possède une adresse IP assignée. S'il n'y a pas une interface avec une adresse IP valide avec accès à la source du paquet, le paquet sera logué, mais le *reset* ne sera jamais envoyé sur le réseau.

- On peut également effectuer des *resets* par l'intermédiaire d'un dispositif physique en utilisant iptables. On utilise le nom `INDEV` fourni par ip_queue comme interface sur laquelle on envoie le *reset*. Il n'est plus nécessaire qu'une adresse IP soit présente sur le pont, il demeure donc furtif.

7.5 Réalisation de l'IPS SIP

Les systèmes de la VoIP utilisent différents protocoles pour la gestion d'appel (par ex. SIP) et pour la livraison de données (par ex. RTP). Une étude menée par le «*Research Departement Qovia, Inc* » montre l'effet des mesures de sécurité contre les intrusions sur la qualité de service de la VoIP. Voici leur conclusion:

> «*Qovia found Snort-inline's operation within the user space deteriorates the QoS greatly as its long travel path incurs significant delays and jitter [20]*»

Autrement dit, cette équipe de testeurs a montré que les opérations menées par Snort Inline détériorent considérablement la qualité de service. Lors du test, Snort Inline prend en charge à la fois l'inspection des paquets de signalisation, mais également les paquets de média. L'IPS réalisé lors de ce travail ne prendra en charge que la partie signalisation de la VoIP. Ainsi la qualité de service sera maintenue. En plus, l'objectif principal est de protéger le proxy. Rappelez-vous qu'en SIP, les flux RTP contournent en général le proxy et sont transmis directement entre les clients. Raison de plus, dans un premier temps en tout cas, pour concentrer tout l'effort sur la signalisation.

[6] Un raw socket est un socket avec un pouvoir «supérieur», il permet d'écrire ou de lire des paquets sur une interface réseau, avec les paramètres souhaités, c'est-à-dire qu'il aura la possibilité de modifier l'entête d'un paquet pour ensuite l'envoyer avec les options de ce même entête modifiées (adresse IP source, flag TCP, etc).

7.5.1 Qu'est-ce que l'IPS est censé réaliser?

L'IPS SIP a pour objectif la détection et l'arrêt des attaques basées sur les vulnérabilités du protocole SIP contre le proxy SIP de l'entreprise.

Il utilise la méthode de détection basée sur une analyse protocolaire. Cette méthode consiste à décoder le paquet et vérifier s'il est conforme à la RFC. Ce contrôle de conformité est également appliqué au comportement de l'application.

Snort Inline offre une fonction de décodage - une opération effectuée par le composant décodeur de paquet - mais elle s'arrête au protocole de transport. Les paquets sont ensuite acheminés jusqu'au préprocesseur.

Comme on désire garder toutes les fonctionnalités actuelles de Snort Inline, mais en plus que le décodage du paquet soit fait jusqu'au protocole applicatif, un ajout de fonctionnalité est inévitable. C'est la raison pour laquelle un préprocesseur SIP a été créé.

Le préprocesseur inspecte les requêtes adressées au proxy. Il les décode, vérifie s'ils sont conformes à la RFC. S'il découvre des irrégularités, il enregistre un log du paquet dans le fichier prévu pour ça. Si en plus il a été configuré pour prendre une mesure plus restrictive, comme rejeter le paquet jugé hostile, il exécute cette action. L'utilisateur a le libre choix d'activer ou pas cette mesure de rejet. Si le paquet est jugé non hostile, il est transmis au proxy. Le préprocesseur se concentre plus sur les requêtes, car la plupart des attaques utilisent ces messages comme moyen d'attaque.

Le prototype réalisé au cours de ce travail détecte et arrête les attaques lancées contre le proxy. Les attaques font parties de la catégorie de celles qui ne respectent pas le protocole applicatif (cf. 7.2 i), c'est-à-dire des attaques par paquets malformés ou des paquets bien construits mais hors contexte. En plus, il détecte et arrête également les attaques de spam ou l'attaque de déni de service par inondation de requêtes *INVITE*.

En raison du peu de temps à disposition, il n'est malheuresement pas possible de fournir une protection contre toutes les différentes attaques présentées à la section 6. Mais avec la mesure de protection fournie, par cet IPS on arrive déjà à soulager considérablement le proxy, c'est-à-dire ne plus le surcharger de paquets «inutiles».

Mais le proxy n'implémentait-il pas déjà des mesures contre de telles situations?

La réponse est oui, la plupart des proxies arrivent à savoir si un paquet est malformé et le refusent en envoyant un message «`603 decline`». Mais si on considère le cas où un attaquant adresse, en continu, plusieurs paquets malformés au proxy, ce dernier est **surchargé** et **occupé** à répondre à ces messages.

Il est occupé à servir un utilisateur qui ne veut même pas faire un appel: c'est une forme de tentative de déni de service.

7.5.2 Qu'est-ce qui se passe dans l'IPS?

Le travail de l'IPS commence quand le firewall iptables dépose des paquets dans le *target* QUEUE.

Le paquet est pris en charge par le décodeur de paquet et ensuite transmis à un préprocesseur. Si le paquet est un message SIP, c'est le préprocesseur SIP qui prend le relais.

Afin de mener à bien son travail, le préprocesseur reçoit en plus d'autres informations complémentaires. Ces dernières ont été prédéfinies dans le fichier de configuration.

L'utilisateur définit dans ce fichier:

- Une liste de numéros de ports utilisés par le service SIP.
- Un entier qui indique le nombre maximum d'appels simultanés que le proxy SIP peut gérer.
- Un entier qui indique le nombre maximum d'appels qu'un utilisateur peut passer ou recevoir pendant un certain temps, lors d'une situation normale.
- Et bien-sûr un entier qui indique le temps d'observation (exprimé en seconde),
- L'adresse IP du proxy et enfin,
- L'action à prendre en présence d'un paquet malveillant: l'action DROP.

Voici à quoi ressemble l'architecture interne du préprocesseur SIP:

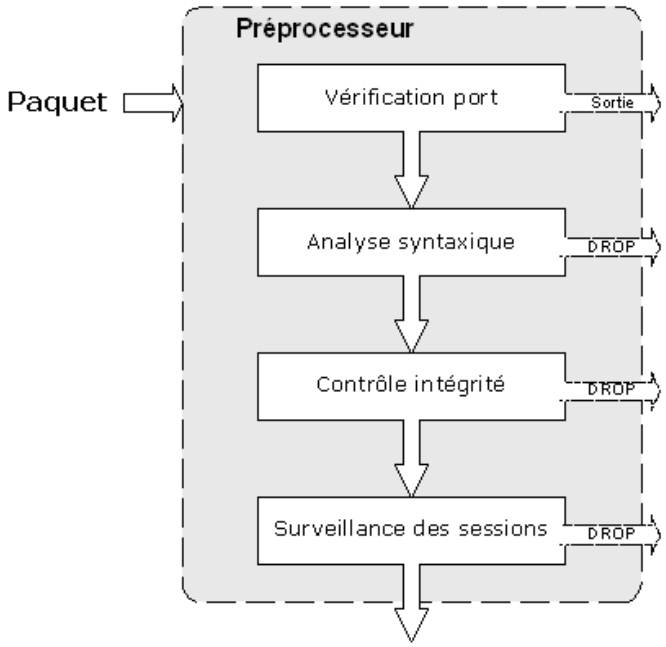


Figure 7-5 - Architecture du préprocesseur SIP

7.5.2.1 Contrôle du numéro de port

La première étape de la détection consiste à vérifier si le paquet reçu par le préprocesseur lui est vraiment destiné. Pour ce faire, il vérifie le port destination du paquet. Il utilise comme référence la liste des ports transmise par le fichier de configuration. Si aucun port n'est mentionné, le préprocesseur utilise le port SIP par défaut 5060. Cette opération limite les paquets à traiter uniquement à ceux appartenant au protocole SIP.

7.5.2.2 Analyse syntaxique

La deuxième étape consiste à analyser le paquet. Le préprocesseur effectue cette analyse en utilisant la bibliothèque SIP «*GNU oSIP library*». Cette dernière est une implémentation de la RFC 3261. Elle se compose de deux parties:

1. *PARSER*

2. *TRANSACTION LAYER*

La partie *parser* est capable d'analyser les requêtes et les réponses SIP. Il est important de souligner que le *parser* mis à disposition dans cette bibliothèque ne vérifie pas complètement si le message est conforme et bien formé.

Ex: `INVITE sip:toto@domain.com:abcd SIP/2.0`, cette ligne comporte un étrange numéro de port, pourtant le *oSIP parser* ne considère pas ceci comme étant une erreur.

Autrement dit le préprocesseur, à ce stade, n'effectue qu'une analyse syntaxique en utilisant la bibliothèque. En effet même si le message SIP est correctement analysé par le *oSIP parser*, il peut être encore non conforme.

Déroulement de l'analyse syntaxique:

Le paquet reçu par le préprocesseur contient d'autres informations que le message SIP car les données de l'application sont encapsulées:

Figure 7-6 - Encapsulation des données en TCP/IP

Et comme la figure le montre, l'information utile pour le préprocesseur se trouve dans le «*user data*» (données utilisateur). C'est la donnée de l'application.

Avant de commencer une analyse, le préprocesseur prend soin d'extraire cette partie du paquet. Il transmet ensuite cette donnée à la bibliothèque *oSIP*. L'API *osip_message_parser* parcourt la donnée et extrait les différentes parties du message SIP.

Un message SIP est composé de:

- ligne de départ (*start line*),
- des entêtes (*header*)
- corps de message (*message body*)

De chacune de ces différentes parties, le *parser* extrait encore les informations qui la composent.

Ce parcours permet aussi à l'API de contrôler la présence des entêtes obligatoires dans chaque message SIP. Autrement dit, il vérifie si la syntaxe imposée par la RFC est respectée.

L'API met le résultat de ces extractions dans une structure de données. Les différents champs d'entête qui composent un message SIP sont accessibles depuis cette structure.

Quand l'analyse n'a pas pu être faite - parce qu'un entête manque ou bien il est présent mais la valeur qu'il contient n'est pas conforme - le préprocesseur génère un log.

> Ex: `CSeq: 1234INVITE`, est interprété comme étant une erreur par le *parser*.
>
> Rappel: L'entête Cseq est composé d'un numéro de séquence et d'une méthode, ce qui est bien le cas de l'exemple. Alors pourquoi est-il considéré comme erroné? Le *parser* cherche plutôt une ligne `CSeq:1234 sp INVITE` (`sp = espace`) car c'est la règle décrite par la RFC.

Si l'utilisateur a précisé une action – *drop* (rejeter) - à prendre dans la configuration du détecteur, le préprocesseur effectue ensuite cette action. Comme mentionné auparavant, le *parser* est assez tolérant et «laisse passer» quelques erreurs. Il est donc important de faire une autre vérification plus stricte du message. C'est la prochaine étape.

7.5.2.3 Contrôle d'intégrité

Ce contrôle ne sera effectué que pour les messages qui ont passé avec succès l'analyse syntaxique. Ce qui suit est donc un contrôle complémentaire à ce que le *parser* a déjà effectué.

Le contrôle d'intégrité est appliqué surtout aux entêtes obligatoires dans chaque requête et réponse, dans le but d'optimiser le préprocesseur.

D'après la RFC 3261 (section 20.1), ces entêtes obligatoires sont:

Header field	where	ACK	BYE	CAN	INV	OPT	REG
Accept	R	-	o	-	o	m*	o
Accept	2xx	-	-	-	o	m*	o
Accept	415	-	c	-	c	c	c
Accept-Encoding	R	-	o	-	o	o	o
Accept-Encoding	2xx	-	-	-	o	m*	o
Accept-Encoding	415	-	c	-	c	c	c
Accept-Language	R	-	o	-	o	o	o
Accept-Language	2xx	-	-	-	o	m*	o
Accept-Language	415	-	c	-	c	c	c
Alert-Info	R	-	-	-	o	-	-
Alert-Info	180	-	-	-	o	-	-
Allow	R	-	o	-	o	o	o
Allow	2xx	-	o	-	m*	m*	o
Allow	r	-	o	-	o	o	o
Allow	405	-	m	-	m	m	m
Authentication-Info	2xx	-	o	-	o	o	o
Authorization	R	o	o	o	o	o	o
Call-ID	c	m	m	m	m	m	m
Call-Info		-	-	-	o	o	o
Contact	R	o	-	-	m	o	o
Contact	1xx	-	-	-	o	-	-
Contact	2xx	-	-	-	m	o	o
Contact	3xx	-	o	-	o	o	o
Contact	485	-	o	-	o	o	o
Content-Disposition		o	o	-	o	o	o
Content-Encoding		o	o	-	o	o	o
Content-Language		o	o	-	o	o	o
Content-Length		t	t	t	t	t	t
Content-Type		*	*	-	*	*	*
CSeq	c	m	m	m	m	m	m
Date		o	o	o	o	o	o
Error-Info	300-699	-	o	o	o	o	o
Expires		-	-	-	o	-	o
From	c	m	m	m	m	m	m
In-Reply-To	R	-	-	-	o	-	-
Max-Forwards	R	m	m	m	m	m	m
Min-Expires	423	-	-	-	-	-	m
MIME-Version		o	o	-	o	o	o
Organization		-	-	-	o	o	o

Table 7-1– Résumé des champs d'entête

Header field	where	ACK	BYE	CAN	INV	OPT	REG
Priority	R	-	-	-	o	-	-
Proxy-Authenticate	407	-	m	-	m	m	m
Proxy-Authenticate	401	-	o	o	o	o	o
Proxy-Authorization	R	o	o	-	o	o	o
Proxy-Require	R	-	o	-	o	o	o
Record-Route	R	o	o	o	o	o	-
Record-Route	2xx,18x	-	o	o	o	o	-
Reply-To		-	-	-	o	-	-
Require		-	c	-	c	c	c
Retry-After	404,413,480,486	-	o	o	o	o	o
	500,503	-	o	o	o	o	o
	600,603	-	o	o	o	o	o
Route	R	c	c	c	c	c	c
Server	r	-	o	o	o	o	o
Subject	R	-	-	-	o	-	-
Supported	R	-	o	o	m*	o	o
Supported	2xx	-	o	o	m*	m*	o
Timestamp		o	o	o	o	o	o
To	c(1)	m	m	m	m	m	m
Unsupported	420	-	m	-	m	m	m
User-Agent		o	o	o	o	o	o
Via	R	m	m	m	m	m	m
Via	rc	m	m	m	m	m	m
Warning	r	-	o	o	o	o	o
WWW-Authenticate	401	-	m	-	m	m	m
WWW-Authenticate	407	-	o	-	o	o	o

Table 7-2 – Résumé des champs d'entête (suite et fin)

Légende :

R: peut apparaître uniquement dans les requêtes

r: peut apparaître uniquement dans les réponses

c: copier depuis la requête

m: obligatoire

o: optionnel

m*: Le champ d'entête DEVRAIT être envoyé, mais les clients/serveurs doivent être préparés à recevoir des messages sans ce champ.

t: Le champ d'entête DEVRAIT être envoyé, mais les clients/serveurs doivent être préparés à recevoir des messages sans ce champ.

*: Le champ d'entête est requis si le corps de message n'est pas vide.

-: pas applicable

Les entêtes obligatoires sont: `Call-ID, Cseq, To, From et Via.`

Call-ID: en général, cet entête est de la forme: `localid@host`. S'il est présent, on contrôle la taille des différentes parties qui le composent. Ceci est fait dans le but de se protéger contre un dépassement de capacité. En effet le Call-ID est stocké par le préprocesseur dans une table, il est donc prudent de contrôler sa taille.

Cseq: s'il est présent, l'IPS contrôle si le nom de la méthode indiqué dans cet entête correspond à la méthode indiquée à la ligne de requête.

Ex:

```
Session Initiation Protocol
  → Request-Line: INVITE sip:1234@192.168.20.40 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.20.30:5060;rport;branch=z9hG4bK00F387AD8CE34FA78F1DB2C82A9EE3C
5
        From: winLaptop <sip:2345@192.168.20.40>;tag=1878808204
        To: <sip:1234@192.168.20.40>
        Contact: <sip:2345@192.168.20.30:5060>
        Call-ID: 1E8BFBC8-1A9F-4522-9A8D-40825F7ACE70@192.168.20.30
      → CSeq: 20985 INVITE
        Max-Forwards: 70
        Content-Type: application/sdp
        User-Agent: X-Lite release 1103m
        Content-Length: 271
    Message body
```

Table 7-3 – Message SIP

Si ce n'est pas le cas, un log est généré et éventuellement l'action drop est exécutée. La même chose se produit si l'entête est absent.

Bien évidemment, cette vérification n'a pas lieu pour les réponses, puisqu'elles ne comportent pas de ligne de requête.

To et **From**: en plus de la vérification de leur présence, l'IPS contrôle également s'ils respectent bien le format `sip:user:password@host:port;uri-parameters?headers` décrit dans la RFC 3261. La section 19.1.2 de la RFC précise un peu plus quel composant de ce format générique doit être présent dans un contexte particulier. Ci-après, une partie du tableau qui indique cette précision:

	To	From
user	o	o
password	o	o
host	m	m
port	-	-
user-param	o	o
method	-	-
maddr-param	-	-
ttl-param	-	-
transp.-param	-	-
lr-param	-	-
other-param	o	o
headers	-	-

Légende : o: optionnel, m: obligatoire, -:pas autorisé.

