Table des Matières

Liste des tableaux

Table des figures

Remerciements

Je tiens à remercier dans un premier temps le professeur Kamel ADI de l'université du Québec en Outaouais (Canada) pour m'avoir encadré tout au long d'année pratique de mon magistère, pour son aide et ses orientations.

Je tiens ensuite à remercier Mr Kamel TARI, le chef de département d'informatique de l'université de Bejaia pour les moyens qui m'a offert pour mettre en place mes expérimentations.

Je remercie également Melle Lamia HAMZA pour ses relectures et ses conseils avisés.

Je remercie tous les enseignants de l'école doctorale ReSyD, et tous les responsables qui ont participé dans la bonne démarche de nos études durant ces deux années de magistère.

Je remercie tous les étudiants de l'école doctorale ReSyD, pour l'environnement de travail très agréable durant ces deux dernières années de magistère.

Je remercie toutes les personnes qui ont lu et relu ce mémoire et en particulier, mesdames et messieurs les rapporteurs.

Enfin, et surtout, je remercie vivement toute ma famille qui m'a toujours supporté moralement et financièrement pendant toutes mes longues années d'étudiant.

Ammar BOULAICHE.

Résumé

UN "honeypot" peut être défini comme un ensemble de pièges permettant de détecter ou de guider des tentatives d'utilisation non autorisées d'un système d'information. Généralement, il se compose d'un ordinateur ou d'un noeud de réseau qui semble faire partie d'un réseau mais qui est isolé et protégé et qui semble contenir de l'information précieuse pour des attaquants. L'utilisation, dans certaines conditions, de leurres informatiques peut s'avérer être d'une très grande utilité pour lutter contre les nouvelles menaces. Cependant, les technologies "*honeypot*" actuelles sont facilement détectables par les intrus et ne jouent donc leur plein rôle comme outil de protection. Donc, par cet axe de recherche, nous souhaitons mettre en place des technologies de leurres informatiques efficaces et non détectables permettant de connaître au plus tôt les nouvelles techniques et méthodes employées par les intrus afin d'anticiper au mieux les nouveaux risques. Ce projet de recherche se propose d'expérimenter les technologies actuelles et d'en proposer une évaluation précise, avec des solutions possibles pour résoudre les problèmes de leur détection.

Mots clés : Détection d'attaques de systèmes, Leurre informatique, Simulation, Audit.

Abstract

A *honeypot* can be defined as a set of traps allowing to detect or to drive a not authorized use attempts of a information system. Generally, it is composed of a computer or a network node which seems be a part of a network but which is isolated and protected and seems contain a precious information for attackers. The use of computer baits, in certain conditions, can turn out have a very big utility to fight against new threats. However, current honeypot technologies are easily detectable by the intruders and do not play so their full role as a protection tools. So, by this axis of search, we wish to set up a efficient and not detectable computer lures technologies allowing to know as soon as possible the new techniques and methods used by the intruders to anticipate at best the new risks. This research project suggests experimenting current technologies and proposing it a precise evaluation, with possible solutions to resolve the problems of their detection.

Keywords : Systems attacks detection, computer Bait, Simulation, Audit.

تلخيص

"الهونيبوت" (قدر العسل) هو عبارة عن مصيدة إلكترونية تسمح باكتشاف و مراقبة المحاولات غير الشرعية لاختراق واستعمال الأنظمة الآلية، فهو يتشكل عموما من جهاز كمبيوتر أو أي جهاز آخر مربوط بالشبكة الآلية بحيث يبدو وكأنه جهاز عادي ككل أجهزة الشبكة الأخرى لكنه في الحقيقة معزول و محمي بحيث يظهر و كأنه يحوي معلومات مهمة ومغرية للمخترق. إن استعمال هذه المصايد الإليكترونية داخل الشبكات تحت شروط معينة يكتسي أهمية بالغة في اكتشاف و مواجهة التهديدات الحديثة التي لم يتم الكشف عن وسائلها بعد. لكن أغلب تقنيات "الهونيبوت" الحالية إن لم نقل كلها يمكن اكتشاف وجودها داخل الشبكة بسهولة تامة من طرف المخترقين مما يجعلها تفقد الكثير من أهميتها كوسيلة لحماية الأنظمة الإلكترونية. في موضوع بحثنا هذا سنحاول تقديم تقنيات "هونيبوت" جديدة أكثر فاعلية و غير قابلة للاكتشاف من طرف المخترقين، والتي تسمح بالكشف عن التقنيات والطرق الحديثة التي يستعملها المخترقون في أقصر وقت ممكن مما يمكننا من تجنب كل هذه التهديدات قبل أن توقع أضرارا بليغة داخل الأنظمة المستهدفة. مشروع هذا البحث يتضمن تجريب تقنيات "الهونيبوت" المستعملة حاليا ومن ثم تقديم تقييم مفصل لهذه التقنيات انطلاقا من النتائج المحصل عليها مع محاولة تقديم اقتراحات لسد الثغرات التي تؤدي إلى اكتشاف هذه التقنيات من طرف المخترقين.

مفاتيح البحث : اكتشاف الهجمات على الأنظمة، المصايد الآلية، المحاكاة، "الأوديت".

Introduction générale

Devant la croissance accrue dans l'utilisation des réseaux et des systèmes informatiques et devant la nature sensible des données manipulées par ces systèmes, il est primordial de mettre en place des mécanismes de sécurité pour assurer l'intégrité[1], la confidentialité[2], la disponibilité[3] ainsi que la non-répudiation [4]. La sécurité informatique en général et la sécurité des réseaux en particulier est devenue aujourd'hui, plus que jamais, un souci majeur au niveau international, et surtout à l'heure où les systèmes informatiques sont devenus de plus en plus complexes, avec des centaines d'applications qui s'exécutent en parallèle, et qui ont été développées par des acteurs dont le soucis majeur est de réduire au maximum le temps de délivrance de la marchandise au marché. Ceci a considérablement augmenté le nombre de vulnérabilité qui permettent aux intrus de s'introduire frauduleusement dans les systèmes avec tout ce que cela suppose comme nuisances.

Comme une première solution pour toutes ces menaces et vulnérabilités, plusieurs mécanismes de sécurité ont été développés pour nous protéger, tels que : Les pare-feux, les IDS (*Intrusion Detection System*), les routeurs filtrants, les protocoles et applications sécurisées (IPv6, IPSec, etc.), les VPN (*Virtual Private Network*), etc. Le plus connu parmi tous ces dispositifs et solutions est le Pare-feu (*Firewall*), qui détermine la limite entre un réseau interne et l'Internet en analysant les informations contenues dans les flux de données entrants et sortants pour bloquer les connexions suspectes qui peuvent provenir de virus, vers ou chevaux de Trois d'une part et pour éviter la fuite non contrôlée d'informations vers l'extérieur d'autre part. Cependant, à lui tout seul, le pare-feu n'est pas très efficace pour garantir la sécurité des systèmes protégés, il est souvent indispensable de lui adjoindre un IDS qui surveille les comportements des utilisateurs du système en détectant des signatures d'attaques indiquant des intentions malicieuses ou suspectes. Cependant, malgré tous ces outils, les intrusions dans les systèmes informatiques sont encore fréquentes, car ces derniers sont surtout de type préventif dont le fonctionnement est basé sur des signatures d'attaques déjà connues

[1] **Intégrité :** assurer que l'information n'a pas été altérée par des moyens non autorisés.
[2] **Confidentialité :** assurer que l'information est accessible uniquement à ceux qui sont autorisés.
[3] **Disponibilité :** assure que l'information est accessible selon les modalités définies.
[4] **Non-répudiation :** assurer que l'auteur d'une action ne peut dénier l'avoir effectué.

et laisse de ce fait la porte grande ouverte pour des attaques inconnues. Il est donc très important et nécessaire de découvrir au plus tôt les nouvelles méthodes que les pirates informatiques emploient pour compromettre les systèmes cibles et affronter ainsi ces attaques. Pour assurer ce besoin, une nouvelle technologie de sécurité informatique appelée "Honeypots" ou *pot de miel* est apparue.

Contrairement aux différents outils de sécurité préventifs traditionnels, les *honeypots* adoptent une approche non défensive de la sécurité, une approche qui consiste à prendre l'initiative d'attirer les attaquants en les invitant à explorer un système informatique, à l'attaquer et à le corrompre tout en analysant dans le détail son comportement. En effet, un *Honeypot* est défini comme une ressource (ordinateur ou programme) volontairement vulnérable, destinée à attirer et à piéger les hackers, afin d'étudier leurs comportement et d'obtenir des informations sur les moyens et les techniques de compromissions qu'ils utilisent bien avant que ces techniques ne deviennent publiques. Un *honeypot* est un système de sécurité qui n'a aucune valeur de production et aucun utilisateur ni aucune autre ressource ne devrait en principe avoir à communiquer avec lui. L'activité ou le trafic attendu sur le *honeypot* étant nul à la base, donc toute activité enregistrée par cette ressource est suspecte par sa nature. Ainsi, tout trafic, tout flux de données envoyé à un *honeypot* est probablement un test, un scan ou une attaque. Tout trafic initié par un *honeypot* doit être interprété comme une probable compromission du système et signifie que l'attaquant est en train d'effectuer des connexions par rebond.

Depuis leur apparition, le développement des technologies *honeypot* n'a jamais cessé de progresser. Initialement ils n'étaient que de simples démons exécutés sur des machines du réseau pour émuler les différents systèmes et leurs services (exp. *honeyd*), l'interaction entre ce type de *honeypots* et les attaquants était très faible à cause de la limite imposée par l'émulation utilisée, ce qui a limité leur utilisation dans le domaine de production. Actuellement, nous avons des *honeypots* qui ne se basent pas sur l'émulation d'un service, mais plutôt sur un vrai système d'exploitation (par exemple, le *honeynet roo*, *ManTrap*, etc.) en offrant une haute interaction avec l'attaquant.

Avec cette évolutions, plusieurs problèmes concernant l'utilisation des *honeypots* ont commencé à se poser. Dans ce projet de recherche nous allons discuter l'un des problèmes les plus importants qui menace l'utilisation future des technologies *honeypot* actuelles : c'est le problème de leurs détection par les intrus.

Contributions

Ces dernières années ont vue l'utilisation des technologies *honeypot* devenir de plus en plus fréquente pour affronter les nouvelles attaques qui apparaissent chaque jour en attirant les pirates informatiques dans un système piégé pour en gagner des informations vitales sur les nouvelles vulnérabilités qui ne sont connues que par un cercle restreint de pirates. Donc, ces technologies forment une vraie menace pour la communauté des pirates d'où l'utilité de détecter toute présence de *honeypot* dans un système informatique. Puisque, le développement de la plupart des technologies *honeypot* actuelles était concentré sur le contrôle et la capture des données des pirates, avec une négligence presque totale du côté de leur détection, la plupart de ces derniers sont faciles à détecter et à désactiver ou à détourner par les pirates expérimentés et même par les amateurs (*script kiddy*) qui utilisent des outils faciles à installer et à piloter.

La conception de technologies *honeypot* furtive (non détectable) est de nos jours un problème d'actualité et plusieurs recherches y sont consacrées. Ainsi, dans notre projet de recherche, nous avons apporté plusieurs solutions afin d'améliorer la furtivité des technologies *honeypot*. En effet, suite à une étude expérimentale poussée sur des produits *honeypot* les plus réputées et les plus connues dans le milieu des communautés des ingénieurs et des experts de la sécurité informatique (la technologie *honeynet roo* et la technologie *honeyd*), nous avons identifié plusieurs failles qui permettent de détecter la présence de *honeypot* et proposé plusieurs solution originales afin de remédier à ces failles. Plus particulièrement, nous avons apporté les contributions suivantes :

- Nous avons réalisé une expérimentation poussée sur deux technologies *honeypot* les plus utilisées actuellement dans le milieu de la sécurité informatique. La première technologie est la technologie *Honeynet ROO CDROM* (c'est la technologie de haute interaction la plus utilisée dans les milieux de recherche), alors que la deuxième technologie est la technologie *Honeyd* (c'est la technologie de faible interaction la plus utilisée dans les milieux de production). Nous avons installé ces deux technologies dans le réseau intranet de l'université de Bejaia, pour une durée de 3 à 4 mois. Cependant, avant de lancer ces expérimentations, nous avons d'abord maîtriser tous les risques possibles qui peuvent accompagner l'installation d'un *Honeypot* dans un réseau de production, car la mise en place de ces dernières est très sensible, et nécessite des connaissances et des configurations particulières, afin de pouvoir prendre toutes les précautions nécessaires pour éviter la possibilité d'utilisation du *honeypot* comme rebond pour conduire de nouvelles attaques contre des systèmes étrangers ou d'autres machines de notre réseau.
- Nous avons proposé une analyse détaillée des données capturées dans chacun des deux *Honeypots* pour découvrir quels sont les systèmes, les protocoles et les ports ciblés par la communauté des hackers. Pour réaliser cette analyse, nous nous

sommes d'abord familiarisé avec les techniques d'attaques et avec les techniques d'analyse des fichiers de log. Suite à cette analyse, nous avons découvert plusieurs failles qui touchent à la fiabilité et à l'efficacité des deux technologies. En effet, l'analyse nous a permis d'extraire des résultats intéressants qui permettent de localiser facilement la présence des *honeypots*.

– Nous avons proposé un ensemble de solutions pour les problèmes de la détection des technologies *honeypots* que nous avons expérimenté.

Organisation du mémoire

Ce mémoire se compose de quatre chapitres :

- Dans le premier chapitre, nous exposons un état de l'art sur les technologies *honeypot*. Nous commençons par un petit historique et une définition du *honeypot*, puis nous décrivons les différents types, implémentations et architectures des *Honeypots*, avec la présentation de quelques technologies *honeypot* actuelles et les avantages et les inconvénients de chacune d'elles. Nous terminons le chapitre par la description de la notion de *honeynet* et son fonctionnement, ses architectures et ses types.
- Dans le deuxième chapitre, nous exposons la première expérimentation effectuée dans le cadre de notre projet. Nous commençons par un petit état de l'art sur la technologie à expérimenter et les différents outils utilisés, puis nous continuons par la description de l'environnement d'expérimentation, et les ressources matérielles et logicielles utilisées, ainsi que l'architecture générale de cette expérimentation. Enfin, nous terminons ce chapitre par une analyse détaillée du trafic capturé avec une conclusion résumant les leçons acquises de cette expérimentation.
- Dans le troisième chapitre, nous exposons la deuxième expérimentation, en commençant toujours par un petit état de l'art sur la technologie à expérimenter. Ensuite, nous faisons une description de l'environnement et des configurations de cette expérimentation. Enfin, et comme dans le chapitre précédent, nous terminons ce chapitre par une analyse du trafic capturé durant les deux mois de cette deuxième expérimentation et tirons les conclusions et les acquis de cette expérimentation.
- Dans le dernier chapitre, nous traitons la problématique de la détection des deux technologies expérimentées. Nous commençons d'abord par la description des failles trouvées dans la première technologie, et les solutions proposées pour les résoudre. Ensuite, nous terminons ce chapitre par la description de la série des tests effectués pour découvrir les failles qui conduisent à la détection de la deuxième technologie, avec les solutions proposées pour remédier toutes ces failles.
- Finalement, nous terminons ce mémoire par une conclusion générale récapitulant le travail accompli avec quelques travaux futurs, et un annexe contenant toutes les configurations importantes effectuées pendant la mise en place des deux expérimentations.

Chapitre 1

Les technologies "honeypot"

1.1 Introduction

Les attaques informatiques représentent aujourd'hui le plus grand défi pour la plupart des entreprises, et des établissements, cependant, malgré les développements et les améliorations des méthodes et des outils de protection, le nombre des attaques succès s'augmente sans cesse, et des nouveaux formes d'attaques et de trous de sécurité sont publiés presque chaque jour. Néanmoins, ni l'augmentation d'utilisation des $firewalls$, ni celle des anti-virus ont pu stopper ces attaques, et encore les dommages des entreprises et des établissements et même des personnes restent en croissance à l'abri des systèmes actuels de protection, de détection et de prévention des attaques qui forment un simple point de protection contre les attaques connues qui sont enregistrées dans leurs bases de règles (bases de signatures), le problème crucial de ces derniers c'est comment mettre à jour leurs bases de signatures par les nouveaux types d'attaques avant que ces derniers peuvent faire des dégâts. Une des solutions les plus importantes pour ce types de problèmes c'est l'utilisation des *honeypots*.

1.2 Les "Honeypots"

1.2.1 Historique

L'utilisation des *honeypots* dans les systèmes informatiques revient au milieu des années 80, où $Clifford\ Stoll$ a rapporté dans son livre [Sto90] les premiers balbutiements du concept de *honeypot*, lorsque dans le cadre d'une investigation qu'il menait à l'université de $Berkeley$ (en 1986), celui-ci a été amené à alimenter un pirate en fausses informations, de façon à garder l'intrus en ligne suffisamment longtemps pour réussir

à le localiser et finalement le faire appréhender par les forces de police. Ainsi l'idée de pot de miel ou *honeypot* naît, dans sa première appréhension, dès le milieu des années 1980. Et au début des années 90, le 7 janvier 1991 pour être exact, *Bill Cheswick* a implémenté et déployé un véritable *Honeypot* au sein des laboratoires de la division de recherche informatique et scientifique de AT&T [Che04].

Bill a présenté le problème de la façon suivante : *"Un cracker, croyant avoir découvert le fameux trou de sécurité utilisant la commande DEBUG de sendmail sur notre passerelle Internet a tenté d'obtenir une copie de notre fichier password. Je la lui ai envoyé."*.

Quelques années plus tard le concept de *honeypot* devient plus sophistiqué, faisant l'objet de véritables études et donnant lieu au développement d'outils spécifiques comme : *Deception Toolki* en 1997, *CyberCop* en 1998 (premier *honeypot* commercial), *BackOfficer Friendly* en 1998, *honeyd* en 2002 et plusieurs autres [MFH04, Spi02].

Actuellement, le domaine des *honeypots* est très évolué incluant des projets complexes comme le *Honeynet Project* qui se développe d'une manière très rapide depuis sa première apparition en 1999, et qui est actuellement installé au niveau de plus de vingtaine de projets distribués sur plus de quinzaine de pays du monde [HP07a].

1.2.2 Définition

La définition d'un *honeypot* comme a été défini par *Lance Spitzner* [Spi02, L.S03a] est : *"Un honeypot est une ressource système dont la seule utilité est de se faire attaquer ou compromettre"*, autrement dit, un *honeypot* est un ordinateur ou un programme volontairement vulnérable destiné à attirer et à piéger les pirates informatiques.

1.2.3 Types de "honeypots"

Selon ce qu'on attend des *honeypots*, nous pouvons distinguer deux principaux types : *honeypots* de production et *honeypots* de recherche [Spi02, Spi03].

1.2.3.1 "Honeypots" de production (*production Honeypots*)

Un *honeypot* de production est utilisé pour sécuriser un réseau opérationnel. Il déroute les attaques orientées vers les différents services de production du système, en les attirant vers lui, ce qui permet de réduire le risque, en renforçant la sécurité qui est assurée par les autres mécanismes de sécurité comme les *firewalls*, les IDS (Systèmes de Détections d'Intrusions), etc. Comme il peut aussi détecter des attaques grâce à ses fichiers d'audit, qui peuvent être aussi utilisés pour corriger les vulnérabilités.

Son rôle dans la protection du système

Un *honeypot* de production joue un rôle important dans une ou plusieurs composantes de la sécurité du système de production telles que :

- **La prévention :** Laisser le hacker jouer sur le *honeypot* au lieu de jouer sur les systèmes de production.
- **La détection :** Toute connexion établie avec un *honeypot* de production est considérée comme tentative d'intrusion au système, il élimine ainsi toutes les fausses alertes (positives et négatives).
- **Le recouvrement :** Le rôle des *honeypots* de production dans le recouvrement se traduit par les deux points suivants :
  - Ils permettent une continuité des services après une attaque produite en leur sein, en les mettant simplement hors service.
  - L'information enregistrée par les *honeypots* de production sera d'un apport considérable pour le recouvrement du système.

1.2.3.2 "Honeypots" de recherche (*research Honeypots*)

Le souci de ce type de *honeypots* n'est pas de sécuriser un système particulier, mais c'est de s'introduire dans un environnement de recherche pour comprendre et étudier comment la communauté *Black Hat* évolue, quelles sont les techniques que cette communauté utilise et qui appartient à cette communauté. Les *honeypots* de recherche sont plus complets que les *honeypots* de production. C'est en général le système en entier qui peut être attaqué (et non pas seulement un seul service), ce qui en fait des systèmes sensibles dans leur gestion et complexes pour l'analyse de leurs résultats.

Son rôle dans la protection du système

Les *honeypots* de recherche ne servent pas la sécurité des systèmes (prévention, détection et recouvrement) d'une manière directe, mais ils offrent des renseignements précieux sur les attaquants et leur comportement. Ces informations permettent une meilleure connaissance de la communauté des hackers, ce qui aide les professionnels de la sécurité informatique dans l'amélioration de méthodes et mécanismes de protection.

1.2.4 Classification des "honeypots"

L'implémentation d'un *honeypot* est reposée principalement sur le niveau d'interaction[1] *"level of involvment"* du *honeypot* utilisé. Ainsi, nous pouvons distinguer trois classes différentes de *honeypots* [Spi02, Spi03] : les *honeypots* à faible interaction, les *honeypots* à moyenne interaction et les *honeypots* à haute interaction.

[1]Le terme interaction désigne l'interaction entre le pirate et le système piraté.

1.2.4.1 "Honeypots" à faible interaction

Un *honeypot* à faible interaction est un *honeypot* virtuel fournit comme le montre son nom une interaction limitée (faible) avec le pirate, il est tout simplement un programme qui simule les services d'un système réel [Ros03] par la mise en place par exemple des sockets d'écoute sur chaque port d'un service, ces sockets ne font que logger les différents paquets qu'elle reçoit, comme le montre la figure Fig.1.1.

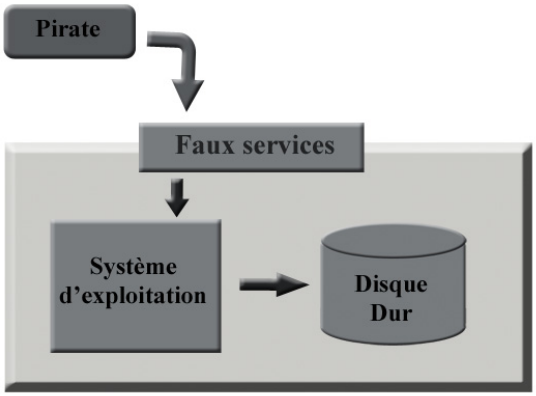


FIG. 1.1 – Schéma d'une interaction faible

Les avantages et les inconvénients principaux de ce type de *honeypots* sont :

- **Avantages :**
  - La mise en place est très simple.
  - La gestion complexe des logs du système est éliminée.
  - La sécurité du système est conservée (mais uniquement s'il est bien configuré et que les faux services implémentés ne possèdent pas eux même un trou de sécurité).
- **Inconvénients :**
  - Sont faciles à détecter par les attaquants à cause d'absence de réponses attendues dues à l'implémentation incomplète des services.
  - Peu d'informations sur l'attaquant sont dérivées (le temps et la date d'attaque, adresse IP (*Internet Protocol*) source et adresse IP destination de la connexion, port source et port destination de la connexion), car le *honeypot* n'offre aucune possibilité au pirate de s'introduire dans le système.

Le but principal de ce type de *honeypots* est la détection des tentatives de connexions non autorisées. Donc ce type est sans doute le plus utilisé dans les *honeypots* de production.

Les *honeypots* à faible interaction les plus connus sont : *Honeyd*, *Specter*.

1.2.4.2 "Honeypots" à moyenne interaction

Un *honeypot* à moyenne interaction est un *honeypot* semi-virtuel qui assure une simulation améliorée des services d'un système par rapport à la simulation fournie par les *honeypots* à faible interaction, en lui ajoutant la possibilité de renvoi des réponses aux attaquants, ces réponses sont généralement fausses de façon à leur donner des pistes ou à les dérouter sans forcément les intriguer, comme le montre la figure Fig.1.2. En plus des services simulés, il offre aussi quelques services réels, mais sans donner la possibilité au pirate de prendre un contrôle total du système.

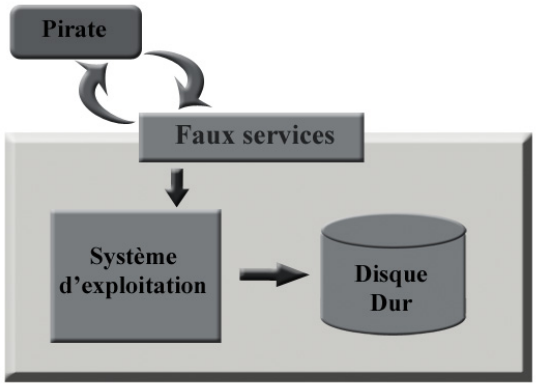


FIG. 1.2 – Schéma d'une interaction moyenne

Les avantages et les inconvénients principaux de ce type de *honeypots* sont :

- **Avantages :**
  - La gestion des logs du système est facile par rapport à celle des *honeypots* à haute interaction et un peu difficile par rapport à celle des *honeypots* à faible interaction.
  - Fournit beaucoup plus d'informations intéressantes à analyser, à cause de variété d'attaques proposées aux pirates ce qui s'avère plus intéressant pour eux.
- **Inconvénients :**
  - Très dur à implémenter en terme de développement, car la fourniture d'un leurre parfait implique une parfaite connaissance des protocoles de chaque faux service pour bannir toute faille de sécurité.
  - Sécurité du système difficile à contrôler, du fait que, plus le niveau de complexité d'un *honeypot* augmente plus il y a de chance qu'il contienne lui même un trou de sécurité qui peut être exploité par le pirate.

Les *honeypots* les plus connus de ce type sont : *homemade honeypots*, *Deception Toolkit*.

1.2.4.3 "Honeypots" à haute interaction

Contrairement aux *honeypots* à faible et à moyenne interaction, un *honeypot* à haute interaction ne se base pas sur l'émulation d'un service, mais plutôt sur un vrai système d'exploitation, ce qui offre une grande interactivité avec l'attaquant, puisque il s'agit bien des systèmes réels avec des failles de sécurité qu'il peut exploiter, comme le montre la figure Fig.1.3. Ce type de *honeypots* est orienté plus à la recherche dont on souhaite qu'un pirate pénètre un système pour l'observer.

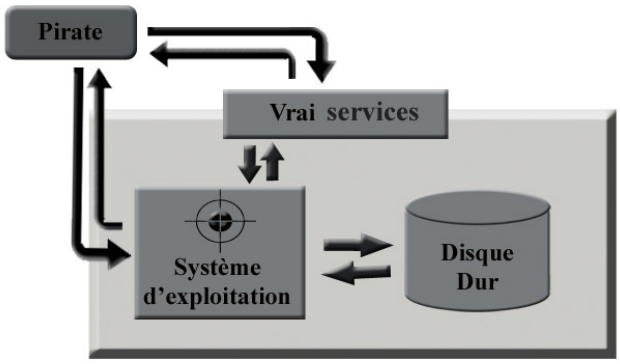


FIG. 1.3 – Schéma d'une interaction forte

Les avantages et les inconvénients principaux de ce type de *honeypots* sont :

- **Avantages :**
  - Très difficile à détecter par les pirates.
  - Fournir beaucoup d'informations sur les activités du pirate.
- **Inconvénients :**
  - Introduire un grand risque dans le système hôte, à cause de la pénétration du pirate au système réel avec une liberté totale, ce qui puisse engendrer une réinstallation périodique du système.
  - La réactivité du monitoring est un facteur de complexité important.
  - La gestion des logs est très compliquée (transférer les informations capturées sur le pirate à un log distant sans se faire repérer par les attaquants).

Les *honeypots* à haute interaction les plus connus sont :$Honeynet\ CDROM\ ROO$, $ManTrap$.

1.2.5 Architecture des "Honeypots"

L'architecture d'un *honeypot* est définie par la nature du système qui l'héberge, qui peut être réel ou virtuel [Spi02] :

1.2.5.1 Architecture réelle

Dans cette architecture, chaque *honeypot* est installé sur sa propre machine physique, c'est-à-dire chaque *honeypot* est représenté par un système réel.

Les avantages et les inconvénients de cette architecture sont :

- **Avantages :**
  - Administration simplifiée.
- **Inconvénients :**
  - Si plusieurs *honeypots* alors plusieurs machines physiques.
  - Le monitoring système sans se faire repérer par le pirate est compliqué.
  - Réinstallation fréquente du système pour chaque *honeypot*.

1.2.5.2 Architecture virtuelle

Dans cette architecture, le *honeypot* est installé sur une machine virtuelle. La création des machines virtuelles est assurée par des outils de virtualisation de systèmes comme : VMWare [vH07] sous Linux et Windows, UML (User-Mode-Linux) [UH07] sous Linux, et Jail [KW00] sous Unix BSD. Ces outils peuvent émuler un ou plusieurs systèmes[2] sur une seule machine, donc il est possible d'installer plusieurs *honeypots* virtuels sur une seule machine. De plus, VMWare peut émuler plusieurs systèmes de natures différentes (windows, linux , ...etc), on peut alors proposer plusieurs *honeypots* de plusieurs systèmes d'exploitation virtuels sur la même machine physique.

Les avantages et les inconvénients de cette architecture sont résumés dans :

- **Avantages :**
  - Sécurité de la machine virtuelle.
  - Economie de machines physiques.
  - Possibilité de monitoring en temps réel des disques virtuels.
  - Facilité de réinstallation par sauvegarde des disques virtuels.
  - Le système hébergeant les systèmes virtuels est rendu invisible pour le pirate.
- **Inconvénients :**
  - Charge importante du système hébergeant les machines virtuelles.
  - Le choix du système virtuel est restreint à ceux qui sont compatibles.

[2] chaque système représente une machine virtuelle

1.2.6 Quelques technologies "honeypot" existantes

Actuellement et après une dizaine d'années depuis l'apparition de la notion de *honeypot*, un grand nombre de technologies *honeypot* ont été développées. Parmi ces technologies on peut citer :

1.2.6.1 BackOfficer Friendly (BOF)

BOF [BFH07] est un *honeypot* simple à faible interaction, développé par *Marcus Ranum*, fonctionne dans un environnement graphique (Windows ou Unix), il émule quelques services de base comme : *http, ftp, telnet, mail,* ou *Back Orifice*[3]. Toutefois, il permet d'activer quelques réponses pour certains services (en utilisant l'option *"faking replies"*), réponses qui restent limitées et non configurables. Le programme ressemble également à un système de détection d'intrusion simple qui logge sous forme d'une liste de *warnings* toutes les tentatives de connexion destinées vers les ports qu'il contrôle. Si cela ne présente que peu d'intérêt pour un utilisateur avancé, les utilisateurs non avertis y trouveront une très bon solution pour se familiariser avec les technologies *honeypot*.

Les avantages et les inconvénients de cette technologie sont résumés dans :

- **Avantages :**
  - Facile à installer, configurer et maintenir,
  - S'exécute sur n'importe quelle plate-forme système comme Windows, BSD, Linux, Solaris, ...etc,
  - Peu de risques, en raison de leur simplicité.
- **Inconvénients :**
  - Limité uniquement à sept ports sur lesquels il peut détecter des attaques,
  - Les ports ne peuvent pas être personnalisés, ce qui augmente la possibilité d'en prise des *fingerprintings* pour le *honeypot*,
  - N'a ni possibilité d'y login à distance, ni fonctionnalité de configuration, ni possibilité d'alerter l'administrateur en cas des attaques.

1.2.6.2 Specter

Specter [TSH07] est un *honeypot* commercial de production, à faible interaction, développé par *NetSec* (société Suissesse de sécurité réseau), fonctionne sous un environnement graphique. Il émule un grand nombre de services, de fonctionnalités et même de systèmes d'exploitation. Il représente un bon outil de détection d'intrusion puisqu'il supporte une grande variété de mécanismes d'alertes et d'enregistrement. Il se caractérise principalement par sa simplicité d'implémentation d'une part, et par la faiblesse du risque qu'il engendre d'autre part.

[3] *Back Orifice* est un service fourni par l'un des trojans (chevaux de troie) les plus réputés.

Les avantages et les inconvénients de la technologie *Specter* sont résumés dans les points suivants :

- **Avantages :**
  - Facile à installer, configurer et maintenir,
  - Emulation considérable de services,
  - Il contrôle deux fois autant de ports que BOF,
  - Il a la capacité de notifier l'administrateur en cas des attaques,
  - Il possède un mécanisme de management à distance.
- **Inconvénients :**
  - Il ne contrôle que 14 ports,
  - Les services émulés sont limités à l'interaction avec seulement les comportements connus,
  - Les contradictions avec la pile IP et le système d'exploitation émulé peuvent mener à la prise de *fingerprinting* du *honeypot*,
  - La quantité d'apprentissage est limitée à cause du nombre limité des ports contrôlés.

1.2.6.3 Nepenthes

Nepenthes [NH07a, BKHD06] est un *honeypot* à faible-interaction, orienté vers la capture automatique des *malwares*[4] sur les plateformes Microsoft en simulant une machine Windows. Les services émulés dans ce *honeypot* offrent aux vers et aux autres logiciels malveillants la possibilité de se télécharger au système *honeypot*. Et dés qu'une connexion est établie à un service émulé, le module de vulnérabilité approprié est chargé pour traiter la tentative d'exploitation entrante, ensuite, la charge utile contenant le *malware* envoyé par l'attaquant sera directement envoyée vers la *sandbox norman*[5].