Ce tableau indique que si les entêtes To et From sont présents, l'URL qu'ils comportent doit au moins contenir la partie *host.* L'IPS ne se contente pas de contrôler la partie host, car si le signe @ est présent, la partie *user* ne doit pas être vide.

En résumé, le contrôle d'intégrité des entêtes To et From est un contrôle du format de l'URL, et doit plus précisément vérifier si cet URL comporte au moins une partie *user* et *host*.

Ex: `winLaptop <sip:2345@192.168.20.40>;` est un entête From valide.

Si l'entête To et/ou From est absent du message, ou bien s'ils sont présents, mais l'URL est vide ou il manque une des parties décrites ci-dessus, un log est généré et éventuellement l'action drop est effectuée.

Via: si cet entête est malformé - c'est-à-dire qu'il lui manque une des parties obligatoires qui le composent: le nom du protocole et la version (SIP 2.0), et le paramètre *branch* – un log est généré et éventuellement l'action drop est effectuée.

Pour les requêtes SIP, l'IPS contrôle, en plus de ces entêtes, le format de l'URI. En effet, le proxy ne doit pas accepter des messages ayant un URI qui ne respecte pas la norme. Notez que la norme définit entre autres: `sip` ou `sips` comme formats valide.

7.5.2.4 Surveillance de la session

Une fois que l'analyse syntaxique ainsi que la vérification de l'intégrité des messages sont effectuées, le préprocesseur passe à la surveillance de la session. Il garde dans une structure de données une trace de toutes les sessions.

Voici à quoi ressemble chaque entrée de cette structure:

Enable	From	To	Call-ID	wait_ack	active	expires
FAUX				FAUX	FAUX	Heure courante [s]

Table 7-4 -Représentation d'une entrée de la table des sessions

Le préprocesseur garde différents paramètres de l'appel dans cette table, ainsi que quelques *flags* qui indiquent soit l'état de l'entrée, soit l'état de la session.

- Le champ *enable* et *expires* sont des *flags* qui indiquent l'état de l'entrée de la table.

Ces informations sont nécessaires pour déterminer à quelle ligne la prochaine écriture aura lieu, mais aussi pour garder une sorte d'historique de la session entre différents utilisateurs.

- Le champ *wait_ack* et *active* sont des *flags* qui indiquent l'état de la session.

- L'ensemble des 3 entêtes suivants forme l'identificateur d'un appel. C'est pourquoi il est intéressant de les garder en mémoire. Il s'agit des entêtes:

- **Call-ID**: identificateur unique d'un ensemble de messages,
- **To**: destinataire de la requête,
- **From**: origine de la requête

La deuxième ligne de la Table 7-4 représente la valeur par défaut de chaque champ.

Le préprocesseur remplit cette table en inspectant les requêtes qui viennent du réseau externe à destination du proxy. Un diagramme d'état de la session a été défini pour mieux situer chaque session.

Rappelez-vous que les sessions SIP sont établies en employant la méthode de la poignée de main à trois voies *(three-way handshake)* (tout comme en TCP).

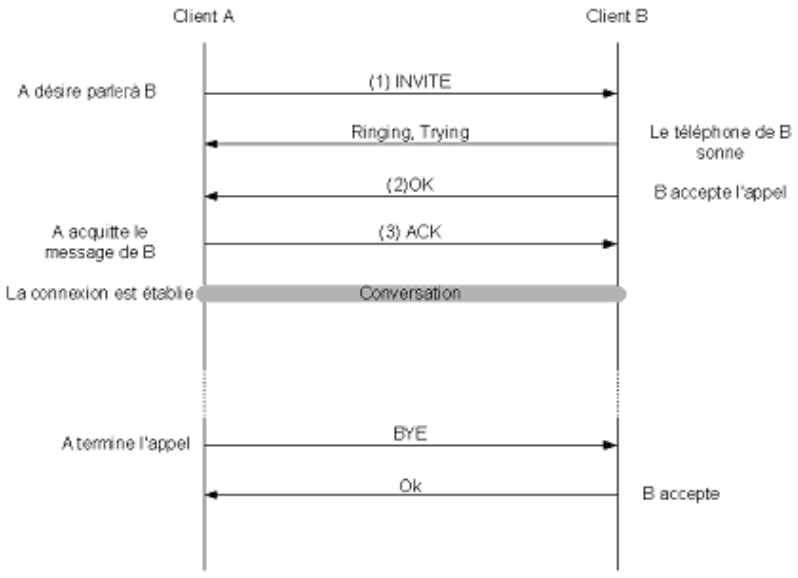


Figure 7-7 - Etablissement et libération de la connexion SIP

Pour le préprocesseur, une session est associée à l'un des états inactive, attente d'une ACK (*wait_ack*) et active.

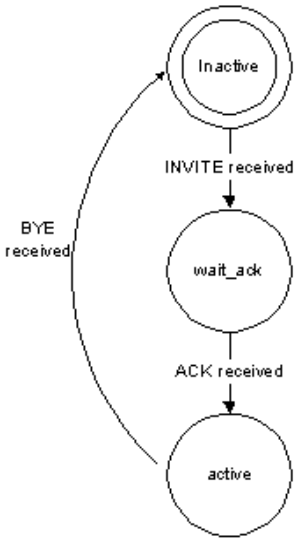


Figure 7-8 - Diagramme d'état de la session

Une session est inactive au départ. Une session inactive est une session qui n'est pas encore établie ou pas complètement. Quand le proxy reçoit une requête INVITE – n'oubliez pas les paquets passent d'abord par l'IPS avant d'arriver au proxy - l'état de la session passe à *wait_ack* (attente d'une ACK). Une entrée est créée pour cette nouvelle session dans la table. La connexion passe à l'état active quand l'ACK pour la précédente INVITE est reçue. Elle passe à inactive quand la requête BYE de cette conversation est reçue.

Mise à jour de la table:

Le préprocesseur met à jour les entrées de la table en suivant l'algorithme décrit ci-dessous (cf. Figure 7-9 et Figure 7-10). Des commentaires ont été ajoutés afin d'aider le lecteur à mieux comprendre ce qui se passe.

Pour chaque paquet, le préprocesseur parcourt la table afin de déterminer si une entrée existe déjà pour cette session.

- L'entrée existe:

Si elle existe – une requête INVITE a été reçue – le préprocesseur met à jour les différentes informations concernant cette session suivant la nature du message reçu.

*Dans le cas d'un **ACK***, il vérifie d'abord si cette session se trouve vraiment à l'état *wait_ack*. Si oui, il actualise l'état à active, sinon un log est généré car c'est un message hors contexte. L'action drop est éventuellement effectuée. Comme l'état de la session est maintenant active, le proxy gère donc une session de plus. Par conséquent, le préprocesseur actualise également la variable qui compte les sessions simultanées gérées par le proxy.

*Dans le cas d'un **BYE***, il vérifie aussi si la session se trouve à l'état active. Si c'est le cas, il actualise l'état à inactive et décrémente le nombre des sessions simultanées. Dans le cas contraire, un log est généré, car il est étrange de recevoir un message BYE qui n'est rattaché à aucune INVITE, donc à aucune session.

- L'entrée n'existe pas:

Dans ce cas, le préprocesseur vérifie si le message reçu est une requête INVITE ou REGISTER. En effet, une entrée inexistante signifie que la session est inactive. Si la session doit être créée une requête INVITE est nécessaire.

Ainsi, si le message reçu est effectivement une **INVITE**, une nouvelle entrée sera créée. Mais avant, il faut assurer que le proxy peut encore gérer cette nouvelle session. S'il ne peut pas – donc si le compteur de session simultanées dépasse la limite définie par l'utilisateur - un log est généré et si l'action drop est activée, le paquet est rejeté. Dans le cas contraire, la nouvelle entrée est créée avec les paramètres From, To et Call-ID de l'appel. La session passe à l'état *wait_ack*.

Si une requête **REGISTER** est reçue, aucune entrée ne sera créée, car cette requête n'a pas de lien direct dans la procédure d'établissement de la session (cf. Figure 7-8).

Si une requête autre que celles citées ci-dessus est reçue, un log est généré.

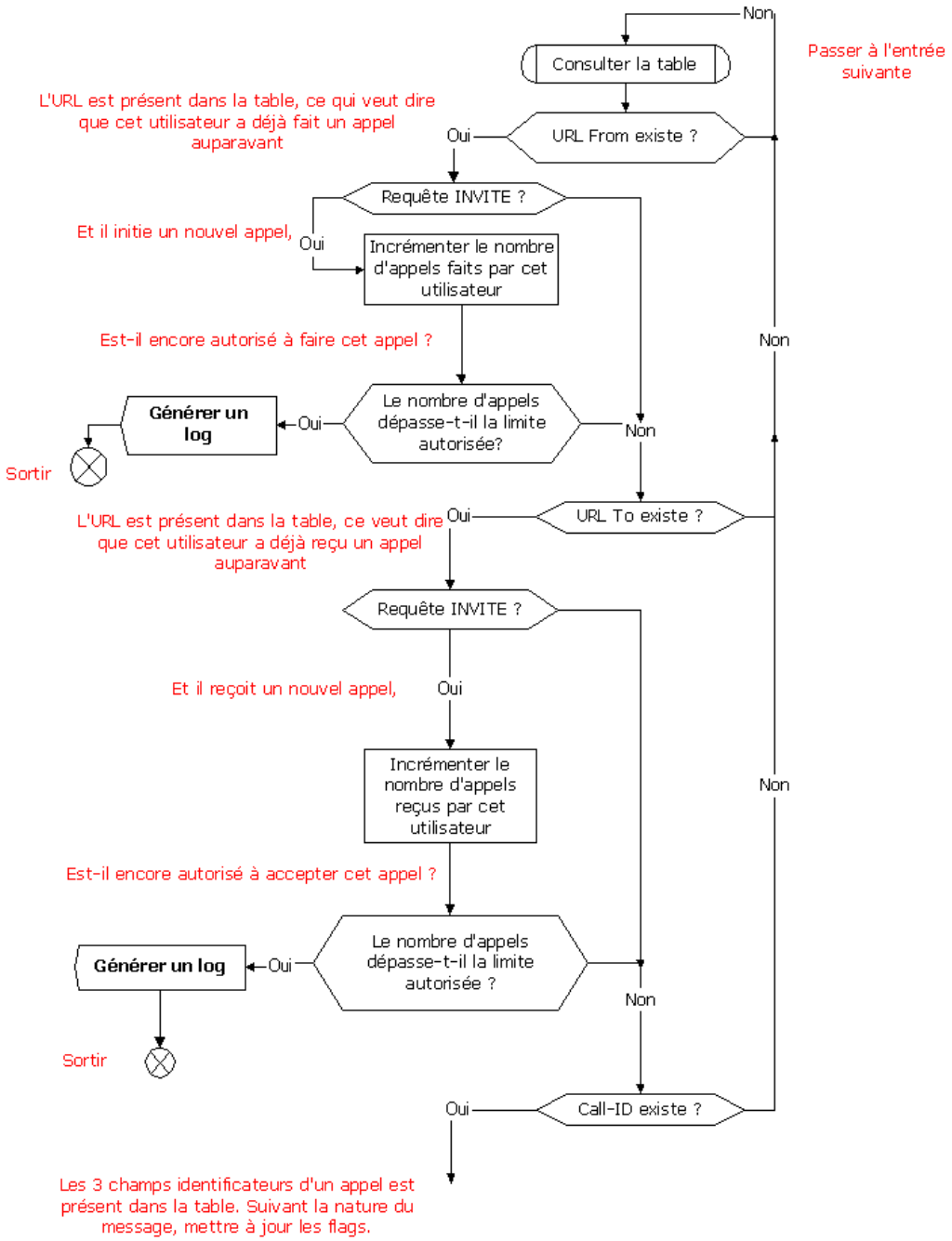


Figure 7-9 - Algorithme de mise à jour de la table de la session

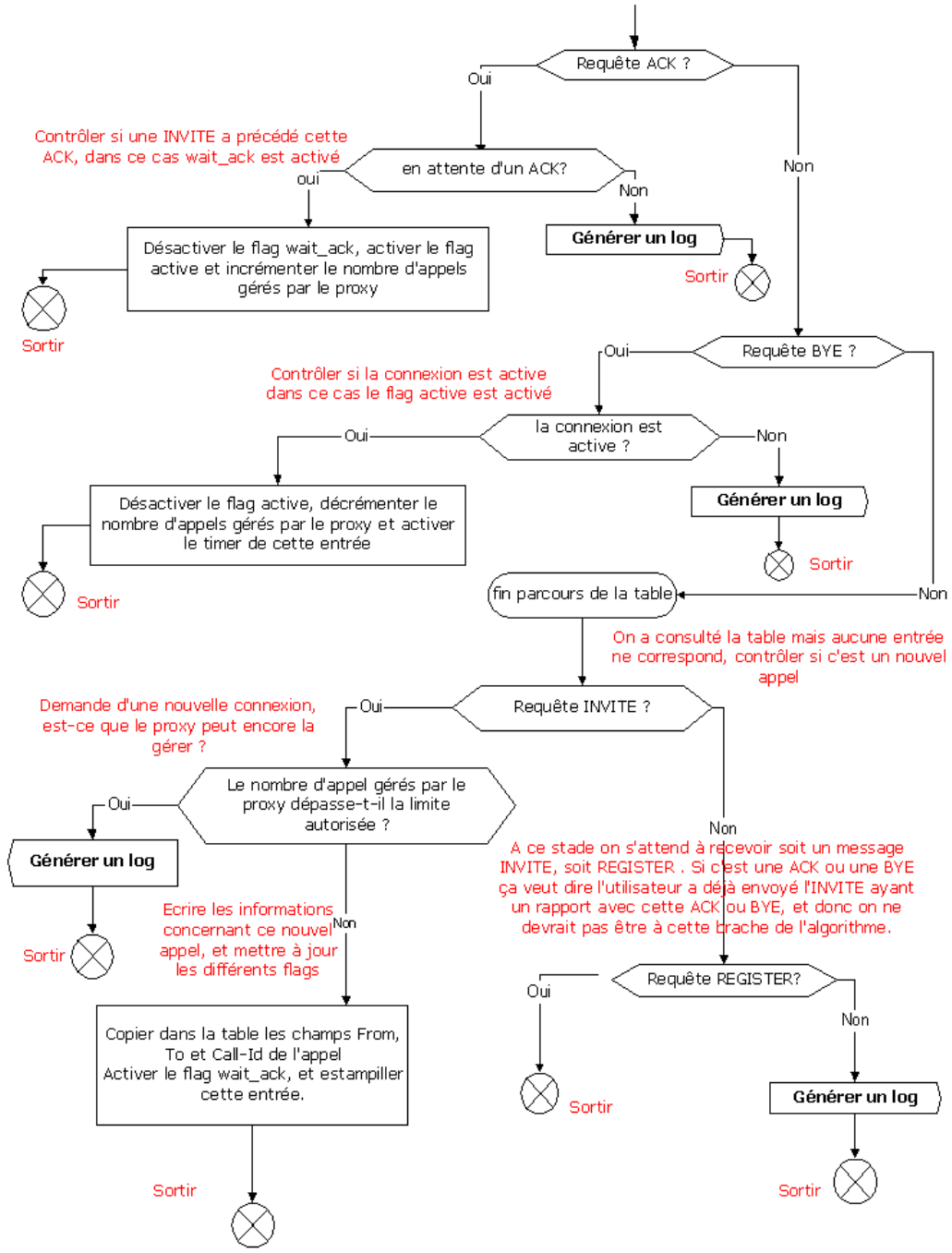


Figure 7-10 - Algorithme de mise à jour de la table de la session (suite)

7.6 Test de l'IPS SIP

7.6.1 Configuration de test

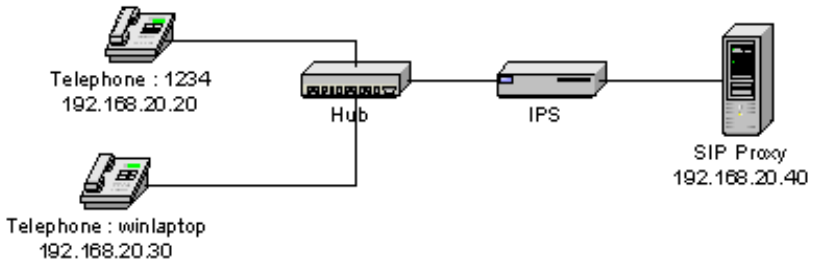


Figure 7-11 - Plateforme de test

Les clients SIP sont connectés à une interface de l'IPS à travers le hub, et le proxy à l'autre interface. L'IPS est situé entre les clients et le proxy, et joue en fait l'IPS le rôle de pont entre eux. La machine qui héberge l'IPS est configurée en bridge. De cette façon, tous les messages à destination du proxy passeront d'abord par l'IPS qui va les examiner et si tout va bien – c'est-à-dire si les paquets ont été jugés non hostiles – il va les acheminer au proxy.

Configuration de l'IPS

Le prototype de système de prévention d'intrusion réalisé lors de ce travail est une extension de la version 2.2.0 de Snort Inline. En effet, on a ajouté à ce dernier un préprocesseur pour qu'il puisse détecter des attaques contre les applicatifs SIP.

L'IPS est déployé en *mode bridge* pour les raisons déjà évoquées à la section 7.3, c'est-à-dire pour que l'IPS puisse être intégré dans le réseau sans toucher à son l'architecture actuelle. Et en ce mode l'IPS reste furtif et est donc mieux protégé.

L'IPS tourne sous Linux. Un script a été écrit pour faciliter son lancement (cf. Annexe Scripts). Voici les paramètres de configuration de l'IPS utilisés lors de ce test:

```
preprocessor sipDecode: ports 5060, action-drop, limit 10, calls 3,
period 600, proxy 192.168.20.40
```

Cette ligne nous dit que SIP utilise le port par défaut 5060. Le détecteur est configuré en mode DROP c'est-à-dire qu'il rejette les paquets malveillants. Le proxy peut gérer, au maximum, 10 appels simultanés. Bien-sûr, l'utilisateur du détecteur est libre de définir ce nombre suivant la capacité de son proxy et de son réseau VoIP.

Un utilisateur est autorisé à recevoir ou passer 3 appels au maximum dans une période de 10 mn (600s). Cela fait en moyenne 3mn par appel, ce qui représente une valeur choisie arbitrairement. Enfin, l'adresse IP du proxy est 192.168.20.40. Ceci permet à l'IPS d'identifier les paquets venant du proxy. A partir de maintenant, tout paquet à destination de cette adresse sera inspecté par l'IPS.

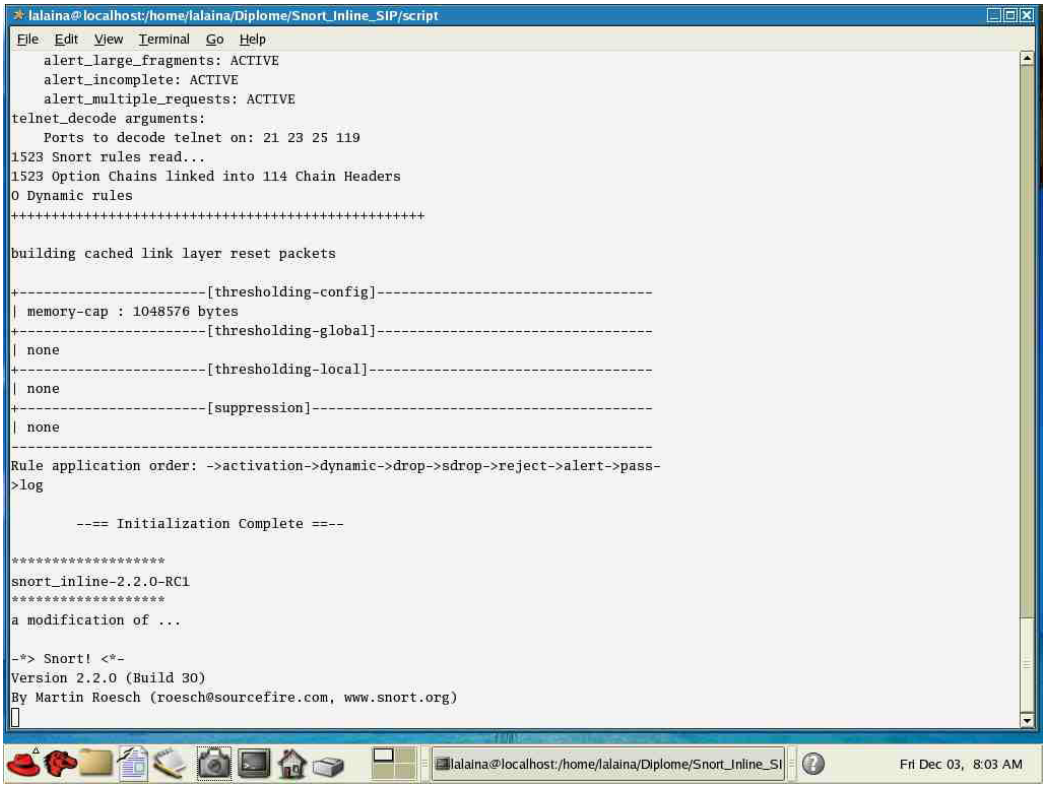


Figure 7-12 - Snort Inline

7.6.2 Les mesures effectuées

Quelques attaques sont simulées pour démontrer les fonctionnalités de l'IPS. Mais avant les simulations, une vérification qui montre que l'ajout de l'IPS dans la topologie réseau ne perturbe pas le fonctionnement de l'application est effectuée.

7.6.2.1 Appel légitime

Voici une illustration de ce qui se passe. L'appelant 2345 adresse sa requête INVITE au proxy. Le paquet passe d'abord par l'IPS. Une fois l'inspection terminée, la requête est transmise au proxy.

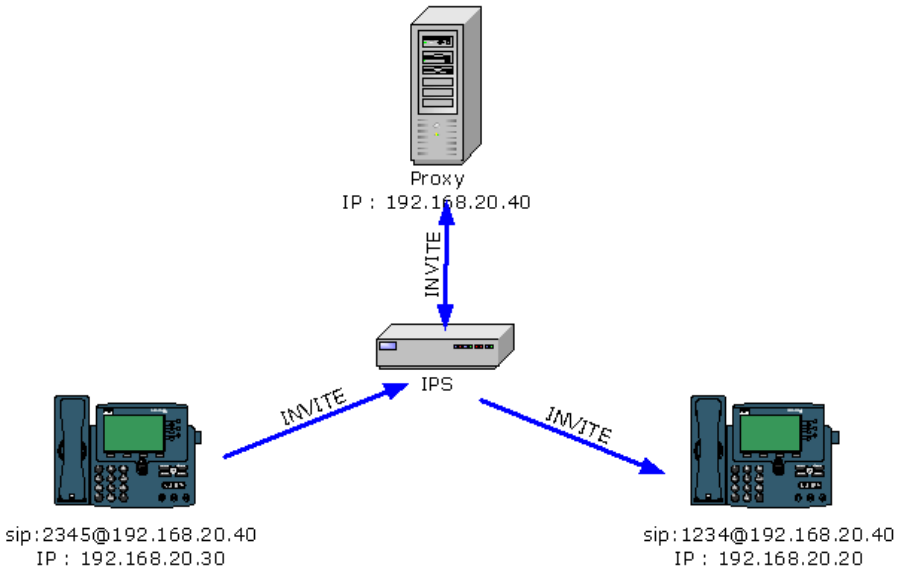


Figure 7-13 - Le client 2345 appelle 1234

Et voici une capture des messages envoyés par le client 2345:

No.	Time	Source	Destination	Protocol	Info
1	10:56:49	192.168.20.30	192.168.20.40	SIP	Request: REGISTER sip:192.168.20.40
2	10:56:49	192.168.20.40	192.168.20.30	SIP	Status: 200 OK (1 bindings)
3	10:57:03	192.168.20.30	192.168.20.40	SIP/SDP	Request: INVITE sip:1234@192.168.20.40, with ses
4	10:57:03	192.168.20.40	192.168.20.30	SIP	Status: 100 Trying
5	10:57:03	192.168.20.40	192.168.20.30	SIP	Status: 180 Ringing
6	10:57:13	192.168.20.40	192.168.20.30	SIP/SDP	Status: 200 Ok, with session description
7	10:57:13	192.168.20.30	192.168.20.40	SIP	Request: ACK sip:1234@192.168.20.20:5060
8	10:57:19	192.168.20.40	192.168.20.30	SIP	Request: BYE sip:2345@192.168.20.30:5060
9	10:57:19	192.168.20.30	192.168.20.40	SIP	Status: 200 Ok

```
▷ Frame 3 (755 bytes on wire, 755 bytes captured)
▷ Ethernet II, Src: 00:c0:9f:17:a5:a4, Dst: 00:02:b3:95:af:10
▷ Internet Protocol, Src Addr: 192.168.20.30 (192.168.20.30), Dst Addr: 192.168.20.40 (192.168.20.40)
▷ User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)
▽ Session Initiation Protocol
  ▷ Request-Line: INVITE sip:1234@192.168.20.40 SIP/2.0
  ▽ Message Header
      Via: SIP/2.0/UDP 192.168.20.30:5060;rport;branch=z9hG4bKF8B9384D1418451F9D744BE4D93C1A4B
    ▷ From: winLaptop <sip:2345@192.168.20.40>;tag=2856522600
    ▷ To: <sip:1234@192.168.20.40>
      Contact: <sip:2345@192.168.20.30:5060>
      Call-ID: F6B60427-FAFA-438F-98CB-B606433B256E@192.168.20.30
      CSeq: 7490 INVITE
      Max-Forwards: 70
      Content-Type: application/sdp
      User-Agent: X-Lite release 1103m
      Content-Length: 271
  ▽ Message body
    ▷ Session Description Protocol
```

Figure 7-14 - Appel initié par 192.168.20.30

Et ceux reçus par le proxy:

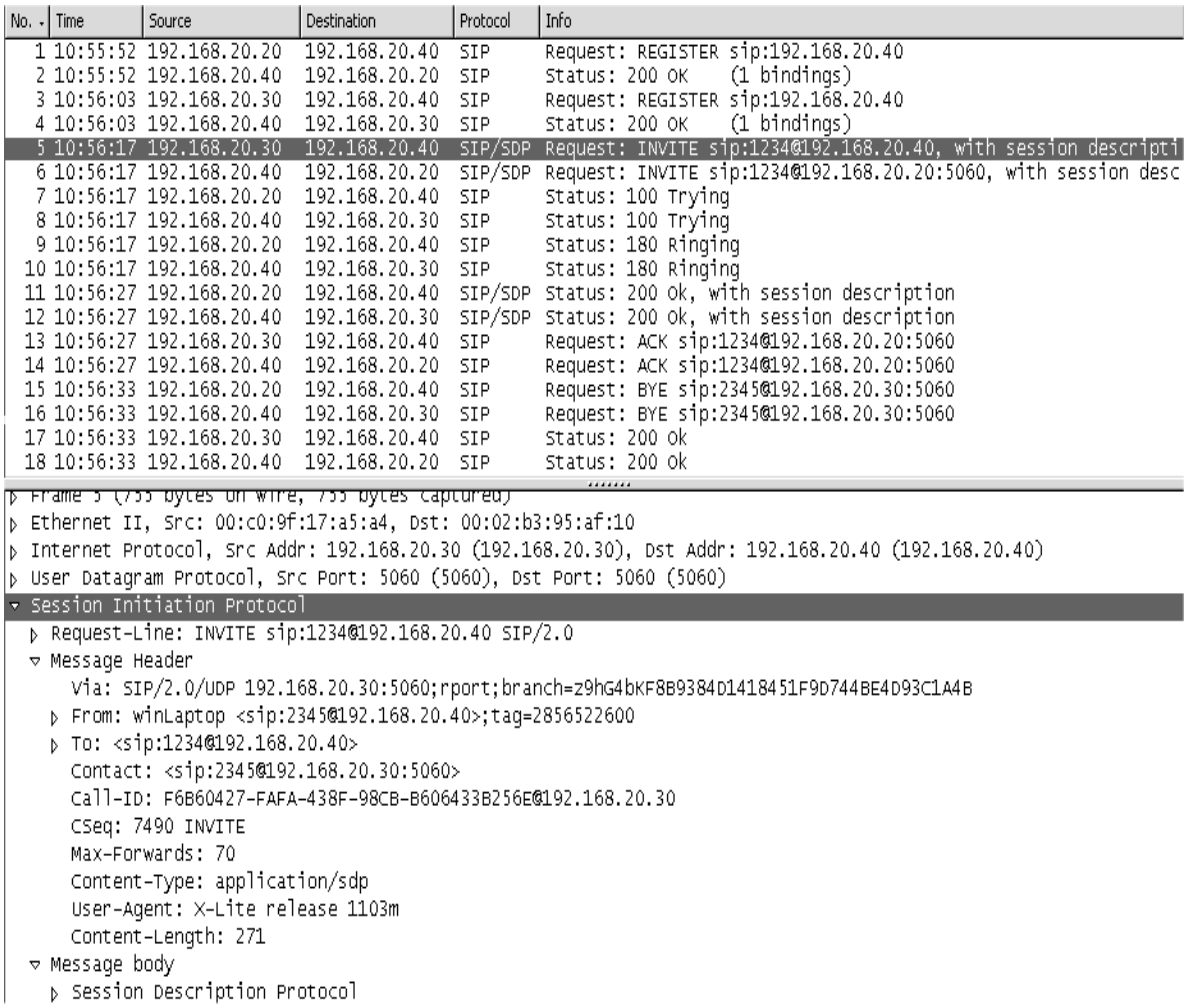


Figure 7-15 - Messages reçus et relayés par le proxy

Les deux clients ont pu établir la session sans aucun problème. Le comportement du proxy est tout à fait inchangé. L'IPS travaille de manière complètement transparente et comme il n'y avait pas de paquet malveillant lors de cette communication, tous les messages sont parvenus à destination.

7.6.2.2 Attaques spam et déni de service par INVITE flooding

La première attaque simulée est une attaque de spam. C'est la même démarche pour une attaque par inondation de requêtes INVITE. Un client se fait appeler plusieurs fois dans un court délai. Ces appels peuvent provenir d'une seule ou de plusieurs personnes, ce qui ne change rien dans l'interprétation de l'événement par l'IPS.

D'après les configurations du détecteur, un utilisateur ne doit pas recevoir plus que 3 appels en 10mn. Pourtant, entre `10:57:03` et `10:57:58`,

l'utilisateur `sip:2345@192.168.20.40` a appelé `sip:1234@192.168.20.40` plus que 3 fois.

Voici ces messages (notez le dernier paquet ayant comme entête `Cseq: 3961 INVITE`):

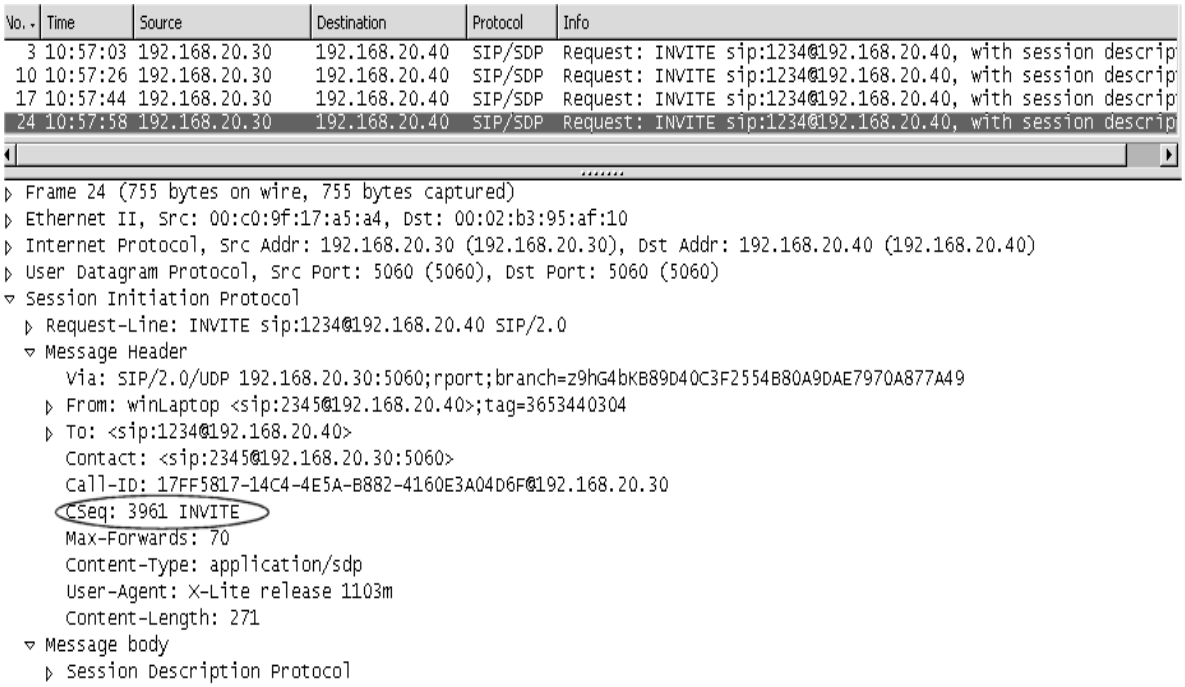


Figure 7-16 - Appels faits par 192.168.20.30

Et voici les paquets reçus par le proxy:

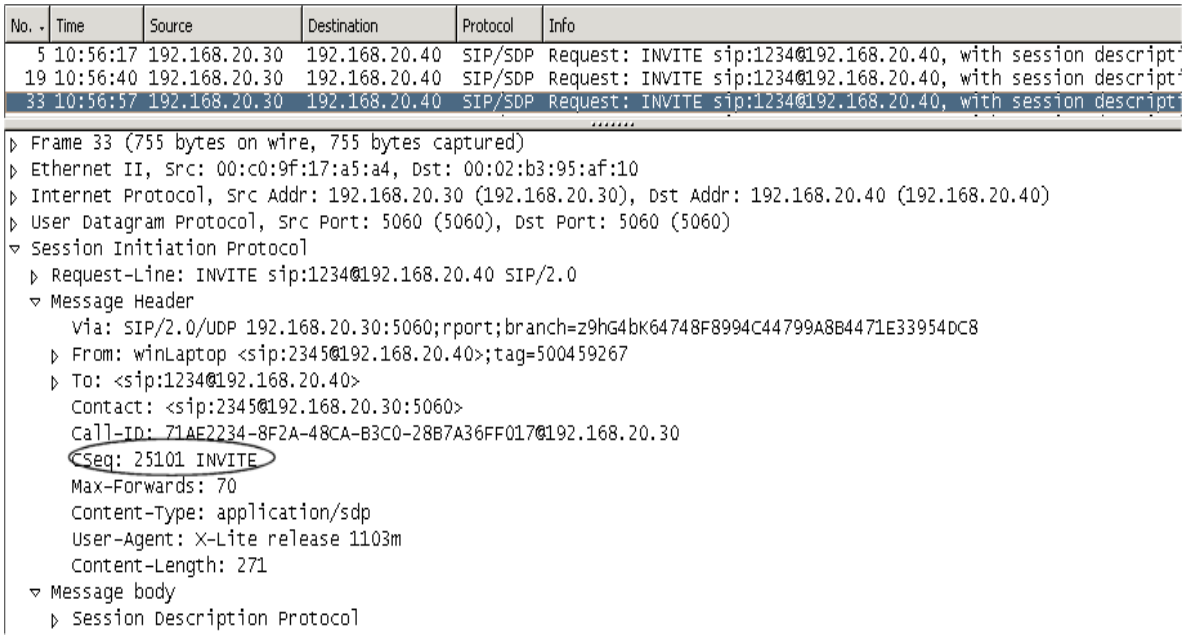


Figure 7-17 - Appels reçus par le proxy

On constate que non seulement le proxy n'a reçu que 3 requêtes INVITE, mais qu'en plus le dernier paquet reçu possède un entête `Cseq:25101 INVITE`.