Les avantages et les inconvénients de cette technologie sont résumés dans :

- **Avantages :**
  - Efficace pour la collection des *malwares*,
  - Facile à implémenter, car il n'émule que les parties nécessaires pour exploiter la vulnérabilité,
  - Au contraire aux attaques humaines, l'attrapage des attaques automatisées ne nécessite que quelques conditions générales qui doivent être accomplies, par conséquent, l'efficacité de l'approche est maximisée.
- **Inconvénients :**
  - Limité aux attaques automatiques via des vers et des logiciels malveillants,

[4] *malware* c'est l'abréviation des deux mots "**mal***icious soft***ware** " (programme malveillant).

[5] *Sandbox Norman* est une boite utilisée par Nepenthes pour stocker tout malware potentiel, afin de l'analyser ensuite en détaille.

- Facile à détecter par les attaquants humains, à cause de l'émulation incomplète des services,
- Limité au système d'exploitation Windows.

1.2.6.4 Honeyd

Honeyd [THH07, Pro04] est un *honeypot* à faible interaction très complet et simple à utiliser, développé par *Niels Provos* de l'université de *Michigan*, il fonctionne sur les systèmes Unix, et porté même sur Windows. Il simule des services et même des systèmes d'exploitations réels sur des adresses IP non utilisées d'un réseau. Grâce à ses fichiers de configuration, *Honeyd* permet d'émuler un grand nombre de services, de personnaliser les réponses aux connexions, de simuler différentes piles IP pour tromper l'attaquant sur la version du système d'exploitation.

Les avantages et les inconvénients de cette technologie sont résumés dans :

- **Avantages :**
  - Il peut contrôler n'importe quel port UDP (*User Datagram Protocol*) ou TCP (*Transmission Control Protocol*) et n'importe quelle architecture réseau,
  - C'est une solution *OpenSource*[6], donc il peut se développer rapidement par n'importe quelle personne de la communauté de sécurité informatique,
  - Il résiste contre la prise des *fingerprintings* du *honeypot* en imitant des systèmes d'exploitation au niveau de la pile IP aussi bien qu'au niveau application.
- **Inconvénients :**
  - Comme est une solution de faible interaction, il ne peut pas fournir aux attaquants un vrai système d'exploitation, donc la quantité d'apprentissage est limitée,
  - Comme est une solution *OpenSource*, il ne fournit aucun support formel pour la maintenance et la localisation de ses failles,
  - Il n'a aucun mécanisme d'alerte en cas des attaques.

1.2.6.5 Deception Toolkit (DTK)

DTK [DTH07] est l'un des *honeypots OpenSource* à moyenne interaction les plus anciens, créé par *Fred Cohen* à la fin d'année 1997. C'est un ensemble de scripts PERL et de codes C qui émulent une variété de vulnérabilités connues sous un système UNIX. Les réponses des faux services sont générées de façon à orienter le pirate sur les vulnérabilités du système. Toutefois un attaquant peut ainsi savoir si ce système est un

[6]OpenSource signifie que : (1) la solution est gratuite et (2) vous avez la possibilité d'accéder au programme source de cette solution, et de le personnaliser.

honeypot en vérifiant l'ouverture du port 365, qui a été attribué officiellement au DTK. Donc on peut ouvrir le port 365 exprès pour faire fuir les attaquants de notre système.

Les avantages et les inconvénients de cette technologie sont résumés dans :

- **Avantages :**
  - C'est une solution *OpenSource*, donc on peut facilement l'obtenir le personnaliser et l'installer chez nous,
  - Très efficace pour la protection des systèmes de production, il donne un temps supplémentaire de réaction à l'administrateur en laissant l'attaquant jouer sur le *honeypot* au lieu de jouer sur les systèmes de production.
- **Inconvénients :**
  - Limité par le fait qu'il n'émule que les vulnérabilités connues,
  - Il fournit une quantité limitée d'informations,
  - Installé uniquement sur les systèmes Unix,
  - Très facile à détecter, en regardant l'ouverture du port 365.

1.2.6.6 ManTrap

$ManTrap$ [SD99, MH07] est un *honeypot* professionnel à haute interaction très complet et très cher, proposé par la société *Symantec*. Il met en œuvre quatre systèmes d'exploitation logiquement séparés et qui sont installés sur la même machine hôte. Chacun de ces systèmes supporte des applications réelles, et apparaît comme système indépendant avec sa propre interface réseau. L'administration du système hôte se fait par une interface utilisateur graphique *Java*. $Mantrap$ peut être utilisé comme *honeypot* de production, notamment pour la détection et la réaction, et il est aussi rentable dans la recherche, mais avec un risque considérable.

Les avantages et les inconvénients de cette technologie sont résumés dans :

- **Avantages :**
  - Il détecte l'activité malveillante sur n'importe quel port de son système grâce au sniffer qu'il emploie,
  - Il capture toute l'activité d'attaquant, incluant le trafic chiffré comme SSH,
  - Il possède un mécanisme d'administration à distance avec un système d'alerte par e-mail.
- **Inconvénients :**
  - C'est un *honeypot* de haute interaction, donc les attaquants peuvent l'employer pour nuire à d'autres systèmes,
  - Les attaquants peuvent prendre des $fingerprintings$ du *honeypot* ou enfuir la cage $ManTrap$ à laquelle ils ont eue accès,
  - Limité aux systèmes d'exploitation Solaris,
  - C'est une solution commerciale, donc on ne peut pas le personnaliser librement.

1.3 Les "Honeynets"

1.3.1 Définition

Un *honeynet* est un réseau de systèmes *honeypots* combiné avec un ensemble de mécanismes de sécurité comme les $firewalls$, les IDS, les serveurs log, ...etc [Spi02, HP07d]. Il donne apparence à tout un environnement de production avec des failles et des informations pertinentes utilisées comme appât pour attirer et piéger les attaquants afin de surveiller et d'enregistrer ensuite toutes leurs actions. Sa structure de réseau permet même d'avoir des renseignements sur les communications entre les attaquants et leurs méthodes de collaboration. A cause de la richesse des informations collectées par lui, un *honeynet* est considéré comme un outil très utile pour apprendre plus sur les techniques d'attaques, et les comportements des attaquants, sur un réseau réel.

1.3.2 Fonctionnement des "Honeynets"

Pour pouvoir atténuer le risque du leurre informatique utilisé, et pour faciliter le processus d'analyse et de réponse, le fonctionnement du *honeynet* doit assurer les trois opérations principales suivantes : le contrôle des données, la capture et la collection des données et l'analyse des données [HP07d, Spi02].

1.3.2.1 Le contrôle des données

Un système *honeynet* doit être capable de contrôler toutes les données dedans, et surtout le trafic sortant (*outbound* $traffic$) qui peut contenir des attaques envers d'autres systèmes réels. Cela est assuré par exemple par un $firewall$ dissimulé derrière un routeur, qui utilise plusieurs techniques comme le blocage du trafic après un nombre limité de connexions. Ainsi le routeur représente une deuxième couche de contrôle du trafic sortant.

1.3.2.2 La capture et la collection des données

Un système *honeynet* doit capturer toutes les informations qui se rapportent sur l'attaquant et ses activités. Cette opération est effectuée sur trois niveaux : le $firewall$, les IDS et les systèmes *honeypots* eux même. Toutes les informations capturées dans ces trois niveaux doivent être collectées et stockées dans un dispositif central loin de tout risque éventuel pour éviter leur suppression par l'attaquant qui cherche toujours à effacer ses traces d'une part, et pour faciliter la tâche d'analyse par la suite d'autre part.

1.3.2.3 L'analyse des données

Cette opération est assurée par un ou plusieurs outils graphiques ou non graphiques utilisés par le système *honeynet* pour visualiser et examiner toutes les informations collectées d'une manière simplifiée et lisible, ce qui permet de faciliter la tâche d'extraction des informations recherchées, telles que les outils, les méthodes et tactiques utilisées par les pirates, ou bien celles qui révèlent les vulnérabilités existantes dans le système.

1.3.3 Architecture des "honeynets"

Les architectures des *honeynets* utilisées actuellement sont classifiées en trois générations, chacune de ces générations se distingue principalement par l'architecture réseau mise en place pour assurer les trois fonctionnalités du *honeynet* (contrôle de données, collecte de données, analyse de données).

1.3.3.1 Architecture de la 1ère génération

L'architecture de cette génération est une architecture multi-couches, dont le contrôle et la capture de données sont assurés par plusieurs dispositifs distincts (plusieurs couches : routeur, pare-feu, IDS)[L.S03a, L.S03b]. Autrement dit, le routeur, le pare-feu et l'IDS de cette génération sont des éléments indépendants comme le montre la figure Fig.1.4 suivante.

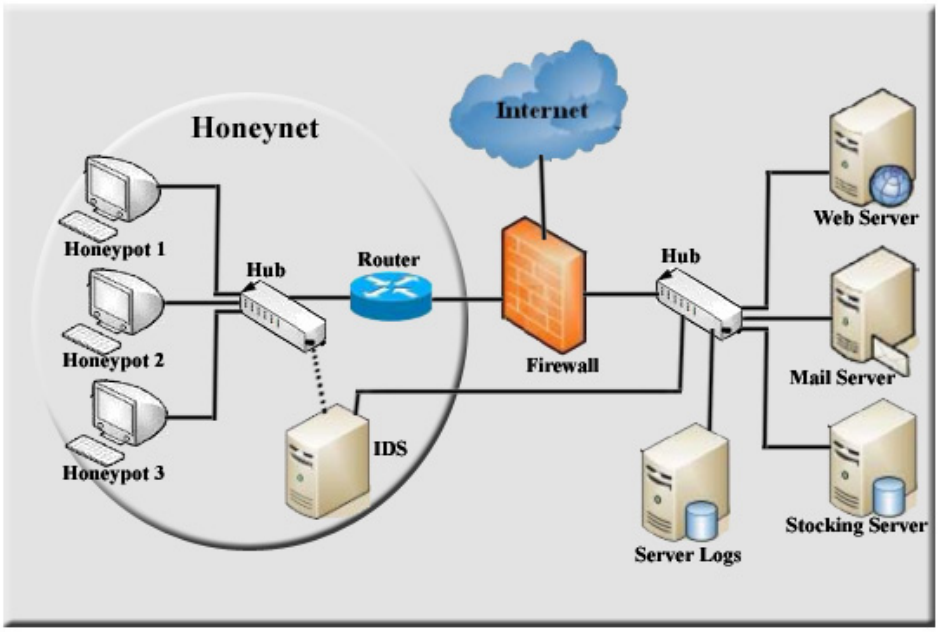


FIG. 1.4 – Architecture réseau d'un *honeynet* de la 1ère génération

Dans cette architecture le contrôle de données est assuré par :

- Le *firewall* (la première couche) qui limite le nombre de connexions entrantes,
- Le routeur (la deuxième couche) qui limite toute connexion vers l'internet et cache le firewall du réseau intérieur.

Tandis que la capture et la collection des données sont assurées par :

- Le *firewall* (la première couche) qui garde un journal de toutes les connexions de l'Internet et du *honeynet*,
- Le IDS (la deuxième couche) qui garde aussi un log de toutes les activités du réseau de *honeynet*. (Il est accessible depuis le réseau de production mais non pas depuis le *honeynet*),
- Les *honeypots* (la troisième couche) qui exploitent les journaux du système (*systme logs*).

Donc avec cette architecture, si un des dispositifs tombe en panne, la surveillance et la capture de données sont encore garanties. Par contre, son administration est relativement difficile à cause de la complexité de l'architecture réseau.

1.3.3.2 Architecture de la 2ème génération

Dans l'architecture de la deuxième génération (voir la figure Fig.1.5), les opérations de capture, contrôle et analyse des données sont assurées par un seul dispositif appelé *"Honeywall"*, ce dispositif est configuré en mode *bridge* (pont), ce qui lui permet de fonctionner sur les deux couches basses du modèle OSI (*Open System Interconnection*)[CSD99], donc il est difficilement détectable par les pirates, car il n'a pas d'adresse IP et il ne décrémente pas le champ TTL (*Time To Live*). Le but du *Honeywall* c'est le routage du trafic malveillant destiné au réseau de production vers le réseau de *honeypots*, plus le contrôle et la capture des données relatives à ce trafic [HP07c]. Cette génération a permis d'améliorer les capacités des *honeynets*, mais elle reste toujours difficile à employer et à maintenir, à cause du nombre élevé de configurations requises (configuration des $Snort\_Inline$, *Sebek*, $IPTables$, ...etc). Dans cette architecture le contrôle de données est assuré par les deux modules $IPTable$ et $Snort\_Inline$ du *Honeywall* [HP07c], le $IPTable$ est utilisé comme une première couche qui permet de limiter le nombre de trafic sortant du *honeynet*, tandis que le $Snort\_Inline$ est utilisé comme une deuxième couche qui assure la prévention des attaques connues (il bloque le trafic sortant lorsqu'il détecte une attaque connue). La même chose pour la capture de données, elle est assurée au niveau de trois couches du *honeywall* : $IPTable\ firwall$, $Sniffer$, et *Sebek*. *Sebek* est un outil très important pour capturer les données encryptées, ces derniers sont capturées au niveau des *honeypots* par les *clients Sebeks*, qui les envoient ensuite au *Sebek server* du *honeywell* via un canal UDP [HP07c].

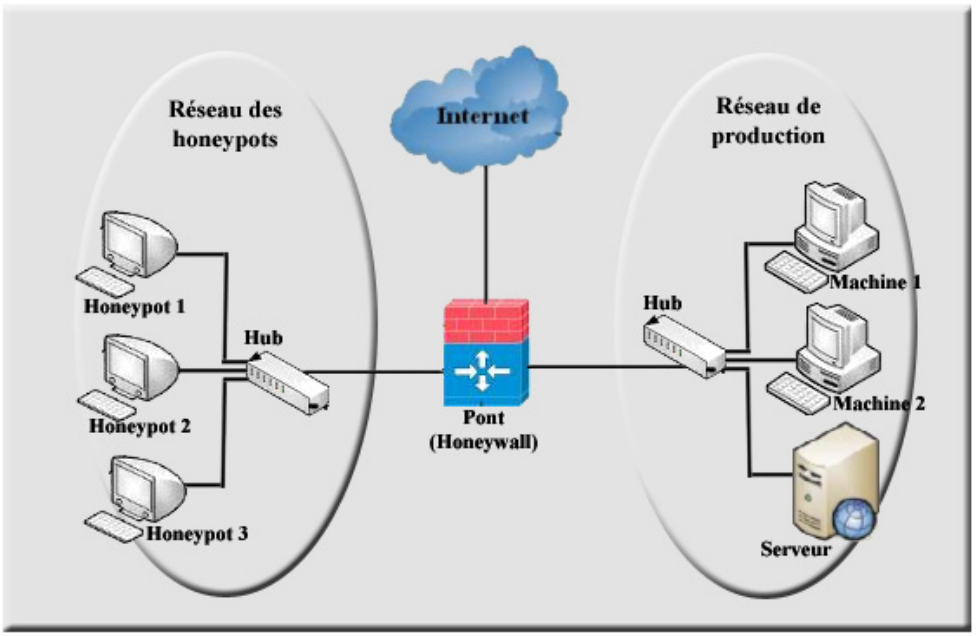


FIG. 1.5 – Architecture réseau d'un *honeynet* de la 2ème génération

1.3.3.3 Architecture de la 3ème génération

L'architecture de la troisième génération [HP07e] a été conçue principalement pour régler le problème de la difficulté de déploiement, de gestion et de maintenance des *honeynets* de la deuxième génération, en intègrant des nouveaux outils nécessaires pour la collection et l'analyse de données avec tous les outils utilisés par le *honeywall* de la deuxième génération dans un seul CD Bootable facile à installer et à configurer. La première version de ce CD est appelée *Eeyore*. Cette version a été améliorée ensuite (en 2005) à une nouvelle version appelée *"Honeynet ROO CDROM"*. Les outils de contrôle de données collectés dans ce CD sont les mêmes que ceux de la deuxième génération ($IPTable$, $Snort\_Inline$), tandis que ceux de capture et de collecte de données sont renforcés par des nouveaux outils comme $p0f$ et $Argus$. $P0f$ est utilisé pour découvrir le système d'exploitation de la machine du pirate et de celle attaquée, et $Argus$ est un outil plus puissant dans le rapatriement d'informations sur les échanges de données par le réseau. $Argus$ permet de connaître l'heure de début et de fin d'une connexion, le nombre d'octets et le nombre de paquets transmis dans chaque direction (cas d'une connexion TCP bi-directionnelle). Le *CDROM honeywall* contient aussi un outil en script *Perl* appelé $Hflow$ qui fusionne toutes les données récupérées par les outils de collection dans une base de données centrale. L'analyse de données de cette génération a été aussi renforcée par une interface web graphique appelée "*walleye*" qui peut être lancée à distance via une connexion web sécurisée (*https*).

1.3.4 Les "Honeynets" virtuels

Un *honeynet* virtuel est une simulation d'une architecture réseau complexe de *honeypots* et de mécanismes de sécurité ($firewalls$, IDS, les serveurs log, ...etc) sur une ou plusieurs machines physiques [HP07b]. Et selon la nature des machines utilisées, nous pouvons classifier les *honeynets* virtuels en deux classes principales :

1.3.4.1 "Honeynet" purement virtuel

Dans cette classe de *honeynets* virtuels, les dispositifs de contrôle et de capture des données ($firewalls$, IDS, les serveurs log, ...etc) ainsi que les *honeypots*, sont déployés sur une seule machine, ce qui permet de faciliter son administration. Toutefois, si un hacker parvient à prendre contrôle de l'hôte, tout le *honeynet* est compromis,

FIG. 1.6 – Architecture d'un *honeynet* purement virtuel

L'utilisation des *honeynets* purement virtuels nous offre plusieurs **avantages** :

- Gestion centralisée.
- Coût réduit comme seulement une machine est exigée.
- Déploiement simple, un seul système suffisant pour implémenter et connecter le *honeynet*.

Cependant, des **limitations** importantes doivent aussi être tenues en considération :

- Les logiciels qui peuvent être utilisés dans le *honeynet* sont limités par le logiciel de virtualisation utilisé, VMWare par exemple ne supporte que les Plates-formes x86, et UML supporte uniquement les systèmes linux.
- N'importe quel problème matériel affecte tout le système *honeynet*.
- La compromission du système hôte, implique la compromission de la totalité du système *honeynet*.
- La virtualisation d'un grand nombre de systèmes clients nécessite un hôte très performent.

1.3.4.2 "Honeynet" virtuel hybride

Dans cette classe de *honeynets* virtuels, les dispositifs de contrôle et de capture des données sont déployés sur une machine séparée de celle des *honeypots*, ce qui permet d'éviter le problème de compromission de tout le *honeynet* en cas où un hacker parvient à prendre contrôle de l'hôte.

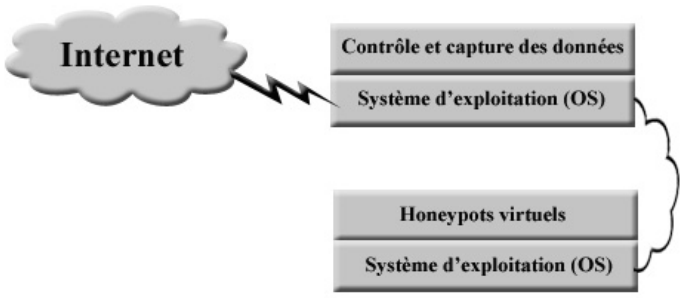


FIG. 1.7 – Architecture d'un *honeynet* virtuel hybride

Les **avantages** de cette classe de *honeynets* virtuels sont :

- Plus sûr, le contrôle et la capture de données sont assurés par un dispositif séparé des *honeypots*.
- Plus flexible, il y a un moindre de limitations impliquées par le logiciel de virtualisation utilisé.

Mais il y a aussi quelques **inconvénients** comme :

- Coût un peu élevé (plus de dispositifs à utiliser).
- Configuration compliquée.

1.4 Conclusion

Comme nous avons montré dans ce chapitre, le *honeypot* n'est pas une alternative des systèmes de détection d'intrusion (IDS) utilisés pour sécuriser un réseau vulnérable. Mais il peut être considéré comme un supplément efficace pour la détection des intrusions dans le réseau. Comme nous avons vu, le terme *honeypot* est un terme très vaste, comporte plusieurs formes différentes dépendantes des besoins et des objectifs pour lesquels il a été conçu, mais le point commun entre toutes ces formes c'est d'attraper les attaquants, et d'en collecter des informations précises. C'est cette diversité des formes qui a permis aux *honeypots* d'être étendus d'une manière prodigieuse ces dernières années, car avec cette diversité chacun peut mettre en place la forme du *honeypot* qui lui permet d'atteindre ses objectifs avec le moindre coût.

Chapitre 2

La technologie "honeynet ROO CDROM"

2.1 Introduction

La technologie *honeynet Roo* est l'une des fruits de développement du projet *honeynet* [HP07f], qui a été fondé en octobre 1999, dés cette date jusqu'à nos jours, le développement de la technologie *honeynet* n'a jamais cessé, ce qu'en a résulté l'apparition de plusieurs générations de *honeynets*. La première génération était rudimentaire, fournissant un contrôle simple de données via le $firewall$ avec quelques capacités limitées de capture de données basées sur les logs du $firewall$ et des systèmes *honeypot* installés. De plus, elle n'était pas capable d'analyser le trafic chiffré, et elle était très facile à détecter par la communauté des hackers à cause du contrôle et de capture des données qui se basaient sur des dispositifs de la couche trois du modèle OSI ($firewall$, ...etc). Tous ces problèmes, ont conduit à l'apparition de la deuxième génération en 2002, dans cette génération le contrôle et la capture de données sont assurés par un dispositif de la couche 2 du modèle OSI appelé "$Honeywall$", via des nouveaux outils comme *snort*, $snort\_inline$, *iptables*,...etc, ce qui a rendu sa détection relativement difficile, de plus, le problème d'analyse du trafic chiffré a été résolu en intégrant un nouveau module de capture des données appelé "*Sebek client*" dans les noyaux des *honeypots* installés. Dans la troisième génération qui a apparu la première fois en mai 2003 dans la version "*Eeyore*" qui a été améliorée ensuite en mai 2005 vers une nouvelle version appelée "*roo*", plusieurs améliorations ont été marquées, que ce soit dans le coté d'utilisation ou dans le coté de la capacité de capture et d'analyse des données. Cette génération a été conçue principalement pour régler le problème de la difficulté de déploiement et de maintenance des *honeynets* de la deuxième génération, en intégrant des nouveaux outils nécessaires pour la collection et l'analyse de données avec tous les outils utilisés

par le *honeywall* de la deuxième génération dans un seul CD Bootable facile à installer et à configurer. Avec l'apparition de cette nouvelle génération et les simplifications qu'elle offre, son utilisation a connu ces dernières années une expansion considérable, ils sont actuellement installés officiellement par des groupes des spécialistes internationaux de la sécurité informatique au niveau de plus de vingtaine de projets distribués sur plus de quinzaine de pays comme le montre la figure Fig.1.1, formants une alliance [HP07f, HP07a] pour partager les connaissances et les résultats obtenus au niveau de chacun d'eux afin de mieux apprendre sur les moyens de compromissions des pirates et de mieux s'en défendre. (Pour plus d'information sur cette alliance consultez le site web : *"http ://www.honeynet.org"*).

South Florida Honeynet Project, Georgia Technical Institute, Azusa Pacific University, USMA Honeynet Project, Pakistan Honeynet Project, Paladion Networks Honeynet Project (India), Internet Systematics Lab Honeynet Project (Greece), Honeynet.BR (Brazil), UK Honeynet, French Honeynet Project, Italian Honeynet Project, Portugal Honeynet Project, German Honeynet Project, Spanish Honeynet Project, Singapore Honeynet Project, China Honeynet Project.

FIG. 2.1 – Les mombres du projet *"Honeynet Research Alliance"*

2.2 Le "Honeywall"

2.2.1 Description

Le *honeywall* [HP07e] est sans doute la partie la plus importante dans la technologie *honeynet roo*, c'est la passerelle dans laquelle s'effectuent toutes les fonctionnalités de contrôle, de capture et d'analyse des données qui représentent l'objectif principal d'un *honeynet*. Le *honeywall* est un dispositif (une passerelle) de la couche 2 du modèle OSI basé sur une version minimisée du système *Fedora Core* 3, il joue le rôle d'un pont entre le réseau externe (le réseau non *honeynet* exp : Internet ou intranet) et le réseau *honeynet*, derrière ce dispositif on trouve les différents *honeypots* du *honeynet*, qui sont des systèmes avec des services et des vulnérabilités réels. Donc, tout trafic entrant ou sortant de ces derniers transite obligatoirement par le *honeywall*. Avec cette position du *honeywall*, on peut facilement capturer et logger tout trafic malveillant entrant ou sortant de notre *honeynet*, comme on peut aussi facilement contrôler tout le trafic transitant le *honeynet* (entrant ou sortant), ce qui nous permet d'empêcher facilement les attaques destinées vers les machines du monde externe à partir d'un de nos *honeypots* (rebondissement d'attaque), en limitant la quantité du trafic sortant et le nombre de ports de destination autorisés, et en bloquant le trafic correspondant à des attaques connues. Le *honeywall* est généralement constitué de trois interfaces réseau (NIC) : eth0, eth1 et eth2. La première interface (eth0) est utilisée pour le connecter au réseau externe, tandis que la deuxième (eth1) est utilisée pour le connecter au réseau interne (réseau *honeynet*). Comme c'est un pont, les systèmes interne et externe sont dans le même réseau IP. La troisième interface (eth2) est une interface facultative de la couche 3 (couche réseau) du modèle OSI, elle est utilisée pour l'administration distante du *honeywall* (ceci inclus le transfert de tous les fichiers de log, ou de données capturées vers un point central). L'avantage majeur de cette architecture c'est l'extrême difficulté de la détection de la passerelle *honeywall*, car au niveau de cette dernière il n'y a ni saut de routage, ni modification du champ TTL (*Time To Live*), ni adresse MAC (*Media Access Control*) y associée. De plus, le déploiement du *honeynet* avec cette architecture est devenu très facile avec la centralisation du contrôle et de la capture des données dans la passerelle *honeywall*.

2.2.2 Les composants d'un "honeywall"

Pour pouvoir assurer un bon fonctionnement du *honeynet*, plusieurs outils ont été intégrés dans la passerelle *honeywall*, chacun de ces outils est conçu principalement pour assurer une des trois fonctionnalités de base de la technologie *honeynet* (contrôle des données, capture et collection des données, analyse des données).

2.2.2.1 Composants de contrôle des données

a. Iptable

Iptable [NH07b] est un pare-feu linux intégré dans le *kernel* du *honeywall*, il est utilisé pour contrôler le trafic qui circule dans le *honeynet* en limitant le nombre de connexions sortantes qu'un attaquant puisse initier depuis un *honeypot*. Il compte le nombre de connexions sortantes, et quand la limite définie est atteinte, il bloque toutes les connexions ultérieures. La limite des connexions sortantes autorisées est définie dans le script "$rc.firewall$", où nous définissons combien de connexions TCP, UDP ou ICMP (*Internet Control Message Protocol*) sortantes qui peuvent être établies par un pirate. La limite définie dans ce script dépend d'une manière directe au risque qu'on est prêt à prendre.

b. Snort_inline

Snort_inline [SIH07] est une variante de *Snort* [SH07a], utilisé dans le *honeywall* pour contrôler les données du *honeynet*. Les paquets sortants d'un *honeypot* sont traités d'abord par *Iptable*, puis ils sont envoyés vers *Snort_inline* pour les analyser. Une fois l'analyse est effectuée, *Snort_inline* rend ces paquets à l'*Iptable* pour continuer leur routage. L'analyse effectuée par *Snort_inline* consiste à tester si un des paquets sortants correspond à une attaque connue qui est définie dans sa base de règles (base de signatures), si c'est le cas, une alerte est déclenchée (comme dans un IDS traditionnel), et le paquet est immédiatement bloqué ou modifié (le rendre inefficace).

c. Swatch

Swatch [SH07b] est un outil de supervision automatique complet, capable de notifier les administrateurs lors d'une attaque potentiellement réussie. *Swatch* surveille les fichiers de log à la recherche de motifs décrits dans un fichier de configuration. Quand un motif est identifié, il propage nativement l'alerte par mail, bip système, appel téléphonique et peut être étendu pour lancer d'autres commandes ou programmes.

2.2.2.2 Composants de capture et de collection des données

a. Libpcap

Libpcap [TH07] est une librairie fournit à l'utilisateur une série de fonctions de capture et de filtrage de paquets. Dans le *honeywall*, *Libpcap* est utilisée par *Snort*, *Argus*, $P0f$ et *Sebek Server* pour capturer le trafic qui cercle dans le *honeynet*.

b. Sebek Server

Sebek Server [Hp07g] est un module noyau du *honeywall*, utilisé pour capturer et collecter les données envoyées par *Sebek client*. *Sebek client* [Hp07g] est un module noyau caché, indétectable par les pirates, installé dans le noyau du *honeypot*, il est utilisé pour capturer l'activité du pirate directement sur le système compromis. Sa position dans le noyau du *honeypot* est très importante pour capturer l'activité du pirate lorsqu'il utilise des protocoles cryptés comme SSH (*Secure SHell*) ou 3DES (*Triple Data Encryption Standard*) pour communiquer avec les ordinateurs compromis, car ce genre de communications est incompréhensible au niveau du *honeywall* à cause du cryptage imposé. Les informations rassemblées par *Sebek client* ne doivent pas être stockées sur le *honeypot*, où elles pourraient être découvertes par l'attaquant. *Sebek client* transmet ces informations via UDP vers une machine à l'écoute (*honeywall*), ces paquets sont invisibles et incapturables par les pirates, car le *honeypot* est modifié de telle manière qu'il ne puisse ni voir ni capturer les paquets avec une adresse MAC source pré-configurée. *Sebek client* envoie alors les données capturées vers la passerelle avec l'adresse MAC source spécifiée. Comme l'ensemble des *honeypots* est contrôlé par *Sebek client*, aucun d'entre eux ne peut être utilisé pour capturer les saisies clavier envoyées par ces derniers.

L'architecture et le fonctionnement du *client/serveur Sebek* sont illustrés dans la figure Fig. 2.2 suivante.

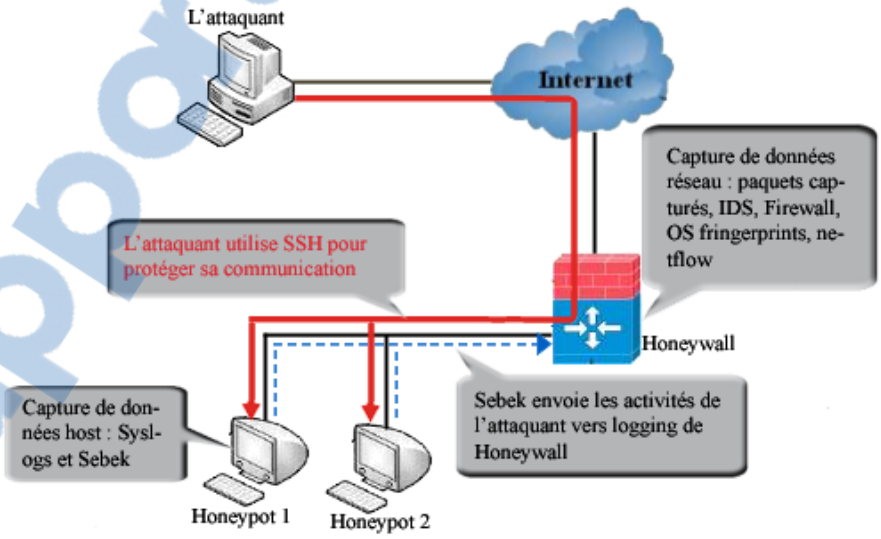


FIG. 2.2 – Architecture et fonctionnement de *Sebek*

c. Snort

Snort [TSP06, SH07a] est un système de détection d'intrusion (IDS) [MH03], il permet de capturer et d'analyser le trafic réseau de type IP. Il peut être configuré pour fonctionner en trois modes différents : le mode $sniffer$, le mode *packet logger* et le mode détecteur d'intrusion réseau (NIDS). Dans le *honeywall*, *Snort* est utilisé pour capturer les données circulant sur le réseau *honeynet*, donc le mode utilisé est le mode $sniffer$, dans ce mode, tout trafic IP capturé sera sauvegardé dans un fichier de log au format *tcpdump*. Il est activé sur l'interface interne $eth1$ du *honeywall*, afin qu'il ne capture que les données relatives au *honeynet*.

d. P0f

$P0f$ [PH07] est un outil passif de prise d'empreintes digitales des systèmes d'exploitation, c'est-à-dire il prend l'empreinte digitale du système d'exploitation sans initier aucune connexion ! Ainsi, uniquement en observant les flux qui transitent sur un réseau. Il est utilisé dans le *honeywall* pour découvrir le système d'exploitation de la machine du pirate et de celle attaquée, il analyse les options de TCP pour garantir avec plus ou moins de fiabilité le système d'exploitation utilisé. $P0f$ propose quatre modes différents de $fingerprinting$:

- *Incoming connection fingerprinting (SYN mode, mode par défaut)* : Pour identifier l'OS (*Operating System*) installé sur un hôte qui se connecte à notre plateforme,
- *Outgoing connection fingerprinting (SYN+ACK mode)* : Pour déterminer le système d'exploitation de l'hôte sur lequel on se connecte,
- *Outgoing connection refused fingerprinting (RST+ mode)* : Pour une prise d'empreinte d'un système qui rejette notre trafic,
- *Established connection fingerprinting (stray ACK mode)* : Pour examiner des sessions en cours sans interférence de sa part.

e. Argus

Argus [AH07, HSBR99] est un logiciel libre qui permet de collecter des informations sur les paquets envoyés entre deux hôtes, comme le nombre de paquets et d'octets du flux (in/out), la durée du flux (date de début, date de fin),...etc. Toutes ces informations sont générées directement à partir du trafic capturé par *Argus* lui-même, et aussi à partir du trafic capturé par un autre sniffer comme $TCPdump$, *Snort* ...etc. L'utilisation d'$Argus$ dans le *honeywall* est pour connaître l'heure de début et de fin d'une connexion, le nombre d'octets et le nombre de paquets transmis dans chaque direction (cas d'une connexion TCP bi-directionnelle).

f. Hflow

Hflow [EC05] est un script *Perl* qui permet de fusionner et d'enregistrer les données récupérées par les outils de capture et de collection de données (*Snort*, *Sebek*, *P0f*, *Argus*) dans une seule base de données centralisée. Le schéma modélisant les opérations depuis la capture des trames jusqu'à leur stockage dans la base de données est donné par la figure Fig. 2.3 suivante.