Mais que c'est-il passé?

En fait, la 4ème requête INVITE est détectée par l'IPS et considérée comme étant une situation qui enfreint la règle «pas plus de 3 appels en 10mn», et donc à rejeter. Il a également généré un log dont voici un extrait:

```
[**] (spp_sip) To many calls from a caller, possible spam [**]
11/26-10:57:54.892109 192.168.20.30:5060 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13244 IpLen:20 DgmLen:741
Len: 713
49 4E 56 49 54 45 20 73 69 70 3A 31 32 33 34 40  INVITE sip:1234@
31 39 32 2E 31 36 38 2E 32 30 2E 34 30 20 53 49  192.168.20.40 SI
50 2F 32 2E 30 0D 0A 56 69 61 3A 20 53 49 50 2F  P/2.0..Via: SIP/
32 2E 30 2F 55 44 50 20 31 39 32 2E 31 36 38 2E  2.0/UDP 192.168.
32 30 2E 33 30 3A 35 30 36 30 3B 72 70 6F 72 74  20.30:5060;rport
3B 62 72 61 6E 63 68 3D 7A 39 68 47 34 62 4B 42  ;branch=z9hG4bKB
38 39 44 34 30 43 33 46 32 35 35 34 42 38 30 41  89D40C3F2554B80A
39 44 41 45 37 39 37 30 41 38 37 37 41 34 39 0D  9DAE7970A877A49.
0A 46 72 6F 6D 3A 20 77 69 6E 4C 61 70 74 6F 70  .From: winLaptop
20 3C 73 69 70 3A 32 33 34 35 40 31 39 32 2E 31   <sip:2345@192.1
36 38 2E 32 30 2E 34 30 3E 3B 74 61 67 3D 33 36  68.20.40>;tag=36
35 33 34 34 30 33 30 34 0D 0A 54 6F 3A 20 3C 73  53440304..To: <s
69 70 3A 31 32 33 34 40 31 39 32 2E 31 36 38 2E  ip:1234@192.168.
32 30 2E 34 30 3E 0D 0A 43 6F 6E 74 61 63 74 3A  20.40>..Contact:
20 3C 73 69 70 3A 32 33 34 35 40 31 39 32 2E 31   <sip:2345@192.1
36 38 2E 32 30 2E 33 30 3A 35 30 36 30 3E 0D 0A  68.20.30:5060>..
43 61 6C 6C 2D 49 44 3A 20 31 37 46 46 35 38 31  Call-ID: 17FF581
37 2D 31 34 43 34 2D 34 45 35 41 2D 42 38 38 32  7-14C4-4E5A-B882
2D 34 31 36 30 45 33 41 30 34 44 36 46 40 31 39  -4160E3A04D6F@19
32 2E 31 36 38 2E 32 30 2E 33 30 0D 0A 43 53 65  2.168.20.30..CSe
71 3A 20 33 39 36 31 20 49 4E 56 49 54 45 0D 0A  q: 3961 INVITE..
4D 61 78 2D 46 6F 72 77 61 72 64 73 3A 20 37 30  Max-Forwards: 70
```

Table 7-5 - Log généré par Snort Inline

L'IPS a bien fait son travail, il a rejeté le bon paquet (cf entête Cseq).

7.6.2.3 Envoi de paquets malformés

Pour la suite, des requêtes malformées ont été adressées au proxy.

Voici par exemple une requête INVITE sans l'entête To adressée au proxy.

```
Time     Source          Destination    Protocol   Info
10:59:14 192.168.20.30  192.168.20.40  SIP        Request: INVITE
sip:1234@192.168.20.40

Session Initiation Protocol
    Request-Line: INVITE sip:1234@192.168.20.40 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 192.168.20.30;branch=z9hG4bK776asdhds
        From: root <sip:root@192.168.20.30>;tag=1928301774
        Call-ID: a84b4c76e66710@192.168.20.30
        CSeq: 123456 INVITE
        Contact: <sip:root@192.168.20.30>
        Max_forwards: 70
        User Agent: SIVuS Scanner
        Content-Type: application/sdp
        Subject: SiVuS Test
        Expires: 7200
        Content-Length: 0
```

Table 7-6 - Message SIP sans l'entête To

Cette requête ne parvient pas jusqu'au proxy car l'IPS l'a détectée et voici le log qu'il a généré:

```
[**] [123:6:1] (spp_sip) security check failed: To Header missing [**]
11/26-10:59:10.884104 192.168.20.30:3518 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13265 IpLen:20 DgmLen:403
Len: 375
```

Table 7-7 - Log généré par Snort Inline

C'est un cas où l'entête n'est pas présent. Que ce passe-t-il si l'entête est bien présent, mais la valeur qu'il contient n'est pas conforme?

Voici un type de message avec un entête To comportant un étrange URI:

```
Time     Source          Destination     Protocol Info
11:00:27 192.168.20.30   192.168.20.40   SIP      Request: INVITE
sip:1234@192.168.20.40

Session Initiation Protocol
    Request-Line: INVITE sip:1234@192.168.20.40 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 192.168.20.30;branch=z9hG4bK776asdhds
        From: root <sip:root@192.168.20.30>;tag=1928301774
        To: 1234@192.168.20.40
        Call-ID: a84b4c76e66710@192.168.20.30
        CSeq: 123456 INVITE
        Contact: <sip:root@192.168.20.30>
        Max_forwards: 70
        User Agent: SIVuS Scanner
        Content-Type: application/sdp
        Subject: SiVuS Test
        Expires: 7200
        Content-Length: 0
```

Table 7-8 - Message SIP ayant un entête To malformé

Cette requête est également adressée au proxy, qui ne l'a jamais reçue. Voici le log généré par l'IPS:

```
[**] [123:14:1] (spp_sip) process packet failed: malformed packet :
parsing error [**]
11/26-11:00:24.583664 192.168.20.30:3524 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13271 IpLen:20 DgmLen:427
Len: 399
```

Table 7-9 - Log généré par Snort Inline

Dans ce cas-ci aussi, l'IPS a bien détecté que le paquet est malformé. Notez la différence de message de log. Ils sont différents car il s'agit bien de deux événements de natures distinctes.

Le paquet suivant est un autre cas de malformation de paquet. En effet, le nom de la méthode contenu dans la ligne de requête n'est pas le même que celui indiqué dans l'entête Cseq.

```
Time     Source          Destination     Protocol Info
10:59:53 192.168.20.30   192.168.20.40   SIP      Request: INVITE
sip:1234@192.168.20.40

Session Initiation Protocol
    Request-Line: INVITE sip:1234@192.168.20.40 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 192.168.20.30;branch=z9hG4bK776asdhds
        From: root <sip:root@192.168.20.30>;tag=1928301774
        To: 1234 <sip:1234@192.168.20.40>
        Call-ID: a84b4c76e66710@192.168.20.30
        CSeq: 123456 ACK
        Contact: <sip:root@192.168.20.30>
        Max_forwards: 70
        User Agent: SIVuS Scanner
        Content-Type: application/sdp
        Subject: SiVuS Test
        Expires: 7200
        Content-Length: 0
```

Table 7-10 - Message SIP avec un Cseq non conforme

Ce paquet non plus n'est pas arrivé à sa destination (le proxy), car l'IPS l'a détecté comme corrompu et l'a donc rejeté. Voici le log généré par l'IPS:

```
[**] (spp_sip) security check failed: Cseq Header missing or corrupted
[**]
11/26-10:59:49.970360 192.168.20.30:3522 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13269 IpLen:20 DgmLen:435
Len: 407
```

Table 7-11 - Log généré par Snort Inline

Le test suivant montre un paquet apparemment normal. C'est une apparence trompeuse car si on observe bien l'entête Cseq – même s'il comporte bien la partie numéro de séquence:`123456` et l'autre partie nom de la méthode: `INVITE` – sa construction n'est pas conforme.

```
Time     Source         Destination      Protocol Info
11:00:02 192.168.20.30  192.168.20.40    SIP      Request: INVITE
sip:1234@192.168.20.40
Session Initiation Protocol
    Request-Line: INVITE sip:1234@192.168.20.40 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 192.168.20.30;branch=z9hG4bK776asdhds
        From: root <sip:root@192.168.20.30>;tag=1928301774
        To: 1234 <sip:1234@192.168.20.40>
        Call-ID: a84b4c76e66710@192.168.20.30
        CSeq: 123456INVITE
        Contact: <sip:root@192.168.20.30>
        Max_forwards: 70
        User Agent: SIVuS Scanner
        Content-Type: application/sdp
        Subject: SiVuS Test
        Expires: 7200
        Content-Length: 0
```

Table 7-12 - Message SIP comportant un entête Cseq malformé

Ce paquet non plus n'arrive à bon port. En effet, l'entête Cseq devrait être plutôt `Cseq:123456 INVITE`. Notez l'espace entre les deux parties. L'IPS a fait son travail. Voici le log qu'il a généré:

```
[**] [123:14:1] (spp_sip) process packet failed: malformed packet :
parsing error [**]
11/26-10:59:58.963442 192.168.20.30:3523 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13270 IpLen:20 DgmLen:437
Len: 409
```

Table 7-13 - Log généré par Snort Inline

On a également procédé à des tests du même genre que ceux présentés ci-dessus sur d'autres entêtes obligatoires du message SIP. Toutefois, tous les messages envoyés et les logs générés par l'IPS ont été omis par souci de clarté du rapport, pour ne pas trop surcharger le lecteur de messages de log. Le test s'est bien déroulé, l'IPS a bien détecté et logué les événements, et en plus, les paquets malformés ont été rejetés.

Remarque: le comportement du proxy sans le détecteur n'est pas le même qu'avec. En effet, chaque proxy a sa façon d'implémenter la RFC. Certains ont implémenté ce qui est optionnel, et d'autres non. L'OnDo SIP server, par exemple, essaie – mais n'y parvient pas – de traiter un paquet qui n'a pas d'entête Via, alors que le SCS SIP proxy répond tout de suite par un «`603 decline`».

7.6.2.4 Envoi de messages hors contexte

Le test suivant montre l'importance de la détection à effet mémoire (*stateful detection*).

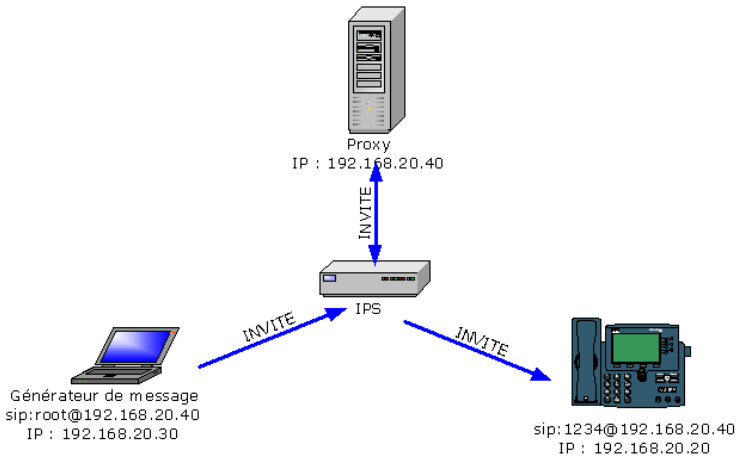


Figure 7-18 - Configuration pour l'expérience d'une détection *stateful*

Notez l'arrivée de l'utilisateur `sip:root@192.168.20.40`. Jusqu'à maintenant, ce dernier n'a fait aucun appel, et n'a pas d'appel en attente non plus. Pourtant, il adresse au proxy une requête BYE, qui signifie une terminaison de session.

On peut voir ce message sur la capture suivante:

```
No. | Time     | Source          | Destination     | Protocol | Info
 50 11:01:03 192.168.20.30 192.168.20.40 SIP Request: BYE sip:1234@192.168.20.40
 51 11:01:11 192.168.20.30 192.168.20.40 SIP Request: INVITE sip:1234@192.168.20.40
 52 11:01:11 192.168.20.40 192.168.20.30 SIP Status: 100 Trying
 53 11:01:11 192.168.20.40 192.168.20.30 SIP Status: 180 Ringing
 54 11:01:17 192.168.20.40 192.168.20.30 SIP/SDP Status: 200 Ok, with session description
 55 11:01:19 192.168.20.40 192.168.20.30 SIP/SDP Status: 200 Ok, with session description
 56 11:01:22 192.168.20.40 192.168.20.30 SIP/SDP Status: 200 Ok, with session description
 57 11:01:28 192.168.20.40 192.168.20.30 SIP/SDP Status: 200 Ok, with session description
 58 11:01:57 192.168.20.30 192.168.20.40 SIP Request: BYE sip:1234@192.168.20.40

▷ Frame 50 (446 bytes on wire, 446 bytes captured)
▷ Ethernet II, Src: 00:c0:9f:17:a5:a4, Dst: 00:02:b3:95:af:10
▷ Internet Protocol, Src Addr: 192.168.20.30 (192.168.20.30), Dst Addr: 192.168.20.40 (192.168.20.40)
▷ User Datagram Protocol, Src Port: 3525 (3525), Dst Port: 5060 (5060)
▽ Session Initiation Protocol
  ▷ Request-Line: BYE sip:1234@192.168.20.40 SIP/2.0
  ▽ Message Header
      Via: SIP/2.0/UDP 192.168.20.30;branch=z9hG4bK776asdhds
    ▷ From: root <sip:root@192.168.20.30>;tag=1928301774
    ▷ To: 1234 <sip:1234@192.168.20.40>
      Call-ID: a84b4c76e66710@192.168.20.30
      CSeq: 123456 BYE
      Contact: <sip:root@192.168.20.30>
      Max_forwards: 70
      User Agent: SIVuS Scanner
      Content-Type: application/sdp
      Subject: SIVuS Test
      Expires: 7200
      Content-Length: 0
```

Figure 7-19 - Appel fait par l'utilisateur "root"

Avouez que c'est plutôt étrange comme comportement. Pour l'IPS, cet événement déclenche la règle «une BYE est toujours rattachée à une INVITE, c'est-à-dire à une session établie». Comme il n'a pas réussi à trouver une session correspondante, il a généré un log.

```
[**] [123:13:1] (spp_sip) update table failed: INVITE request expected
[**]
11/26-11:00:59.991825 192.168.20.30:3525 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13285 IpLen:20 DgmLen:432
Len: 404
```

Table 7-14 - Log généré par Snort Inline

En observant la capture, on voit que l'utilisateur `root` a cette fois envoyé une INVITE (cf. capture Figure 7-19 paquet n°51) avant la requête BYE (cf. capture Figure 7-19 paquet n°58). Pourtant l'IPS génère un log:

```
[**] [123:12:1] (spp_sip) update table failed: BYE found with no
INVITE [**]
11/26-11:01:54.655586 192.168.20.30:3528 -> 192.168.20.40:5060
UDP TTL:128 TOS:0x0 ID:13287 IpLen:20 DgmLen:432
Len: 404
```

Table 7-15 - Log généré par Snort Inline

Pourquoi, vous demandez-vous?

Une session se trouve dans l'un des différents états définis à la section 7.5.2.4 (cf. Figure 7-8). L'utilisateur a envoyé l'INVITE, la session passe à l'état wait_ack. Mais si on observe attentivement la capture de messages (Figure 7-19), on ne voit aucun message ACK. Ceci veut dire que les 3 phases d'établissement de la session ne sont pas encore complètement effectuées et que cette session n'est pas encore «active». C'est la raison pour laquelle l'IPS trouve que le message BYE est hors contexte.

7.6.2.5 Détection de signature

Les tests effectués et présentés ci-dessus démontrent la fonctionnalité de l'IPS. Il est utile de rappeler que c'est le préprocesseur SIP qui a scruté chaque paquet à la recherche d'un signe d'intrusion. Pour cela, il a fait différents analyses et contrôles (cf. Figure 7-5). Les attaques ont été bien détectées et arrêtées.

Avec Snort, l'utilisateur a la possibilité d'écrire ses propres règles, c'est-à-dire de définir une signature de trafic. Ces règles seront appliquées au trafic par le moteur de détection.

Les règles Snort sont divisées en deux sections logiques: l'entête de la règle et les options de la règle. L'entête de règle contient comme informations l'action de la règle, le protocole, les adresses IP source et destination, les masques réseau, et les ports source et destination. La section options de la règle contient les messages d'alerte et les informations sur les parties du paquet qui doivent être inspectées pour déterminer si l'action de la règle doit être acceptée.