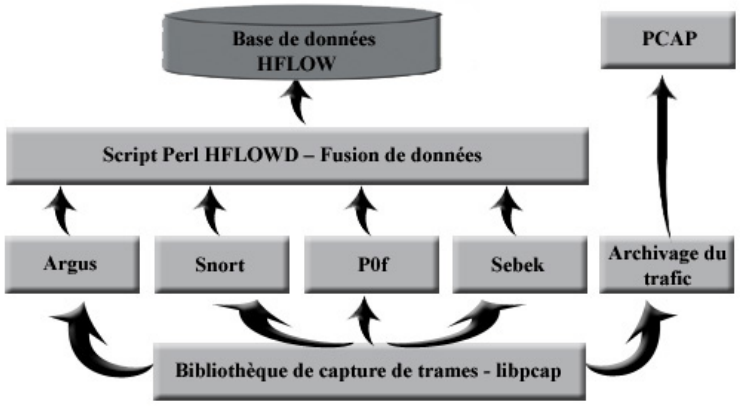


FIG. 2.3 – Capture et fusion des données dans un *honeynet* GenIII

2.2.2.3 Composants de configuration et d'analyse des données

a. Walleye

Walleye [HP07e, HP06] est une interface web qui permet à l'utilisateur de se connecter à distance au système *honeywall*, via une connexion web sécurisée (connexion *https*) basée sur le protocole SSL (*Secure Socket Layer*). Elle permet à l'utilisateur de contrôler, de configurer et de maintenir le système *honeywall*, ainsi que de visualiser et d'analyser graphiquement toutes les informations et les données capturées par le *honeywall*, en utilisant des simples cliques. L'utilisation du *Walleye* nécessite l'existence de la troisième interface réseau (interface d'administration distante) dans le *honeywall*, car le *Walleye* est inaccessible localement.

b. Dialog Menu

Dialog Menu [HP07e, HP06] est un menu graphique utilisé pour configurer et contrôler graphiquement les différents composants du *honeywall* via des simples choix. Ce menu peut être utilisé, soit localement sur le *honeywall*, soit à distance.

c. HWCTL

HWCTL [HP07e, HP06] est une ligne de commande qui permet à l'utilisateur du *honeywall* de gérer et de configurer les variables systèmes utilisées par les différents composants du *honeywall*, à l'aide d'un ensemble de commandes simples à employer et à conserver. Comme *Dialog Menu*, cette ligne de commande est utilisée, soit localement sur le *honeywall*, soit à distance via un accès SSH.

2.3 VMWare

2.3.1 Description

VMWare [vH07, HP07h] est un environnement logiciel qui offre la possibilité d'émuler plusieurs machines virtuelles sur une seule machine physique, dont chacune de ces machines virtuelles est capable de réaliser l'installation et la configuration complète d'un système d'exploitation invité (guest OS) et de ses applications. Avec *VMWare* les ressources matérielles de la machine physique hôte sont présentées sous forme de ressources logiques aux machines virtuelles, c'est-à-dire le système d'exploitation d'une machine virtuelle n'a aucun accès direct aux ressources physiques du matériel. Donc avec *VMWare*, plusieurs machines virtuelles peuvent partager les mêmes ressources physiques du matériel du système hôte.

2.3.2 Mise en réseau virtuel

Les machines virtuelles créées par *VMWare* peuvent être reliées entre elles dans un ou plusieurs réseaux virtuels, grâce à l'ensemble des dispositifs virtuels : *switchs*, *hubs*, serveur *NAT* (*Network Addresses Translation*), serveur *DHCP* (*Dynamic Host Configuration Protocol*), ...etc, fournis par *VMWare* [vH07]. Il fournit huit commutateurs (*switchs*) virtuels notés *VMnet*0, *VMnet*1, ..., *VMnet*7, dont les machines de chaque réseau virtuel sont reliées à un de ces commutateurs.

Les réseaux virtuels fournis par *VMWare* sont de trois types :

2.3.2.1 Réseau virtuel ponté (*bridged*)

Dans ce type de réseaux, la machine virtuelle est vue comme connectée directement au réseau local réel (physique), puisqu'elle est reliée directement au commutateur virtuel *VMnet*0[1] qui est relié par défaut à l'adaptateur Ethernet de la machine hôte via un

[1]par défaut le commutateur virtuel d'un réseau ponté est *VMnet*0

pont virtuel, comme le montre la figure Fig. 2.4 suivante.

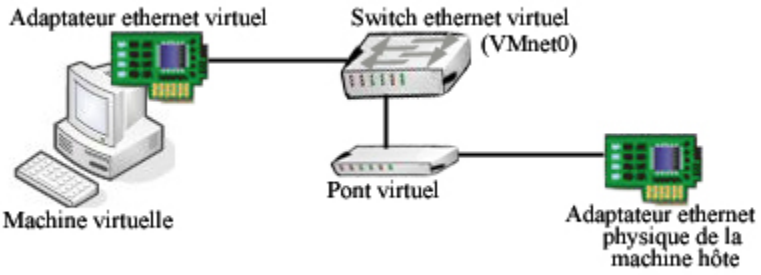


FIG. 2.4 – La mise en réseau ponté

Un réseau virtuel ponté est considéré comme une extension du réseau physique de la machine hôte, donc les machines virtuelles de ce réseau doivent prendre des adresses IP libres du réseau physique, c'est à dire une machine physique et une machine virtuelle ne doivent pas avoir la même adresse IP.

2.3.2.2 Réseau virtuel hôte uniquement ($host - only$)

Un réseau virtuel hôte uniquement est un réseau virtuel privé non connecté à l'Internet. Le commutateur virtuel par défaut de ce type de réseaux est le $VMnet1$. Les machines virtuelles de ce type de réseaux ne peuvent communiquer avec aucune autre machine (physique ou virtuelle) d'un autre réseau. La machine hôte est reliée à un réseau virtuel de ce type via un adaptateur Ethernet virtuel, comme le montre la figure Fig. 2.5 suivante.

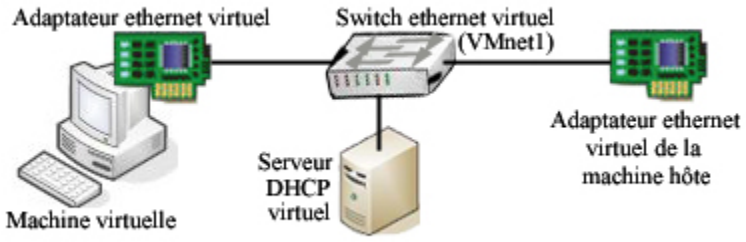


FIG. 2.5 – La mise en réseau hôte uniquement ($host - only$)

Les adresses IP des machines virtuelles de ce type de réseaux sont fournies par le serveur DHCP virtuel relié à leur réseau.

2.3.2.3 Réseau virtuel avec translation d'adresses réseau (NAT)

Dans ce type de réseaux, la machine virtuelle est connectée au réseau local[2] de la machine hôte via une passerelle virtuelle de translation d'adresses simulée par $VMWare$, comme le montre la figure Fig. 2.6 suivante.

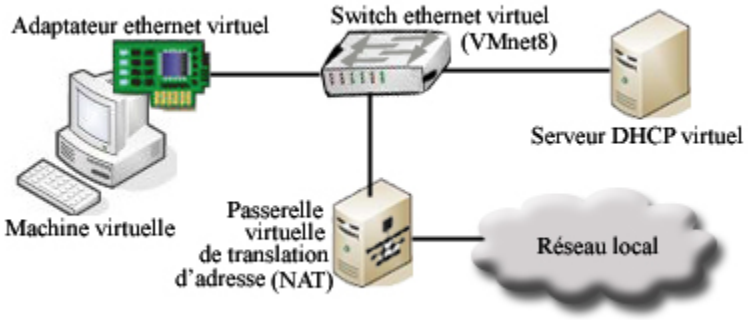


FIG. 2.6 – La mise en réseau avec translation d'adresses (NAT)

Le commutateur virtuel par défaut de ce type de réseaux est le $VMnet8$. Chaque machine virtuelle de ce type de réseaux prend son adresse IP auprès du serveur DHCP virtuel qu'elle y est relié.

Le périphérique NAT virtuel du réseau virtuel transmet les paquets de données des machines virtuelles vers leurs destinations dans le réseau externe, en remplaçant leurs adresses IP sources par son adresse IP, et lorsqu'il reçoit des paquets de données de l'extérieur, il identifie la machine virtuelle concernée et les lui envoie.

2.4 Expérimentation

2.4.1 Environnement d'expérimentation

L'installation de notre *Honeynet* a été réalisée dans le réseau local de notre université, qui est un réseau privé d'adresse IP 172.16.0.0/16, dont sa passerelle est protégée par un pare-feu qui interdit tout trafic entrant vers le réseau privé, sauf celui destiné à une machine de la zone démilitarisée (DMZ). En effet, pour pouvoir recevoir des attaques externes, le *honeynet* doit être placé dans la zone démilitarisée (DMZ) où il y a moins de restrictions, mais malheureusement l'administrateur réseau au niveau d'université a refusé totalement cette idée, ce qui nous a obligé de placer le *honeynet* dans

[2]réseau physique de la machine hôte

l'intranet, et donc nous ne pouvons capturer que les attaques qui s'effectuent au niveau de l'intranet. Heureusement, notre intranet couvre deux pôles universitaires avec un grand nombre de machines (plus de 500 machines) ce qui croit la probabilité d'avoir un grand nombre d'attaques, surtout les attaques résultantes des machines compromises par des botnets, des vers, des chevaux de troie et des virus. L'architecture générale de notre environnement est illustrée dans la figure Fig. 2.7 suivante.

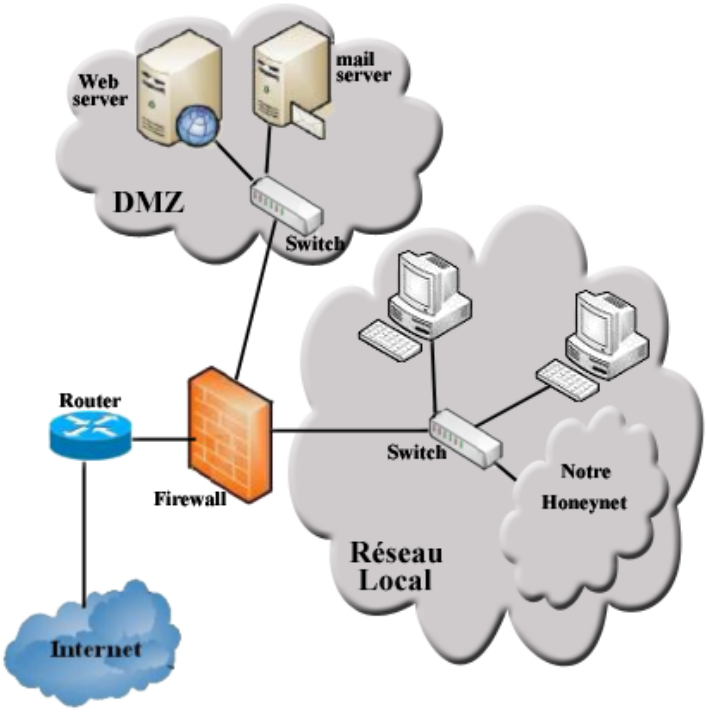


FIG. 2.7 – Environnement de notre expérimentation

2.4.2 Ressources d'expérimentation

Pour diminuer le coût des ressources exigées pour la réalisation de notre expérimentation, nous avons décidé d'utiliser un *honeynet* purement virtuel, dont tous ses composants sont installés sur des machines virtuelles (clients) dans une seule machine physique (hôte). La création des machines virtuelles est assurée par un logiciel de virtualisation comme $VMWare$ [vH07] ou UML [UH07], dans notre cas nous avons opté pour $VMWare$ bien que UML est plus puissant, mais malheureusement il ne fonctionne qu'avec GNU/Linux comme système invité, par contre $VMWare$ nous offre la possibilité de faire fonctionner plusieurs types de systèmes d'exploitations (Windows, Linux, ...etc) et leurs applications simultanément dans un seul PC.

Cependant, l'installation de plusieurs systèmes sur une seule machine via $VMWare$ nécessite une machine hôte puissante, pour cette raison nous avons utilisé une machine avec un processeur $Pentium$ 4 de vitesse 2.8 GHZ, et une mémoire RAM de taille 512 Mo, et un disque dur de taille 40 Go, et disposant de deux adaptateurs Ethernet, le premier est utilisé pour relier les *honeypots* avec le monde externe, et le deuxième est utilisé pour relier l'interface de management du *honeywall* avec la machine distante de management, cette séparation entre le trafic des attaquants et celui de la management est effectuée pour donner à ce dernier un plus de sécurité. Le système d'exploitation installé dans la machine hôte est un système *Red Hat Enterprise Linux* 4 ($RHEL4$).

Le nombre de machines virtuelles créées dans la machine hôte est de trois machines, dont deux machines représentent deux *honeypots*, et une machine pour installer le *honeywall*. Le premier *honeypot* est un système $Windows\ XP$ qui offre plusieurs services comme : *http*, *ftp*, *telnet*, ...etc, alors que le deuxième *honeypot* est un système *Red Hat Enterprise Linux* 4 qui offre également plusieurs services comme : SSH ($OpenSSH$), $HTTP$, $Telnet$, $Sendmail$, FTP, ...etc. Dans chacun de ces deux *honeypots*, nous avons installé l'outil *Sebek client* (*Sebek-Win32-3.0.4* pour le premier *honeypot* et *sebek-linux26-3.2.0b.tar.gz* pour le deuxième). Le *honeywall* utilisé est le *ROO CDROM* (roo-1.0.hw-189) qui est basé sur une version minimisée du système $Fedora\ Core$ 3. La machine du *honeywall* est une machine virtuelle avec une mémoire RAM de taille 276 Mo, et un disque dur de taille 20 Go, alors que la machine du premier *honeypot* ($Windows\ XP$) est une machine virtuelle avec une mémoire RAM 96 Mo, et un disque dur 4 Go, et celle du deuxième *honeypot* (*Red Hat Enterprise Linux* 4) est une machine virtuelle avec une mémoire RAM 128 Mo, et un disque dur 4 Go.

2.4.3 Architecture d'expérimentation

Comme il est illustré dans la figure Fig. 2.8 dans la page suivante, l'architecture de notre *honeynet* virtuel est une architecture purement virtuelle, constituée de trois sous-réseaux virtuels, dont chacun d'eux est relié à une des interfaces réseau du *honeywall*. Le premier sous-reseau ($VMNet0$) est un réseau bridgé, accessible du réseau externe[3], il est conçu pour relier la première interface réseau du *honeywall* ($eth0$) au réseau externe. Le deuxième sous-réseau ($VMNet1$) c'est le réseau privé ($host-only$) des *honeypots* de notre *honeynet*, pour pouvoir y accéder de l'extérieur on doit passer par le *honeywall* qui y est relié via sa deuxième interface réseau ($eth1$). Le troisième sous-réseau ($VMNet2$) c'est un autre réseau bridgé inaccessible depuis l'intranet, il est relié à la deuxième interface réseau physique ($eth1$) de la machine hôte, qui est lui même reliée directement à la machine d'administration via un câble réseau croisé, ce

[3]réseau physique d'intranet

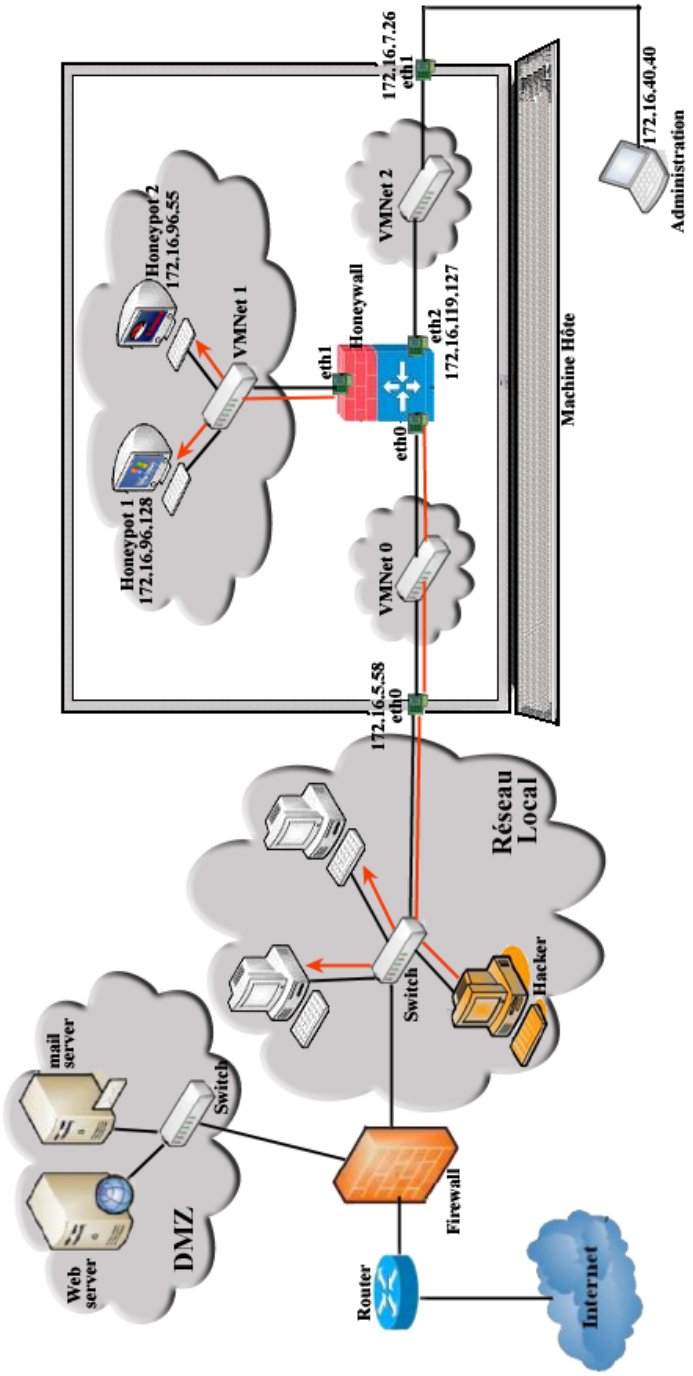


FIG. 2.8 – Architecture de notre première expérimentation

troisième sous-réseau est conçu pour relier la troisième interface du *honeywall* (interface d'administration *eth*2) à un réseau isolé invisible pour les hackers, pour assurer une administration distante sécurisée.

Le trafic des hackers est acheminé directement vers le réseau privé des *honeypots* via le *honeywall*.

2.5 Analyse du trafic capturé

Dans cette partie, nous allons essayer d'analyser les différents trafics capturés durant le premier mois d'expérimentation de notre *honeynet ROO CDROM*, et puisque notre travail se concentre principalement sur le problème de la détection du *honeynet ROO CDROM* à cause de la politique utilisée pour contrôler le trafic sortant (rebondissement d'attaques), nous allons consacrer une sous partie de cette analyse pour le trafic sortant, en essayant de collecter le maximum d'informations sur ce genre de trafics, pour pouvoir obtenir des idées qui peuvent conduire à la résolution de ce problème. Ensuite, tout trafic semble intéressant sera analysé en détail dans la partie d'analyse détaillée.

2.5.1 Analyse statistique

2.5.1.1 Trafic total (entrant et sortant)

Juste quelques minutes après la mise en place du *honeynet* dans l'intranet, plusieurs attaques y ont été destinées. Une analyse du trafic généré par ces attaques durant le premier moins d'expérimentation montre que la majorité de ces trafics (99.82 %) sont en relation avec le premier *honeypot* (Windows XP- 172.16.96.128), et que 80.95% de ces trafics sont des rebondissement d'attaques (trafics sortants), comme le montre le tableau Tab. 3.1 et le diagramme de la figure Fig. 2.9.

	Honeypot 1 (Windows XP) 172.16.96.128	Honeypot 2 (RHEL 4) 172.16.96.55	Total
Trafic entrant (Inbound)	15976	107	16083
Trafic sortant (Outbound)	67418	36	67454
Total	83394	143	83537

TAB. 2.1 – Quantité du trafic entrant et sortant pour chaque *honeypot*

Ce résultat montre que le premier *honeypot* (*Windows XP*) est compromis par des vers ou des botnets, qui cherchent à se propager vers les autres machines du réseau, ce

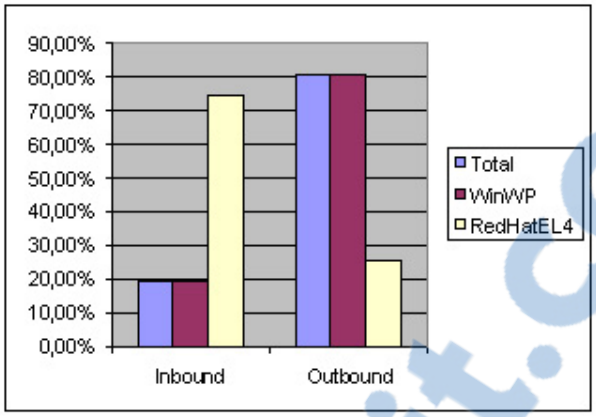


FIG. 2.9 – Trafic entrant et sortant pour chaque *honeypot*

qui explique tout ce trafic sortant de ce premier *honeypot*, au contraire du deuxième *honeypot* (*RedHat Enterprise Linux* 4), où le trafic sortant est relativement faible par rapport au trafic entrant.

a. Trafic capturé en fonction des protocoles utilisés

Le trafic capturé en fonction des protocoles utilisés dans la couche transport est illustré dans le diagramme de la figure Fig. 2.10 suivante.

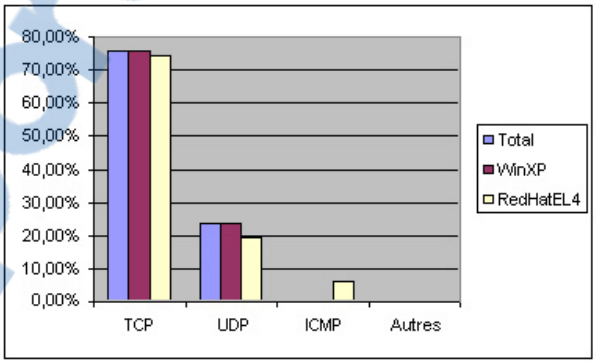


FIG. 2.10 – Trafic capturé par protocole

Comme il est illustré dans ce diagramme, le protocole dominant dans la plupart du trafic capturé est le protocole TCP, avec un pourcentage de 75.55% pour le premier *honeypot* (*WindowsXP*), et 74.12% pour le deuxième *honeypot* (*RedHat Enterprise Linux* 4). Tandis que le protocole UDP domaine sur le reste du trafic avec un pourcentage de 23.64% pour le premier *honeypot*, et 19.58% pour le deuxième *honeypot*.

On remarque aussi que le pourcentage du trafic total est très proche à celui du premier *honeypot* (*Windows XP*), et ça à cause de la plupart du trafic (99.82%) qu'était destiné vers ce dernier.

b. Trafic capturé en fonction des ports de destination

Lorsque on prend le trafic capturé en fonction des ports de destination, on peut voir le résultat indiqué dans le diagramme de la figure Fig. 2.11 suivante.

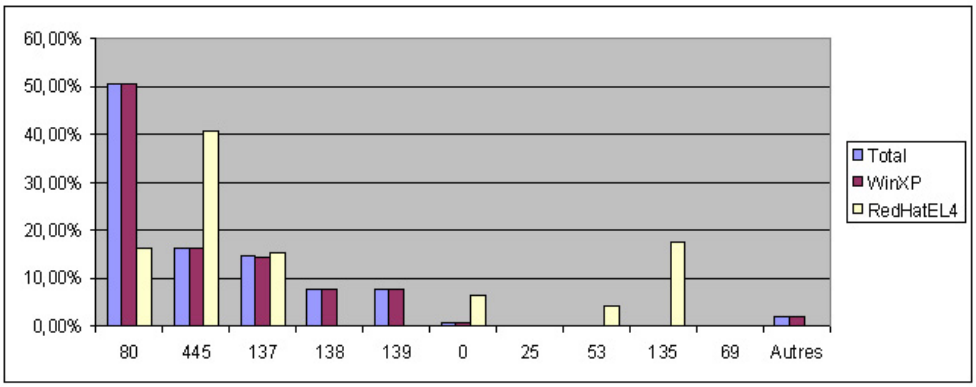


FIG. 2.11 – Trafic capturé en fonction des ports de destination

Comme il est illustré dans ce diagramme (Fig. 2.11), il est évident que les ports de destination du trafic capturé sont relatifs au système qui est installé dans le *honeypot*. On remarque par exemple que le port le plus attaqué dans le *honeypot RHEL* 4 est le port 445 qui est réservé pour le service $microsoft - ds$[4], avec un pourcentage de 40,55%, tandis que le port le plus attaqué dans le *honeypot Windows XP* est le port 80 qui est réservé pour le service *http*, avec un pourcentage de 50,64%.

c. Trafic et Alertes générées en fonction des ports de destination

Pour les alertes générées par *Snort_inline*, la remarque la plus importante qui est remarquée dans le résultat obtenu, c'est l'absence des alertes pour le *honeypot RHEL*4, et ça peut être à cause du nombre faible des machines linux qui sont installées dans l'intranet ce qui a diminué les tentatives d'attaque vers ce type de systèmes, par contre plusieurs alertes concernant le *honeypot Windows XP* ont été générées. Et comme il est illustré dans le tableau Tab. 3.4 et le diagramme de la figure Fig. 2.12 dans la page suivante, le port qui génère le plus grand nombre d'alertes est le port 139 (65,86%) qui est réservé pour le service $netbios - ssn$[5] (*NETBIOS Session Service*), dont parmi les

[4]service de partage de ressources
[5]service d'accès aux ressources partagées

6415 flux de ce port, il y a 3104 alertes (1 alerte pour chaque 2 flux). Le deuxième port qui génère beaucoup d'alertes est le port 445 (22,06%), dont parmi les 13421 flux de ce port, il y a 1040 alertes (1 alerte pour chaque 13 flux). Donc 87,92% d'alertes ont été générées par ces deux ports seuls. La deuxième remarque importante qu'on peut remarquer d'après ce résultat, c'est que plus de 96% d'alertes générées sont des alertes générées par le trafic TCP, et aucune alerte n'a été générée par les deux ports UDP : 137 (netbios-ns : *NETBIOS Name Service*) et 138 (netbios-dgm : *NETBIOS Datagram Service*), malgré la grande quantité du trafic capturé qui a été destinée vers ces deux ports (12252 flux pour le premier et 6501 flux pour le deuxième).

Port	Honeypot 1 (Windows XP) 172.16.96.128	
	trafic	alerte
netbios-ns (137/UDP)	12258	0
http (80/TCP)	42233	359
netbios-ssn (139/TCP)	6415	3104
netbios-dgm (138/UDP)	6501	0
0/ICMP	569	0
microsoft-ds (445/TCP)	13421	1040
tftp (69)	11	88
smtp (25)	198	5
domain (53/UDP)	145	0
epmap (135/TCP)	27	31
autres	1616	86

TAB. 2.2 – Nombre d'alertes générées en fonction des ports de destination

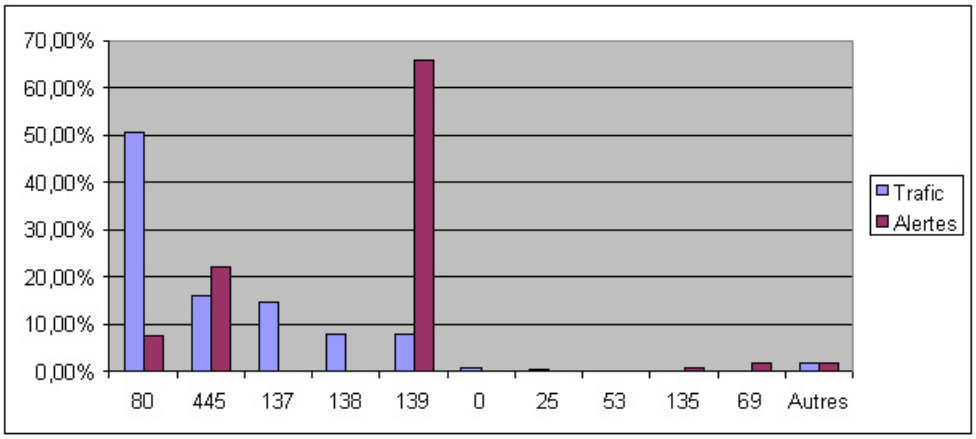


FIG. 2.12 – Trafic et Alertes générées en fonction des ports de destination

2.5.1.2 Trafic sortant

a. Trafic sortant en fonction des protocoles utilisés

Le trafic sortant de notre *honeynet* en fonction des protocoles utilisés dans la couche transport est illustré dans le diagramme de la figure Fig. 2.13 suivante.

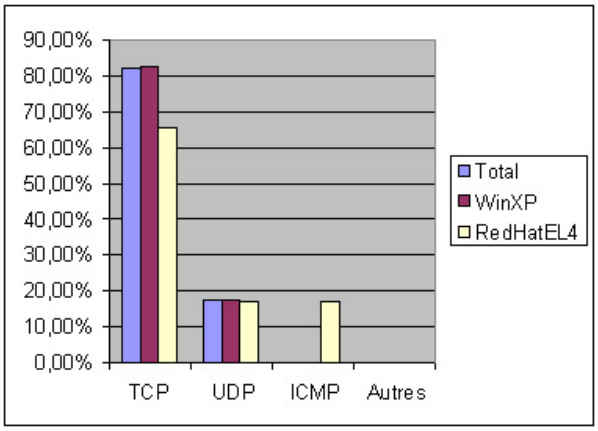


FIG. 2.13 – Trafic sortant en fonction des protocoles utilisés

La même chose comme pour le trafic total, le protocole dominant dans la plupart du trafic sortant est le protocole TCP avec un pourcentage de 82,38% pour le *honeypot* $WindowsXP$ et de 65,71% pour le *honeypot* $RedHat$.

b. Trafic sortant en fonction des ports de destination

Pour le trafic sortant de notre *honeynet* en fonction des ports de destination, et comme il est illustré dans le diagramme de la figure Fig. 2.14 dans la page suivante, le port le plus attaqué dans les deux *honeypots* ($Windows\ XP$ et $RHEL$ 4) est le port 80 avec un pourcentage de 62,71% pour le *honeypot* $Windows\ XP$ et 65,11% pour le *honeypot* $RedHet$, Par contre dans le reste des ports on remarque un renversement des ports attaqués par rapport aux deux *honeypots*, où on remarque que les ports les plus attaqués dans le *honeypot* $Windows\ XP$ après le port 80 sont respectivement les ports 445, 138 et 137, tandis qu'il n'y a aucune attaque correspondante à ces trois ports dans le *honeypot* $RHEL4$, et la même chose pour les deux ports 53 et 0/ICMP les plus attaqués dans le *honeypot* $RHEL4$, où il n'y a aucune attaque correspondante à ces deux ports dans le *honeypot* $Windows\ XP$. On peut remarquer aussi une absence quasiment total du trafic sortant pour le reste des ports (25, 69, 135 et 139).

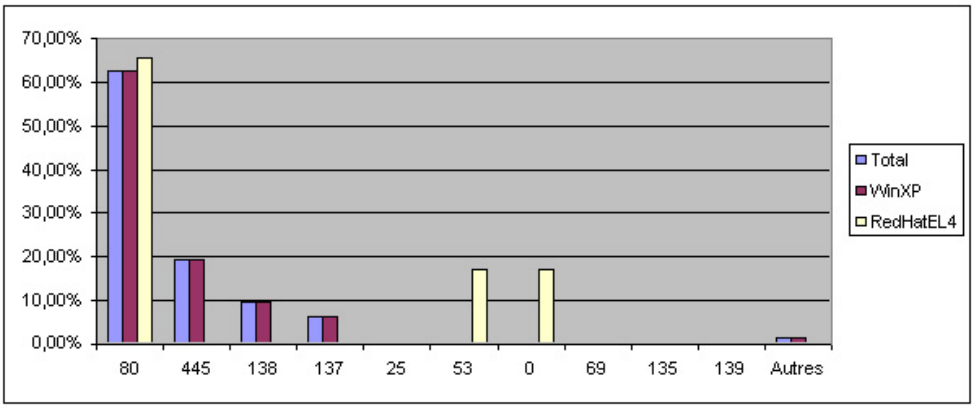


FIG. 2.14 – Trafic sortant en fonction des ports de destination

c. Alertes générées par le trafic sortant en fonction des ports de destination

La distribution des alertes générées par *Snort_inline* pour le trafic sortant en fonction des ports de destination est illustré dans le tableau Tab. 2.3 et le diagramme de la figure Fig. 2.15.