Cette fonctionnalité est maintenue dans la version de l'IPS proposée ici. Pour la tester, quelques règles pour les trafics SIP sont écrites.

```
# ************************************************************
# SIP RULES
# ************************************************************
drop udp $EXTERNAL_NET any -> $INTERNAL_NET $SIP_PORTS (msg:"Non Via
header SIP message"; content: "|53 49 50|";content: !"|56 69
61|";nocase;)
drop udp $EXTERNAL_NET any -> $INTERNAL_NET $SIP_PORTS (msg:"Non From
header SIP message";content: "|53 49 50|";content:!"|46 72 6F
6D|";nocase;)
drop udp $EXTERNAL_NET any -> $INTERNAL_NET $SIP_PORTS (msg:"Non To
header SIP message";content: "|53 49 50|";content:!"|54 6F
3A|";nocase;)
drop udp $EXTERNAL_NET any -> $INTERNAL_NET $SIP_PORTS (msg:"Non Call-
ID field SIP message";content: "|53 49 50|";content:!"|43 61 6C 6C 2D
49 44|";nocase;)
drop udp $EXTERNAL_NET any -> $INTERNAL_NET $SIP_PORTS(msg:"Non Cseq
field SIP message";content: "|53 49 50|";content:!"|43 53 65
71|";nocase;)
```

Table 7-16 - Règles Snort pour le protocole SIP

Ces règles disent à l'IPS de rejeter le paquet qui correspond aux critères mentionnés. Comme Snort ne gère que les protocoles TCP, UDP et ICMP, on ne peut pas écrire explicitement SIP comme protocole auquel s'applique la règle. Mais rappelez-vous: SIP utilise le protocole de transport UDP.

La règle s'applique à tous les paquets venant de l'extérieur depuis n'importe quel port à destination du réseau interne, port `$SIP_PORT`. L'extérieur est défini par la plage d'adresses `$EXTERNAL_NET` et l'intérieur par celle de `$INTERNAL_NET`.

Dans les options un message décrivant la règle déclenchée est enregistré avec le paquet. Tandis que `content` est utilisé pour rechercher un contenu spécifique par correspondance de motifs dans la charge du paquet. Dans ce cas-ci, on recherche d'abord la présence du mot SIP, ensuite la non présence d'un champ d'entête spécifique tel que Via, From, To, Call-ID ou Cseq. Ces derniers sont exprimés en format binaire.

Notez que les variables `$EXTERNAL_NET`, `$INTERNAL_NET` et `$SIP_PORTS` ont été prédéfinies dans le fichier de configuration.

Un message SIP est envoyé par l'utilisateur root au proxy:

```
INVITE sip:laptop@192.168.20.40 SIP/2.0
Via: SIP/2.0/UDP 192.168.20.30;branch=z9hG4bK776asdhds
From: root <sip:root@192.168.20.30>;tag=1928301774
Call-ID: a84b4c76e66710@192.168.20.30
CSeq: 1 INVITE
Contact: <sip:root@192.168.20.30>
Max_forwards: 70
User Agent: SIVuS Scanner
Content-Type: application/sdp
Subject: SiVuS Test
Expires: 7200
Content-Length: 0
```

Table 7-17 - Message SIP sans l'entête To

Comme c'est un message qui ne comporte pas d'entête To, une règle est déclenchée. Un log est généré et le paquet est rejeté.

```
11/12-10:35:45.028609  [**] [1:0:0] Non To header SIP message [**]
[Priority: 0] {UDP} 192.168.20.30:3090 -> 192.168.20.40:5060
```

Table 7-18 - Log généré par l'IPS

Même si cette méthode est assez efficace, créer une signature qui cible au mieux le trafic n'est pas simple et donc le risque des faux positifs augmente considérablement. En plus, toutes les attaques possibles contre un système de la VoIP ne peuvent pas être écrites en règles Snort. Créer une signature pour détecter une attaque de déni de service par requête BYE (cf 6.1.1.2) en est un exemple. Notez en plus que les options relatives au contenu sont les plus gourmandes en ressources et donc dans la mesure du possible, il est recommender d'éviter de les employer. C'est pour cette raison que cette solution est laissée au bénéfice de la détection effectuée par le préprocesseur.

8 Conclusion

Les produits NIDS ont incontestablement aidé les entreprises à combattre les intrusions, mais avec la hausse du nombre d'attaques, les limites de l'approche passive sont devenues évidentes. Les systèmes de prévention des intrusions représentent une nouvelle technologie prometteuse pour la sécurité des réseaux. Ces systèmes peuvent automatiquement prendre des mesures pour bloquer les attaques et intrusions. Un système IPS offre un niveau de protection que ne peut pas atteindre un système NIDS.

Un prototype d'IPS destiné aux réseaux VoIP a été conçu lors de ce travail. L'IPS proposé ici est un système en ligne capable de détecter les attaques basées sur les vulnérabilités du protocole SIP et de les bloquer. Il s'intègre facilement aux environnements de gestion de la sécurité. L'IPS fonctionne avec le protocole de signalisation de la VoIP: **SIP**. De cette manière, la qualité de service est maintenue. En effet les médias qui sont livrés par le protocole RTP ne constituent pas de menaces importantes mais surchargeraient le détecteur.

La simulation de différents scénarios de test a permis de démontrer que l'IPS peut détecter et arrêter les attaques de spam et le déni de service par INVITE flooding contre un proxy SIP. Il arrive aussi à détecter et rejeter les paquets non conformes à la RFC.

L'IPS utilise la technique de détection d'anomalie de protocole ainsi que la détection par correspondances de motifs en mode *stateful* (avec état). Il ne serait toutefois pas capable de détecter toutes les attaques contre un système de la VoIP sans compléter ces techniques par la méthode comportementale. Malheureusement par manque de temps cette dernière n'a pas pu être implémentée et constituerait une suite intéressante à ce travail.

Yverdon-les-Bains, le 17 décembre 2004

Lalaina KUHN

9 Références

[1] O.Arkin, Sys-Security Group, "Security Risk Factors with IP", 2002.

[2] Top Layer Networks, "Au-delà des systèmes de détection des intrusions, les notions fondamentales sur la prévention des intrusions sur le réseau", 2003.

[3] K. Kanellakis, "VoIP Security", 2003.

[4] Cisco Systems, "IP Telephony Security in Depth", white paper, 2003.

[5] Bureau de la Protection des Infrastructures Essentiels et de la Protection Civile, note d'information " Protection VoIP", 2003.

[6] K.Steinklauber, "VoIP Security in Small Business", 2003.

[7] C.Endorf, E.Schultz, J.Mellander "Intrusion detection & Prevention, 2004.

[8] R. Bace, P.Mell "Intrusion detection Systems", NIST Special Publication.

[9] P.Lindstrom, "Intrusion Prevention Systems: next generation Firewall", A Spire Research Report, 2004.

[10] E. Ahlm Vigilar INC, "Is Intrusion Prevention Changing Information Security", 2004.

[11] R.Panko, "Sécurité des systèmes d'information et des réseaux", Pearson 2004.

[12] McAfee Security, "Host and Network Intrusion Prevention: competitors or Partners? ", White Paper, 2004.

[13] NSS Group, "Intrusion Prevention System", 2004.

[14] T.Holland, "Understanding IPS and IDS", 2004.

[15] Pingtel and CheckPoint software technologie, "Secure IP Telephony For The Enterprise", 2004.

[16] Defense Information Systems Agency (DISA), "VoIP Security Technical Implentation", 2004.

[17] N.Fischbach, Sécurité.org, "Sécurité de la voix sur IP".

[18] Tipping Point, "Intrusion Prevention: The Future of VoIP Security".

[19] C.H.Abdul Razak, "A new Challenge", 2004.

[20] Qovia Inc, "Network Intrusion QoS impact within VoIP", 2004.

[21] Nortel Network, "Security for service Provider VoIP Networks".

[22] J.Rosenberg, "The SIP and spam", draft-rosenberg-sipping-spamm-00, 2004.

[23] O.Arkin, @stake, "The Next Generation of Phreaking ".

[24] N.Desai, "IPS : The Next Step in the Evolution of IDS".

[25] J.Koziol, "Snort 2", 2003.

[26] RFC 3261 : Session Initiation Protocol

Annexe A. Description des logiciels utilisés lors du test

- Les clients SIP sont des softphones qui tournent sous MacOSX et MS Windows XP pro SP2,

- L'IPS tourne sur une machine Linux Red Hat 9 (kernel 2.4-27) avec deux interfaces, et enfin

- Le proxy SIP tourne sous MS Windows XP pro SP2.

Les softphones

Xlite (v.2.0) un est softphone gratuit du constructeur Xten (www.xten.com). Il fonctionne sous MacOSX et MS Windows. Il supporte le protocole SIP.

Figure 1 - softphone Xlite

SJphone™ est un softphone gratuit développé par SJ Labs (www.sjlabs.com) et fonctionne sous MS Windows. Il permet de communiquer, en utilisant l'Internet, avec tous les téléphones conventionnels, portables ou d'autres softphones. Il supporte les protocoles SIP et H.323.

Figure 2 - Softphone SJPhone

Générateur de message

En plus de ces clients softphones, des générateurs de message SIP sont aussi utilisés. Ces générateurs permettent de personnaliser et d'envoyer un message SIP. Il s'agit de SiVuS et SIPNess Messenger.

SiVuS (**S**ip **Vu**lnerability **S**canner), comme son nom l'indique, est un scanner de vulnérabilité du protocole SIP (RFC 3261). Il permet également la génération des messages SIP. Il a été développé par ***V****oice* ***o****ver* ***P****acket Security (VoPSecurity)* (www.vopsecurity.org) pour les plateformes MS Windows. SiVuS permet la génération des requêtes INVITE (avec SDP comme corps de message), REGISTER, BYE, CANCEL, ACK et OPTIONS.

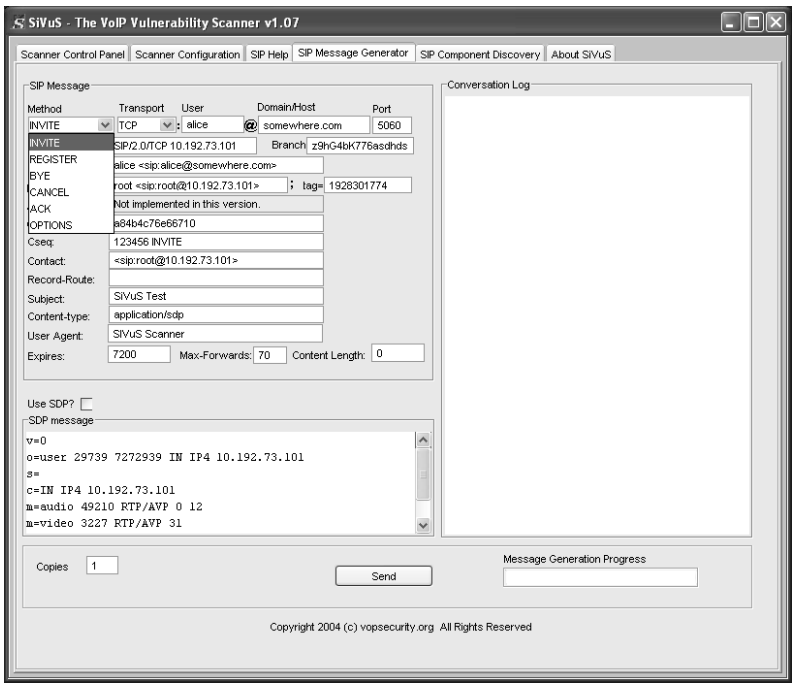


Figure 3 - Générateur de message SiVuS

Le **SIPNess™ messenger** d' *ortena Network* (www.ortena.com) est un outil qui fournit à l'utilisateur une manière facile de construire et d'envoyer des messages personnalisés SIP à un interlocuteur distant, et en même temps de recevoir et surveiller les messages SIP entrants. La version 1.06 utilisée lors de ce travail supporte les messages INVITE, Ringing, Trying, CANCEL, BYE, ACK, OK, REGISTER, OPTIONS, et Moved temporarily.

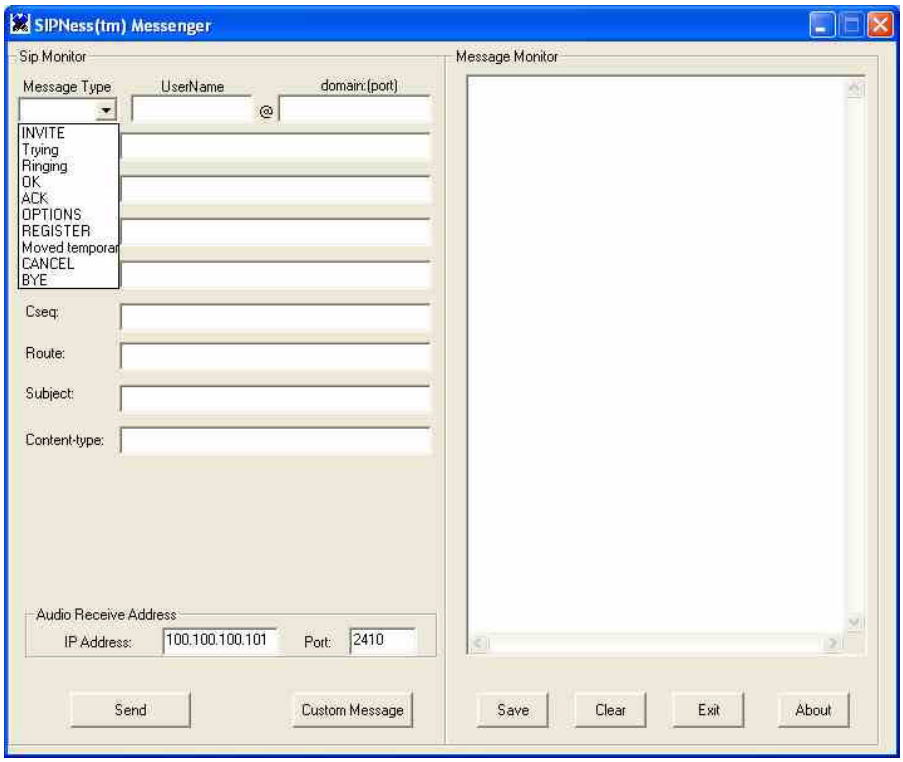


Figure 4 - Générateur de message SIPNess Messenger

Proxy SIP

En ce qui concerne le proxy, on a utilisé différents proxies SIP.

OnDO SIPServer

C'est un serveur basé sur le protocole SIP. On peut utiliser avec ce serveur des *hardphones* SIP, des *softphones* SIP, et les *gateways* SIP-PSTN (*Public Switched Telephones Network*) pour des communications VoIP.

OnDO SIPServer tourne sous MS Windows, Linux, MacOSX et Solaris. Il a été développé par Brekeke Software (www.brekeke.com) et est basé sur la technologie Java. La version 1.2 utilisée lors de ce travail offre les fonctions suivantes: *Registrar*, *Routing* et *SIP-NAT*. C'est une version enregistrée sous une licence «*Personnal use*» et donc gratuite.

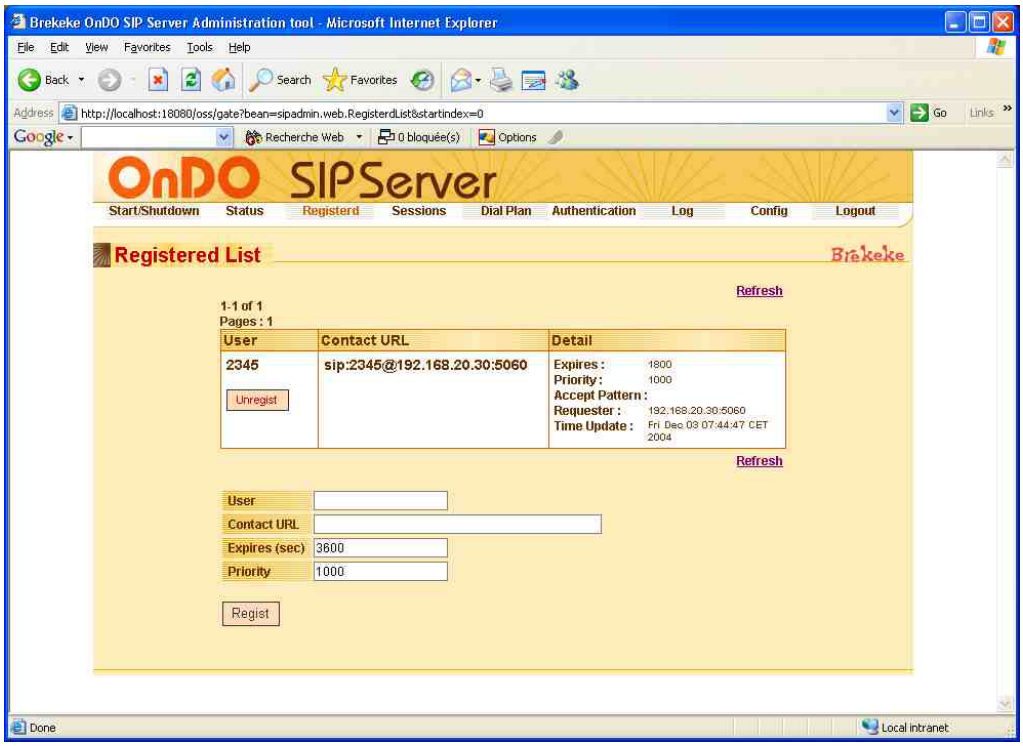


Figure 5 - OnDO SIPServer

La fonctionnalité de filtrage constitue une particularité de ce proxy. En effet, OnDO SIPServer possède ce que le constructeur appelle *Dial Plan*. C'est un moyen pour définir des règles pour le routage des appels, ou encore pour mettre en place un filtrage des sessions.

Par exemple, la règle:
```
$request=^INVITE
$addr=192\.168\.0\.1$
$action=603
```

rejette en envoyant une réponse «`603 decline`» les appels depuis l'adresse IP 192.168.0.1.

Interaction SIP Proxy

Il est compatible avec la dernière RFC 3261 pour le protocole SIP. Il peut être configuré pour fonctionner comme un *stateful proxy*, un *redirect proxy* ou un *load-balance proxy*. Il assure également des services d'enregistrement SIP (*SIP registrar*). Intercation SIP Proxy fonctionne avec les téléphones SIP et les *gateways.*

Il a été développé par *Interactive Intelligence, Inc* (www.inin.com) et tourne sous MS Windows.

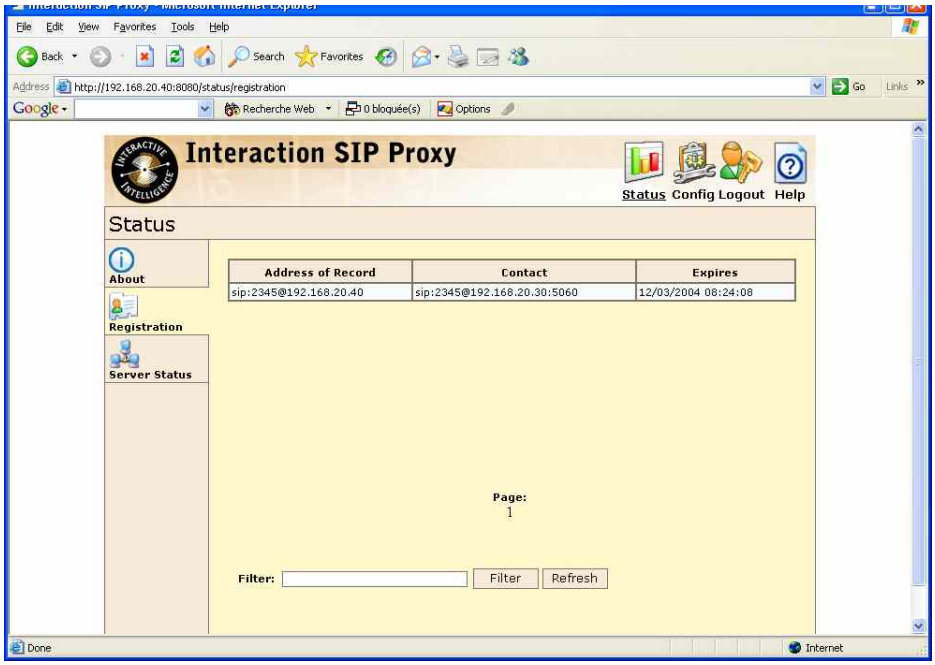


Figure 6 - Interaction SIP Proxy

SCS SIP Proxy

Ce proxy est basé sur et conforme à la RFC 3261. La version 1.0.0.3 utilisée lors de ce travail fournit les fonctionnalités suivantes: *Registrar* (max. 100 enregistrements), proxy (en mode *stateless* ou *redirect*), *Routing*, Authentification. Il a été développé par *Siemens Communication Systems* (www.mysip.ch) et tourne sous MS Windows.

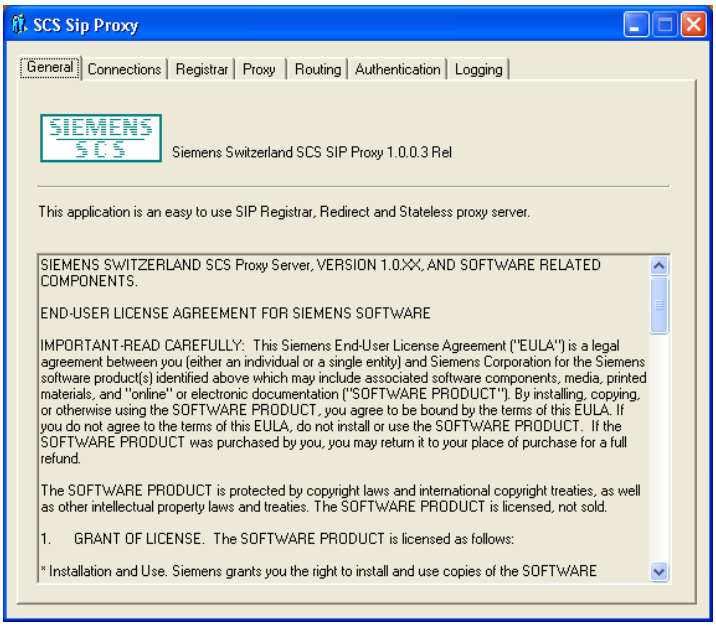


Figure 7 - SCS SIP Proxy

JAIN SIP proxy

C'est un proxy SIP développé par NIST (***N**ational **I**nstitute of **S**tandards and **T**echnology*) (http://snad.ncsl.nist.gov/proj/iptel/). Il est écrit en Java. Ce proxy possède les fonctionnalités suivantes: serveur de *location*, *Routing*, *Registrar*, et serveur d'authentification.

Figure 8 - NIST JAIN SIP Proxy

Annexe B. Installation d'un bridge firewall

Introduction

L'idée est d'ajouter à une machine Linux la fonction de *bridge,* appelée aussi un pont. Nous verrons également comment cette fonction peut être perfectionnée en *bridging firewall* (pont filtrant). Pour ajouter cette fonction à la machine Linux, il faut au minimum que cette dernière possède deux cartes réseau. Chaque carte réseau devient alors l'équivalent d'un port du *bridge*. Le *bridge* fonctionnera comme un *switch* Ethernet classique: il apprend tout seul les adresses MAC qui sont derrière ses interfaces réseau et aiguille les paquets Ethernet comme un *switch*. Par contre, contrairement à un *switch* classique, il ne croise pas la connexion réseau: il faudra donc relier le *bridge* aux autres ordinateurs par des câbles croisés, et aux autres *switches* par des câbles droits.

Ce document explique toutes les étapes pour monter un bridging firewall en utilisant **Red Hat Linux 9.0 (kernel 2.4-27)**

Installer un bridge

Il y a deux principales choses à faire. D'abord il faut installer l'utilitaire `bridge-utils`. Cet utilitaire permet la gestion et l'installation des ponts sous Linux. Vérifier s'il est déjà installé, sinon la dernière version peut être téléchargée sur la page http://bridge.sourceforge.net.

```
# rpm -q bridge-utils
package bridge-utils is not installed
# apt-get install bridge-utils
Reading Package Lists... Done
Building Dependency Tree... Done [...]
0 packages upgraded, 1 newly installed, 0 removed and 0 not upgraded.
# rpm -q bridge
bridge-utils-0.9.3-8
```

La deuxième chose à faire est la préparation du *kernel*. Il faut activer

```
Networking options --> 802.1d Ethernet Bridging
```

Ensuite recompiler le *kernel* et l'installer.

Configurer le bridge

Pour configurer un bridge qui utilise les interfaces `eth0` et `eth1`, exécuter le script fourni avec ce document.

Le STP (*Spanning Tree Protocol*) est désactivé parce que, dans ce cas-ci, il n'est pas utile, et aussi pour ne pas avoir de trafic inutile. ARP (*Address Resolution Protocol*) est également désactivé puisque les interfaces réseau n'ont pas une pile IP et parce qu'il pourrait indiquer leur identité et leur position.

Bridging firewall

Appelé également firewall transparent, il possède les avantages:

Zéro configuration. D'un point de vue de gestion de réseau, il n'y a pratiquement aucun changement. Le bridge firewall est branché en ligne avec le réseau qu'il protège. Ceci signifie qu'on peut le mettre entre deux routeurs, ou entre un routeur et un *switch*. On peut également le mettre devant une simple machine. Il achemine simplement les trames entre les interfaces après les avoir inspectées. C'est-à-dire qu'il n'y a aucun changement à faire dans le réseau existant. Il est complètement transparent. Plus de prise de tête ou de mise à jour de configuration de sous-réseau n'est nécessaire.

Performance. Puisque c'est un dispositif simple, il y a moins de maintenance.

Caché. Caractéristique principale de ce dispositif. Il fonctionne à la couche 2 du modèle OSI, donc les interfaces réseaux n'ont aucune adresse IP. Sans adresse IP, ce dispositif est inaccessible et invisible au monde extérieur.

Il existe deux possibilités pour réaliser un bridge firewall: avec `ebtables` ou avec `iptables`. Avec `ebtables`, on se concentre plus sur la couche OSI 2 ou 3, tandis qu'avec `iptables`, c'est plutôt sur les couches OSI 3 et 4.

`Ebtables` est plus approprié dans le scénario où le pont relie deux segments de réseau d'un même sous-réseau. `Iptables` est plus approprié dans le cas où le pont est placé avant un routeur qui relie un sous-réseau à un autre.

Installer un bridge firewall

D'abord on a besoin d'un pont. La précédente section présente comment l'installer et le configurer. Deuxièmement il faut installer le paquetage `ebtables`.

La prochaine étape est d'ajouter l'outil de filtrage `bridge-nf` au *kernel*.

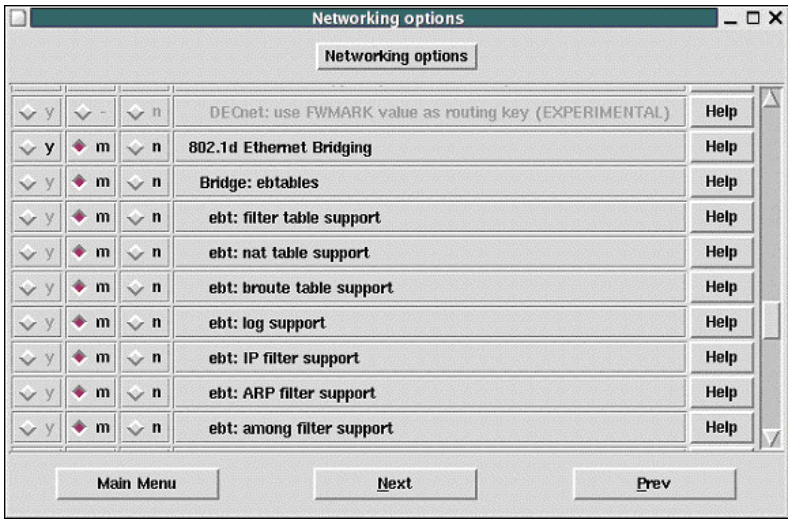
`Ebtables` et `bridge-nf` sont inclus dans le *kernel* 2.6 standards. Il y a un patch disponible pour les *kernels* 2.4 stables. Si on veut installer un bridge firewall en utilisant une version standard d'un *kernel* 2.4.x, il est impératif de télécharger un patch approprié à la version du *kernel*.

`Ebtables` et le patch `bridge-nf` sont disponibles sous: http://ebtables.sourceforge.net/.

Appliquer le patch:

```
# gzip -d ebtables-brnf-6_vs_2.4.27.diff.gz
# cd /usr/src/linux-2.4.27
# patch -p1 < /ebtables-brnf-6_vs_2.4.27.diff
```

En choisissant les options du *kernel*, dans `Networking options`, sélectionner comme module: `802.1d Ethernet bridging`, `Bridge: ebtables`, ainsi que toutes les options d'`ebt`.



Après avoir sauvegardé les options du *kernel*, le compiler et l'installer.

```
make dep ; make bzImage ; make modules ; make modules_install
```

La machine Linux est maintenant prête pour faire du *bridging firewall*. On peut utiliser soit `iptables`, soit `ebtables` pour écrire le firewall. Utiliser le script fourni avec ce document pour mettre en place le firewall.

Annexe C. Installation de Snort Inline 2.2.0

Installation sous Red Hat 9.0 (kernel 2.4-27)

Décompresser la source fournie avec ce document et se placer dans le nouveau répertoire.

```
tar -xzf snort-inline-2.2.0.tar-gz
cd snort-inline-2.2.0
```

Configurer ensuite Snort Inline.

```
./configure
```

Suivre les étapes de compilation et d'installation classique.

```
make; make install
```

L'application Snort Inline est maintenant installée.

Créer un répertoire pour les règles et les fichiers de configuration de Snort Inline.

```
mkdir /etc/snort_inline
```

Placer les fichiers appropriés dans le répertoire correspondant. Exécuter les commandes suivantes à partir du répertoire qui héberge le code source de Snort Inline.

```
cd etc
cp * /etc/snort_inline
```

Télécharger ensuite un système de règles récent à l'adresse: http://www.snort.org/dl/rules/

Extraire les fichiers de règles:

```
tar -xzf snortrules-snapshot-2_2.tar.gz
```

Se déplacer dans le répertoire des règles et le placer dans le répertoire `/etc/snort_inline`

```
mv rules /etc/snort_inline
```

Le script `convert.sh` qui se trouve dans le répertoire `/etc/snort_inline` convertit les règles de snort en règles snort_inline. Copier ce fichier dans le répertoire des règles, lui donner les permissions nécessaires et l'exécuter.

```
cp convert.sh rules/
chmod +rx rules/convert.sh
cd rules
./convert.sh
```

Configuration de snort_inline.conf

On configure les paramètres Snort_inline dans son principal fichier de configuration: `snort_inline.conf`. On définit les plages d'adresses IP à surveiller, les préprocesseurs à activer, les modules de sortie à employer et les règles à exploiter. Les variables qui doivent être changées sont les suivantes:

var INTERNAL_NET cette variable indique la plage d'adresses IP internes que Snort Inline doit surveiller à la recherche d'intrusion. On peut spécifier l'un des quatre différents types de plages d'adresses IP.

- 192.168.20.40/24
- ou au format `CIDR` [192.168.20.0/24, 10.0.0.1/24]
- `$interface_ADDRESS` (ex: `eth1_ADDRESS`). Cette méthode ne fonctionne pas si l'interface est en mode furtif.
- `any`, pour surveiller toutes les adresses IP internes.

var EXTERNAL_NET cette variable est l'équivalent pour les adresses externes de la variable `INTERNAL_NET`. Elle supporte les quatre même options que `INTERNAL_NET`.

var RULE_PATH

Cette variable indique à Snort Inline l'emplacement des fichiers de règles. La positionner sur le répertoire qui contient les règles. Dans ce cas-ci:

```
var RULE_PATH /etc/snort_inline/rules
```

L'application Snort Inline est maintenant prête à être exécutée. Pour la lancer, un script a été écrit: `Start_Snort_Inline.sh`

Annexe D. Scripts

Script pour configurer le bridge: bridge.sh

```
#! /bin/sh

# Script pour mettre en place le bridge
# Auteur: L.KUHN , Nov 2004

echo "starting up bridging ..."

# Mise en service du bridge
brctl addbr br0

# Desactiver le Spanning Tree
brctl stp br0 off

# Attacher des interfaces physiques
brctl addif br0 eth0
brctl addif br0 eth1

# Retirer les adresses IP
ifconfig eth0 down
ifconfig eth1 down
ifconfig eth2 down
ifconfig eth0 0.0.0.0 up -arp
ifconfig eth1 0.0.0.0 up -arp

# Affecter une adresse IP a l'interface logique
ifconfig br0 0.0.0.0 up

# Afficher les informations concernant le bridge
echo ""
echo "Bridge's configuration"
echo ""
brctl show
```

Script pour mettre en place le firewall: firewall.sh

```
#! /bin/sh

# Author: L.KUHN, Nov 2004

# Clear old rules
iptables -F

# Insert kernel mode
modprobe ip_queue

# Check to see if it worked
lsmod | grep ip_queue

# Queue packet
iptables -A FORWARD -j QUEUE

# Display table's content
echo ""
echo "iptables's content"
echo ""
iptables -L -v
```

Scripts pour démarrer Snort Inline:

Start-snort_inline.sh

```
#!/bin/bash

# Original Work Of
# Created Honeynet Project <project@honeynet.org>
# March 18, 2000
#
# Modified By
# Shomiron Das Gupta <devilscrow@sify.com>
# LAk-IPS Version 0.2
# Last Modified: July 14, 2003
#
# Modified By
# L.Kuhn, Nov 2004
#
#
PATH=/bin:/usr/local/bin

echo ""
echo ""
echo " Running Snort_Inline startup script...."
echo ""

PID=/var/run/snort_inline.pid
echo " Location of snort_inline.pid : /var/run/snort_inline.pid"

DIR=/var/log/snort
echo " Location of logs : /var/log/snort"

DATE=`date +%b_%d`

SNORT=/usr/local/bin/snort_inline

USER=snort

### Kill snort
if [ -s $PID ]; then
  PRO=`cat $PID`
  echo ""
  echo "Previous version of snort_inline running"
  echo "Killing snort_inline, PID $PRO"
  echo ""
  kill -9 $PRO
fi

# Make directory based on date, if already exists do nothing.
if [ -d $DIR/$DATE ]; then
        :
else
        mkdir $DIR/$DATE

fi
echo ""
echo ""
echo " The snort_inline.conf contains the configuration for running"
echo " snort_inline. It contains the variables for the Internal"
echo " and the external network, similarly it contains the path to"
```

```
echo " the rules directory."
echo ""
echo -n " Did you set var $RULES_PATH [yes] : "
read rules_opt

case "$rules_opt" in
   "no" )
   echo " "
   echo   "   Sorry,   snort_inline   will   not   run   properly   till
snort_inline.conf"
   echo " is configured properly. Please edit the configuration file
and "
   echo " run ./snort_inline.sh again."
   exit 0
   ;;

   * )
   echo ""
   ;;
esac

# Snort options explanation
# -b log packets in tcpdump format
# -c configuration file
# -d log packet details
# -D daemon mode
# -l log directory
# -i interface in our case eth0, this option is required when using
#    the -Q option.
# -Q (used ONLY with Snort-Inline for QUEUE mode)
# -u $USER run snort as UID $USER in our case nobody

### Start snort for the Honeynet
$SNORT  -Dd  -c  /etc/snort_inline/snort_inline.conf  -Q  -i  eth0  -l
$DIR/$DATE

echo " "
echo " tail /var/log/messages to check if snort_inline"
echo " has been successfully initialised"
exit
```