Port	Honeypot 1 (Windows XP) 172.16.96.128	
	trafic	alerte
netbios-ns (137/UDP)	4264	0
http (80/TCP)	42233	371
netbios-ssn (139/TCP)	0	0
netbios-dgm (138/UDP)	6463	0
0/ICMP	25	0
microsoft-ds (445/TCP)	12992	586
tftp (69)	11	88
smtp (25)	198	5
domain (53/UDP)	133	0
epmap (135/TCP)	0	0
autres	1099	63

TAB. 2.3 – Nombre d'alertes du trafic sortant par port de destination

Comme il est illustré dans le diagramme de la figure Fig. 2.15 dans la page suivante, le port qui génère le plus grand nombre d'alertes est le port 445 (52,65%), dont parmi les 12992 flux de ce port, il y a 586 alertes (1 alerte pour chaque 22 flux). Le deuxième port qui génère beaucoup d'alertes est le port 80 (21,99%), parmi les 42233 flux de ce port, il y a 371 alertes (1 alerte pour chaque 114 flux). Tout ça c'est par rapport aux

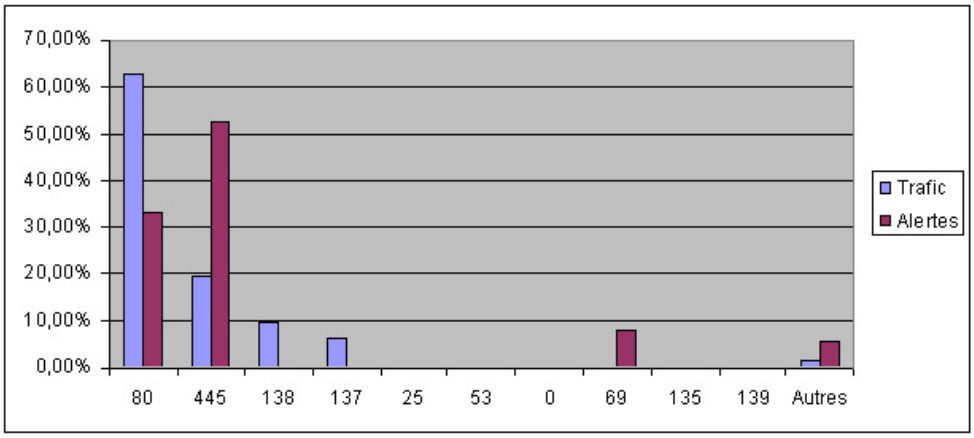


FIG. 2.15 – Alertes et Trafics sortants en fonction des ports de destination

alertes totales, mais lorsque on prend les alertes générées par rapport au trafic total de chaque port, on trouve que le port génèrant le plus grand nombre d'alertes est le port 69 (tftp), dont parmi les 11 flux de ce port, il y a 88 alertes (8 alertes pour chaque 1 flux).

2.5.2 Analyse détaillée

D'après le trafic capturé durant toute cette première expérimentation, on peut distinguer deux classes de trafics, la première classe représente le trafic des attaques inconnues (sans alertes)(94,35% du trafic total), et la deuxième classe représente le trafic des attaques connues (avec alertes) (05,65% du trafic total) dont leurs signatures sont enregistrées dans la base de données du *Snort_inline*. Dans cette partie nous allons analyser en détail quelques trafics de chacune de ces deux classes.

2.5.2.1 Trafic des attaques inconnues (sans alertes)

Pendant l'analyse du trafic de cette classe, nous avons remarqué que parmi les flux de cette dernière, il y a des flux qui effectivement ne conduisent à aucun dommage sur la machine attaquée (Trafic de faible risque), en même temps nous avons remarqué l'existence de certains flux très dangereux (Trafic de risque élevé), qui peuvent conduire à un dommage considérable sur la machine attaquée, cependant, malgré cette dangerosité, *Snort_inline* n'a généré aucune alerte.

a. Trafic de faible risque

a.1 Trafic TCP

Pour pouvoir analyser le trafic capturé en détail nous avons utilisé l'outil *Ethereal* [Com07], avec cet outil on peut examiner, interpréter et anatomiser la structure des paquets de tout trafic soupçonné d'une manière facile, lisible et efficace, en les décodant jusqu'à la couche applicative. L'utilisation de cet outil dans l'analyse du trafic capturé par notre *honeywall*, nous a permis de remarquer et de déduire plusieurs choses intéressantes concernant la nature de ce trafic, une de ces déductions c'est que la plupart des flux TCP similaires à celui de la figure Fig.2.16, sont des flux de scan de ports, utilisés par l'attaquant pour collecter des informations sur la machine cible afin de déterminer si cette dernière mets certaines conditions ou bien non.

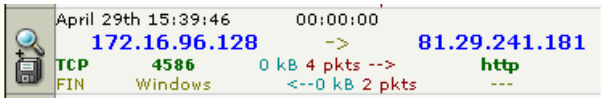


FIG. 2.16 – Exemple d'un trafic de scan du port *http*

Le trafic illustré dans cette figure (Fig.2.16), c'est un trafic TCP constitué de 6 paquets, 4 paquets envoyés par la machine d'attaquant (notre *honeypot* d'adresse IP 172.16.96.128) et 2 paquets envoyés par la machine attaquée (machine d'adresse IP 81.29.241.181). Lorsque on analyse ce trafic en utilisant *Ethereal* nous allons obtenir le résultat indiqué dans la figure Fig.2.17 suivante.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.96.128	81.29.241.181	TCP	4586 > http [SYN] Seq=0 Len=0 MSS=1460
2	0.003545	81.29.241.181	172.16.96.128	TCP	http > 4586 [SYN, ACK] Seq=0 Ack=1 Win=6
3	0.006032	172.16.96.128	81.29.241.181	TCP	4586 > http [ACK] Seq=1 Ack=1 Win=64240 L
4	0.006039	172.16.96.128	81.29.241.181	TCP	4586 > http [FIN, ACK] Seq=1 Ack=1 Win=64
5	0.008342	81.29.241.181	172.16.96.128	TCP	http > 4586 [FIN, ACK] Seq=1 Ack=2 Win=6
6	0.009602	172.16.96.128	81.29.241.181	TCP	4586 > http [ACK] Seq=2 Ack=2 Win=64240 L

FIG. 2.17 – Flux des paquets de scan du port *http* donné par *Ethereal*

Ce résultat montre que la machine 172.16.96.128 (notre *honeypot*) a initié une connexion vers le port *http* de la machine 81.29.241.181 en envoyant un paquet contenant le bit de contrôle SYN, cette dernière a répondu par un paquet contenant SYN-ACK, ensuite la machine 172.16.96.128 a envoyé un paquet contenant ACK pour confirmer l'établissement de la connexion, suivi par un autre paquet contenant FIN pour demander la fermeture de la connexion, la machine 81.29.241.181 a répondu en envoyant un paquet contenant FIN-ACK pour acquitter et demander de son coté la fermeture de la connexion, enfin la machine 172.16.96.128 a envoyé un paquet ACK pour confirmer

la fermeture de la connexion. Donc une connexion *http* a été établie et fermée immédiatement, ce qui montre que ce trafic ni qu'un scan du port *http* (80) pour déterminer si ce dernier est accessible ou non. Donc ce genre de trafics ne représente généralement aucun risque direct sur la machine attaquée.

a.2 Trafic UDP

D'après l'analyse statistique de cette première expérimentation, nous avons remarqué que plus de 99% du trafic UDP a été destiné vers les trois ports 137, 138 et 53, où le premier port (137) est utilisé par le serveur $netbios - ns$ pour assurer le service de résolution des noms NetBIOS en adresses IP, ainsi que l'affectation des noms uniques aux différentes machines du réseau. Le deuxième port (138) est utilisé par le serveur $netbios - dgm$ essentiellement pour offrir un service de diffusion de datagrammes d'exploration des ressources disponibles sur les machines du réseau local pour les explorateur Windows, ainsi que d'autres services comme la diffusion des datagrammes d'élection d'un nouveau *master browser* d'un domaine ou d'un groupe de travail [LN97, MB01]. Le troisième port (53), c'est le port qui est utilisé par le serveur DNS (*Domain Name Service*) pour la résolution des noms Internet en adresse IP. Donc tous ces trois ports sont utilisés pour fournir des informations sur le réseau, et ne conduisent généralement à aucun dommage sur la machine de destination, c'est pour ça que nous n'avons remarqué aucune alerte pour le trafic UDP destiné à un de ces trois ports. Un exemple de ce type de trafics est illustré dans la figure Fig. 2.18 suivante.

FIG. 2.18 – Exemple d'un flux $netbios - dgm$ capturé par le *honeywall*

Lorsque nous analysons le fichier "$*.pcap$" correspondant à ce flux de trafic en utilisant l'outil *Ethereal*, nous allons obtenir le résultat indiqué dans la figure Fig. 2.19.

Dans cette figure on peut voir qu'il y a plusieurs paquets (6 paquets) identiques envoyés par le *honeypot Windows XP* (172.16.96.128) vers le port 138/UDP ($netbios - dgm$) de la machine 172.16.4.52. Ces paquets encapsulent des frames du protocole *Microsoft windows Browser*, ces frames comme le montre le segment (5) de cette figure représentent des réponses *«Get Backup List»* envoyées par le *master browser* (explorateur maître, dans notre cas c'est notre *honeypot*) pour répondre à des requêtes *«Get Backup List»* envoyées par un *browser* client (le *browser* de la machine 172.16.4.52), ces réponses portent des listes contenant des identités des serveurs de sauvegarde (*backup servers* ou *backup browsers*), dans notre cas et comme le montre le segment (6) de la figure, cette liste est de taille 1, et comme le montre le segment (8) de la figure,

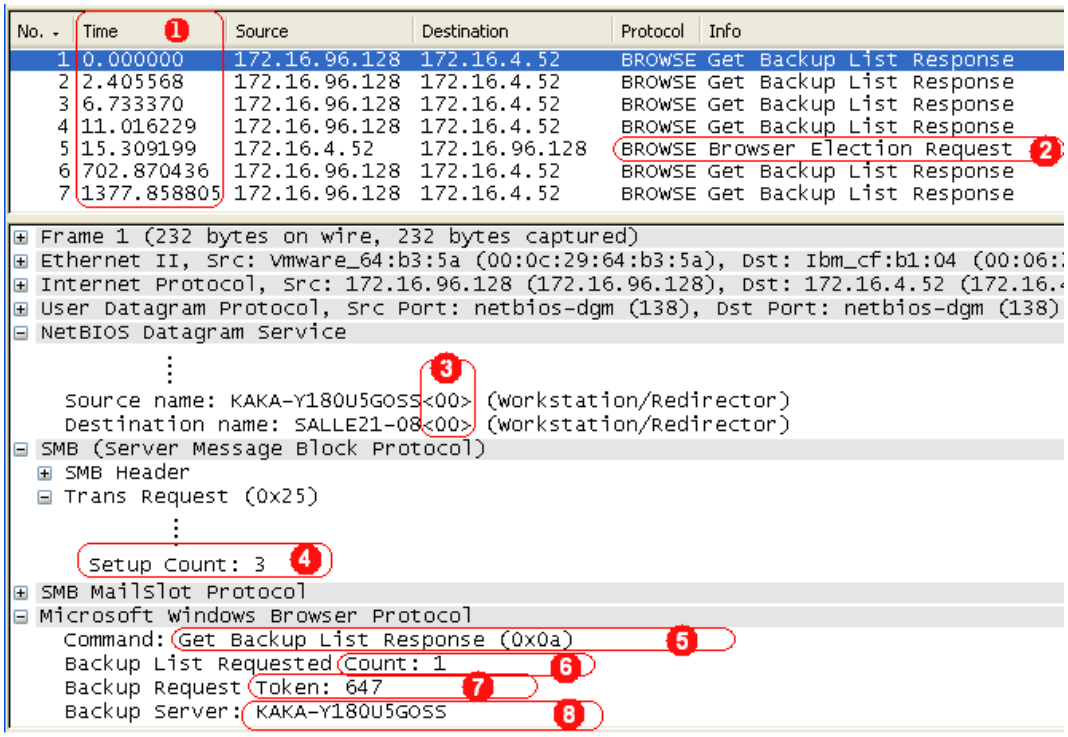


FIG. 2.19 – Analyse détaillée du flux *netbios − dgm* avec *Ethereal*

cette dernière contient le nom du seul serveur de sauvegarde qui est notre *honeypot* nommé "KAKA". Le *token* indiqué dans le segment (7) doit prendre la même valeur du *token* envoyé dans la requête client, il est utilisé par le client pour différencier entre les réponses des multiples requêtes envoyées. Ces frames sont encapsulés dans des messages du protocole *SMB Mailslot (Server Message Block Mailslot)* qui sont eux même encapsulés dans des *CIFS TRANSACT SMBs*, la seule transaction autorisée dans le protocole *SMB Mailslot* c'est la transaction *«mailslot write»*, c'est pour ça qu'on trouve la valeur "3" dans le paquet SMB du segment (4) de la figure, cette valeur représente une taille de trois octets, dont le premier prend la valeur "1" indiquant la transaction *«mailslot write»*. Les paquets SMB sont encapsulés ensuite dans des datagrammes du serveur *netbios − dgm* qui est en écoute dans le port 138/UDP, dans l'entête de ces datagrammes on trouve (segment (3)) le seizième octet du nom NetBIOS des deux machines (source et destination) prend la valeur "00", ce qui veut dire que le service concerné par ces paquets est le service de station de travail standard. Dans le segment (2) de la figure on voit un autre type de datagrammes *netbios − dgm*, qui représente l'envoie d'un paquet d'élection d'un nouveau maître explorateur, qui centralise la liste des machines présentes avec des informations sur chacune de ces machines, cette liste sera diffusée ensuite à qui la demande. Donc, tout ce trafic représente le fonctionnement normal de l'API NetBIOS de la machine Windows, qui envoie à intervalle régulier des paquets

138/udp à destination de différentes machines de son sous-réseau, afin de "découvrir" son environnement. C'est typiquement le fonctionnement par défaut du mécanisme de partage de fichiers utilisé par Windows. Donc, généralement ce genre de trafics ne provoque aucun dommage dans le système de destination, et le seul dommage possible c'est le déni de service dans le cas de transmission d'un grand nombre de paquets à des temps réduits, mais comme il est montré dans le segment (1) de la figure, le temps entre deux envois successives de ces paquets est considérable, ce qui démunie la probabilité d'existence d'une attaque de DoS (*Denial of Service*). C'est presque la même remarque pour le trafic destiné vers le port 137/UDP et le port 53/UDP.

b. Trafic de risque élevé

Pendant l'analyse du trafic capturé par le *honeywall*, nous avons remarqué une série anormale du trafic sortant de notre *honeypot* vers le port *http* de la machine 81.29.241.181, ce trafic consiste à établir une connexion TCP vers ce port et la fermer immédiatement à chaque minute pour plusieurs fois successives (jusqu'à son blocage par le *honeywall*, lorsque la limite du trafic sortant est atteinte), le port source est changé pour chaque nouvelle connexion comme le montre la figure Fig. 2.20 suivante.

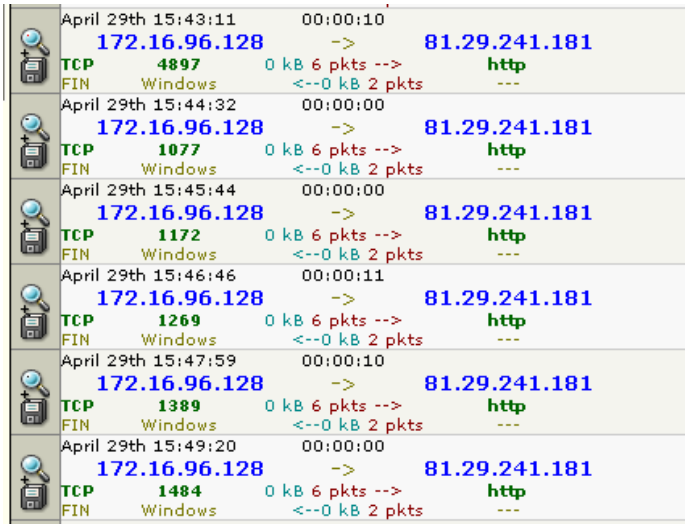


FIG. 2.20 – Exemple d'une serie de flux *http* capturée par le *honeywall*

Ce genre de trafics est probablement utilisé pour participer dans une attaque de déni de service distribuée (DDoS) contre le serveur web de la machine 81.29.241.181. Comme il peut être aussi utilisé dans une attaque de fraude pour gagner l'argent, car il y a des sites qui offrent à leurs clients des bénéfices pour chaque accès d'un internaute à leurs propres pages hébergées dans ces sites. Et pour garantir le maximum de bénéfices, il y a des clients qui diffusent des programmes malicieux (vers, virus, chevaux de troie,

botnets, ...etc) qui s'occupent d'une manière automatique par l'opération d'accès à la page concernée à partir des sites différents. Mais l'analyse détaillée de ce trafic via *ethereal* nous a montré que tous les paquets de ce trafic sont des paquets d'ouverture et de fermeture de connexion TCP avec le port *http*, et il n'y a aucune charge utile dans tous ces paquets, ce qui élimine la deuxième supposition. Donc, il y a une forte probabilité qu'il s'agit d'une attaque de déni de service distribuée.

2.5.2.2 Trafic des attaques connues (avec alertes)

Comme nous avons vu dans l'analyse statistique, le pourcentage des alertes générées par rapport au trafic total capturé est de 05,65%, nous avons vu aussi que les ports qui génèrent le plus grand nombre d'alertes sont : 139 et 445 (87,92% d'alertes). Dans chacun de ces deux ports nous avons remarqué comme le montre la figure 2.21 que plus de 90% des alertes générées dans le port 445/TCP ($microsoft-ds$) sont des alertes *"SHELLCODE x86 inc ebx NOOP"*, et plus de 80% des alertes générées dans le port 139/TCP ($netbios-ssn$) sont des alertes *"NETBIOS SMB IPC$ unicode share access"* et *"NETBIOS SMB Session Setup AndX request unicode username overflow attempt"*.

FIG. 2.21 – Exemple des flux avec alertes capturés par le *honeywall*

a. L'attaque "SHELLCODE x86 inc ebx NOOP"

"SHELLCODE x86 inc ebx NOOP" est une alerte correspondante à une attaque de dépassement de tampon (*buffer overflow*)[Smi97, Blo04, DD07]. Comme son nom l'indique, il s'agit d'un bug dans un exécutable (processus) laissant à l'utilisateur la possibilité de stocker dans la pile plus d'octets que prévu, ceci provoque l'écrasement de l'adresse de retour du procédure qui est stockée au sommet de cette pile, l'attaquant exploite cette faille en transmettant une grande quantité de données vers le processus d'un service de la machine cible, ce dernier va stocker ces données dans sa pile dans le sens inverse (dans le TAS), et puisque la taille de ces données est supérieure à la taille disponible dans le TAS (heap), le stockage se poursuit en dehors des limites du TAS, donc le sommet de la pile sera écrasé par ces données, et l'adresse de retour du procédure sera remplacée par une autre adresse qui pointe vers un code appelé *shellcode* écrit par l'attaquant, ce code est porté dans les données transmises. Pour assurer que la nouvelle adresse pointe vraiment sur le *shellcode* envoyé, l'attaquant utilise deux techniques dans les données envoyées, la première technique c'est la duplication d'adresse de retour plusieurs fois après le *shellcode*, jusqu'à débordement du TAS, et la deuxième technique c'est la duplication d'une série d'instructions vides comme NOP ($No\ OPeration$) ou *inc ebx*, ...etc avant le *shellcode*, et comme ça même si l'adresse de retour pointe vers une adresse avant le *shellcode*, cette adresse sera correspondante à une de ces instructions vides qui seront exécutées une par une jusqu'à l'arrivée au *shellcode*. Cette technique d'attaque est illustrée dans les deux figures Fig. 2.22 et Fig. 2.23.

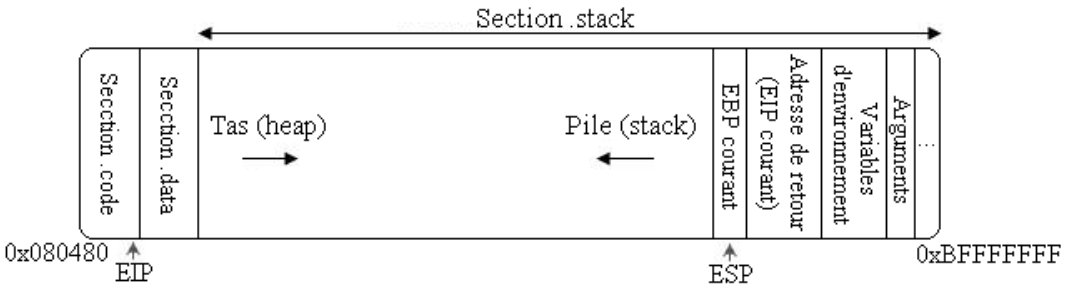


FIG. 2.22 – Structure de la pile avant l'attaque *buffer overflow*

Dans les attaques *"SHELLCODE x86 inc ebx NOOP"*, les instructions vides utilisées ne sont pas des NOPs, mais sont des instructions INC EBX codées en ASCII par des lettres "CC" ("CC" = 0x43h = INC EBX).

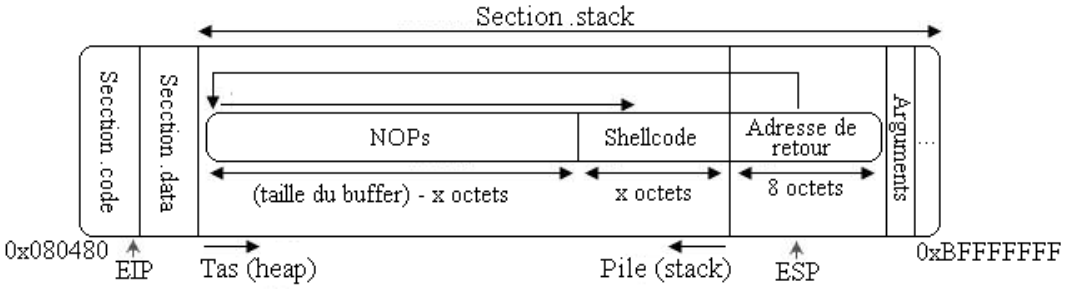


FIG. 2.23 – Structure de la pile après l'attaque *buffer overflow*

Les attaques *"SHELLCODE x86 inc ebx NOOP"* sont détectée par *Snort_inline* grâce à la règle de la figure Fig. 2.24 suivante.

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE
x86 inc ebx NOOP"; content:"CCCCCCCCCCCCCCCCCCCCCCCCC";
classtype:shellcode-detect; sid:1390; rev:5;)
```

FIG. 2.24 – Règle *Snort_inline* pour une attaque *"SHELLCODE x86 inc ebx NOOP"*

Où :

- *alert* : L'action de la règle.
- *ip* : Le protocole de la règle.
- $EXTERNAL\_NET$: L'adresse IP source du paquet capturé.
- $SHELLCODE\_PORTS$: Le port source du paquet capturé.
- $HOME\_NET$: L'adresse IP de destination du paquet capturé.
- *any* : Le port de destination du paquet capturé ("*any*" signifie n'importe quel port).
- *(msg :"SHELLCODE x86 inc ebx NOOP"; content :"CCCCCCCCCCCCCCCCCCCCCCCCC"; classtype :shellcode-detect; sid :1390; rev :5;)* : Les options de la règle, dont "*msg*" c'est le message à afficher à l'utilisateur lorsque la règle est déclenchée, et "*content*" c'est une partie du contenu du paquet capturé qui provoque le déclenchement de cette règle (c'est la signature de la règle).

Donc *Snort_inline* détecte ce type d'attaques lorsqu'il découvert le contenu "CCCCCCCCCCCCCCCCCCCCCCCCC" (une suite d'instructions INC EBX) dans un des paquets capturés.

Pendant l'analyse du trafic capturé dans notre expérimentation nous avons remarqué un grand nombre d'alertes de ce genre, une de ces alertes est indiquée dans la figure Fig. 2.25 suivante.

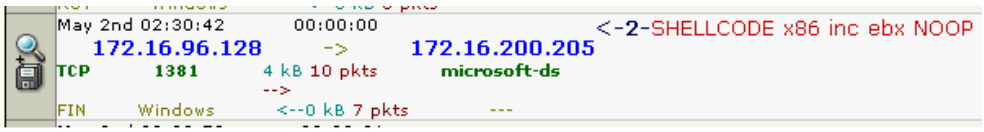


FIG. 2.25 – Trafic avec une alerte *"SHALLCODE x86 inc NOOP"*

C'est une attaque destinée vers le port 445/TCP ($microsoft - ds$) de la machine 172.16.253.100. Pour analyser le *shellcode* envoyé à l'intérieur de ce trafic, nous avons utilisé l'outil *Ethereal* qui nous a permis de lire le fichier "*.*pcap*" correspondant, en utilisant l'option *"Follow TCP stream"* qui permet d'afficher "en clair" tout le dialogue effectué entre la machine du pirate et la machine victime dans une fenêtre. Une partie de ce dialogue est illustrée dans la figure Fig. 2.26 suivante (on peut pas visualiser la totalité du dialogue car ça va prendre plus de quatre pages) :

```
000000BD  10 00 00 00 00 d4 00 00  80 7e 10 60 82 10 7a 06 ........ .~.`..z.
000000CD  06 2b 06 01 05 05 02 a0  82 10 6e 30 82 10 6a a1 .+...... ..n0..j.
    :
000000ED  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41 AAAAAAAA AAAAAAAA
000000FD  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41 AAAAAAAA AAAAAAAA
    :
000004DD  41 41 41 41 41 41 41 41  41 41 41 41 03 00 23 82 AAAAAAAA AAAA..#.
000004ED  0c 57 03 82 04 0a 00 90  42 90 42 90 42 90 42 e8 .W...... B.B.B.B.
000004FD  ff ff ff ff c0 5d 89 eb  31 c9 81 e9 23 fe ff ff .....].. 1...#...
    :
000006D1  98 84 98 84 86 81 8c 87  87 85 8f 99 b6 de 36 60 ........ ......6`
000006E1  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42 BBBBBBBB BBBBBBBB
000006F1  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42 BBBBBBBB BBBBBBBB
    :
000008F1  42 42 42 42 42 42 42 42  42 42 42 42 23 0a 03 08 BBBBBBBB BBBB#...
00000901  00 f8 0f 01 00 f8 0f 01  23 82 08 39 03 82 04 11 ........ #..9....
    :
00000A51  e0 74 07 c1 cf 0d 01 c7  eb f4 3b 7c 24 24 75 e1 .t...... ..;|$$u.
00000A61  8b 5a 24 01 eb 66 8b 0c  4b 8b 5a 1c 01 eb 8b 04 .Z$..f.. K.Z.....
00000A71  8b 01 e8 89 44 24 1c 61  c2 08 00 eb fe 43 43 43 ....D$.a .....CCC
00000A81  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43 CCCCCCCC CCCCCCCC
00000A91  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43 CCCCCCCC CCCCCCCC
    :
00000D15  43 43 43 43 43 43 43 43  43 43 43 43 43 23 82 04 CCCCCCCC CCCCC#..
00000D25  20 03 09 00 eb 06 90 90  90 90 90 90 03 82 04 11  ....... ........
00000D35  00 44 44 44 44 44 44 44  44 44 44 44 44 44 44 44 .DDDDDDD DDDDDDDD
00000D45  44 44 44 44 44 44 44 44  44 44 44 44 44 44 44 44 DDDDDDDD DDDDDDDD
```

FIG. 2.26 – Contenu des paquets d'une attaque *"SHALLCODE x86 inc NOOP"*

Donc comme il est illustré dans cette figure (Fig. 2.26) l'attaquant a envoyé des données massives contenant un *shellcode* vers la victime. Malheureusement dans tous ces octets de données, il n'y a aucune combinaison qui nous permet de déterminer la position exacte du *shellcode*, ce qui nous oblige d'analyser tous les octets hexadécimaux

situés entre les "AAAA...", "BBBB..." et "CCCC...". Pour pouvoir analyser ces hexadécimaux, on doit d'abord les convertir en assembleur en utilisant la commande "*gdb*" de linux (on crée un petit programme C dans lequel on stocke nos hexadécimaux dans un tableau de caractères, puis on le compile avec l'option $-ggdb$, ensuite on lance le débuggeur de ce programme pour extraire le code assembleur stocké dans le tableau de caractère). Après l'analyse du code assembleur correspondant aux hexadécimaux situés entre "AAAA..." et "BBBB...", nous avons trouvé que ce code ne correspond pas à un code assembleur, car toutes ses instructions sont aléatoires et sans sens, comme le montre la liste d'instructions de la figure Fig. 2.27 suivante :

```
0x08049734  shellcode+468>:     es
0x08049735 <shellcode+469>:     es
0x08049736 <shellcode+470>:     faddp  %st,%st(2)
0x08049738 <shellcode+472>:     ret    $0x8cc6
0x0804973b <shellcode+475>:     cltd
```

FIG. 2.27 – Une partie de code assembleur du *shellcode* extrait

Donc, le *shellcode* est situé entre "BBBB..." et "CCCC...". La génération du code assembleur correspondant à ces hexadécimaux nous a donné un code de grande taille (303 lignes), l'analyse de ce code nous a permis de détecter deux codes identiques, où les lignes de chacun d'eux sont liées entre elles ce qui montre qu'il s'agit du *shellcode* cherché. Une partie de ce code est illustrée dans la figure Fig. 2.28 suivante.

```
    :
0x08049641 <shellcode+225>:     pusha
0x08049642 <shellcode+226>:     mov    $0x7ffdf020,%edi
0x08049647 <shellcode+231>:     mov    (%edi),%ebx
0x08049649 <shellcode+233>:     mov    0x8(%esi),%eax
0x0804964c <shellcode+236>:     mov    %eax,(%edi)
0x0804964e <shellcode+238>:     mov    0xfffffff8(%edi),%edi
0x08049651 <shellcode+241>:     add    $0x178,%edi
0x08049657 <shellcode+247>:     mov    %edi,%ecx
0x08049659 <shellcode+249>:     cmp    %ebx,(%ecx)
0x0804965b <shellcode+251>:     je     0x8049661 <shellcode+257>
0x0804965d <shellcode+253>:     mov    (%ecx),%ecx
0x0804965f <shellcode+255>:     jmp    0x8049659 <shellcode+249>
0x08049661 <shellcode+257>:     mov    %edi,%edx
0x08049663 <shellcode+259>:     cmp    %ebx,0x4(%edx)
0x08049666 <shellcode+262>:     je     0x804966d <shellcode+269>
0x08049668 <shellcode+264>:     mov    0x4(%edx),%edx
0x0804966b <shellcode+267>:     jmp    0x8049663 <shellcode+259>
0x0804966d <shellcode+269>:     mov    %edx,(%ecx)
0x0804966f <shellcode+271>:     mov    %ecx,0x4(%edx)
0x08049672 <shellcode+274>:     movb   $0x1,0xfffffffd(%ebx)
0x08049676 <shellcode+278>:     popa
0x08049677 <shellcode+279>:     ret
    :
```

FIG. 2.28 – Une autre partie du *shallcode* porté dans les paquets envoyés

La recherche de la fonctionnalité de ce *shellcode* est très fastidieuse à cause de sa grande taille, mais pendant l'analyse de ce dernier nous avons remarqué l'existence d'une boucle infinie dans ce code, ce qui montre qu'en plus d'exploiter la vulnérabilité, ce *shellcode* peut conduire à une consommation inutile des ressources de la machine. Cette boucle est illustrée dans la figure Fig. 2.29 suivante.

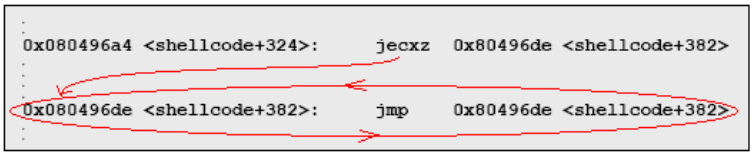


FIG. 2.29 – Code assembleur correspondant à une boucle infinie

b. L'attaque "NETBIOS SMB IPC$ unicode share access"

"NETBIOS SMB IPC$ unicode share access" est une attaque correspondante à une tentative d'accès aux partages réseaux *Windows* de la machine cible, en essayant d'accéder à des dossiers localisés dans un lecteur disque de la machine victime, en exploitant la vulnérabilité du service de partage de fichiers dans les systèmes *Windows*, qui utilisent par défaut le format *%DRIVE_LETTER%+$* pour leur partage des disques, donc n'importe qui peut accéder aux partages avec les droits d'administrateur. Ce type d'attaques arrive généralement juste après un scan réussi du port 137/UDP ($netbios - ns$) qui permet de fournir des informations sur le serveur de partage de fichiers (SMB) exécuté dans le port 139/TCP ($netbios - ssn$). Cette attaque est généralement effectuée pour télécharger des *rootkits*[6] vers la victime, pour pouvoir la contrôler ensuite à distance.

L'exemple de la figure Fig. 2.30 suivante représente une attaque provoquant une alerte de ce genre :

FIG. 2.30 – Trafic avec une alerte *"NETBIOS SMB IPC$ unicode share access"*

L'analyse de ce trafic via *Ethereal* nous a donné le résultat illustré dans la figure Fig. 2.31 suivante.

[6]Un *rootkit* est un ensemble d'outils utilisés par les hackers pour effectuer leurs différentes attaques

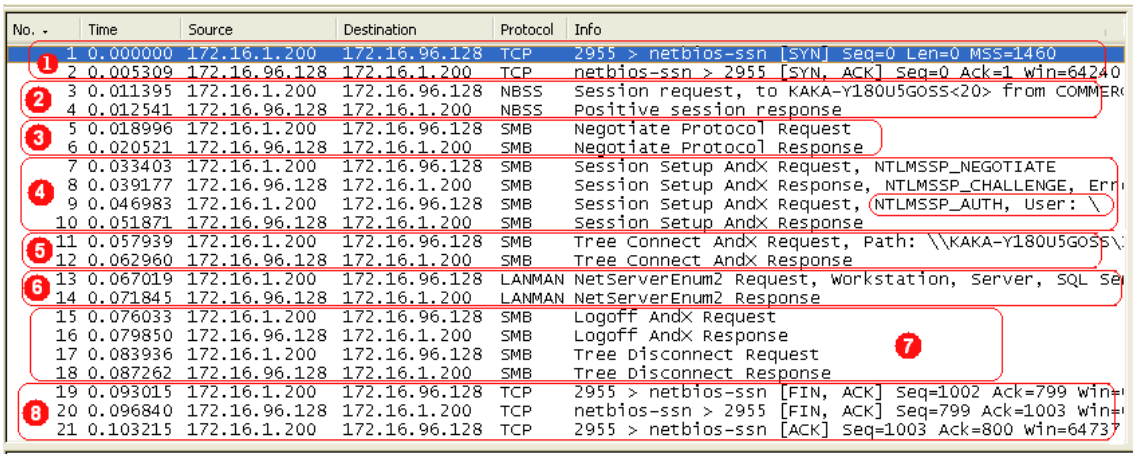


FIG. 2.31 – Contenu d'une attaque *"NETBIOS SMB IPC$ unicode share access"*

Comme il est illustré dans cette figure (Fig. 2.31), l'attaquant a envoyé un paquet SYN vers le port 139 de notre *honeypot* pour initier une connexion au service $netbios - ssn$, sa demande a été acceptée par notre *honeypot* qui a répondu par un paquet $SYN\text{-}ACK$ (segment 1 de la figure), après l'établissement de la connexion, l'attaquant a envoyé un autre paquet pour établir une session $NBSS$ ($NetBIOS\ Session\ Service$) qui lui permet d'accéder au partages disponibles dans notre *honeypot*, ce dernier a accepté cette demande, et la session a été établie avec succès (segment 2 de la figure). Ensuite, il a commencé la négociation nécessaire pour établir une session SMB (segment 3 de la figure), cette négociation consiste à décrire les dialectes qu'il supporte pour établir la session, comme par exemple les dialectes d'authentification, et d'encryption des clés. Après la fin de la négociation, l'attaquant est arrivé à la partie la plus importante dans l'établissement de la session SMB (segment 4 de la figure), dont il doit s'authentifier pour pouvoir établir la session. Comme il est indiqué dans le segment 4 de la figure, l'attaquant a commencé l'authentification par l'envoie d'un paquet contenant le mot de passe d'administrateur, notre *honeypot* a répondu par une erreur, dont son statut est : "$Status\_More\_Processing\_Required$", mais ça c'est normal, puisque l'attaquant n'a pas envoyé toutes les informations d'authentification, c'est pour ça, il a complété les informations d'authentification manquantes (partie finale du mot de passe et le nom d'utilisateur) dans la deuxième requête, et comme il est illustré dans la figure, le nom d'utilisateur envoyé est *root* (\) ($NTLMSSP\_AUTH, User$: \), ensuite notre *honeypot* a accepté l'authentification, donc l'attaquant a pu ouvrir une session SMB avec les privilèges d'administrateur. Après l'ouverture de cette session SMB, l'attaquant a lancé une requête (segment 5 de la figure) pour ouvrir une session vers le *stub* du service *IPC (Inter-Process Communication)*, le contenu de cette requête qui est illustré dans la figure Fig. 2.32 suivante (longueur de mot de passe égale à 1) montre

que l'attaquant a essayé d'ouvrir une session *NULL* (*session SMB unauthentifiable*) [Mar05] vers l'*IPC$*.

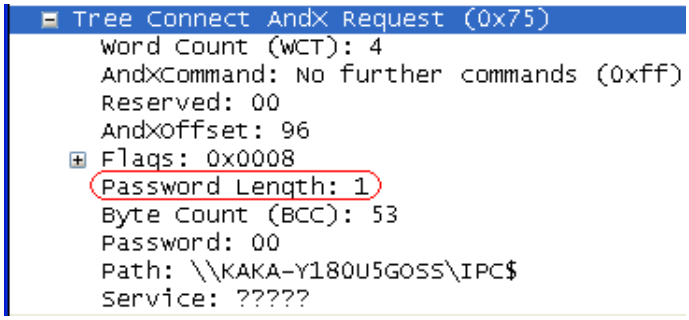


FIG. 2.32 – Requête d'établissement d'une session NULL

L'ouverture de cette session a été terminée avec succès, puis l'attaquant a commencé l'invocation des procédures (fonctions, méthodes) distantes via cette session (segment 6 de la figure Fig. 2.31). La fonction invoquée par l'attaquant est la fonction "*NetServerEnum2*" qui lui permet de récupérer la liste de tous les serveurs actives dans la machine cible (notre *honeypot*) [Mar05]. Le résultat récupéré par cette fonction est illustré dans la figure Fig. 2.33 suivante :

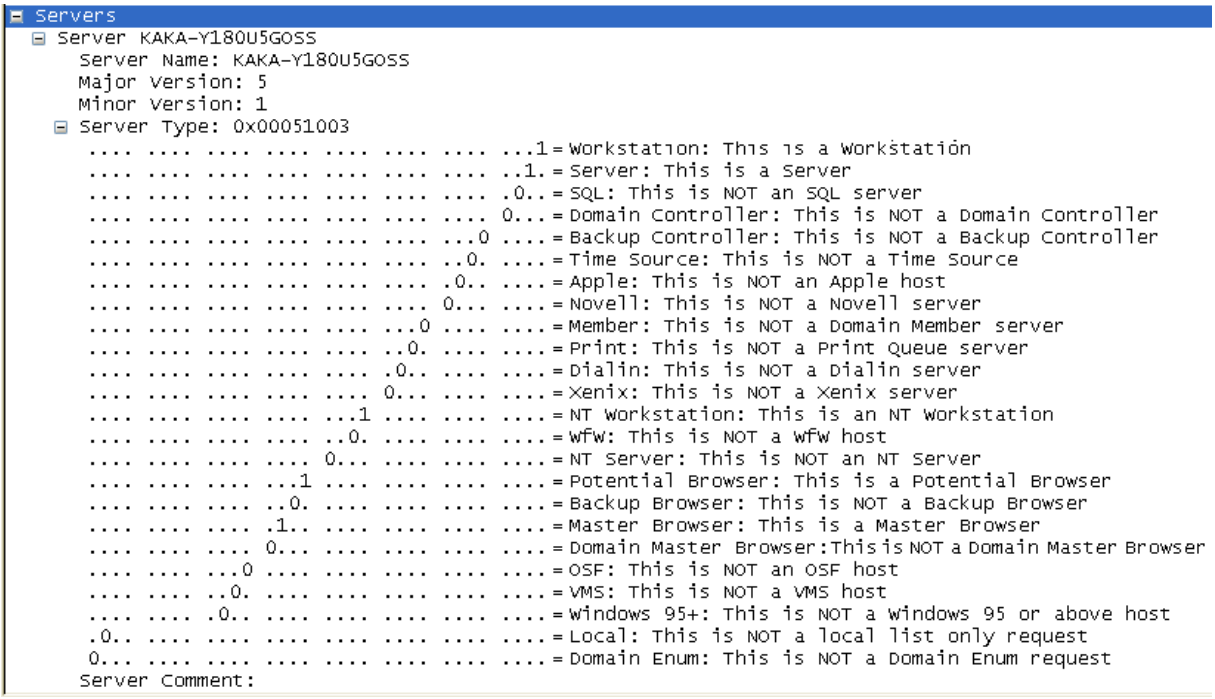


FIG. 2.33 – Résultat de la fonction *NetServerEnum2*

Après la récupération de ces informations, l'attaquant a fermé toutes les sessions qui a ouvert précédemment (sessions $NBSS$ et SMB. Segment 7 de la figure Fig. 2.31), ainsi que la connexion TCP (segment 8 de la figure Fig. 2.31).

c. L'attaque "NETBIOS SMB Session Setup AndX request unicode username overflow attempt"

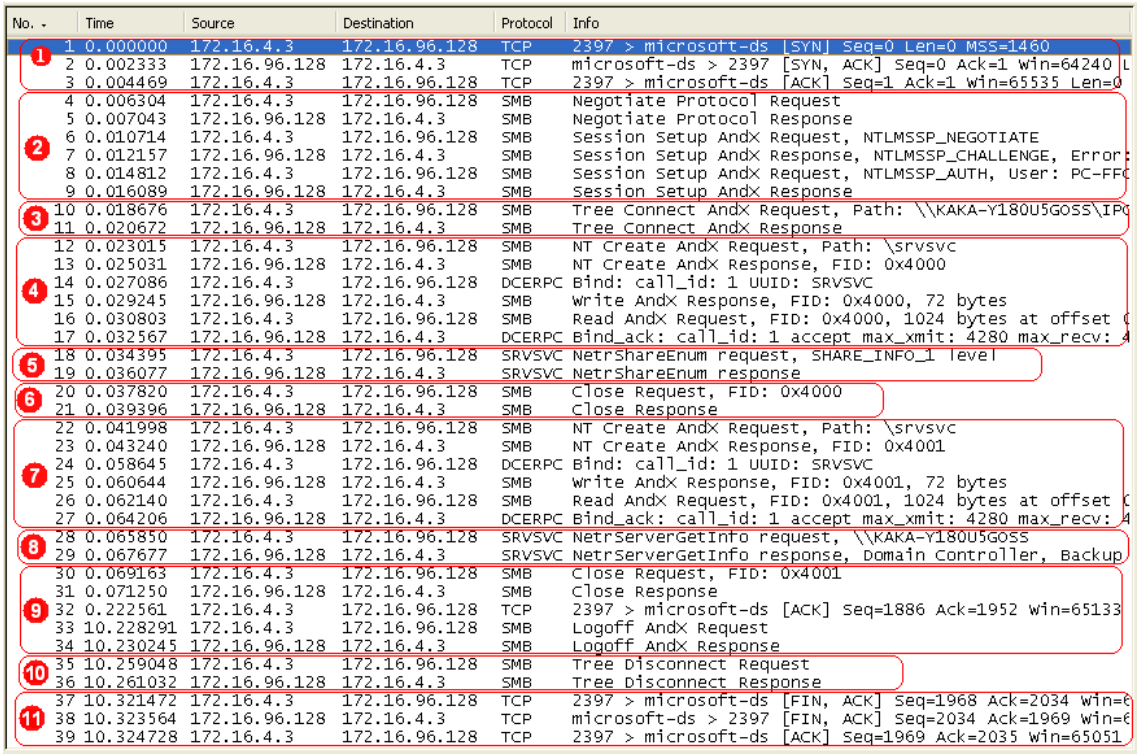
"NETBIOS SMB Session Setup AndX request unicode username overflow attempt" est une attaque correspondante à une tentative d'exploitation de la vulnérabilité existante dans les produits *ISS RealSecure*, utilisés dans la plupart des systèmes pour surveiller le trafic réseau, par le biais d'un agent de réseau, dans le but de découvrir des signatures d'attaques. Dans ce type d'attaques, l'attaquant envoie un paquet simple de SMB (*Server Message Block*) contenant un nom de compte supérieur à 300 Octets, ce qui génère un débordement de tampon (*buffer overflow*) dans le module d'analyse d'ISS (*Internet Security Systems*), et donc le service de ce dernier ne peut plus répondre, et les paquets SMB incluant le code exploit seront décodés par le système, et comme ça le code exploit sera exécuté sur la machine avec des privilèges d'administrateur. Cette attaque est utilisée généralement, soit pour désactiver les sondes d'ISS, ou pour prendre le contrôle de la machine victime. Dans notre expérimentation nous avons remarqué que cette attaque est généralement combinée avec l'attaque précédente (voir la figure Fig. 2.21).

Un exemple d'une attaque provoquant ce genre d'alertes est donné dans la figure Fig. 2.34 suivante.

FIG. 2.34 – Trafic avec une alerte *"NETBIOS SMB-DS Session Setup ... "*

L'analyse de ce trafic avec *Ethereal* nous a donné le résultat de la figure Fig. 2.35.

Comme il est illustré dans le segment "1" de cette figure (Fig. 2.35), l'attaquant a commencé son attaque par l'établissement d'une connexion TCP avec le port 445 (port de $microsoft - ds$), puis il a commencé un processus de négociation pour ouvrir une session SMB avec les privilèges de l'administrateur, et comme il est illustré dans le segment "2" de la figure, l'attaquant a terminé l'authentification et l'ouverture de la session avec succès, puis il a continué pour ouvrir une session nulle (*anonymous SMB session.*) vers le *stub* (*soush*) du service IPC (*Inter- Process Communication*) appelé IPC\$, comme le montre le segment "3" de la figure. Après l'ouverture de la session



No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.4.3	172.16.96.128	TCP	2397 > microsoft-ds [SYN] Seq=0 Len=0 MSS=1460
2	0.002333	172.16.96.128	172.16.4.3	TCP	microsoft-ds > 2397 [SYN, ACK] Seq=0 Ack=1 Win=64240 L
3	0.004469	172.16.4.3	172.16.96.128	TCP	2397 > microsoft-ds [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.006304	172.16.4.3	172.16.96.128	SMB	Negotiate Protocol Request
5	0.007043	172.16.96.128	172.16.4.3	SMB	Negotiate Protocol Response
6	0.010714	172.16.4.3	172.16.96.128	SMB	Session Setup AndX Request, NTLMSSP_NEGOTIATE
7	0.012157	172.16.96.128	172.16.4.3	SMB	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error:
8	0.014812	172.16.4.3	172.16.96.128	SMB	Session Setup AndX Request, NTLMSSP_AUTH, User: PC-FF
9	0.016089	172.16.96.128	172.16.4.3	SMB	Session Setup AndX Response
10	0.018676	172.16.4.3	172.16.96.128	SMB	Tree Connect AndX Request, Path: \\KAKA-Y180U5GOSS\IPC
11	0.020672	172.16.96.128	172.16.4.3	SMB	Tree Connect AndX Response
12	0.023015	172.16.4.3	172.16.96.128	SMB	NT Create AndX Request, Path: \srvsvc
13	0.025031	172.16.96.128	172.16.4.3	SMB	NT Create AndX Response, FID: 0x4000
14	0.027086	172.16.4.3	172.16.96.128	DCERPC	Bind: call_id: 1 UUID: SRVSVC
15	0.029245	172.16.96.128	172.16.4.3	SMB	Write AndX Response, FID: 0x4000, 72 bytes
16	0.030803	172.16.4.3	172.16.96.128	SMB	Read AndX Request, FID: 0x4000, 1024 bytes at offset 0
17	0.032567	172.16.96.128	172.16.4.3	DCERPC	Bind_ack: call_id: 1 accept max_xmit: 4280 max_recv: 4
18	0.034395	172.16.4.3	172.16.96.128	SRVSVC	NetrShareEnum request, SHARE_INFO_1 level
19	0.036077	172.16.96.128	172.16.4.3	SRVSVC	NetrShareEnum response
20	0.037820	172.16.4.3	172.16.96.128	SMB	Close Request, FID: 0x4000
21	0.039396	172.16.96.128	172.16.4.3	SMB	Close Response
22	0.041998	172.16.4.3	172.16.96.128	SMB	NT Create AndX Request, Path: \srvsvc
23	0.043240	172.16.96.128	172.16.4.3	SMB	NT Create AndX Response, FID: 0x4001
24	0.058645	172.16.4.3	172.16.96.128	DCERPC	Bind: call_id: 1 UUID: SRVSVC
25	0.060644	172.16.96.128	172.16.4.3	SMB	Write AndX Response, FID: 0x4001, 72 bytes
26	0.062140	172.16.4.3	172.16.96.128	SMB	Read AndX Request, FID: 0x4001, 1024 bytes at offset 0
27	0.064206	172.16.96.128	172.16.4.3	DCERPC	Bind_ack: call_id: 1 accept max_xmit: 4280 max_recv: 4
28	0.065850	172.16.4.3	172.16.96.128	SRVSVC	NetrServerGetInfo request, \\KAKA-Y180U5GOSS
29	0.067677	172.16.96.128	172.16.4.3	SRVSVC	NetrServerGetInfo response, Domain Controller, Backup
30	0.069163	172.16.4.3	172.16.96.128	SMB	Close Request, FID: 0x4001
31	0.071250	172.16.96.128	172.16.4.3	SMB	Close Response
32	0.222561	172.16.4.3	172.16.96.128	TCP	2397 > microsoft-ds [ACK] Seq=1886 Ack=1952 Win=65133
33	10.228291	172.16.4.3	172.16.96.128	SMB	Logoff AndX Request
34	10.230245	172.16.96.128	172.16.4.3	SMB	Logoff AndX Response
35	10.259048	172.16.4.3	172.16.96.128	SMB	Tree Disconnect Request
36	10.261032	172.16.96.128	172.16.4.3	SMB	Tree Disconnect Response
37	10.321472	172.16.4.3	172.16.96.128	TCP	2397 > microsoft-ds [FIN, ACK] Seq=1968 Ack=2034 Win=6
38	10.323564	172.16.96.128	172.16.4.3	TCP	microsoft-ds > 2397 [FIN, ACK] Seq=2034 Ack=1969 Win=6
39	10.324728	172.16.4.3	172.16.96.128	TCP	2397 > microsoft-ds [ACK] Seq=1969 Ack=2035 Win=65051

FIG. 2.35 – Contenu des paquets d'une attaque *"NETBIOS SMB-DS Session Setup ..."*

nulle, l'attaquant a créé un chemin vers l'interface "*srvsvc*" pour énumérer toutes les informations du partage (segment "4" de la figure), puis il a invoqué la méthode *NetrShareEnum* [Mar05] de cette interface (segment "5" de la figure) pour récupérer toutes les informations du partage. Les informations récupérées par cette méthode sont données dans la figure Fig. 2.36.

Après la récupération de ces informations, l'attaquant a fermé le chemin destiné vers l'interface "*srvsvc*" (segment "6" de la figure Fig. 2.35), puis il a créé un nouveau chemin vers la même interface "*sevsvc*" (segment "7" de la figure Fig. 2.35), et après la création de ce nouveau chemin, l'attaquant a invoqué la méthode $NetrServerGetInfo$ (segment "8" de la figure Fig. 2.35) pour récupérer toutes les informations concernant les serveurs actives dans cette machine, les informations récupérées sont similaires à celles récupérées par la fonction $NetServerEnum2$ (voir l'exemple de l'attaque précédente). Après la récupération de ces informations, l'attaquant a fermé le chemin (segment "9" de la figure Fig. 2.35), puis il a fermé la session nulle avec l'IPC\$ (segment "10" de la figure Fig. 2.35), ainsi que la connexion TCP (segment "11" de la figure Fig.2.35).

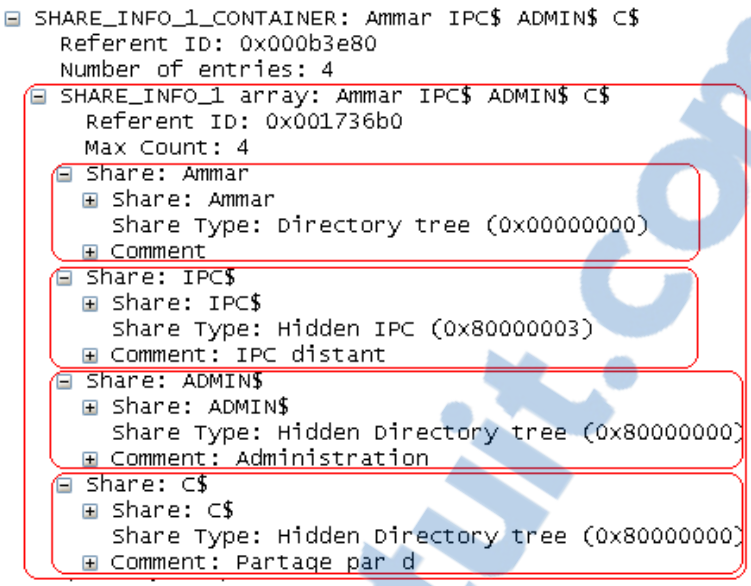


FIG. 2.36 – Résultat de la méthode *NetrShareEnum*

2.6 Conclusion et Leçons acquises

La mise en place de la technologie *honeynet ROO CDROM* nous a permis de découvrir plusieurs résultats intéressants concernant les attaques informatiques et la communauté des hackers. A partir de cette expérimentation, nous avons pu savoir quelles sont les techniques les plus utilisées par la communauté des hackers pour pénétrer un système, nous avons vu que la plupart des attaques sont basées soit sur l'exploit d'un partage disponible dans une machine, en exploitant une des vulnérabilités des protocoles de partage de fichiers, comme les protocoles $NETBIOS$ ($netbios-ns$, $netbios-dgm$, $netbios-ssn$) et $microsoft-ds$, ou bien en se basant sur la technique de dépassement de tampon, pour pouvoir exécuter des codes qui leur permettent par exemple d'ouvrir des *backdoor*[7] dans la victime. Avec cette expérimentation, nous avons pu aussi savoir quels sont les ports qui intéressent plus les hackers, et quels sont les scénarios d'attaques les plus utilisés par ces derniers pour exploiter ces ports. Mais, malgré tous ces acquis, cette technologie reste encore avoir besoin de certaines améliorations de plusieurs côtés, à cause de différents problèmes et pointes faibles que nous avons remarqué durant notre expérimentation, que ce soit dans le côté d'utilisation ou dans le côté de performance.

Dans le côté d'utilisation par exemple, nous avons rencontré une grande difficulté pour analyser les données capturées, à cause de leur énorme quantité, donc il faut chercher des techniques et des méthodes pour limiter le volume d'informations à analyser, comme par exemple le regroupement des données capturées en des classes, dont chaque

[7]Un *backdoor* est un petit programme installé par les hackers dans la machine victime, pour y créer une voie d'accès secrète.

classe doit contenir un genre différent de trafics (exp. La classe du trafic avec alertes, la classe des attaques inconnues, ...etc), et comme ça la personne qui s'intéresse uniquement aux attaques inconnues, peut directement se destiner vers la classe appropriée. Ou bien on utilise des techniques de *data mining* pour classifier le trafic capturé en "clusters", de telle manière que tous les flux identiques sont mis dans le même "cluster", donc, au lieu d'analyser tous les flux, on analyse uniquement un seul flux pour chaque "cluster", la même chose pour les alertes (chaque type d'alertes dans un "cluster").

Dans le côté de performance et d'efficacité, le point faible le plus important que nous pouvons extraire à partir de l'analyse du trafic capturé, c'est la limitation du trafic sortant qui ne se contente pas uniquement par la permission aux attaquants de détecter la présence du *honeynet*, mais elle conduit aussi à une pauvreté formidable dans la quantité d'informations collectées sur la communauté des hackers, car dans la plupart des cas, la quantité du trafic qui sortit d'un *honeypot* est plus grande que celle qui y entré. Par exemple, pendant l'analyse du trafic capturé dans cette expérimentation, nous avons remarqué que 80.95% des trafics capturés sont des trafics sortants (rebondissants), donc la limitation de ce trafic touche massivement l'efficacité de cette technologie dans l'apprentissage qui représente l'objectif majeur des technologies *honeypot*, donc il faut chercher une autre technique efficace pour remplacer la limitation du trafic sortant.

Un autre point faible que nous avons remarqué dans cette expérimentation est le manque de distinction entre le trafic sortant résultant d'une attaque et le trafic sortant résultant du fonctionnement normal du système installé, comme dans le cas du trafic généré par le protocole *"Microsoft Windows Browser Protocole"* qui est basé sur le protocole $netbios - dgm$ (port 138/UDP), ce protocole envoi périodiquement des paquets d'exploration de voisinage réseau [LN97], comme par exemple : "$HostAnnouncement$" pour annoncer sa présence au maître explorateur, "$NetServerEnum2$" pour prendre la liste des serveurs de son domaine ainsi que la liste des domaines du réseau, ...etc. et comme nous avons vu pendant l'analyse du trafic capturé dans cette première expérimentation, il y a eu plus de 10% du trafic sortant qui correspond à ce protocole, donc ce type de trafics va accélérer l'atteint à la limite maximum définie, avec un pourcentage de 10%, ce qui augmente la probabilité d'alerte du pirate d'une part, et limite le pourcentage d'apprentissage d'autre part. Donc il faut chercher un moyen efficace pour assurer la distinction entre le trafic de ce type et le reste du trafic.

Enfin, nous pouvons dire que le *honeynet roo* est actuellement un des meilleurs *honeypots* de haute interaction, mais il reste avoir besoin de plusieurs améliorations dans plusieurs côtés pour devenir le meilleur parmi tous les autres *honeypots*.

Chapitre 3

La technologie "Honeyd"

3.1 Introduction

Honeyd [Pro03, THH07, SP03] est un outil de simulation de systèmes d'exploitation et de services réseaux, développé par *Niels Provos* de l'université de *Michigan*. C'est un outil *OpenSource* qui peut s'exécuter sous plusieurs systèmes comme *BSD*, *Linux* et *Solaris*, et même sous *Windows* récemment. Avec cet outil il est possible d'installer un ou plusieurs *honeypots* virtuels à faible interaction avec différentes personnalités (systèmes) et services sur une seule machine, en leur associant des adresses IP qui ne sont pas encore utilisées dans notre réseau réel. Il a été conçu principalement pour détecter les activités non autorisées, car toute tentative de connexion à une adresse IP non encore utilisée est considérée comme une activité malveillante, mais avec l'ajout de possibilité d'émulation de différents services TCP/IP comme *HTTP*, *SMTP*, *SSH* ...etc, son utilisation a évolué, il est actuellement possible de l'utiliser pour construire des *honeypots* de recherche pour découvrir les nouvelles attaques qui ne sont pas encore recensées, en observant le comportement des attaquants.

3.2 Compilation et installation de "honeyd"

Pour pouvoir compiler *honeyd* sur une telle machine, cette dernière doit disposer des bibliothèques indépendantes suivantes [THH07] :

- **Libpcap :** C'est la bibliothèque qui contient toutes les fonctions nécessaires à la capture des paquets circulant dans le réseau, elle est utilisée dans de nombreux sniffers comme : tcpdump, ethereal, snort, ...etc.
- **Libdnet :** C'est une bibliothèque développée par *Dug Song*, elle offre une interface pour les différentes fonctionnalités liées au réseau, comme la manipulation

de cache *arp*, la gestion de règles de pare-feu, les opérations de routage...etc.

– **Libevent :** C'est une bibliothèque développée par *Niels Provos* également, elle fournit une API (*Application Programming Interface*) permettant d'associer une fonction de $call - back$ à un événement donné (time-out, signal, événement sur un descripteur de fichier, ...).

Une fois toutes ces trois bibliothèques sont installées, la compilation et l'installation de *honeyd* devient possible, mais son fonctionnement nécessite encore un autre outil qui lui redirige les attaques circulant dans le réseau, cet outil est appelé *arpd* :

– **Arpd** [SP03] est un démon ARP qui écoute sur le réseau les requêtes ARP (*Address Resolution Protocol*) destinées aux adresses IP gérées par *honeyd*. Et lorsqu'il reçoit une requête ARP pour le MAC d'une adresse IP de la plage gérée par *honeyd*, il cherche d'abord dans sa table ARP s'il y a une entrée MAC correspondante à cette adresse IP. S'il n'y a pas de référence, il diffuse lui-même une requête pour déterminer si la machine correspondante à cette adresse IP est présente sur le réseau ou non. s'il n'y a pas de machine présente sur le réseau avec cette adresse IP, il répond à la requête ARP avec l'adresse MAC de l'hôte *honeyd*.

3.3 Architecture interne de "honeyd"

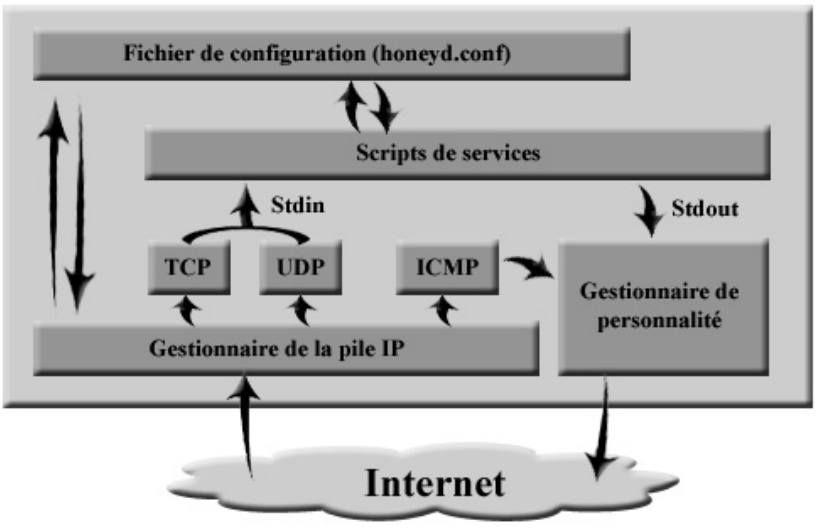


FIG. 3.1 – Architecture interne de *honeyd*

Comme il est illustré dans cette figure (Fig.3.1), l'architecture interne de *honeyd* est constituée de quatre composants suivants :

3.3.1 Fichier de configuration (*honeyd.conf*)

Comme nous avons dit, *honeyd* est utilisé pour simuler un ou plusieurs systèmes d'exploitation pour créer des *honeypots* à faible interaction, chacun de ces systèmes simulés est indiqué dans le fichier de configuration "*honeyd.conf*" sous forme d'un chablon (*template*), à l'interieur de chaque chablon on trouve :

- **La personnalité (*personality*) :** Celle-ci permet d'indiquer quel est le système d'exploitation du *honeypot* à simuler. Elle est simulée en foncton du comportement de la pile réseau du système spécifié, grâce à des fichiers empreintes utilisés par des logiciels tels que NMAP [NH07c].
  Exemple : *set windows personnality "Windows NT/4.0 SP4".*
- **L'uptime :** Il permet de déterminer le temps depuis lequel le *honeypot* est en fonctionnement, ceci est très important pour rendre les *honeypots* plus crédibles.

  Exemple : *set windows uptime 1728650.*
- **UserID et GroupID :** Ceux-ci permettent d'associer des droits d'accès aux programmes simulant des services, afin de ne pas les faire tourner avec des droits *root*, ce qui pourrait être désastreux en cas de faille.

  Exemple : *set windows uid 65534 gid 65534.*
- **L'état des ports :** Ceci permet d'indiquer quels sont les ports ouverts dans le *honeypot* (**exemple**, *"add windows udp port 137 open"*), et quels sont les comportements par défauts des ports fermés (**exemple**, *set windows default tcp action reset*). Pour un service émulé d'un port, son comportement est indiqué par un lien vers le script qui l'émule (**exemple**, *add windows tcp port 80 "sh script/web.sh"*).
- **Adresses IP associées :** Celles-ci permettent d'indiquer quelles sont les adresses IP qui ne sont pas encore utilisées dans le réseau réel et qui sont associées à ce chablon. Donc chaque chablon correspond à une ou plusieurs machines virtuelles (*honeypots*) qui ont la même configuration système qui y est indiquée.

  Exemple : *bind 172.16.2.92 172.16.45.23 windows*

Donc le chablon d'un système *Windows* est donné comme dans la figure suivante :