Annexe E. Code source du préprocesseur SIP

```
/* $Id: spp_sipDecode.c, 2004/11/04 l.kuhn $ */
/* Snort Preprocessor Plugin Source File */

/* spp_sipDecode.c
 *
 * Purpose: deals with the messages sip.  Check if all required
 * headers within each request and answer messages are present.
 * Keep track connection in a table.
 *
 * Arguments: This plugin takes a list of integers representing the
 * ports that the user is used in having sip. Integer limit of
 * simultaneous calls handle by the proxy. Integer indicate
 * limit of calls made or received by a user within a period.
 * Integer indicate this period.Action to be taken in case of
 * malicious packet. Proxy's IP adress.
 *
 * Effect: generate log and plus dropped if option activated
 *
 * Comments:
 *
 */************************-INCLUDE-*********************************/

#include <sys/types.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <osipparser2/osip_parser.h>

#include "decode.h" /* Packet */
#include "plugbase.h" /* RegisterPreprocesor */
#include "parser.h" /* file_name, file_line */
#include "debug.h" /* DebugMessage, DEBUG_WRAP */
#include "util.h" /* FatalError */
#include "mstring.h" /* mSplit, mSplitfree*/
#include "snort.h" /* pv.homenet */
#include "detect.h"
#include "generators.h" /* Sip preprocessor IDs*/
#include "event_wrapper.h" /* GenerateSnortEvent*/
#ifdef GIDS
#include "inline.h" /* InlineDrop */
#endif

/************************-DEFINE-**********************************/

#define CALLIDNUM_SIZE             64
#define CALLIDHOST_SIZE            32
#define BUFFERSIZE                65535
#define STS_FALSE                   0
```

```
#define STS_TRUE                    1
#define STS_SUCCESS                 0
#define STS_FAILURE                 1
#define CNXMAP_SIZE                40 /* connection table size*/

static char SipDecodePorts[65536/8];
static int counter = 0; /* number of simultaneous calls
                         * handle by the proxy*/
static int nbFrom = 0; /* number of calls from this caller
                        * within a period */
static int nbTo = 0; /* number of calls received by this callee
                      * within a perdiode*/

/*************************-DATA-STRUC -*****************************/
/* table to hold the client connection */
struct cnxmap_s {
  int enable;  /* record's state */
  int active; /* connection established*/
  int wait_ack; /* connection wait for a ACK*/
  time_t expires; /* keep this entry how long? */
  /* maintain from, to and call-ID, 'coz this 3 fields form the
   * call leg */
  osip_uri_t *from;
  osip_uri_t *to;
  osip_call_id_t *callid;
};

/* The config struct */
struct SipDecodeConfig{
  /* calls accepted from / to a user within a period */
  int calls;
  int limit; /* number of simultaneous connection*/
  int period;/* observation time */
  char drop; /* actions */
  char proxy_ip[16]; /* SIP proxy's IP address*/
} sipcnf;

/* connection mapping table */
static struct cnxmap_s cnxmap[CNXMAP_SIZE];

/*************************-PROTOTYPES-*****************************/
void SipDecodeInit(u_char *);
void ParseSipDecodeArgs(char *);
void PreprocSipDecode(Packet *);
void SetSipPorts(char *);
void update_table (osip_message_t *, Packet *, int , int , int);
int security_check_sip(osip_message_t *, Packet *);
int compare_url(osip_uri_t *, osip_uri_t *);
int compare_callid(osip_call_id_t *, osip_call_id_t *);

/*****************************************************************/
 * Function: SipDecodeInit(u_char *)
 * Purpose: Calls the argument parsing function, performs final setup
 * on data structs, links the preproc function into the function
 * list.
 * Arguments: args => ptr to argument string
 * Returns: void function
***** ************************************************************/

void SipDecodeInit(u_char *args)
{
```

```
  /* parse the argument list into config */
  ParseSipDecodeArgs(args);

  /* initialize the connection mapping table */
  int i;
  for (i=0; i<CNXMAP_SIZE; i++)
  {
    cnxmap[i].enable = STS_FALSE;
    cnxmap[i].active = STS_FALSE;
    cnxmap[i].wait_ack = STS_FALSE;
    cnxmap[i].from = NULL;
    cnxmap[i].to = NULL;
    cnxmap[i].callid = NULL;
    cnxmap[i].expires = time(NULL);
  }

  /* set the preprocessor function into the function list */
  AddFuncToPreprocList(PreprocSipDecode);

  DEBUG_WRAP(DebugMessage(DEBUG_SIP,"spp_SipDecode Initialized\n"));
}
/*************************************************************************
 * Function: ParseSipDecodeArgs(char *)
 *
 * Purpose: reads the options.
 *
 * Arguments: pointer to string with options
 *
 * Returns: void function
*************************************************************************/

void ParseSipDecodeArgs(char *args)
{
  char **toks;
  int num_toks;
  int i = 0;
  char *index;
  int ports_done = 0;

  char **limittoks;
  int num_limittoks = 0;
  char *limit;

  char **callstoks;
  int num_callstoks = 0;
  char *calls;

  char **periodtoks;
  int num_periodtoks = 0;
  char *period;

  char **proxyiptoks;
  int num_proxyiptoks=0;
  char *ipProxy;

  /* set the default values */
#ifdef GIDS
  sipcnf.drop = 0;
#endif /* GIDS */
  sipcnf.limit = 20;
  sipcnf.calls = 5;
```

```
  sipcnf.period= 600;
  strcpy(sipcnf.proxy_ip,"0.0.0.0");

  /* if no args, load the default config, else process the args */
  if (args != NULL)
  {
     toks = mSplit(args, ",", 11, &num_toks, 0);

     for(i = 0; i < num_toks; i++)
     {
        index = toks[i];
        while(isspace((int)*index)) index++;
        if(!strncasecmp(index, "ports", 5))
        {
           SetSipPorts(index+6);
           ports_done = 1;
        } /* if token = ports*/

#ifdef GIDS
        else if(!strncasecmp(index, "action-drop", 11))
        {
           sipcnf.drop = 1;
        } /* if token = action-drop*/
#endif /* GIDS */
        else if (!strncasecmp(index,"proxy",5))
        {
           /* Get the argument for the option */
           proxyiptoks = mSplit(index, " ", 1, &num_proxyiptoks,0);
           ipProxy = proxyiptoks[1];
           /* Copy it to the sipcnf */
           if (strcpy(sipcnf.proxy_ip,ipProxy) == NULL)
               FatalError("%s(%d) =>Unknown argument to SipDecode "
                       "preprocessor: \"%s\"\n",
                       file_name, file_line, proxyiptoks[0]);
        } /* if token = proxy */

        else if(!strncasecmp(index, "limit", 5))
        {
            /* get the argument for the option */
            limittoks = mSplit(index, " ", 1, &num_limittoks, 0);

            /* copy it to the sipcnf */
            limit = limittoks[1];

            if(isdigit((int)limit[0])) sipcnf.limit = atoi(limit);
            else
            {
               FatalError(" %s(%d) => Unknown argument to SipDecode "
                          "preprocessor: \"%s\"\n",
                           file_name, file_line, limittoks[0]);
            }
            mSplitFree(&limittoks, num_limittoks);
        } /* if token = limit */

        else if(!strncasecmp(index, "calls", 5))
        {
           /* get the argument for the option */
           callstoks = mSplit(index, " ", 1, &num_callstoks, 0);

           /* copy it to the sipcnf */
           calls = callstoks[1];
```

```
            if(isdigit((int)calls[0])) sipcnf.calls = atoi(calls);
            else
            {
               FatalError(" %s(%d) => Unknown argument to SipDecode "
                          "preprocessor: \"%s\"\n",
                          file_name, file_line, callstoks[0]);
            }
            mSplitFree(&callstoks, num_callstoks);
         } /* if token = calls */

         else if(!strncasecmp(index, "period", 6))
         {
            /* get the argument for the option */
            periodtoks = mSplit(index, " ", 1, &num_periodtoks, 0);

            /* copy it to the sipcnf */
            period = periodtoks[1];
            if(isdigit((int)period[0])) sipcnf.period = atoi(period);
            else
            {

                FatalError(" %s(%d) => Unknown argument to SipDecode "
                          "preprocessor: \"%s\"\n",
                          file_name, file_line, periodtoks[0]);
            }
            mSplitFree(&periodtoks, num_periodtoks);
         } /* if token = period */
      }/* for */

      mSplitFree(&toks, num_toks);

   }/* if not NULL*/
}
/*************************************************************************
 * Function: PreprocSipDecode(Packet *)
 *
 * Purpose: Perform the preprocessor's intended function.  This can be
 *          simple (statistics collection) or complex(IP
 *          defragmentation) as you like. Try not to destroy the
 *          performance of the whole system by trying to do too
 *          much.
 *
 * Arguments: p => pointer to the current packet data struct
 *
 * Returns: void function
 *
*************************************************************************/
 void PreprocSipDecode(Packet *p)
{
   int i = 0; /* loop */
   int sts;
   int msg_ack = STS_FALSE;
   int msg_bye = STS_FALSE;
   int msg_invite = STS_FALSE;
   int msg_reg = STS_FALSE;
   osip_message_t *sipmsg;
   u_int8_t *pkt_data;
   char Buffer [BUFFERSIZE];

   /* check the port list */
   if(!(SipDecodePorts[(p->dp/8)] & (1<<(p->dp%8)))) return;
```

```
/* init the oSIP parser */
parser_init();

/* process packet : copy data */
pkt_data = p->data;

for (i=0;i<p->dsize;i++)
{
   Buffer[i]= (char)*pkt_data;
   pkt_data += 1;
}
Buffer[p->dsize]= '\0';
/* init sip_msg */
sts=osip_message_init(&sipmsg);
sipmsg->message=NULL;

/* parse the received message, if not succeed generate log*/
sts=osip_message_parse(sipmsg, Buffer);
if (sts == -1)
{
   if (p->iph->ip_src.s_addr != inet_addr(sipcnf.proxy_ip))
   {
      GenerateSnortEvent(p,GENERATOR_SPP_SIPDECODE,
         PARSERROR,1,0,0,SIPDECODE_PARSERROR_STR);
      if (sipcnf.drop)InlineDrop();
      return;
   }
}

DEBUG_WRAP(DebugMessage(DEBUG_SIP,"SIP message parsing done \n"));

/* check for mandatory fields*/
DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Start packet integrity check
 \n"));

sts=security_check_sip(sipmsg,p);

DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Packet integrity check done,
           return %d\n",sts));

/* Stateful detection,only if passed all message integrity check*/
if (sts == STS_SUCCESS)
{
   DEBUG_WRAP(DebugMessage(DEBUG_SIP,
                       "Start SIP packet stateful detection \n"));

   if (MSG_IS_INVITE(sipmsg)) msg_invite = STS_TRUE;
   else if (MSG_IS_ACK(sipmsg)) msg_ack = STS_TRUE;
   else if (MSG_IS_BYE(sipmsg) || MSG_IS_CANCEL(sipmsg))
     msg_bye = STS_TRUE;
   else if (MSG_IS_REGISTER(sipmsg)) msg_reg = STS_TRUE;

   /* Update table only for request (ack, invite, bye) packet */
   if ((msg_invite == STS_TRUE || msg_ack == STS_TRUE ||
        msg_bye == STS_TRUE))
   {
     DEBUG_WRAP(DebugMessage(DEBUG_SIP,"with invite= %d, ack= %d,
        bye= %d\n", msg_invite, msg_ack, msg_bye));
      update_table(sipmsg, p, msg_ack, msg_bye, msg_invite);
   }
```

```
    }
}
/*************************************************************************
 * Function: SetupSipDecode()
 * Purpose: Registers the preprocessor keyword and initialization
 *          function into the preprocessor list. This is the function
 *          that gets called from InitPreprocessors() in plugbase.c.
 * Arguments: None.
 *
 * Returns: void function
 *
*************************************************************************/
void SetupSipDecode()
{
   /* link the preprocessor keyword to the init function in the
    * preproc list */
   RegisterPreprocessor("SipDecode", SipDecodeInit);
   DEBUG_WRAP(DebugMessage(DEBUG_SIP,"spp_SipDecode is setup...\n"));
}
/*************************************************************************
 * Function: SetSipPorts(char *)
 *
 * Purpose: Reads the list of port numbers from the argument string
 *          and parses them into the port list data struct
 *
 * Arguments: portlist => argument list
 *
 * Returns: void function
 *
*************************************************************************/
void SetSipPorts(char *portlist)
{
   char portstr[STD_BUF];
   char **toks;
   int is_reset = 0;
   int num_toks;
   int num;

   if(portlist == NULL || *portlist == '\0') portlist = "5060 5061";

   DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Parsing arguments\n"));

   /* tokenize the argument list */
   toks = mSplit(portlist, " ", 31, &num_toks, '\\');

   /* convert the tokens and place them into the port list */
   for(num = 0; num < num_toks; num++)
   {
      if(isdigit((int)toks[num][0]))
      {
         /* used to determine last position in string */
         char *num_p = NULL;
         long t_num;

         t_num = strtol(toks[num], &num_p, 10);

         if(*num_p != '\0')
         {
            FatalError("Port Number invalid format: %s\n", toks[num]);
```

```
         }
         else if(t_num < 0 || t_num > 65335)
         {
            FatalError("Port Number out of range: %ld\n", t_num);
         }
         /* user specified a legal port number and it should override
          * the default port list, so reset it unless already done */
         if(!is_reset)
         {
            bzero(&SipDecodePorts, sizeof(SipDecodePorts));
            portstr[0] = '\0';
            is_reset = 1;
         }

         /* mark this port as being interesting using some portscan2-
          * type voodoo and also add it to the port list string while
          * we're at it so we can later print out all the ports with a
          * single LogMessage() */
         SipDecodePorts[(t_num/8)] |= 1<<(t_num%8);
      }
      else
      {
         FatalError(" %s(%d) => Unknown argument to SipDecode "
                    "preprocessor: \"%s\"\n",
                    file_name, file_line, toks[num]);
      }
   }
   mSplitFree(&toks, num_toks);

   DEBUG_WRAP(DebugMessage(DEBUG_SIP,"SIP ports set\n"));
}
/************************************************************************
 * Function: update_table(osip_message_t *, Packet *, int , int ,
int);
 *
 * Purpose: tracking connection.
 *
 * Arguments: sip message received, request methode
 *
 * Returns: void function
 *
************************************************************************/
void update_table (osip_message_t *sipmsg, Packet *p,
                   int ack, int bye, int invite)
{
   int i;
   int j = -1;
   int g;
   osip_uri_t *url1_from, *url2_from;
   osip_uri_t *url1_to, *url2_to;
   osip_call_id_t *cid1, *cid2;
   time_t time_now;
   in_addr_t Ip ;

   /* initialization*/
   url1_from = sipmsg->from->url;
   url1_to = sipmsg->to->url;
   cid1 = sipmsg->call_id;
   time(&time_now);
   nbFrom = 1;
   nbTo = 1;
```

```
Ip = inet_addr(sipcnf.proxy_ip);

DEBUG_WRAP(DebugMessage(DEBUG_SIP,"START UPDATE TABLE...\n"));
DEBUG_WRAP(DebugMessage(DEBUG_SIP,"IP source = %s, IP proxy= %s\n",
inet_ntoa(p->iph->ip_src),(sipcnf.proxy_ip)));

/* if the packet come from proxy (means internal network)
 * don't wast time checking it once again. Check has already done
 * since proxy do relay only request between UA */

if (p->iph->ip_src.s_addr == Ip )
{
   DEBUG_WRAP(DebugMessage(DEBUG_SIP,
              "Packet from proxy, exit update table\n"));
   return;
}

/* print tables's content, print only if entry exist */
for (g=0; g<CNXMAP_SIZE;g++)
{
   if ((cnxmap != NULL) && (cnxmap[g].from != NULL) &&
       (cnxmap[g].to != NULL) && (cnxmap[g].callid != NULL) &&
       (cnxmap[g].from->username != NULL) &&
       (cnxmap[g].to->username != NULL) &&
       (cnxmap[g].callid->number != NULL))
   {
      DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Table's content\n"));
      DEBUG_WRAP(DebugMessage(DEBUG_SIP,"ENABLE %d,ACTIVE %d,
                 WAIT_ACK %d\n", cnxmap[g].enable,cnxmap[g].active,
                 cnxmap[g].wait_ack));
      DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Call_ID %s, From %s,
                 To %s\n", cnxmap[g].callid->number, cnxmap[g].
                 from->username, cnxmap[g].to->username));
   }
}

/* UA already made a call ? */

for (i=0; i<CNXMAP_SIZE; i++)
{
   /* Look at where to write, verify if the entry is still up
    * = not yet expired. If it's already expired but still enabled
    * means we have received an INVITE request but not the BYE who