```
create windows                                  #créer un chablon pour un système windows
set windows personnality "Windows NT/4.0 SP4"          # définir la personnalité
set windows default tcp action reset   # réaction par defaut pour les ports TCP fermés
set windows default udp action reset   # réaction par defaut pour les ports UDP fermés
add windows tcp port 80 "sh script/web.sh"     # les paquets 80 sont traités par 'web.sh'
bind 172.16.2.92 172.16.45.23 windows          # associer deux adresses IP au chablon
```

FIG. 3.2 – Définition d'un chablon pour un système Windows

3.3.2 Scripts de services

L'émulation des services d'un système virtuel (*honeypot*) créé par *honeyd* se fait généralement par des applications scripts développées en perl, python, ou encore en shell-script, chacun de ces scripts accepte des données en entrée standard *stdin*, puis après avoir traité ces données selon le protocole émulé ($HTTP$, FTP, ...etc), il renvoi le résultat du traitement en sortie standard *stdout*, Comme le montre la figure suivante :

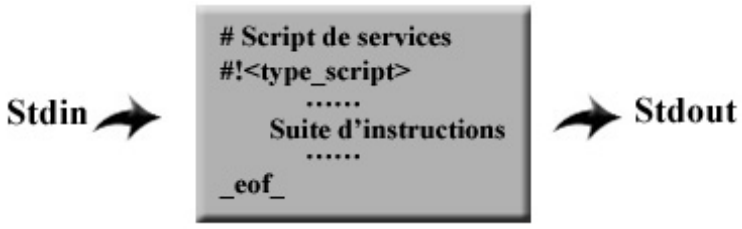


FIG. 3.3 – Structure d'un script *honeyd*

L'entrée standard *Stdin* et la sortie standard *Stdout* de chaque script sont reliées directement aux ports qu'ils simulent, grâce au fichier de configuration *honeyd.conf* comme nous avons indiqué dans la section précédente.

3.3.3 Gestionnaire de personnalité

Le gestionnaire de personnalité est un programme qui permet de transformer le format des données à envoyer vers une autre machine au format standard du système simulé. Pour assurer cette fonctionnalité, il utilise des fichiers empreints utilisés par des logiciels tels que NMAP, qui permet de scanner tous types de systèmes, ces fichiers sont : *nmap.prints*, fourni en standard avec la version 0.8 de *honeyd*, associé à *xprobe*2.*conf* [XH07] et enfin *nmap.assoc*, qui permet de faire l'association des deux fichiers de signatures.

3.3.4 Gestionnaire de la pile IP

Le gestionnaire de la pile IP est un programme qui permet de desencapsuler les paquets TCP/IP reçus pour extraire leurs adresses (source et destination) et leurs ports (source et destination) afin de savoir leur prochaine destination dans le *honeyd*, après bien sûre des tests dans le fichier de configuration *honeyd.conf* comme par exemple, le test d'existence de chablon pour l'adresse de destination, et test d'état du port de destination et son script d'émulation s'il existe, ...etc.

3.4 Fonctionnement de "honeyd"

Le fonctionnement de *honeyd* est basé sur celui de *arpd*. Sans *Arpd*, *Honeyd* ne peut pas travailler, car c'est *arpd* qui redirige les attaques vers *Honeyd*. Une fois un paquet est arrivé à *honeyd*, il passe d'abord par le gestionnaire de la pile IP qui va extraire son adresse IP de destination ainsi que son port de destination, puis il consulte dans le fichier de configuration si l'adresse IP de destination du paquet correspond est attribuée à un chablon ou non, si c'est le cas, il traite le paquet au niveau IP (vérification de la longueur du paquet, check du CRC, décrémentation du TTL, ...etc), et selon le protocole de la couche supérieure (couche transport) indiqué dans l'entête IP, il envoie le paquet vers un programme qui joue le rôle de ce protocole (TCP, UDP, ICMP), si ce dernier c'est le protocole TCP ou UDP alors, il va d'abord tester l'état et le comportement du port de destination dans le fichier de configuration *honeyd.conf*, s'il trouve que le port est émulé par un script, alors les données du paquet seront envoyées vers l'entrée standard *stdin* de ce script, et après avoir terminé le traitement de ces données, le script envoie les données du résultat vers le gestionnaire de personnalité via sa sortie standard *Stdout*, et à partir des données reçues le gestionnaire de personnalité crée un paquet sur mesure selon la pile TCP/IP correspondante à la personnalité du *honeypot*, et l'envoie vers le réseau. Sinon, si le protocole de la couche transport du paquet est le ICMP, alors ce paquet sera traité par le programme qui joue le rôle de ce protocole, et le résultat sera envoyé directement vers le gestionnaire de personnalité pour simuler la pile TCP/IP de l'OS du *honeypot*, sans faire aucune vérification dans le fichier de configuration et sans passer par aucun script [Spi02, Pro03].

3.5 Expérimentation

L'expérimentation de la technologie *honeyd* a été effectuée directement sur l'architecture de la première expérimentation (expérimentation de la technologie *honeynet ROO CDROM*). Plus précisément, le *honeyd* est installé sur le deuxième *honeypot* (sur la machine virtuelle *Red Hat Enterprise Linux* 4 créée par *VMWare*), après bien sûre l'installation de l'outil *arpd* et les trois bibliothèques (*libpcap*, *libdnet*, et *libevent*) sur cette machine virtuelle. Les versions installées sont : *honeyd-1.5b.tar.gz, arpd-0.2.tar.gz, libpcap-0.9.5.tar.gz, libdnet-1.11.tar.gz, libevent-1.2a.tar.gz.*

Á partir de *honeyd* nous avons construit les trois *honeypots* suivants :

1. **Le *honeypot "Microsoft Windows XP Professional SP1"***, qui offre les services suivants :

   Service de transfert de fichier (FTP : port 21/tcp), Service de transfert des messages électroniques (SMTP : port 25/tcp), Service Web (HTTP : port 80/tcp),

Service POP3 (Post Office Protocol version 3 : port 110/tcp), Service de messagerie sur Internet (IMAP : port 143/tcp), Service epmap (port 135/tcp), Service netbios-ns (port 137/udp), Service netbios-dgm (port 138/udp), Service netbios-ssn (port 139/tcp), Service microsoft-ds (port 445/udp & 445/tcp).

Les cinq premiers services sont simulés par des programmes en *scripts shell*, tandis que les cinq derniers ne sont que des ports ouverts et aucun service entre eux n'a été simulé (voir le fichier de configuration dans l'annexe). L'adresse IP associée à cet *honeypot* est : 172.16.113.12.

2. **Le *honeypot "Linux Suse 8.0"***, qui offre les services suivants :

   Service de transfert de fichier (FTP), Service du shell sécurisé (SSH : port 22/tcp), Service Telnet (port 23/tcp), Service de transfert des messages électroniques (SMTP), Services finger pour les informations de contacts d'utilisateurs (port 79/tcp), Service Web, Service POP3, Service de messagerie sur Internet (IMAP), Service Spooleur d'imprimante ligne (lpr : port 515/tcp), Service Squid Web proxy cache (port 3128/tcp), Service de cache du World Wide Web (port 8080/tcp), Service Proxy transparent (port 8081/tcp), Service de journalisation de systèmes UNIX (port 514/udp), Services de noms de domaine (port 53/udp).

   Tous ces services sont simulés par des programmes en *scripts shell*, sauf le dernier (services de noms de domaine) qui rebondit directement vers son correspondant dans la machine du serveur DNS du réseau (172.17.1.11) (voir le fichier de configuration dans l'annexe). L'adresse IP associée à cet *honeypot* est : 172.16.113.11.

3. **Le *honeypot "Windows 2000"***, qui offre les services suivants :

   Service Web, Service POP3, Service netbios-ns, Service netbios-dgm, Service netbios-ssn, Service microsoft-ds.

   Les deux premiers services sont simulés par des programmes en *scripts shell*, par contre les quatre derniers sont configurés pour rebondir vers leurs correspondants dans la machine source (voir la configuration dans l'annexe). L'adresse IP associée à cet *honeypot* est : 172.16.113.13.

Le reste des adresses IP de la plage 172.16.113.0-172.16.113.254, qui ne sont pas encore affectées à aucun des *honeypots* courants, sont affectées automatiquement par *arpd* aux machines virtuelles par défaut créées par *honeyd*, dont tous leurs ports sont fermés (voir le chablon $default$ dans le fichier de configuration de *honeyd* dans l'annexe).

La nouvelle architecture globale de l'expérimentations des deux technologies *honeynet ROO* et *honeyd* est illustrée dans la figure Fig. 4.3 suivante.

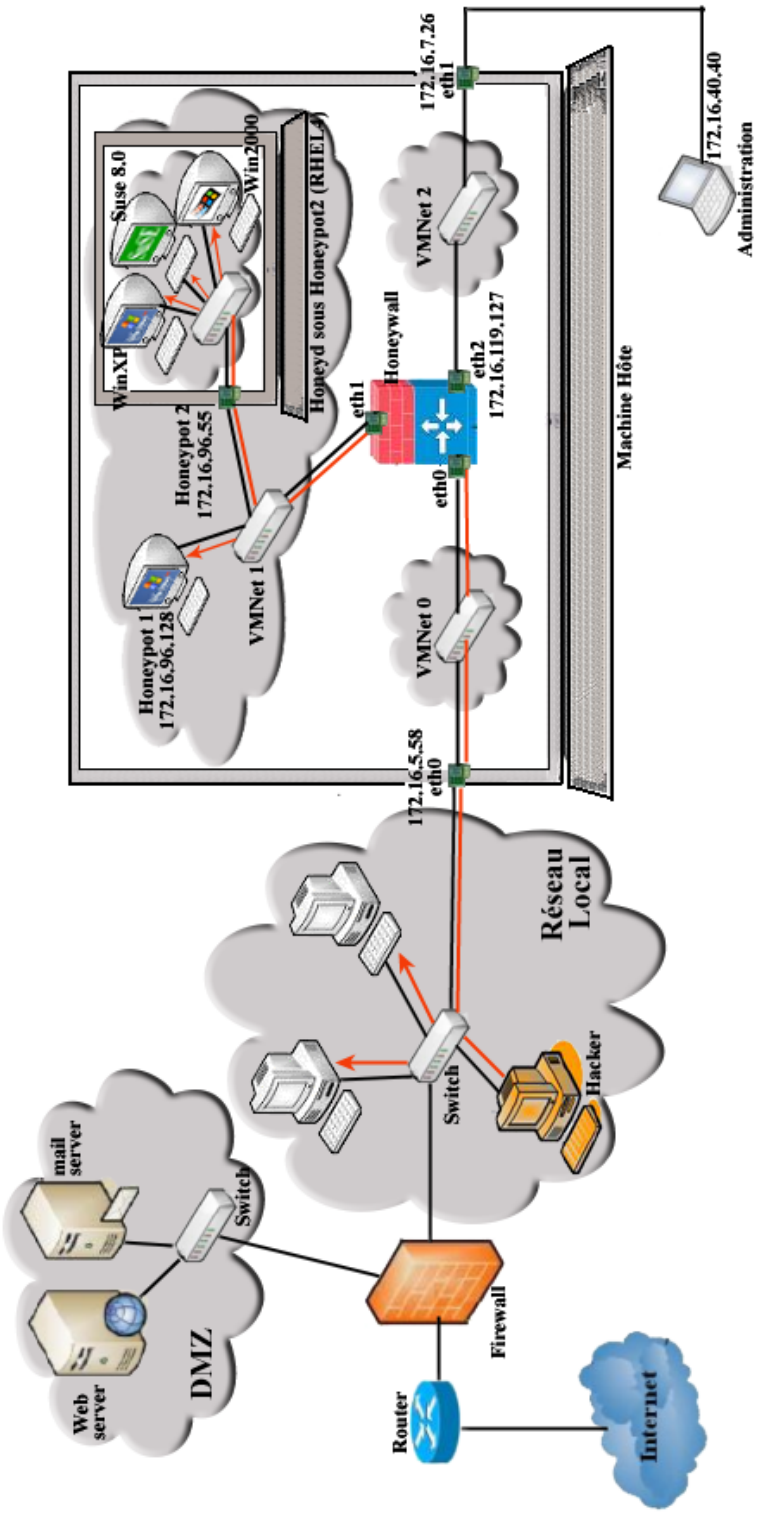


FIG. 3.4 – Architecture globale des deux expérimentations *honeynet ROO* et *honeyd*

3.6 Analyse du trafic capturé

3.6.1 Trafic total (entrant et sortant)

Deux mois après l'installation de la deuxième technologie, nous avons pu capturé 148401 flux de paquets, dont chaque flux est constitué de 1 jusqu'à 20 (ou plus) paquets, 95.78% (142510 flux) de ces flux sont en relation avec le *honeypot* $Windows\ XP$ installé sur la première machine virtuelle créée par $VMWare$. En outre, 68.75% (101584 flux) de ces flux capturés sont des rebondissement d'attaques (trafic sortant), comme le montre le tableau Tab. 3.1 et le diagramme de la figure Fig. 4.4 suivants.

	Honeypots à haute interaction (installés sur des machines de vmware)		Honeypots à faible interaction (créés par Honeyd)			Total
	Honeypot 1 (Windows XP) 172.16.96.128	Honeypot 2 (RHEL 4) 172.16.96.55	Honeypot 3 (Suse 8.0) 172.16.113.31	Honeypot 4 (Windows XP) 172.16.113.32	Honeypot 5 (Wind 2000) 172.16.113.33	
Trafic entrant (Inbound)	41535	3987	629	365	301	46817
Trafic sortant (Outbound)	100615	969	0	0	0	101584
Total	142150	4956	629	365	301	148401

TAB. 3.1 – Quantité du trafic entrant et sortant pour chaque *honeypot*

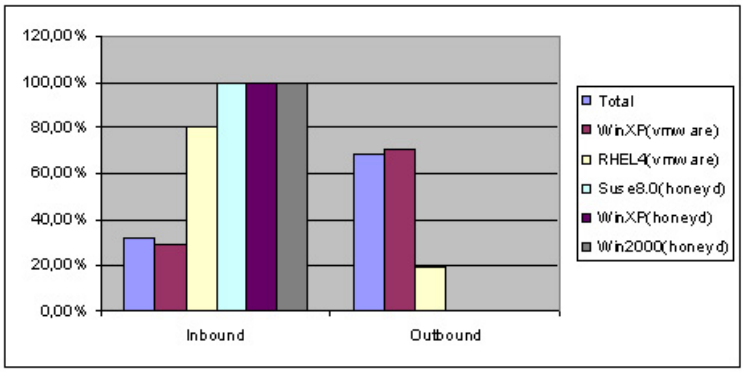


FIG. 3.5 – Trafic capturé en fonction de sa destination

La première remarque importante qu'on peut extraire de ce résultat, c'est la quantité du trafic capturé par les deux premiers *honeypots* (*honeypots* à haute interaction) par rapport à celle du trafic capturé par les trois autres *honeypots* (*honeypots* à faible interaction), il y a 147106 flux (99.12%) pour les deux premiers *honeypots* avec uniquement 1295 flux (0.87%) pour les trois autres *honeypots*, ce qui montre la différence

entre les deux types de *honeypots* en ce qui concerne la quantité du trafic capturé et donc la quantité d'informations révélées sur les activités des attaquants. La deuxième remarque, c'est l'absence du trafic sortant des trois derniers *honeypots*, mais ça c'est normal, car il n'y a pas de vrai système pour ces derniers, en outre les services émulés sont programmés uniquement pour répondre aux différentes requêtes des hackers, donc le système émulé ne pourra jamais initier une connexion vers un autre système.

a. Trafic capturé en fonction des protocoles utilisés.

Le trafic capturé en fonction des protocoles de la couche transport est donné dans le tableau Tab. 3.2 et le diagramme de la figure Fig. 4.5 suivants.

Protocole	Honeypot 1 (Windows XP) 172.16.96.128	Honeypot 2 (RHEL 4) 172.16.96.55	Honeypot 3 (Suse 8.0) 172.16.113.31	Honeypot 4 (Windows XP) 172.16.113.32	Honeypot 5 (Wind 2000) 172.16.113.33	Total
TCP	100318	310	118	87	80	100913
UDP	41084	3899	499	271	219	45972
ICMP	600	745	10	7	2	1364
Autres	148	2	2	0	0	152
Total	142150	4956	629	365	301	148401

Tab. 3.2 – Quantité du trafic capturé par protocole

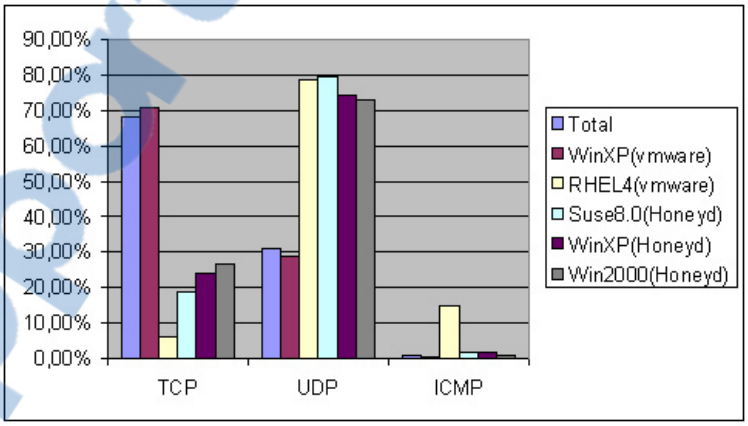


Fig. 3.6 – Trafic capturé en fonction des protocoles utilisés

Comme il est illustré dans ce diagramme, le protocole dominant dans la plupart du trafic capturé par le premier *honeypot* (Win XP) est le protocole TCP, avec un pourcentage de 70.75%, alors que celui dominant dans le reste des *honeypots* est le protocole UDP avec un pourcentage de 78.67% pour le deuxième *honeypot* (RHEL4), 79.33% pour le troisième *honeypot* (Suse 8.0), 74.24% pour le quatrième *honeypot* (Win

XP), et 72.75% pour le cinquième *honeypot* (Win 2000). Il est évident que le pourcentage du trafic total est très proche à celui du premier *honeypot* (Win XP), car la plupart du trafic (95.78%) est en relation avec ce dernier.

b. Trafic capturé en fonction des ports de destination

Lorsque on prend le trafic capturé en fonction des ports de destination, on va obtenir le résultat indiqué dans le tableau Tab. 3.3 et le diagramme de la figure Fig. 4.6.

Protocole	Honeypot 1 (Windows XP) 172.16.96.128	Honeypot 2 (RHEL 4) 172.16.96.55	Honeypot 3 (Suse 8.0) 172.16.113.31	Honeypot 4 (Windows XP) 172.16.113.32	Honeypot 5 (Wind 2000) 172.16.113.33	Total
0/ICMP	271	682	10	7	2	972
smtp (25)	42	0	0	0	0	42
domain (53/UDP)	107	99	0	0	0	206
tftp (69)	31	0	0	0	0	31
http (80/TCP)	66278	20	6	0	0	66304
epmap (135/TCP)	8	7	4	2	2	23
netbios-ns (137/UDP)	30623	3789	499	271	219	35401
netbios-dgm (138/UDP)	8804	0	0	0	0	8804
netbios-ssn (139/TCP)	10506	4	0	0	0	10510
microsoft-ds (445/TCP)	21523	265	108	84	77	22057
autres	3957	90	2	1	1	4051

TAB. 3.3 – Quantité du trafic capturé par port de destination

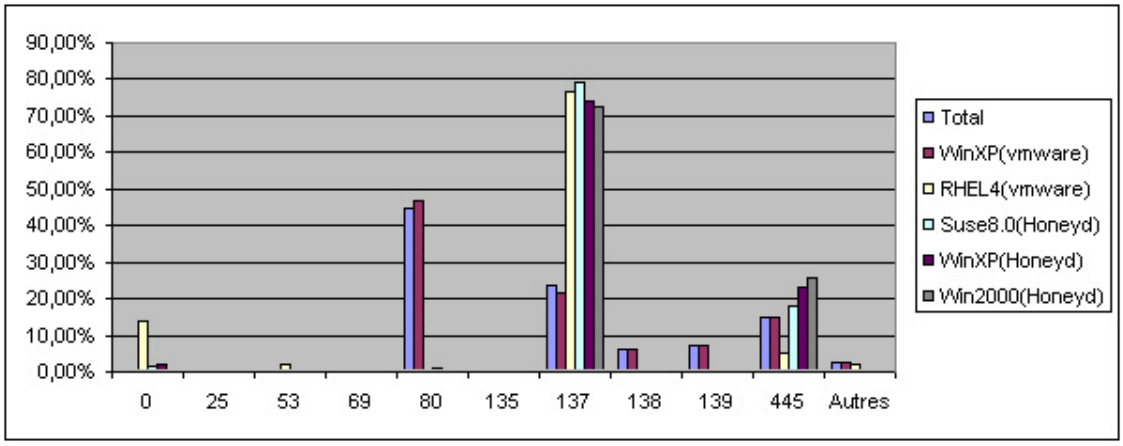


FIG. 3.7 – Trafic capturé en fonction des ports de destination

Comme il est illustré dans cette figure, Les ports les plus attaqués dans le premier *honeypot* (Win XP) sont respectivement : 80 (46.62%), 137 (21.54%), 445 (14.86%), 139 (7.08%), 138 (5.93%), alors que ceux les plus attaqués dans le reste des *honeypots* sont respectivement : le port 137 avec un pourcentage de 76,45% pour le deuxième *honeypot*,

79,33% pour le troisième *honeypot*, 74,24% pour le quatrième *honeypot* et 72,75% pour le cinquième *honeypot*, et le port 445 avec un pourcentage de 05,34% pour le deuxième *honeypot*, 17,77% pour le troisième *honeypot*, 23,01% pour le quatrième *honeypot* et 25,58% pour le cinquième *honeypot*.

c. Trafic et Alertes générées en fonction des ports de destination

Les alertes générées pour chaque port de destination sont illustrées dans le tableau Tab. 3.4 et le diagramme de la figure Fig. 4.7.

Ports	Honeypot 1 (Windows XP) 172.16.96.128		Honeypot 2 (RHEL 4) 172.16.96.55		Honeypot 3 (Suse 8.0) 172.16.113.31		Honeypot 4 (Windows XP) 172.16.113.32		Honeypot 5 (Wind 2000) 172.16.113.33		Total	
	Trafics	alertes	Trafics	alertes	Trafics	alertes	Trafics	alertes	Trafics	alertes	Trafics	alertes
0/ICMP	271	0	682	0	10	0	7	0	2	0	972	0
smtp (25)	42	0	0	0	0	0	0	0	0	0	42	0
domain (53/UDP)	107	3	99	0	0	0	0	0	0	0	206	3
tftp (69)	31	178	0	0	0	0	0	0	0	0	31	178
http (80/TCP)	66278	1714	20	11	6	0	0	0	0	0	66304	1725
epmap (135/TCP)	8	9	7	0	4	0	2	2	2	0	23	11
netbios-ns (137/UDP)	30623	0	3789	0	499	0	271	0	219	0	35401	0
netbios-dgm (138/UDP)	8804	0	0	0	0	0	0	0	0	0	8804	0
netbios-ssn (139/TCP)	10506	4871	4	0	0	0	0	0	0	0	10510	4871
microsoft-ds (445/TCP)	21523	1744	265	48	108	0	84	166	77	152	22057	2110
autres	3957	173	90	0	2	0	1	0	1	0	4051	173

TAB. 3.4 – Nombre d'alertes générées en fonction des ports de destination

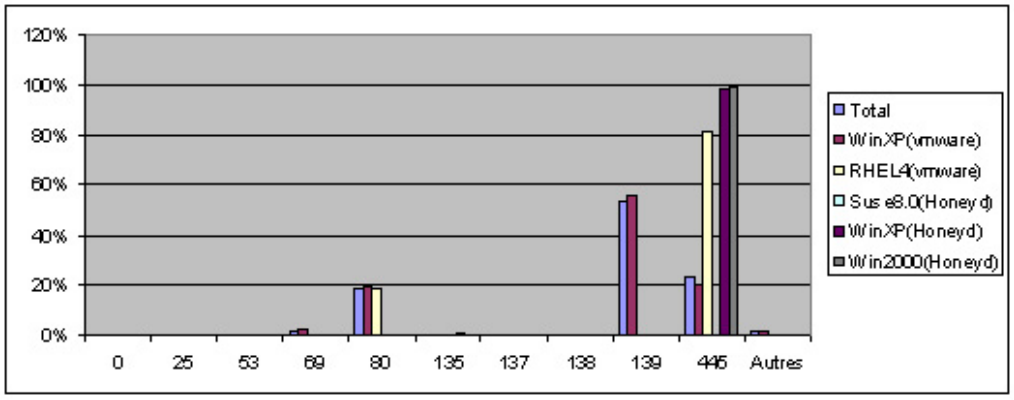


FIG. 3.8 – Alertes générées en fonction des ports de destination

Comme il est illustré dans cette figure, le port qui génère le plus grand nombre d'alertes dans le premier *honeypot* (Win XP) est le port 139 (56,04%), dont parmi les 10506 flux de ce port, il y a 4871 alertes (1 alerte pour chaque 2 flux), alors que celui qui génère le plus grand nombre d'alertes dans les autres *honeypots* est le port 445 avec

un pourcentage de 81.35% pour le deuxième *honeypot* et 98.80% pour le quatrième *honeypot* et 100% pour le cinquième *honeypot*, alors que le troisième *honeypot* (Suse 8.0) n'a généré aucune alerte. Donc la plupart des alertes sont générées par des ports TCP (plus de 96%). Mais lorsque on prend les alertes générées par rapport au trafic total de chaque port, on trouve que le port qui génère le plus grand nombre d'alertes dans le premier *honeypot* (Windows XP) est le port 69 (tftp), dont parmi les 31 flux de ce port, il y a 178 alertes (6 alertes pour chaque 1 flux).

3.6.2 Trafic sortant

Comme nous avons vu dans le tableau Tab. 3.1 précédent, il n'y a pas de trafic sortant pour les *honeypots* à faible interaction créés par *honeyd*, donc nous n'allons pas analyser ce genre de trafics ici car ce dernier est similaire à celui qui a été déjà analysé pendant l'analyse du résultat de la première expérimentation. La même chose pour l'analyse détaillée.

3.7 Conclusion

D'après cette deuxième expérimentation, on peut extraire plusieurs choses intéressantes concernant les deux technologies *honeypot* (*honeynet roo* et *honeyd*), comme par exemple la quantité du trafic capturée pour chaque type de *honeypots*, qui nous a révélé la grande différence entre la quantité capturée pour les deux *honeypots* de la technologie *honeynet roo* par rapport à celle capturée pour les trois *honeypots* de la technologie *honeyd*. Cette expérimentation nous a montré donc que l'utilisation de la technologie *honeyd* dans la recherche est totalement déconseillée à cause de la petite quantité du trafic qu'elle peut capturer, par contre, son utilisation dans les environnements de production comme un détecteur d'intrusions est très efficace, à cause de la possibilité d'émulation d'un grand nombre de machines sur une seule machine réelle d'une part, et l'absence des faux positifs[1] d'autre part. De plus, son utilisation dans l'étude du trafic malicieux destiné à un service bien défini est trés efficace, car dans ce cas il suffit de faire une simulation complète de ce service dans un script, et de configurer ensuite *honeyd* pour rediriger le trafic destiné à ce service vers ce script, et d'analyser enfin tout le trafic logé. Enfin, avec l'évolution des équipes qui travaillent sur cet outil, qu'en a résulte un développement plus rapide et plus perfectionné, on peut dire que cet excellent logiciel puisse devenir un outil indispensable pour le déploiement d'un *honeynet* virtuel, et donc il pourrai être inséré dans les milieux professionnels comme il pourrai aussi être combiné avec d'autres technologies, pour les renforcer.

[1] Alerte en l'absence d'attaque

Chapitre 4

Détection des "honeypots"

4.1 Introduction

AU cours de ces dernières années, et avec l'augmentation considérable des nouvelles attaques inattendues, les technologies *Honeypot* ont suscité un grand intérêt au milieu des communautés de sécurité, qui cherchent à protéger leurs données d'une manière proactive et non plus simplement passive en attirant et en déroutant les éventuelles attaques vers les *Honeypots* pour laisser un temps supplémentaire de réaction à l'administrateur d'une part, et pour observer les moyens de compromissions des pirates d'autre part. Mais avec l'expansion d'utilisation de ces derniers, leur détection est devenue un des objectifs majeurs pour les hackers, qui ne veulent pas que quelqu'un observe leurs actions, puisque cela pourrait mener à la fuite de l'information. Pour cette raison, les hackers essayent toujours d'inventer des nouvelles techniques pour détecter et circonvenir les *Honeypots* et tous les autres environnements suspects. Donc, ils font toujours tous leurs efforts pour détecter s'il y a un *Honeypot* installé dans le système à attaquer avant de lancer leurs attaques. Avec tous ces enjeux, et avec les gros dégâts éventuels qui puisse faire un hacker lorsqu'il détecte l'existence du *honeypot*, la conception de nouvelles technologies *Honeypot* efficaces et indétectables, ainsi que le renforcement des technologies *Honeypot* actuelles par des patchs qui les rendent indétectable, est devenu une des choses les plus indispensable pour les équipes de recherche qui travaillent autour des technologies *Honeypot*, s'ils voulaient vraiment affronter la course aux armements qui a été lancée par la communauté des hackers.

Dans ce chapitre nous allons traiter les problèmes de la détection des deux technologies *honeypnet roo* et *honeyd* les plus utilisées actuellement dans les milieux de sécurité informatique, en essayant de proposer quelques solutions possibles.

4.2 Détection de la technologie "honeynet ROO CDROM"

4.2.1 Problématique

Grâce aux avantages multiples qu'elle offre, que ce soit dans le côté d'utilisation (simplicité d'installation, de configuration et d'analyse de données capturées) ou dans le côté d'efficacité (contrôle et capture de données efficaces), la technologie *honeynet ROO CDROM* est considérée actuellement une des technologies les plus utilisées dans les milieux de recherche en sécurité informatique pour apprendre et découvrir les nouvelles techniques, tactiques et outils utilisés par la communauté des hackers. Et pour pouvoir capturer le maximum d'informations sur le hacker, le *honeynet ROO CDROM* doit bien entendu accepter toutes les connexions entrantes et au contraire limiter les connexions sortantes pour s'assurer qu'un pirate ne puisse pas utiliser un système du *honeynet* pour attaquer ou casser un système extérieur à celui-ci. Cependant, il ne faut en aucun cas interdire toutes les connexions sortantes pour ne pas alerter le pirate. Un bon compromis entre sécurité et risque de découverte du leurre est donc nécessaire. Pour assurer ce bon compromis, la technologie *honeynet ROO CDROM* utilise deux techniques différentes. La première technique consiste à utiliser un système de prévention d'intrusion réseaux ($NIPS$) pour bloquer (ou déjouer) les attaques connues (le $NIPS$ utilisé est $Snort\_inline$), alors que la deuxième technique, c'est la technique de dénombrement des connexions qui consiste à limiter le nombre de connexions sortantes issues du *honeynet* en utilisant l'outil $IPTables$ intégré dans la passerelle *honeywall*, dans lequel nous définissons combien de connexions TCP, UDP, $ICMP$ ou autre ($OTHER$) qui peuvent être établies par un pirate.

Grâce à la limitation de nombre de connexions sortantes, il est difficile de faire beaucoup de dommages dans les systèmes non *honeypots* attaqués à partir du *honeynet*, mais malheureusement, elle rend la détection du leurre très facile, il suffit par exemple de lancer un grand nombre de paquets à partir d'un des *honeypots* compromis du *honeynet* pour découvrir immédiatement le blocage de ces paquets. En outre, cette technique est efficace uniquement lorsque les limites définies sont très petites, mais dans ce cas, la quantité d'apprentissage sera diminuée ce qui écorche d'une manière directe l'objectif principal pour lequel elle a été conçue, surtout lorsque on sait que dans la plupart des cas, la quantité du trafic sortant d'un *honeypot* est plus grande que celle qui y entre, par exemple dans la première expérimentation, nous avons remarqué que 80.95% des trafics capturés sont des trafics sortants. Il est donc très intéressant de proposer de nouvelles techniques efficaces, qui permettent de remplacer la technique de limitation des paquets sortants, pour résoudre le problème de la détection du *honeynet ROO CDROM* d'une part, et pour renforcer la capacité d'apprentissage d'autre part.

4.2.2 Présentation générale de notre technique

En réalité, le seul moyen pour cacher l'existence des *honeypots* c'est d'éviter complètement le blocage du trafic sortant, sauf si ce trafic forme un formidable risque pour le système cible, à part ça la détection du *honeypot* est toujours possible quelque soit sa nature, car il suffit toujours pour l'attaquant de lancer une série de paquets à partir d'un des *honeypots* compromis vers une machine qu'il contrôle, pour remarquer ensuite l'absence de la trace de ce trafic dans les logs de cette machine, ce qui lui montre qu'aucun des paquets transmis n'y est arrivé, donc il détecte directement la présence du *honeypot*. Et pour pouvoir trouver une technique efficace permettant d'éliminer le blocage du trafic sortant sans provoquer des dégâts sur les machines attaquées du monde externe, nous devons d'abord connaître le principe général des attaques réseaux.

En effet, les attaques réseaux s'appuient sur des vulnérabilités liées directement aux implémentations des protocoles de communication réseau comme IP, TCP et UDP, et celles des services des ports ouverts dans la machine cible et des applications qu'ils mettent en œuvre, donc le seul moyen pour pouvoir attaquer et endommager une machine distante est d'exploiter une des vulnérabilités des protocoles et des services des ports ouverts dans cette machine [Det05, DPD04]. Autrement dit, les attaques réseaux sont applicables sur n'importe quelle machine qui offre le service vulnérable.

En se basant sur ce principe, nous pouvons proposer une nouvelle technique de contrôle du trafic sortant qui va nous permettre de résoudre le problème de la détection du *honeynet ROO CDROM*. Le principe de cette nouvelle technique consiste à insérer un relais (*proxy*) entre le *honeynet* et les machines réelles non *honeypots*, ce relais va jouer le rôle d'une cuirasse de protection pour protéger les machines du monde externe contre les attaques rebondissantes du *honeynet*, donc tous les paquets rebondissants seront d'abord passés, traités et testés par ce relais avant de sortir du *honeynet*, et pour pouvoir assurer son rôle d'une manière efficace, le relais doit être capable de traiter tous les paquets que le transitent quelque soit la nature des protocoles qu'ils utilisent, donc il doit avoir une connaissance particulière sur tous les protocoles réseaux, ce qui rend son implémentation trés difficile voire impossible, et pour résoudre ce problème nous proposons d'utiliser une autre machine (appelée "machine de test") dans laquelle nous allons ouvrir d'une manière standard le maximum possible de ports réseaux, avec l'activation de toutes les options des services de ces ports, et comme ça, il suffit pour le relais de transférer le paquet rebondissant vers cette machine pour le tester s'il va générer un dommage sur celle-ci ou non avant de le laisser passer vers la machine du monde externe. De cette manière, les dommages éventuels seront produits sur la machine de test dédiée au lieu d'être produits sur la machine réelle.

L'architecture générale de cette nouvelle technique est illustrée dans la figure Fig.4.1 suivante.

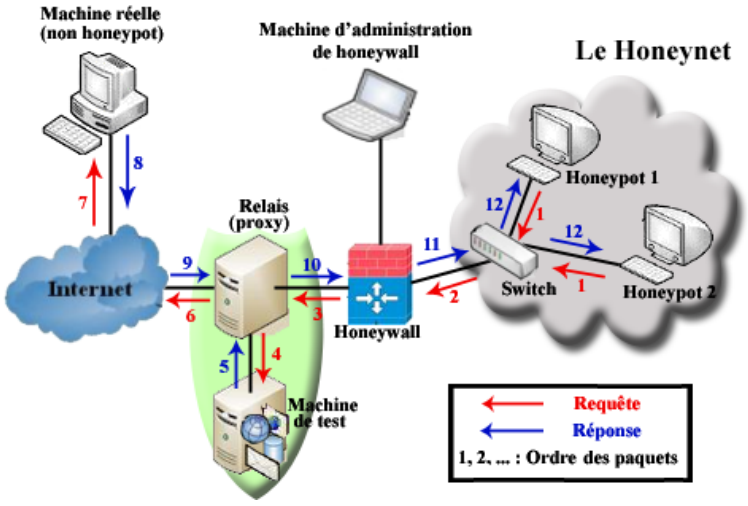


FIG. 4.1 – Architecture générale de notre nouvelle technique

La mise en place de cette architecture nécessite deux autres machines dédiées que celles nécessaires pour mettre en place l'ancienne technique, donc il y a un coût additionnel exigé pour réaliser cette nouvelle technique. De plus, le relais doit fonctionner en mode *bridge* comme le *honeywall* pour être indétectable, ce qui nécessite un effort additionnel pour l'implémenter. Pour résoudre ces deux problèmes, il suffit d'implémenter le relais directement sur le *honeywall*, et comme ça nous n'avons pas besoin d'une nouvelle implémentation en mode bridge d'une part, et nous allons économiser une machine de celles exigées d'autre part. Donc l'architecture générale de notre technique devient comme le montre la figure Fig.4.2 suivante.

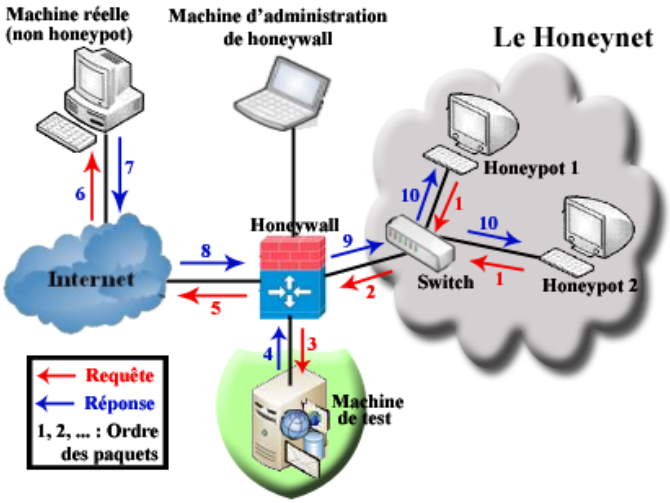


FIG. 4.2 – Architecture générale réduite de notre nouvelle technique

4.2.3 Modes de fonctionnement du relais de notre technique

Pour donner un plus de souplesse dans le traitement de risques, nous pouvons enrichir le fonctionnement principal du relais par un ou plusieurs modes de fonctionnement supplémentaires, chacun de ces modes offre un degré de risque disparate. De cette manière, chaque personne va prendre le mode qui lui permet d'atteindre ses objectifs avec le moindre risque qu'il est en mesure d'assumer.

4.2.3.1 Mode avec risque

C'est le mode principal par défaut du relais, et tous les autres modes ne sont que des extensions de ce mode. Le rôle principal du relais dans ce mode se concentre principalement sur la redirection du trafic sortant vers la machine de test pour tester sa dangerosité avant de le retransmettre vers sa destination réelle, afin d'assurer que le trafic dangereux ne sera jamais destiné vers les machines des autres tant que le service visé est offert par la machine de test, car dans ce cas, le dommage éventuel va être produit dans cette dernière au lieu d'être produit sur la machine du monde externe.

Donc d'une manière générale, nous pouvons résumer le fonctionnement standard du relais de ce mode dans les points suivants :

- Lorsque le relais reçoit un paquet rebondissant du *honeynet*, il vérifie d'abord dans le fichier de configuration si le port de destination de ce paquet est ouvert dans la machine de test, si ce n'est pas le cas, il laisse le trafic sortir sans aucun test. Sinon il mémorise une copie de ce paquet dans un tampon, puis il change l'adresse de destination de ce paquet par celle de la machine de test et l'achemine vers elle et arme un temporisateur d'attente de la réponse.
- Lorsque la machine de test reçoit ce paquet, elle le traite puis elle répond vers le relais.
- Lorsque le relais reçoit la réponse de la machine de test, il teste de son côté si cette réponse ne correspond pas à un dommage de la machine de test, puis il retransmit la copie mémorisée dans le tampon vers sa destination réelle.

Les dommages de la machine de test causés par les paquets rebondissants sont détectés par le relais par une des deux façons suivantes :

1. **La temporisation a expiré sans réception de réponse de la machine de test :** Dans ce cas, le relais renvoie quelques paquets de confirmation vers le port concerné de la machine de test, et s'il n'y a pas toujours de réponse, il considère que le service est endommagé, et dans ce cas il génère une alerte et ferme la connexion établie avec la machine réelle (envoi d'un paquet TCP "FIN") sans éveiller les soupçons d'attaquant qui n'attend aucun paquet de la victime après l'attaque. Un exemple simple de ce type d'attaques est celui de *"ping of death"*.