    * turned off the enable's flag. In this case let's change the
    * enable flag's value, so next time we can write at this
    * entry.*/

   if (cnxmap[i].expires < time_now)
   {
      if (cnxmap[i].enable == STS_TRUE)
            cnxmap[i].enable == STS_FALSE;
      else /* enable = 0 */
      {
        if (j < 0)j=i; /* remember first hole */
        continue;
      }
   }

   url2_from = cnxmap[i].from;
   url2_to = cnxmap[i].to;
   cid2 = cnxmap[i].callid;
```

```
if ((compare_url(url1_from,url2_from) == STS_SUCCESS)||
      (compare_url(url1_from,url2_to) == STS_SUCCESS))
{
   /* this caller have already made a call earlier,
    *  how many in a PERIOD? */

    if (invite==STS_TRUE)
    {
        if((compare_url(url1_from,url2_from)==STS_SUCCESS)&&
          (cnxmap[i].expires > time_now)) nbFrom += 1;

        DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Calls made by %s are
                   %d\n",url1_from->username,nbFrom));
    }

    /* He can make this new one as well ?*/

  if ((invite == STS_TRUE) && (nbFrom > sipcnf.calls))
     {
       GenerateSnortEvent(p,GENERATOR_SPP_SIPDECODE,
       FROM_NUMBER,1,0,0,SIPDECODE_FROM_NUMBER_STR);

       if (sipcnf.drop)
       {
          InlineDrop();
          return;
       }
     }

    if ((compare_url(url1_to,url2_to)==STS_SUCCESS)||
        (compare_url(url1_to,url2_from)==STS_SUCCESS))
    {
       /* this callee have already received a call,
        * how many in a PERIOD? */
       if (invite==STS_TRUE)
       {
    if((compare_url(url1_to,url2_to)==STS_SUCCESS)&&
                  (cnxmap[i].expires > time_now)) nbTo += 1;

       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
                  "Calls received by %s are %d\n",
                   url1_to->username, nbTo));
       }

       /* He can accept this new one as well ?*/
       if ((invite == STS_TRUE) && (nbTo > sipcnf.calls))
       {
          GenerateSnortEvent(p,GENERATOR_SPP_SIPDECODE,
          TO_NUMBER,1,0,0,SIPDECODE_TO_NUMBER_STR);

          if (sipcnf.drop)
          {
             InlineDrop();
             return;
          }
       }

       if(compare_callid(cid1,cid2)==STS_SUCCESS)
       {
          /* record exists, update related information :
```

```
- ACK message, check if a INVITE exists (wait_ack is
  true).
If so increase the counter, make state active and
 make false wait_ack. If not generate event !
- BYE message, check if state is active, if so
connection terminated update expires field, decrease
counter make state inactive. if not generate event */

if (ack == STS_TRUE)
{ /* it's a ACK request */
   DEBUG_WRAP(DebugMessage(DEBUG_SIP,
   "ACK request received (cnxmap[%d].wait_ack =
     %d)\n",
   i, cnxmap[i].wait_ack));

   if (cnxmap[i].wait_ack == STS_TRUE)
   {/* invite sent wait for a ACK */
      cnxmap[i].wait_ack = STS_FALSE;
      cnxmap[i].active = STS_TRUE;
      counter += 1;

      DEBUG_WRAP(DebugMessage(DEBUG_SIP,
      "spp_sipdecode connection table updated \n"));
      return;
   }/* if wait_ack */

   else
   {/* no related INVITE */
      GenerateSnortEvent(p,GENERATOR_SPP_SIPDECODE,
      ACK,1,0,0,SIPDECODE_ACK_STR);
      if (sipcnf.drop)InlineDrop();
      return;
    }/* else wait_ack */
}/* if ack */

/* It's not a ACK request, is it a BYE request ?*/

if (bye == STS_TRUE)
{ /* it's a BYE */

   DEBUG_WRAP(DebugMessage(DEBUG_SIP,
   "BYE request received (cnxmap[%d].active = %d)
    \n",
   i,cnxmap[i].active));

   if (cnxmap[i].active == STS_TRUE)
   {
      /* connection was established */
      cnxmap[i].enable = STS_FALSE;
      cnxmap[i].active = STS_FALSE;
      cnxmap[i].expires = time_now + sipcnf.period;
      counter -=1;

      DEBUG_WRAP(DebugMessage(DEBUG_SIP,
      "spp_sipdecode connection table updated \n
      return;
   }/* if active */

   else
   { /* connection was never established*/
      GenerateSnortEvent(p, GENERATOR_SPP_SIPDECODE,
```

```
                    BYE,1,0,0,SIPDECODE_BYE_STR);
                    /* if (sipcnf.drop) InlineDrop(); */
                    return;
                 } /* else active*/

              } /* if bye */

              /* It's not a ACK request nor a BYE, is it a INVITE?
               * if everything goes well we should not be here,
               * coz a call_id is different for every call, means a
               * new entry is created for each new invite.*/

            /* Check is this connection can be handle by the proxy.
              (counter < limit)*/

              DEBUG_WRAP(DebugMessage(DEBUG_SIP,"simultaneous calls
                        %d\n",counter));

              if(counter > sipcnf.limit)
              {
                 GenerateSnortEvent(p, GENERATOR_SPP_SIPDECODE,
                 LIMIT,1,0,0,SIPDECODE_LIMIT_STR);
                 if (sipcnf.drop) InlineDrop();
                 return;
              }/* if counter */

              /* Normally we should not receive use this */
              if (invite == STS_TRUE)
              {
                 cnxmap[i].wait_ack = STS_TRUE;
                 DEBUG_WRAP(DebugMessage(DEBUG_SIP,
                           "spp_sipdecode INVITE received,
                           update field wait_ack \n"));
                 return;
              }

          }  /* if compare call-ID */

       }/*if compare to*/

    }/* if compare from*/

 } /* for */

 /* No entry found, first call?
   write new entry if it's a INVITE request else generate event */

 if (invite == STS_TRUE)
 {
    DEBUG_WRAP(DebugMessage(DEBUG_SIP,
    "spp_sipdecode INVITE request received \n"));

    if ( (j < 0) && (i >= CNXMAP_SIZE) )
    {
       /* oops, no free entries left... */
       FatalError("CNXMAP is full !!!");
       return;
    }/* if j<0*/

    /*Check is this connection can be handle by the proxy.
    (counter < limit)*/
```

```
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,"simultaneous calls
                  %d\n",counter));
       if(counter >= sipcnf.limit)
       {
          GenerateSnortEvent(p, GENERATOR_SPP_SIPDECODE,
          LIMIT,1,0,0,SIPDECODE_LIMIT_STR);
          if (sipcnf.drop)InlineDrop();
          return;
       }/* if counter */

       /* write entry */
       cnxmap[j].enable = STS_TRUE;
       osip_uri_clone(sipmsg->from->url, &cnxmap[j].from);
       osip_uri_clone(sipmsg->to->url,&cnxmap[j].to);
       osip_call_id_clone(sipmsg->call_id,&cnxmap[j].callid);
       cnxmap[j].wait_ack = STS_TRUE;
       cnxmap[j].expires = time_now;

       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
       "TABLE UPDATED\n"));
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
       "cnxmap[%d].from->host: %s\n", j, cnxmap[j].from->username));
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
       "cnxmap[%d].to->host: %s\n", j, cnxmap[j].to->username));
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
       "cnxmap[%d].wait_ack: %d\n", j, cnxmap[j].wait_ack));
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
       "cnxmap[%d].callid->number: %s\n", j, cnxmap[j].
        callid->number));

       return;
    }/* if invite */
    else if ((ack == STS_TRUE) || (bye == STS_TRUE))
    {
       GenerateSnortEvent(p, GENERATOR_SPP_SIPDECODE,
       REQUEST,1,0,0,SIPDECODE_REQUEST_STR);
       if (sipcnf.drop) InlineDrop();
       return;
    }/* else if ack or bye */

    DEBUG_WRAP(DebugMessage(DEBUG_SIP,"simultaneous
               calls%d\n",counter));
    DEBUG_WRAP(DebugMessage(DEBUG_SIP,"EXITING UPDATE TABLE...\n"));

} /* void update_table ()*/
/*****************************************************************
 * Function: int compare_url(osip_uri_t *, osip_uri_t *);
 *
 * Purpose: compares two URLs
 *          (by now, only scheme, hostname and username are compared)
 *
 * Arguments: url to be compared
 *
 * Returns: STS_SUCCESS if equal
 *          STS_FAILURE if non equal or error
 *
*****************************************************************/
int compare_url(osip_uri_t *url1, osip_uri_t *url2)
{

   if (url1 == NULL || url2 == NULL) return STS_FAILURE;
```

```
   /* compare SCHEME (if present) case INsensitive */
   if (url1->scheme && url2->scheme)
   {
      if (osip_strcasecmp(url1->scheme, url2->scheme) != 0)
          return STS_FAILURE;
   }

   /* compare username (if present) case sensitive */
   if (url1->username && url2->username)
   {
      if (strcmp(url1->username, url2->username) != 0)return
          STS_FAILURE;
   }
   /* compare hostname strings case INsensitive */
   if (osip_strcasecmp(url1->host, url2->host) != 0)return
STS_FAILURE;

   /* the two URLs did pass all tests successfully - MATCH */
   return STS_SUCCESS;
}
/**********************************************************************
 * Function: int compare_callid(osip_call_id_t *, osip_call_id_t *);
 * Purpose: compares two Call IDs
 *          (by now, only hostname and username are compared)
 * Arguments: call-ID to be compared
  * Returns: STS_SUCCESS if equal
             STS_FAILURE if non equal or error
**********************************************************************/
int compare_callid(osip_call_id_t *cid1, osip_call_id_t *cid2)
{ /* Check number part: if present must be equal,
   * if not present, must be not present in both cids */

   if (cid1 == NULL || cid2 == NULL) return STS_FAILURE;
   if (cid1->number && cid2->number)
   { /* have both numbers */
      if (strcmp(cid1->number, cid2->number) != 0) return STS_FAILURE;
   }
   else
   {
      /* at least one number missing, make sure that both are empty */
      if ( (cid1->number && (cid1->number[0]!='\0')) ||
           (cid2->number && (cid2->number[0]!='\0'))) return
            STS_FAILURE;
   }

   /* Check host part: if present must be equal,
    * if not present, must be not present in both cids */
   if (cid1->host && cid2->host)
   {  /* have both hosts */
      if (strcmp(cid1->host, cid2->host) != 0) return STS_FAILURE;
   }
   else
   {
      /* at least one host missing, make sure that both are empty */
      if ( (cid1->host && (cid1->host[0]!='\0')) ||
           (cid2->host && (cid2->host[0]!='\0')))return STS_FAILURE;
   }
   return STS_SUCCESS;
}
```

```
/*************************************************************************
 * Function: int security_check_sip(osip_message *, Packet *)
 * Purpose: Do security and integrity checks on the received packet
 *          (parsed buffer)
 * Arguments: message sip to be checked, packet *
 * Returns: STS_SUCCESS if ok
 *          STS_FAILURE if the packet did not pass the checks
 *
*************************************************************************/
  /* source : RFC 3261,
 * => Mandatory for ALL requests and responses
 * Call-ID                  c       r    m   m   m   m   m   m
 * CSeq                     c       r    m   m   m   m   m   m
 * From                     c       r    m   m   m   m   m   m
 * To                       c(1)    r    m   m   m   m   m   m
 * Via                      R      amr   m   m   m   m   m   m */

int security_check_sip(osip_message_t *sip, Packet *pkt)
{
   in_addr_t Ip = inet_addr(sipcnf.proxy_ip);
   osip_via_t *via;

   DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Start security check \n"));

   if (pkt->iph->ip_src.s_addr == Ip )
   {
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,
                 "Packet from proxy, exit security check\n"));
       return STS_SUCCESS;
    }

   if (MSG_IS_REQUEST(sip))
   {
      /* check for existing SIP URI in request */
      if ((sip->req_uri == NULL) || (sip->req_uri->scheme == NULL))
      {
         GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
           NULL_SIP_URI,1,0,0,SIPDECODE_NULL_SIP_URI_STR);
         if (sipcnf.drop) InlineDrop();
         return STS_FAILURE;
      }

      /* check SIP URI scheme */
      if (osip_strcasecmp(sip->req_uri->scheme, "sip"))
      {
         GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
           UNKNOWN_SCHEME,1,0,0,SIPDECODE_UNKNOWN_SCHEME_STR);
         if (sipcnf.drop) InlineDrop();
         return STS_FAILURE;
      }
   }

   /* check for existing Call-ID header */
   if ((sip->call_id==NULL)||
       ((sip->call_id->number==NULL)||(sip->call_id->host==NULL)))
   {
        GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
          NULL_CALL_ID,1,0,0,SIPDECODE_NULL_CALL_ID_STR);
        if (sipcnf.drop) InlineDrop();
        return STS_FAILURE;
   }
```

```
/* check for existing Via: header list */
osip_message_get_via(sip,0,&via);

if (via == NULL)
{
   GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
   NULL_VIA_HDR ,1,0,0,SIPDECODE_NULL_VIA_HDR_STR);
   if (sipcnf.drop)InlineDrop();
   return STS_FAILURE;
}

/* check for existing From: header */
if ((sip->from==NULL)||
    (sip->from->url==NULL)||(sip->from->url->host==NULL))
{
    GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
    NULL_FROM_HDR,1,0,0,SIPDECODE_NULL_FROM_HDR_STR);
    if (sipcnf.drop)InlineDrop();
    return STS_FAILURE;
}

/* check for existing To: header */
if ((sip->to==NULL)||
    (sip->to->url==NULL)||(sip->to->url->host==NULL))
{
   GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
      NULL_TO_HDR ,1,0,0,SIPDECODE_NULL_TO_HDR_STR);
   if (sipcnf.drop)InlineDrop();
   return STS_FAILURE;
}

/* life insurance: check size of received call_id strings */

if (strlen(sip->call_id->number) > CALLIDNUM_SIZE)
{
     GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
       NUMSIZE_TO_BIG ,1,0,0,SIPDECODE_NUMSIZE_TO_BIG_STR);
     if (sipcnf.drop)InlineDrop();
     return STS_FAILURE;
}

if (strlen(sip->call_id->host) > CALLIDHOST_SIZE)
{
   GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
     HOSTSIZE_TO_BIG,1,0,0,SIPDECODE_HOSTSIZE_TO_BIG_STR);
   if (sipcnf.drop)InlineDrop();
   return STS_FAILURE;
}

/* check for existing CSeq header */
if ((sip->cseq == NULL) || (sip->cseq->method == NULL) ||
   (sip->cseq->number) == NULL)
{
   goto alert;
}
else
{

   /* Verify if methods match */
   DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Verify if methods in Cseq
              match\n"));
```

```
      if (MSG_IS_REQUEST(sip))
      {
          DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Packet is a request:
          %s\n",
          sip->sip_method));

          if ((MSG_IS_INVITE(sip)) && (strcmp("INVITE", sip->
              cseq->method)!= 0))
          {
             DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Cseq method = %s\n",
             sip->cseq->method));
             goto alert;
          }
          else if ((MSG_IS_ACK(sip)) && (strcmp("ACK",sip->cseq
                                              ->method) != 0))
          {
             DEBUG_WRAP(DebugMessage(DEBUG_SIP,"cseq method = %s\n",
             sip->cseq->method));
             goto alert;
          }
          else if ((MSG_IS_BYE(sip)) && (strcmp("BYE",sip->cseq
                                                  ->method) != 0))
          {
             DEBUG_WRAP(DebugMessage(DEBUG_SIP,"cseq method = %s\n",
             sip->cseq->method));
             goto alert;
          }
          else if ((MSG_IS_REGISTER(sip)) &&
                    (strcmp("REGISTER",sip->cseq->method) != 0))
          {
             DEBUG_WRAP(DebugMessage(DEBUG_SIP,"cseq method = %s\n",
             sip->cseq->method));
             goto alert;
          }
      }
      else if (MSG_IS_RESPONSE(sip))
      {
       DEBUG_WRAP(DebugMessage(DEBUG_SIP,"Packet is a response\n"));
      return STS_SUCCESS;
      }
   }
   return STS_SUCCESS;

alert:
   GenerateSnortEvent(pkt,GENERATOR_SPP_SIPDECODE,
           NULL_CSEQ_HDR,1,0,0,SIPDECODE_NULL_CSEQ_HDR_STR);
    if (sipcnf.drop) InlineDrop();
        return STS_FAILURE;
 }
```

Annexe F. Planning

	Task Name	Start	Duration
1	Correction Rapport Travail de semestre	Tue 9/28/04	5 days
2	**Etude la problématique IPS**	**Tue 10/5/04**	**20 days**
3	Description d'un IPS et IDS	Tue 10/5/04	5 days
4	Vulnérabilités de la VoIP	Mon 10/11/04	5 days
5	Recherche d'un produit existant	Mon 10/18/04	3 days
6	Etude et test du produit exisant	Thu 10/21/04	7 days
7	Rédaction Donnée et corrigé du laboratoire FW/ALG SIP	Mon 11/1/04	3 days
8	Réalisation d'une solution IPS	Thu 11/4/04	20 days
9	Rédaction Rapport	Wed 12/1/04	10 days