2. **Il reçoit un paquet de la machine de test indiquant le dommage du service demandé,** comme par exemple "Hôte inaccessible", "port fermé (réception de paquet RST)", "port occupé (réception de paquet RST+ACK)"...etc, et dans ce cas il fait la même chose, il renvoie quelques paquets de confirmation vers le port concerné, et s'il toujours reçoit le même message, alors il génère une alerte et il ferme directement la connexion établie avec la machine externe, puis il fait passer les réponses de la machine de test vers le *honeypot*, après faire bien sûr les changements nécessaires (changement d'adresse IP source et destination, changement du TTL, ...etc) pour éviter tout soupçon de la part de l'attaquant. Et lorsque le dommage est spontanément disparu, alors la conversation entre le *honeypot* et la machine externe sera rétablie de nouveau.

Avec ce mode de fonctionnement, la détection du *honeynet* est très difficile voire impossible, quelque soit la volonté, l'expérience et l'intelligence d'attaquant tant qu'il n'y a pas de blocage du trafic sortant, mais il reste encore possible de faire des dégâts dans les machines externes de *honeynet* lorsque le service utilisé dans l'attaque n'est pas offert par la machine de test, donc pour minimiser le maximum de risques il faut offrir le maximum de services (ports) dans la machine de test avec l'activation de toutes leurs options. Donc, ce mode de fonctionnement est conçu principalement pour ceux qui veulent apprendre le maximum sur les activités des pirates expérimentés qui ont la volonté de détecter le *honeynet* avant de commencer leurs attaques.

4.2.3.2 Mode avec risque réduit

Le rôle principal du relais de ce mode est le même que celui du mode précédent avec quelques améliorations qui permettent de minimiser le risque le plus que possible, en éliminant la permission au trafic destiné vers les ports fermés de la machine de test de sortir vers l'extérieur, et en ajoutant la possibilité de mémorisation des dernières réponses des machines externes dans un tampon pour pouvoir les retourner rapidement s'il reçoit une requête identique, ce qui lui permet d'éviter les dommages causés par les attaques de déni de service classiques et celles de déni de service distribuées qui sont généralement basées sur l'envoi d'une série de paquets identiques vers la victime pour saturer son service attaqué en le rendant inaccessible, comme le cas par exemple d'une attaque *"TCP SYN Flood"*.

Donc dans ce mode, lorsque le relais reçoit un paquet rebondissant du *honeynet*, il vérifie d'abord s'il y a une requête identique dans le tampon, si c'est le cas il répond directement par la réponse mémorisée avec cette requête, sinon il vérifie dans le fichier de configuration si le port de destination de ce paquet est ouvert dans la machine de test, si ce n'est pas le cas, il réponde directement par un paquet indiquant que le port

demandé est fermé. Sinon il mémorise une copie de ce paquet dans le tampon, puis il change son adresse de destination par celle de la machine de test et l'achemine vers elle et arme un temporisateur d'attente de la réponse.

Avec ce mode de fonctionnement, nous pouvons éviter toutes les actions qui peuvent éveiller les soupçons d'attaquant qui ne cherche pas à détecter le *honeynet*, ce qui nous permet d'en apprendre le maximum sans provocation des dégâts dans les systèmes externes de *honeynet*. Mais en effet sa détection reste encore possible tant qu'il y a un blocage du trafic, tel qu'il suffit pour un attaquant expérimenté de développer une *socket server* de test sur un port quelconque non utilisé par les services connus, et de l'installer dans une machine qu'il contrôle, et de lancer ensuite une grande série de paquets à partir d'un des *honeypots* compromis vers cette *socket* pour pouvoir détecter la non arrivée de ces paquets. Donc, ce mode de fonctionnement est conçu pour ceux qui veulent apprendre le maximum sur les activités des pirates inexpérimentés qui ne cherchent pas à détecter le *honeynet* avec la minimisation maximum des risques.

4.2.3.3 Mode sans risque

Dans ce mode de fonctionnement, le relais va nous permettre d'éviter complètement tous les risques possibles, en se basant sur l'interception de tous les paquets sortants, en les redirigeant directement vers la machine de test, et lorsqu'il reçoit la réponse de cette dernière, il fait les changements nécessaires sur les champs du paquet de cette réponse pour éviter tout soupçon (changement d'adresse source, changement de TTL, ...etc), puis il l'achemine vers le *honeypot* qui a envoyé la requête correspondante.

Avec ce mode de fonctionnement, les attaques rebondissantes se font uniquement entre les *honeypots* et la machine de test, donc aucun paquet ne plus sortit vers l'extérieur du *honeynet*, ce qui permet d'éviter complètement tous les risques éventuels. Mais en effet, ce mode ne permet pas de résoudre le problème de détection du *honeynet*, car sa détection est relativement facile, tel qu'il suffit par exemple d'initier une connexion à partir d'un des *honeypots* compromis vers un service d'une machine qu'il sait d'avance qu'elle ne le dispose pas pour remarquer qu'il y a des réponses malgré l'absence de ce service. Donc l'utilisation de ce mode est conseillée uniquement pour les personnes qui ne s'intéressent pas par le problème de la détection du *honeynet*, et pour ceux qui veulent éviter tous les types d'attaques rebondissantes, que ce soit des attaques conduisant à des dommages ou des attaques qui ne conduisent pas à des dommages.

Remarque : Le choix entre ces trois modes de fonctionnement est effectué pendant la configuration du relais.

4.2.4 Avantages de notre technique par rapport à l'ancienne technique

En effet, notre nouvelle technique ne se contente pas uniquement par la résolution du problème de détection de la technologie *honeynet*, mais elle offre d'autres avantages par rapport l'ancienne technique que nous pouvons les résumer dans le tableau suivant :

Notre technique	**L'ancienne technique**
Détection de *honeynet* très difficile.	Détection de *honeynet* très facile.
Quantité d'apprentissage illimitée.	Quantité d'apprentissage limitée à cause du blocage du trafic sortant.
Les dommages causés par les paquets rebondissants sont immédiatement remarquables en temps réel, ce qui accélère la création des signatures correspondantes aux nouveaux types d'attaques.	Les dommages causés par les paquets rebondissants sont indétectables, ce qui retarde la création des signatures correspondantes aux nouveaux types d'attaques.
Elle offre un autre niveau de capture de données dans la machine de test, ce qui offre la possibilité d'apprentissage sur les conséquences du trafic sortant.	L'apprentissage sur les conséquences du trafic sortant est impossible.
Nous pouvons éviter le risque même s'il est porté dans les premiers paquets rebondissants, comme celui du *"ping of death"*.	Le risque porté dans les premiers paquets est incontrôlable.

TAB. 4.1 – Avantages de notre technique par rapport à l'ancienne technique

4.2.5 Problèmes qui peuvent être rencontrés lors d'implémentation de notre technique

Pour implémenter notre technique, nous devons prendre en considération les deux problèmes suivants :

– Le premier problème que nous devons le prendre en considération pendant l'implémentation de notre technique, c'est le problème de minimisation du temps perdu pendant le test du trafic sortant, où l'implémentation de cette technique doit assurer que ce temps perdu LT (*lost time*) plus le temps de réponse de la machine réelle RT (*response time*) soit inférieur au temps de retransmission RTO (*Retransmission Time Out*) : LT + RT < RTO.

Heureusement, ce problème ne se pose que rarement, car la nature et la structure d'Internet imposent un calcul dynamique du RTO en fonction des délais d'acquittement des segments précédents ("*segment round-trip time*") ce qui permet à l'émetteur de s'adapter à tous les débits et à tous les temps de réponse selon l'état et la charge présente du réseau Internet. Mais il y a toujours une valeur maximum (limite) de la temporisation de retransmission notée UBOUND (max(RTO)=UBOUND) qui peut être initiée manuellement lors de la configuration du système. Généralement, UBOUND prend une valeur suffisante pour faire un aller retour entre les deux points les plus éloignés dans le réseau Internet, en prenant en considération tous les traitements effectués par les routeurs transités (de traitement de la couche physique jusqu'à la couche réseau et de cette dernière jusqu'à la couche physique) et le temps perdu dans les files d'attente de ces derniers. Donc UBOUND est généralement suffisante pour faire le test du trafic sortant et pour recevoir la réponse de la machine réelle, donc ce problème ne se pose que lorsque le réseau Internet est trop chargé.

Et pour augmenter la chance d'évitement de ce problème, il faut utiliser des machines compétentes pour installer le relais et la machine de test.

– Le deuxième problème que nous allons rencontrer pendant l'installation de notre technique, c'est le problème de choix d'implémentation du service à utiliser dans le cas où il y a plusieurs implémentations différentes pour le même service à installer, par exemple pour le service *HTTP* il y a plusieurs implémentations différentes comme : *"Apache HTTP Server", "Microsoft IIS", "Sun ONE", "Zeus Web",* ...etc. Pour résoudre ce problème, nous pouvons utiliser une des deux solutions suivantes :

  1. Utiliser l'implémentation la plus utilisée. Donc dans le cas du serveur *HTTP* nous allons utiliser le serveur *"Apache HTTP Server"* qui sert environ 69% des sites Web en octobre 2007 selon *Netcraft*[1].
  2. Installer l'implémentation la plus utilisée dans le port standard, et les autres implémentations dans des ports différents, ensuite c'est le relais qui décide quelle implémentation va utiliser, suivant l'implémentation utilisée dans la machine réelle (il utilise un outil passif de *fingerprinting* pour découvrir l'implémentation utilisée dans la machine réelle), sinon il fera passer le paquet sur toutes les implémentations, mais ça va prendre un temps considérable.

Remarque : C'est la même chose pour le problème de choix du noyau système (Windows, Linux) à utiliser dans la machine de test.

[1]Netcraft est une entreprise spécialisée dans les sondages automatisés d'Internet.

4.3 Détection de la technologie "honeyd"

Une des méthodes les plus simples et les plus efficaces pour découvrir les failles de la technologie *honeyd* qui conduisent à sa détection, est d'analyser les comportements des différents paquets générés par cette technologie, et de les comparer ensuite avec les comportements des mêmes types de paquets générés par une autre machine non *honeypot*, afin de découvrir ensuite les comportements anormaux générés par *honeyd*, et puisque les paquets les plus générés par la technologie *honeyd* sont des paquets ARP, donc c'est ce type de paquets qui sera testé dans l'expérimentation que nous allons effectuer dans ce chapitre.

4.3.1 Classification des machines de notre réseau

Pour faciliter la distinction entre les machines physiques réelles et les différentes machines virtuelles créées par *honeyd*, nous allons classifier les machines du réseau de notre expérimentation en trois classes différentes :

- **Classe 1 :** Machines physiques réelles non *honeypots*,
- **Classe 2 :** Machines virtuelles créées par *honeyd*, correspondantes à des chablons (*templates*) différents de chablon par défaut définis dans le fichier de configuration. Dans cette classe le nombre de machines virtuelles et leurs adresses IP sont fixés d'avance dans le fichier de configuration,
- **Classe 3 :** Machines virtuelles créées par *honeyd*, correspondantes au chablon par défaut défini dans le fichier de configuration. Dans cette classe le nombre de machines virtuelles est dynamique et leurs adresses sont dépendantes des adresses libres dans le réseau physique.

Tout au long de cette partie nous allons essayer de comparer le comportement des machines des deux dernières classes avec le comportement de celles de la première classe, en essayant d'extraire quelques comportements anormaux qui peuvent conduire à la détection de la technologie *honeyd*.

4.3.2 Expérimentation

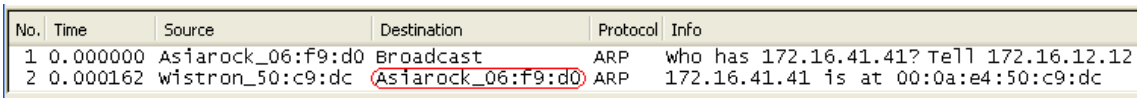
Pour montrer les failles de la technologie *honeyd*, nous avons effectué l'expérience suivante. Nous avons installé *honeyd* dans une machine hôte située dans le réseau local de notre université, et à partir de *honeyd* nous avons construit trois autres machines virtuelles (*honeypots*). Le premier *honeypot* est un *Linux Suse* 8.0, Le deuxième *honeypot* est un $Microsoft\ Windows\ XP\ Professional\ SP1$, et le dernier *honeypot* est un $Windows$ 2000. Chacun de ces *honeypots* offre les services : $http$, ftp, $smtp$, $telnet$, $netbios-ns$, $netbios-dgm$, $netbios-ssn$, $microsoft-ds$. Les adresses IP associées

à ces trois machines sont successivement : 172.16.12.13, 172.16.12.14, 172.16.12.15.

Le reste des adresses de la plage 172.16.12.0/24 qui ne sont pas encore affectées ni aux machines de la première classe (physiques réelles) ni aux celles de la deuxième classes (les trois machines précédentes) sont affectées automatiquement aux machines de la troisième classe définie dans le chablon par defaut.

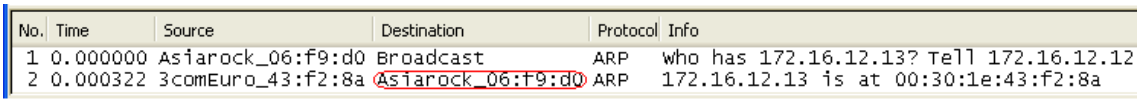
4.3.2.1 Le premier test et la faille détectée

Dans le premier test, nous avons lancé une capture du trafic ARP entrant et sortant de notre machine de test en utilisant le sniffer *Ethereal*, puis nous avons lancé trois *pings*, le premier a été destiné vers la machine 172.16.41.41 de la première classe (machine réelle), le deuxième a été destiné vers la machine 172.16.12.13 de la deuxième classe et le troisième a été destiné vers la machine 172.16.12.1 de la troisième classe. Le trafic ARP capturé par *Ethereal* pendant l'exécution de ces trois *pings* est illustré dans les trois figures Fig.4.3 Fig.4.4 et Fig.4.5 suivantes :



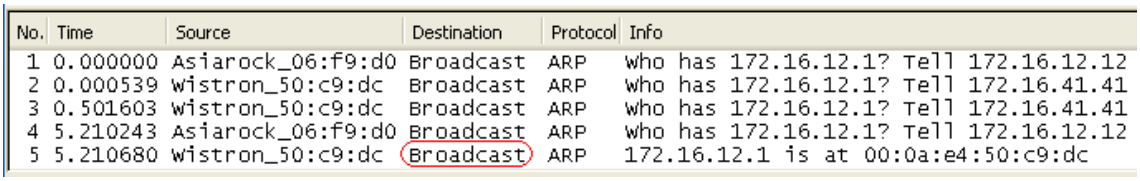
No.	Time	Source	Destination	Protocol	Info
1	0.000000	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.41.41? Tell 172.16.12.12
2	0.000162	Wistron_50:c9:dc	Asiarock_06:f9:d0	ARP	172.16.41.41 is at 00:0a:e4:50:c9:dc

FIG. 4.3 – Trafic ARP pour le premier *ping*



No.	Time	Source	Destination	Protocol	Info
1	0.000000	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.12.13? Tell 172.16.12.12
2	0.000322	3comEuro_43:f2:8a	Asiarock_06:f9:d0	ARP	172.16.12.13 is at 00:30:1e:43:f2:8a

FIG. 4.4 – Trafic ARP pour le deuxième *ping*



No.	Time	Source	Destination	Protocol	Info
1	0.000000	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.12.1? Tell 172.16.12.12
2	0.000539	Wistron_50:c9:dc	Broadcast	ARP	Who has 172.16.12.1? Tell 172.16.41.41
3	0.501603	Wistron_50:c9:dc	Broadcast	ARP	Who has 172.16.12.1? Tell 172.16.41.41
4	5.210243	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.12.1? Tell 172.16.12.12
5	5.210680	Wistron_50:c9:dc	Broadcast	ARP	172.16.12.1 is at 00:0a:e4:50:c9:dc

FIG. 4.5 – Trafic ARP pour le troisième *ping*

Dans les trois figures, la machine de test a diffusé un paquet ARP porte la question *"who has $IP_ADDRESS"* pour obtenir l'adresse MAC de la machine de destination (voir la première ligne dans les trois figures), puis chaque machine d'adresse IP en

question a répondu par un paquet ARP porte son adresse MAC. Pour la machine de la première classe (machine réelle) et celle de la deuxième classe, la réponse a été destinée directement vers la machine de test qui a demandé cette adresse (voir la destination de la réponse dans le petit cercle rouge des deux premières figures), par contre, pour la machine de la troisième classe (machine virtuelle par défaut), le paquet de réponse a été diffusé vers toutes les machines du réseau (voir la destination de la réponse dans le petit cercle rouge de la dernière figure). Cette diffusion est considérée comme un comportement anormal du protocole ARP géré par *honeyd*, donc en exploitant cette faille, un attaquant pourra facilement détecter la présence de *honeyd* avec toutes les machines qu'il contrôle, en lançant un scan de machines du réseau avec une capture de tous les paquets ARP reçus, pour extraire ensuite toutes les adresses des machines qui ont diffusé leur réponse. Ces adresses extraites représentent les machines contrôlées par *honeyd*.

Pour résoudre ce problème, il faut modifier le code source du daemon *arpd*, en annulant la diffusion des réponses ARP.

4.3.2.2 Le deuxième test et les deux failles détectées

Dans le deuxième test, et à partir de la machine de test 172.16.12.12, nous avons lancé une série de *pings* vers 5 machines de la troisième classe. Le trafic ARP capturé par *Ethereal* pendant l'exécution de cette série de *pings* est illustré dans la figure Fig.4.6 suivante.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.1? Tell 172.16.12.12
2	0.000560	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.1? Tell 172.16.41.41
3	0.501334	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.1? Tell 172.16.41.41
4	5.427305	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.1? Tell 172.16.12.12
5	5.427644	Wistron_50:c9:dc	Broadcast	ARP	172.16.12.1 is at 00:0a:e4:50:c9:dc
6	11.294091	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.2? Tell 172.16.12.12
7	11.294681	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.2? Tell 172.16.41.41
8	11.795319	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.2? Tell 172.16.41.41
9	16.427041	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.2? Tell 172.16.12.12
10	16.427390	Wistron_50:c9:dc	Broadcast	ARP	172.16.12.2 is at 00:0a:e4:50:c9:dc
11	23.267208	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.3? Tell 172.16.12.12
12	23.267756	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.3? Tell 172.16.41.41
13	23.768163	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.3? Tell 172.16.41.41
14	28.426856	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.3? Tell 172.16.12.12
15	28.427197	Wistron_50:c9:dc	Broadcast	ARP	172.16.12.3 is at 00:0a:e4:50:c9:dc
16	33.921141	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.4? Tell 172.16.12.12
17	33.921676	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.4? Tell 172.16.41.41
18	34.422287	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.4? Tell 172.16.41.41
19	38.926778	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.4? Tell 172.16.12.12
20	38.927114	Wistron_50:c9:dc	Broadcast	ARP	172.16.12.4 is at 00:0a:e4:50:c9:dc
21	43.903856	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.5? Tell 172.16.12.12
22	43.904402	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.5? Tell 172.16.41.41
23	44.405493	Wistron_50:c9:dc	Broadcast	ARP	who has 172.16.12.5? Tell 172.16.41.41
24	48.926258	Asiarock_06:f9:d0	Broadcast	ARP	who has 172.16.12.5? Tell 172.16.12.12
25	48.926578	Wistron_50:c9:dc	Broadcast	ARP	172.16.12.5 is at 00:0a:e4:50:c9:dc

FIG. 4.6 – Paquets ARP capturés par *Ethereal* pendant l'exécution des 5 *pings*

Dans cette figure nous pouvons remarquer la deuxième et la troisième failles qui peuvent conduire à la détection de *honeyd* :

- La deuxième faille concerne le trafic encadré en rouge dans la figure (les grands cadres dans la figure), dans ce type de trafics nous remarquons que à chaque fois que le daemon *arpd* reçoit une requête ARP pour le MAC d'une des adresses IP des machines de la troisième classe, il diffuse lui-même une requête identique à celle reçue pour déterminer s'il y a une machine de la première classe (machine réelle) qui porte cette adresse IP ou non. Si ce n'est pas le cas, il répond à la requête ARP avec l'adresse MAC de l'hôte *honeyd*. Mais malheureusement, ce comportement est considéré comme suspect par un hacker, tel qu'il suffit pour lui de lancer un scan ARP vers toutes les machines du réseau, pour extraire ensuite l'adresse de la machine qui reproduit la diffusion de la même requête en cours pour plusieurs requêtes différentes. Cette adresse extraite représente l'adresse de la machine hébergeant *honeyd*.
- La troisième faille concerne les adresses MAC des réponses ARP. Comme le montre la figure (les petits cadres bleus dans la figure), toutes les adresses des réponses sont identiques, bien que nous avons donné un type différent pour chaque machine de la troisième classe dans le chablon par défaut du fichier de configuration de *honeyd*, ce comportement signifier que toutes ces différentes adresses IP ont la même adresse MAC, mais en réalité ça c'est impossible, donc ce comportement est suspect, et offre la possibilité de détection de *honeyd* aux hackers, qui leur suffit de lancer un scan ARP vers toutes les machines du réseau, pour extraire ensuite les adresses IP qui ont la même adresse MAC, ces dernières représentent les machines de la troisième classe (machines virtuelles du chablon par défaut) gérées par *honeyd*.

4.3.2.3 Le troisième test et la faille détectée

Dans le troisième test nous avons affecté l'adresse IP d'une machine de la première classe (machine réelle) à une nouvelle machine de la deuxième classe créée par *honeyd*. Cette adresse est : 172.16.41.41. Ensuite, nous avons relancé *honeyd*, puis nous avons lancé un *ping* vers cette machine (machine 172.16.41.41). Le trafic ARP capturé par *Ethereal* pendant l'exécution de ce *ping* est illustré dans la figure Fig.4.7 suivante :

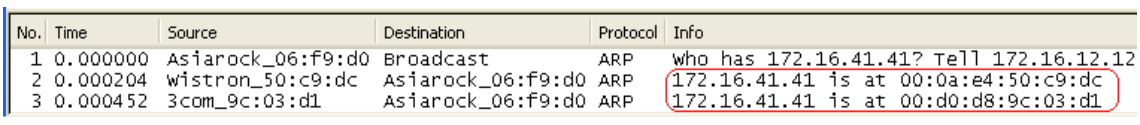


FIG. 4.7 – Paquets ARP correspondants à la quatrième faille

Comme il est montré dans cette figure, pour la requête ARP correspondante à cette commande *ping* (le cadre rouge de la figure), le daemon *arpd* répond directement sans confirmation si cette adresse est déjà associée à une autre machine ou non, puisque il considère que cette adresse est réservée pour un de ses *honeypots*, donc si cette adresse est déjà affectée à une autre machine réelle comme dans notre cas, alors nous aurons un conflit d'adresses MAC comme le montre la figure. Ce conflit peut être détecté facilement par un hacker sniffant le réseau, comme il peut aussi conduire à des problèmes dans notre réseau.

4.3.3 Solutions proposées

Pour remédier tous ces problèmes, nous proposons d'enrichir la technologie *honeyd* par quatre nouveaux modules, chacun d'eux se charge par le patching d'une de ces failles, ces quatre modules sont :

- Module de construction de la liste des adresses IP allouées ($ModIPAlloues$),
- Module de translation d'adresse MAC ($ModTransMAC$),
- Module générateur d'adresses MAC virtuelles ($ModGnrateurMAC$),
- Module de vérification des adresses IP des machines virtuelles de la deuxième classe ($ModIPHoneypots$).

La nouvelle structure générale de la technologie *honeyd* après l'intégration de ces quatre modules devient comme le montre la figure Fig.4.8 suivante :

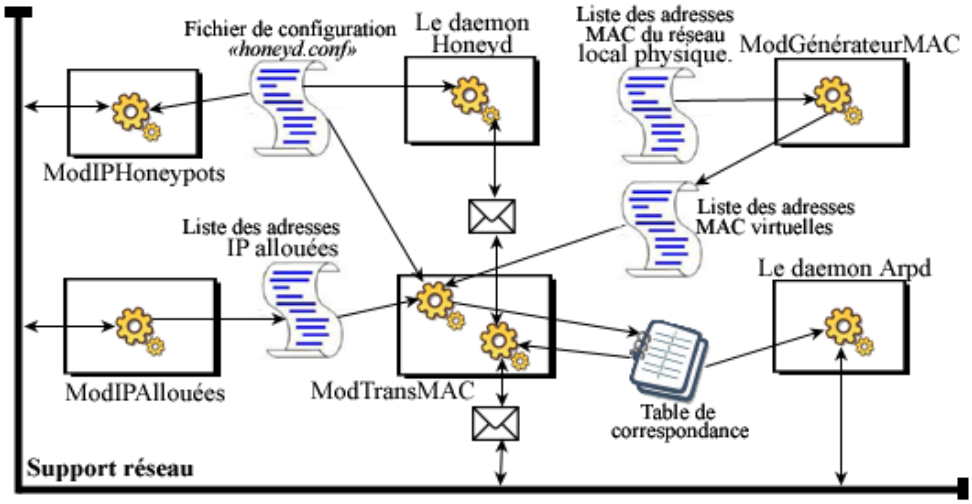


FIG. 4.8 – La nouvelle structure générale de la technologie *honeyd*

1. **Module de construction de la liste des adresses IP allouées ($ModIPAlloues$) :**
   Ce module est conçu pour résoudre le problème de collection des informations sur

les adresses IP libres de la plage des adresses contrôlées par le daemon *arpd*, l'ancienne implémentation utilise la technique de rediffusion des requetes ARP reçues, mais comme nous avons vu dans le deuxième test, cette technique conduit facilement à la détection de la technologie *honeyd*, donc pour résoudre ce problème nous devons changer complètement cette politique, en la remplaçant par une nouvelle technique qui nous permet d'obtenir la liste des adresses IP libres sans éveiller les soupçons d'attaquant, soit en utilisant une technique passive, soit en utilisant des techniques actives sans *broadcasting*.

En effet, lorsque le réseau local utilise un serveur DHCP pour affecter dynamiquement les adresses IP aux différentes machines du réseau, la meilleure façon pour obtenir la liste des adresses IP déjà affectées aux différentes machines réelles c'est de l'extraire de ce serveur DHCP, soit passivement en sniffant les paquets entrants et sortants de ce serveur, soit activement en l'en demandant périodiquement. Et pour assurer cette opération d'une manière efficace, la machine hébergeant *honeyd* doit être installée dans le même segment réseau de la machine DHCP, pour pouvoir assurer le sniffing de tous les paquets sortants et entrants de cette dernière, en évitant les blocages des commutateurs (*switchs*).

A partir de la liste des adresses libres obtenue, le daemon *arpd* répond directement aux requêtes ARP reçues sans attendre la confirmation que personne n'utilise cette adresse, ce qui lui permet d'éviter complètement tous les soupçons. Donc de cette manière, la deuxième faille est totalement remédiée.

Pour montrer comment il est possible d'acquérir la liste des adresses IP libres en utilisant la technique de sniffing des paquets DHCP, nous avons effectué le test suivant :

Nous avons installé un serveur DHCP dans une machine Linux (RHEL4), et à partir d'une autre machine Windows XP, nous avons lancé le sniffer *Ethereal*. Puis, à partir du panneau de configuration de Windows, nous avons indiqué que la carte réseau doit recevoir une adresse IP dynamiquement.

Le trafic DHCP capturé par *Ethereal* est affiché dans la figure suivante :

No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x20a(
2	0.001307	Wistron_50:c9:dc	Broadcast	ARP	Who has 172.16.0.9? Tell 172.16.41.41
3	0.204457	172.16.41.41	172.16.0.9	DHCP	DHCP Offer - Transaction ID 0x20a(
4	0.206365	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x4b7z
5	0.210659	172.16.41.41	172.16.0.9	DHCP	DHCP ACK - Transaction ID 0x4b7z
6	0.302459	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.0.9? Gratuitous ARP
7	0.865112	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.0.9? Gratuitous ARP
8	1.865117	Asiarock_06:f9:d0	Broadcast	ARP	Who has 172.16.0.9? Gratuitous ARP
9	2.793477	172.16.0.9	172.16.255.255	NBNS	Registration NB BOULAICH-98FE11<00>

FIG. 4.9 – Le trafic DHCP capturé par *Ethereal*

(1) Pour obtenir une adresse IP dynamiquement, la machine Windows a diffusé (broadcast général "255.255.255.255") un paquet "*DHCP Discover*" pour chercher un serveur DHCP qui lui fournira une adresse IP. Elle a utilisé l'adresse source 0.0.0.0 puisqu'elle n'a pas encore d'IP.

(2) Le serveur DHCP de la machine linux, qui a l'intention d'offrir à ce client l'IP 172.16.0.9, diffuse une requête ARP sur cette adresse, pour voir si elle est réellement disponible sur le réseau.

(3) Comme le serveur DHCP ne reçoit pas de réponse à sa requête, il offre cette adresse au client en lui envoyant un "*DHCP Offer*".

(4) La machine Windows a répondu à l'offre du serveur DHCP pour confirmer la prise de l'adresse IP envoyé (puisqu'il pourrait y avoir plusieurs DHCP qui répondent, il faut donc qu'il y ait une confirmation au serveur choisi par le client).

(5) Le serveur DHCP répond par "*DHCP ACK*" pour accepter l'affectation.

(6)(7)(8) Le client (la machine Windows) fait trois broadcasts ARP pour vérifier de son côté que l'IP 172.16.0.9 n'est pas dupliquée sur le réseau.

(9) Comme il ne reçoit pas de réponse à ses broadcasts ARP, il registre sa nouvelle adresse,

En effet la durée de validité des adresses IP affectées par le serveur DHCP est limitée par la durée définie dans le fichier de configuration *"/etc/dhcp.conf"* (dans notre cas, elle est de 20 secondes), donc si le client reste connecté plus longtemps que cette durée définie, il va devoir la renouveler périodiquement, en envoyant un *"DHCP request"*, pour recevoir une *"DHCP ACK"* du serveur DHCP, portant une nouvelle durée égale à la précédente.

La liste des informations portées dans les paquets *"DHCP Offer"* ainsi que les paquets *"DHCP ACK"* est donnée dans la figure suivante :

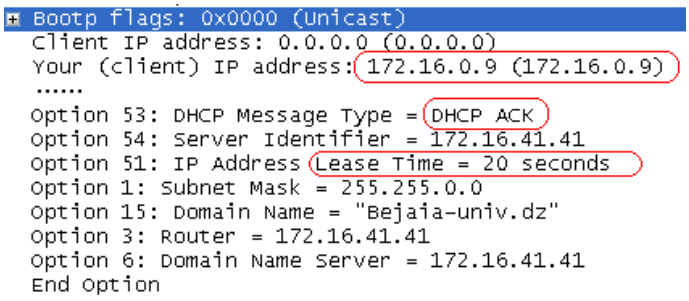


FIG. 4.10 – Le bail de l'informations envoyé par le serveur DHCP

Ce qui nous intéresse dans toutes ces informations capturées, c'est les paquets d'accord (DHCP ACK) envoyés par le serveur DHCP, et en particulier : l'adresse affecté au client (les quatre octets de numéros : 16, 17, 18, 19 du paquet DHCP

ACK) et la durée de validité de cette adresse (les six octets de numéros : 249, 250, 251, 252, 253, 254).

Donc, notre sniffer va être programmé pour capturer uniquement les paquets " DHCP ACK ", pour sauvegarder dans un fichier texte toutes les adresses affectées avec la durée de validité de chacune d'elles.

En plus de la sauvegarde de ces information, notre sniffer teste périodiquement le contenu de ce fichier pour extraire la liste des adresses déjà expirées, afin de leur envoyer des paquets de test pour confirmer s'elles sont encore actives ou non.

Donc, l'algorithme générale de notre sniffer est :

```
Algorithme ModIPAllouées
Var ip_serveur_dhcp : chaîne de caractères ;
    Fichier_adresses_allouées : ObjetFichierTexte ;
Debut
|  DebutPARALLELE
|  | Sniffer(eth0, ip_serveur_dhcp, Fichier_adresses_allouées);
|  PARALLELE
|  | Test_Expiration(fichier_adresses_allouées) ;
|  FinPARALLEL
Fin.
```

```
Procedure Sniffer(eth0, ip_serveur_dhcp, Fichier_adresses_allouées)
Var traf : Objet de chaîne de caractères ;
    paq_dhcp : Objet de paquet DHCP ;
Debut
| Tantque vrai Faire
| | Capturer_trafic(ethi, traf);
| | Si traf.Adr_ip_src() == ip_serveur_dhcp Alors
| | |  paq_dhcp := traf.Extraire_paquet_DHCP();
| | |  Si paq_dhcp.Type() == "DHCP ACK" Alors
| | |  |  Fichier_adresses_allouées.Ajouter(paq_dhcp.Adr_allouées(),
| | |  |     paq_dhcp.Durée_Validité());
| | |  FinSi
| | FinSi
| FinTantque
FinProcedure
```

```
Procedure Test_Expiration(fichier_adresses_allouées)
Var liste_ip_expirées : Objet de liste chaînée ;
    liste_ip_expirées_libres : liste chaînée ;
Debut
| Tantque vrai Faire
| | Sleep(T); // le test d'expiration des addresses IP allouées
| |           // est effectué périodiquement.
| | liste_ip_expirées := fichier_adresses_allouées.IP_Expirées();
| | liste_ip_expirées_libres := liste_ip_expirées.Test_ip_expirées_libres();
| | fichier_adresses_allouées.Supprimer(liste_ip_expirées_libres);
| FinTantque
FinProcedure
```

2. **Module de translation d'adresse MAC (*ModTransMAC*) :** Ce module est conçu pour remédier la troisième faille, le fonctionnement de ce module est similaire à celui du proxy NAT *(Network Addresses Translation)* [CSD99], il comporte une liste d'adresses MAC virtuelles similaires aux adresses MAC fournies par *VMWare*, à partir de cette liste et la liste des adresses libres fournie par le module *ModIPAlloues* précédent ainsi que les adresses du fichier de configuration "*honeyd.conf*", il va générer une table de correspondance, dont chacune de

ses entrées est correspondante à une adresse IP d'une machine virtuelle gérée par *honeyd* avec l'adresse MAC virtuelle affectée à cette adresse IP. Cette table de correspondance sera exportée au daemon *arpd*, qui l'utilisera pour répondre aux requêtes ARP (lorsqu'il reçoit une requête ARP pour une adresse IP virtuelle, il va chercher son adresse MAC virtuelle dans cette table, puis il la charge dans le paquet de réponse et l'envoyer au demandeur). Ce module est programmé pour écouter dans le réseau en mode promiscuité[2], et dés qu'il reçoit une trame, il vérifie son adresse MAC de destination, si cette adresse a une entrée dans la table de correspondance, alors elle sera remplacée par l'adresse MAC de la machine hébergeant *honeyd*, et la trame sera envoyée vers ce dernier. Les trames générées par *honeyd* sont envoyées d'abord vers ce module qui change leurs adresses MAC source par des adresses MAC virtuelles selon les adresses IP source des paquets encapsulés dans ces trames, et les envoie ensuite vers leur destination finale.

Donc, l'algorithme général de ce module est :

```
Algorithme ModTransMAC
Importer honeyd.conf, Fichier_adresses_allouées,
         Fichier_adresses_MAC_virtuelles ;
Var table_corresp : table d'objets;
Debut
|  table_corresp.Initialisation(honeyd.conf,
|      Fichier_adresses_allouées, Fichier_adresses_MAC_virtuelles);
| DebutPARALLELE
| | Tantque vrai Faire
| | | Sleep(T); // la mise à jour de la table de correspondance
| | |           // s'effectue périodiquement.
| | | table_corresp.MAJ(Fichier_adresses_allouées,
| | |           Fichier_adresses_MAC_virtuelles);
| | FinTantque
| PARALLELE
| | Gérer_trafic(table_corresp);
| FinPARALLELE
Fin.
```

```
Procedure Gérer_trafic(table_corresp)
Var traf : Objet de chaîne de caractères ;
Debut
| DebutPARALLELE // Trafic destiné vers honeyd.
| | Tantque vrai Faire
| | | Capturer_trafic(eth0, traf) ;
| | | Si Appartient(traf.Adr_ip_dest(), table_corresp) Alors
| | | |  traf.Changer_adr_MAC_dest(adr_MAC_honeyd);
| | | |  Envoyer_vers_honeyd(traf) ;
| | | FinSi
| | FinTantque
| PARALLELE  // Trafic sortant de honeyd.
| | Tantque vrai Faire
| | | Recevoir_trafic_sortant_honeyd(traf);
| | | adr_MAC := table_corresp.Extraire_adr_MAC(traf.Adr_IP_src());
| | | traf.Changer_adr_MAC_dest(adr_MAC);
| | | Envoyer_vers_réseau(traf);
| | FinTantque
| FinPARALLELE
FinProcedure
```

[2]Le **mode promiscuité**, c'est le mode qui permet à la carte réseau d'intercepter tous les paquets circulant sur le segment réseau, même ceux qui ne lui sont pas destinés.

3. **Module générateur d'adresses MAC virtuelles ($ModGnrateurMAC$) :** Comme son nom l'indique, ce module est conçu pour générer une liste d'adresses MAC virtuelles non utilisées dans le réseau local physique où *honeyd* est installé, cette liste sera utilisée ensuite par le module de translation d'adresses MAC pour générer la table de correspondance.

   En effet, le problème de conflit d'adresses MAC ne se pose qu'au niveau des réseaux locaux, donc la création de la liste des adresses MAC virtuelle est relativement simple, il suffit de générer aléatoirement un grand nombre d'adresses MAC virtuelles de différents constructeurs usuels, et d'en supprimer par la suite toutes les adresses existantes déjà dans notre réseau local physique, par une simple comparaison avec un fichier comportant la liste de tous les adresses MAC de notre réseau local physique.

4. **Module de vérification des adresses IP des machines virtuelles de la deuxième classe ($ModIPHoneypots$) :** Ce module est conçu pour résoudre les problèmes de conflit d'adresses, comme ceux remarqués dans le dernier test. A chaque démarrage de *honeyd*, ce module va vérifier les adresses IP affectées aux différents chablons du fichier de configuration, pour tester s'il n y a pas de machine réelle qui a pris une de ces adresses, en utilisant la même technique utilisée par le mécanisme DHCP. et dés qu'il détecte qu'une de ces adresses est déjà associée à une autre machine réelle, il affiche un message d'erreur pour signaler ce conflit à l'administrateur, et ne recommence son fonctionnement qu'après le réglage de ce problème, et pour faciliter la tâche de l'administrateur, ce mécanisme doit être aussi capable de chercher des adresses libres pour les proposer à ce dernier avec le message d'erreur.

   Donc, l'algorithme général de ce module est :

```
Algorithme ModIPHoneypots
Importer fichier_adresses_allouées, honeyd.conf;
Var list_IP_honeypots, list_IP_erreur : Liste d'Objets;
Debut
|  list_IP_honeypots[] := Extraire_ip(honeyd.conf);
| Pour i=1 à list_IP_honeypots[].longueur() Faire
| | Si Allouée(list_IP_honeypots[i]) Alors
| | |  list_IP_erreur.Ajouter(list_ip_honeypots[i]);
| | FinSi
| FinPour
|
| Si Non Vide(list_IP_erreur) Alors
| | ip_libres := Extraire_ip_libres(Fichier_adresses_allouées);
| | Alerter_administrateur(msg_alert, list_IP_erreur);
| | Offrir_à_administrateur(ip_libres) ;
| FinSi
Fin.
```

Avec l'implémentation de tous ces quatre modules et leur intégration avec la technologie *honeyd* actuelle, il n'y aura plus de possibilité dans le futur pour exploiter les failles découvertes dans ce projet. Donc, d'après tout ce travail nous pouvons dire que la technologie *honeyd* a vraiment acquis une plus d'immunité contre les attaques destinées vers la détection des *honeypots*.

4.4 Conclusion

Dans ce chapitre, nous avons présenté les différentes techniques qui peuvent être utilisées par la communauté des hackers pour détecter la présence des deux technologies *honeynet roo* et *honeyd*, avec quelques solutions possibles pour les résoudre. Par exemple, pour la première technologie, nous avons remarqué que la limitation du trafic sortant conduit facilement à la détection de la présence du *honeynet*, et nous avons vu que le seul moyen possible pour résoudre ce problème, c'est d'éviter complètement le blocage du trafic sortant, et de le remplacer par une nouvelle technique de contrôle de données comme celle proposée dans ce chapitre, qui assure le contrôle du trafic sortant d'une manière efficace et sans aucun blocage de celui-ci, en se basant sur la création d'une cuirasse pour protéger les machines du monde externe, et qui permet également de résoudre d'autres problèmes, comme celui de la limitation d'apprentissage causé par la limitation du trafic sortant, et le problème d'apprentissage sur les conséquences du trafic sortant qui n'était pas assuré dans l'ancienne technique. Pour la deuxième technologie, nous avons pu mettre la main sur quelques failles de sa détection, en se basant sur les comportements anormaux générés par celle-ci, que nous les avons montré par des tests expérimentaux, et que nous avons pu leur présenter des solutions qui permettent de les remédier complètement. Mais bien entendu, ces failles découvertes ne sont pas les seules, il reste sûrement d'autres failles de la détection de ces technologies, qui ne sont pas encore découvertes, et qui nécessitent d'autres efforts sérieux dans le futur pour les découvrir, avant qu'elles soient exploitées par des hackers malicieux.

Conclusion générale et Perspectives

LA technologie *honeynet ROO CDROM*, ainsi que la technologie *honeyd* sont actuellement parmi les *honeypots* les plus utilisés et les plus préférés par la communauté de sécurité informatique, grâce aux avantages multiples qu'elles offrent, comme leur source libre qui a permis à un grand nombre de chercheurs de s'introduire dans leur développement, ce qu'en a résulté un développement plus rapide et plus perfectionné. Cependant, les différences de fonctionnement et de gestion entre ces deux technologies sont nombreuses, ce qui rend le choix entre elles dépend fortement du domaine dans lequel nous voulons les utiliser, en prenant en considération les particularités ainsi que les avantages et les inconvénients de chacune d'elles. En effet, les avantages de l'une pouvent être considérés comme des inconvénients de l'autre, par exemple le risque impliqué par la technologie *Honeynet ROO CDROM* est totalement négligeable dans la technologie *Honeyd*. De plus, et comme nous avons vu dans la deuxième expérimentation, la quantité du trafic capturée par la première est beaucoup plus supérieure que celle capturée par la deuxième, où la quantité du trafic capturée par les deux *Honeypots* de la technologie *Honeynet ROO CDROM* était 147106 flux (99.12% du trafic total), tandis que celle capturée par les trois *Honeypots* de la technologie *Honeyd* n'était que 1295 flux (0.87% du trafic total). En outre, la possibilité de détection de la technologie *Honeynet ROO CDROM* est négligeable par rapport à la possibilité de détection de la technologies *Honeyd*, à cause des limitations imposées par les émulations incomplètes des services fournis par cette dernière. Nous pourrons donc en conclure qu'elles sont complètement différentes, et que suivant l'utilisation que l'on voudra en faire, on choisira soit la première, soit la deuxième. Par exemple nous choisirons la première lorsque nous voulons mettre en place un *Honeypot* de recherche avec lequel nous voulons collecter le maximum d'informations sur les activités des hackers sans éveiller leurs soupçons, et nous choisirons la deuxième lorsque nous voulons mettre en place un *Honeypot* de production utilisé comme un détecteur d'intrusion, où l'intérêt de la détection du *honeypot* et celui de la quantité des informations collectées est relativement faible.

En effet, l'objectif final de ce projet qu'était la mise en place de ces deux technologies *Honeypot*, ainsi que l'extraction des différentes failles de détection de ces dernières et de leur proposer quelques solutions a finalement été achevé, malgré les problèmes

récurrents rencontrés pendant l'installation et la manipulation de ces deux technologies, à cause de la nature des machines requises pour réaliser ces installations par rapports à celles dédiées, qui n'étaient pas puissantes comme il faut. Ces problèmes nous a coûté beaucoup de temps, mais enfin de compte, nous avons pu les installer et nous avons pu capturer et analyser les différents flux malicieux relatifs aux *Honeypots* installés, ce qui nous a permis de définir la nature des attaques comptées par les hackers, ainsi que la nature des protocoles et des ports qui les attirent beaucoup. Et puisque l'utilisation future des technologies *honeypot* en général et de ces deux technologies en particulier s'arrêt sur leur capacité de se cacher de la communauté des hackers, nous avons consacré tout un chapitre pour traiter cette menace, ce qui nous a permis de mettre la main sur quelques problèmes (failles) de la détection de ces dernières, et de leur proposer des solutions intéressantes, mais bien entendu ces failles découvertes ne sont pas les seules, il reste sûrement d'autres failles de détection de ces technologies qui ne sont pas encore découvertes, et qui nécessitent d'autres efforts sérieux dans le futur pour les découvrir avant qu' elles soient exploitées par des hackers malicieux. De plus, nous devons aussi souligner que les solutions que nous avons proposées ne sont pas complètes à cent pour cent, ils comportent sûrement des carences qui nécessitent une réévaluation fréquente pour pouvoir les patcher et les améliorer.

Pour finir, nous dirons que tout ce travail a été très enrichissant techniquement et pratiquement, car il nous a ouvert un peu plus les portes de la sécurité informatique, et plus particulièrement ceux des technologies de leurre informatique. Cette étude nous a permis d'affirmer que les technologies *Honeypot* sont des bonnes solutions pour la sécurité réseau, avec bien sûr leurs propres particularités. Et leur utilisation montre d'un jour à l'autre qu'il arrivera un jour où on les trouvera comme un composant indispensable pour la sécurité réseau de toutes les entreprises, les établissements et les universités qui ont une culture de sécurité relativement avancée.

Annexe A

A.1 Configurations de la première expérimentation

A.1.1 Configuration de notre *honeywall*

La phase la plus importante dans l'installation d'un *honeywall* est la configuration de ses différentes variables, cette configuration est effectuée juste après le premier lancement du *honeywall* après son installation, pour configurer ces variables on utilise une des trois méthodes suivantes :

- La lecture directe du fichier de configuration par défaut qui se trouve dans le répertoire "/etc/Honeywall.conf.orig" du *honeywall*,
- La lecture directe d'une configuration personnalisée stockée dans une disquette,
- La réponse à une série de questions (c'est la méthode la plus utilisée).

La configuration des variables du *honeywall* de notre projet est donnée comme suite :

1. **HwHPOT_PUBLIC_IP :** Les adresses IP's des *honeypots* de notre *honeynet*.

   HwHPOT_PUBLIC_IP=172.16.96.128 172.16.96.55

2. **HwLAN_IP_RANGE :**L'adresse IP de notre réseau *honeynet* en notation CIDR.

   HwLAN_IP_RANGE=172.16.0.0/16

3. **HwLAN_BCAST_ADDRESS :** L'adresse IP de broadcast dans notre réseau *honeynet*.

   HwLAN_BCAST_ADDRESS=172.16.255.255

4. **HwQUEUE :** Permettre ou non au support Queue d'être intégré avec Snort-Inline.

   HwQUEUE=yes

5. **HwSCALE :** L'unité de mesure pour les limites des connexions sortantes de notre *honeynet* (second, minute, hour, day, week, month, year).

   HwSCALE=hour

6. **HwTCPRATE :** Nombre de connexions TCP sortantes autorisées par unité de mesure (HwSCALE).

   HwTCPRATE=50

7. **HwUDPRATE :** Nombre de connexions UDP sortantes autorisées par unité de mesure.

   HwUDPRATE=50

8. **HwICMPRATE :** Nombre de connexions ICMP sortantes autorisées par unité de mesure.

   HwICMPRATE=60

9. **HwOTHERRATE :** Nombre d'autres connexions sortantes autorisées par unité de mesure.

   HwOTHERRATE=30

10. **HwSEBEK :** Autoriser ou non l'envoie des données collectées par *sebek* à un système distant.

    HwSEBEK=no

11. **HwWALLEYE :** Permettre ou non l'utilisation de l'interface web *walleye*.

    HwWALLEYE=yes

12. **HwSEBEK_FATE :** Spécifie s'il faut laisser les paquets SEBEK sortir à l'extérieur (ACCEPT) ou non (DROP).

    HwSEBEK_FATE=DROP

13. **HwSEBEK_DST_PORT :** Port de destination des paquets *sebek*.

    HwSEBEK_DST_PORT=1101

14. **HwSEBEK_LOG :** Permettre ou non à *sebek* de logger dans les logs de firewall du *honeywall*.

    HwSEBEK_LOG=yes

15. **HwMANAGE_IFACE :** L'interface réseau du *honeywall* qui est utilisée pour le management distant.

    HwMANAGE_IFACE=eth2

16. **HwMANAGE_IP :** L'adresse IP de l'interface utilisée pour le management distant.

    HwMANAGE_IP=172.16.119.127

17. **HwMANAGE_NETMASK :** Le masque réseau de l'interface de management.

    HwMANAGE_NETMASK=255.255.0.0

18. **HwMANAGE_GATEWAY :** Passerelle par défaut de l'interface de management.

    HwMANAGE_GATEWAY=172.16.0.1

19. **HwMANAGE_DNS :** Les serveurs DNS utilisés par l'interface de management.

    HwMANAGE_DNS=172.17.1.11

20. **HwALLOWED_TCP_IN :** Les ports TCP ouverts dans l'interface de management.

    HwALLOWED_TCP_IN=22 443

21. **HwRESTRICT :** Permettre ou non au *honeywall* de limiter les ports de destination des trafics sortants.

    HwRESTRICT=yes

22. **HwALLOWED_TCP_OUT :** Les ports TCP's de destination autorisés.

    HwALLOWED_TCP_OUT=22 25 43 80 443

23. **HwALLOWED_UDP_OUT :** Les ports UDP's de destination autorisés.

    HwALLOWED_UDP_OUT=53 123

24. **HwALERT :** Permettre ou non l'envoie des emails d'alerte.

    HwALERT=yes

25. **HwALERT_EMAIL :** L'adresse email utilisée pour les emails d'alerte.

    HwALERT_EMAIL=root@localhost.localdomain

26. **HwBWLIST_ENABLE :** utiliser ou non le Blacklist et Whitelist.

    HwBWLIST_ENABLE=no

27. **HwFENCELIST_ENABLE :** utiliser ou non le Fencelist.

    HwFENCELIST_ENABLE=no

A.1.2 Configuration de *Sebek client*

Les variables de configuration de *sebek client* les plus importantes sont les suivantes :

- **SOURCE_PORT :** Port source utilisé par *sebek client* pour envoyer les données collectées sur les activités d'attaquant.
- **DESTINATION_IP :** Adresse IP de la machine de destination des données envoyées par *sebek client* (peut prendre n'import quelle adresse).
- **DESTINATION_MAC :** Adresse MAC de la machine de destination des données envoyées par *sebek client*.
- **DESTINATION_PORT :** Port de destination utilisé par *sebek* server pour recevoire les données collectées sur les activités d'attaquant envoyées par *Sebek client* (les *honeypots* du même *honeynet* doivent employer le même port de destination).
- **MAGIC_VAL :** La valeur magique est employée avec le port de destination pour déterminer quels paquets à cacher dans le *honeynet*. Elle doit être prendre une valeur difficile à être estimée. (les *honeypots* du même *honeynet* doivent employer la même valeur magique).
- **MODULE_NAME :** Le nouveau nom du module *sebek client*. Il est très important de changer le nom du module *Sebek*. Si cette variable prend la valeur vide alors le système va donner un nom aléatoire au module *sebek*, mais il est préférable que c'est nous qui donnons le nom à *Sebek*, car le nom aléatoire donné par le système peut produire des soupçons pour les attaquants.

a. Configuration de *Sebek client* du *honeypot* Windows XP

La configuration de *Sebek client* installé dans le *honeypot* Windows XP est donnée comme suite :

- *SOURCE_PORT = 1101*
- *DESTINATION_IP = 0.0.0.0*
- *DESTINATION_MAC = 00 :90 :FB :04 :B5 :91*
- *DESTINATION_PORT = 1101*
- *MAGIC_VAL = 1163890700*
- *MODULE_NAME = shoes*

a. Configuration de *Sebek client* du *honeypot* RHEL4

La configuration de *Sebek client* installé dans le *honeypot* Red hat enterprise linux 4 est la même que celle de *Sebek client* du *honeypot* Windows XP, sauf la variable "$MODULE\_NAME$" qui vaut ici la valeur "*boby*"

A.1.3 Test de fonctionnement du *honeynet*

Avant de lancer l'expérimentation on doit s'assurer que le *honeynet* est bien configuré pour éviter tout risque qui peut résulter d'une male configuration. Cette bonne configuration est assurée par les tests suivants :

1. Tester si le temps et la date dans le *honeywall* sont bien configurés, et la même chose pour ceux des *honeypots* :
   - Dans l'invité de commande du *honeywall* (resp. *honeypots*), exécuter la commande "date".
   - Vérifier si la date et le temps affichés dans le résultat sont corrects.

   Remarque : répéter ce test plusieurs fois pour s'assurer que l'horloge se décale correctement.
2. Tester si les *honeypots* sont capables d'établir des connexions depuis et vers les autres machines du réseau intranet :
   - Dans l'invité de commande des *honeypots*, exécuter la commande *"ping <@IP d'une machine du réseau intranet>"*, et vérifier s'il y a des réponses.
   - Dans l'invité de commande d'une machine du réseau intranet, exécuter la commande *"ping <@IP du honeypot>"*, et vérifier s'il y a des réponses.
3. Tester si les *honeypots* sont capables de résoudre les adresses Internet à partir des serveurs DNS :
   - Dans l'invité de commande des *honeypots*, exécuter la commande *"nslookup www.google.com"*, et vérifier s'il y a de réponse contenant l'adresse IP correspondante à cette adresse internet.
4. Tester si le trafic entrant et sortant est loggé dans le *honeywall* :
   - Login en root dans le *honeywall*, et entrer dans le menu de dialogue en exécutant la commande *"menu"*, puis suivre le chemin *"Status->Inbound connexions"*, et vérifier l'existence des paquets ICMP résultants de l'exécution des pings effectués dans les tests précédents.

   Remarque : faire la même chose pour les logs *"menu->Status->Outbound connexions"*.
5. Tester si l'interface Web *walleye* est accessible à partir de la machine distante de management :
   - Dans la machine distante de management, ouvrir un navigateur Web et connecter à l'adresse : *https ://@IP de management/walleye.pl*, et vérifier si la connexion est établie correctement.
   - Login dans le *honeywall* à partir de cette interface et vérifier la date et le temps affichés dedans.
   - Cliquer sur le lien *"last 1 hour"*, et dans l'interface affichée sélectionner *"detailed on view"*, puis cliquer sur le bouton *"send request"*, et vérifier l'existence

des paquets ICMP résultant de l'exécution des pings effectués dans les tests précédents.

6. Tester si les messages d'alerte sont envoyés vers notre boite de lettres :
   - Ouvrir la boite email, et vérifier l'existence d'un message d'alerte correspond aux pings effectués dépuis les *honeypots* dans les tests précédents. L'entête de ce message contient :

     "—— ALERT ! OUTBOUND ICMP ——"

     et son corps contient :

     "DST=<@IP d'une machine externe>"

7. Tester si *Sebek* est en exécution.
   - Dans l'interface *walleye* chercher un flux avec @IP_source = IP du *honeypot* dans lequel *sebek client* s'exécute, et @IP_destination = 0.0.0.0, et port_destination = 1101. ce flux est correspondant aux données collectées par *Sebek client* sur l'activité d'envoie des pings à partir du *honeypot* qui ont été effectués dans les tests précédents.

A.2 Configurations de la deuxième expérimentation

A.2.1 Fichier de configuration "*honeyd.conf*"

```
# ------------------------------------------------------------------
# ---------- Microsoft Windows XP Pro SP1 ----------
# ------------------------------------------------------------------

# Création d'un chablon Microsoft Windows XP Pro SP1
create windows

# Personnalité
set windows personality "Microsoft Windows XP Professional SP1"

# l'uptime
set windows uptime 1728650

# UserID et GroupID
set windows uid 65534 gid 65534

# Ports ouverts
add windows tcp port 21 "sh scripts/win2k/msftp.sh $ipsrc $sport $ipdst $dport"
add windows tcp port 25 "sh scripts/win2k/exchange-smtp.sh $ipsrc $sport $ipdst
$dport"
add windows tcp port 80 "sh scripts/win2k/iis.sh $ipsrc $sport $ipdst $dport"
add windows tcp port 110 "sh scripts/win2k/exchange-pop3.sh $ipsrc $sport $ipdst
$dport"
add windows tcp port 143 "sh scripts/win2k/exchange-imap.sh $ipsrc $sport $ipdst
$dport"
add windows tcp port 135 open # epmap
add windows udp port 137 open # netbios-ns
add windows udp port 138 open # netbios-dgm
add windows tcp port 139 open # netbios-ssn
add windows tcp port 445 open # microsoft-ds
add windows udp port 445 open # microsoft-ds

# Actions par défaut sur les ports non spécifiés précédemment
set windows default tcp action reset
set windows default udp action reset
set windows default icmp action open

# type de la carte Ethernet
set windows ethernet "3com"
```

```
#-------------------------------------------------------------------------
# ---------- Linux Suse 8.0 ----------
#-------------------------------------------------------------------------

# Création d'un chablon Linux Suse 8.0 :
create suse80

# Personnalité
set suse80 personality "Linux 2.4.7 (X86)"

# l'uptime
set suse80 uptime 354136546

# UserID et GroupID
set suse80 uid 65534 gid 65534

# Ports ouverts
add suse80 tcp port 21 "sh scripts/suse8.0/proftpd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 22 "sh scripts/suse8.0/ssh.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 23 "sh scripts/suse8.0/telnetd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 25 "sh scripts/suse8.0/sendmail.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 79 "sh scripts/suse8.0/fingerd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 80 "sh scripts/suse8.0/apache.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 110 "sh scripts/suse8.0/qpop.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 143 "sh scripts/suse8.0/cyrus-imapd.sh $ipsrc $sport $ipdst
$dport"
add suse80 tcp port 515 "sh scripts/suse8.0/lpd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 3128 "sh scripts/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 8080 "sh scripts/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 8081 "sh scripts/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 udp port 514 "sh scripts/suse8.0/syslogd.sh $ipsrc $sport $ipdst $dport"
add suse80 udp port 53 proxy 24.35.0.12 :53

# Actions par défaut sur les ports non spécifiés précédemment
set suse80 default tcp action reset
set suse80 default udp action reset
set suse80 default icmp action open

# type de la carte Ethernet
set suse80 ethernet "3com"
```

```
#-------------------------------------------------------------------------
# ---------- Microsoft Windows 2000 ----------
#-------------------------------------------------------------------------

# Création d'un chablon Microsoft Windows 2000 :
create win2000

# Personnalité
set win2000 personality " Microsoft Windows 2000 Server SP3"

# l'uptime
set win2000 uptime 54346475

# UserID et GroupID
set win2000 uid 65534 gid 65534

# Ports ouverts
add win2000 tcp port 80 "sh scripts/win2k/iis.sh $ipsrc $sport $ipdst $dport"
add win2000 tcp port 110 "sh scripts/win2k/exchange-pop3.sh $ipsrc $sport $ipdst
$dport"
add win2000 tcp port 137 proxy $ipsrc :137 # netbios-ns
add win2000 tcp port 138 proxy $ipsrc :138 # netbios-dgm
add win2000 tcp port 139 proxy $ipsrc :139 # netbios-ssn
add win2000 tcp port 445 proxy $ipsrc :445 # microsoft-ds
add win2000 udp port 137 proxy $ipsrc :137 # netbios-ns
add win2000 udp port 138 proxy $ipsrc :138 # netbios-dgm
add win2000 udp port 445 proxy $ipsrc :445 # microsoft-ds

# Actions par défaut sur les ports non spécifiés précédemment
set win2000 default tcp action reset
set win2000 default udp action reset
set win2000 default icmp action open

# type de la carte Ethernet
set windows2000 ethernet "intel"

#-------------------------------------------------------------------------
# ---------- Machines par défaut ----------
#-------------------------------------------------------------------------

# Chablon par défaut
create default

# Toute tentative de connexion est bloquée (l'hôte n'existe pas)
set default default tcp action block
set default default udp action block
set default default icmp action block
```

```
#-------------------------------------------------------------------------
# ---------- Création de machines virtuelles ----------
#-------------------------------------------------------------------------

# associer les adresses des honeypots aux différents chablons
bind 172.16.113.11 windows
bind 172.16. 113.12 suse80
bind 172.16. 113.13 win2000

#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
```


A.2.2 Lancement de *arpd* et *honeyd*

Pour lancer l'exécution de *arpd* et *honeyd*, nous avons créé deux scripts shell ($start-arpd.sh$ et $start-honeyd.sh$), dans le premier script nous avons écrit la commande de lancemnet de *arpd* comme suite :

./arpd -d 172.16.113.0/24

Cette commande va permettre de lancer le daemon *arpd* pour résoudre toutes les adresses IP situées dans la plage 172.16.113.0/24 et les diriger vers *honeyd*. L'option **-d** est pour afficher les informations de debug.

Dans le deuxième script nous avons écrit la commande de lancement de *honeyd* suivante :

./honeyd -d -i eth0 -f honeyd.conf -p nmap.prints -x xprobe2.conf -a nmap.assoc -0 pf.os -l /var/log/honeyd 172.16.113.1-172.16.113.50

La signification des options utilisées dans cette commande est :

- **-d** est pour afficher les informations de debug,
- **-i** est pour spécifier l'interface réseau à utiliser,
- **-f** est pour spécifier le fichier de configuration,
- **-p** est pour spécifier le fichier d'empreinte de format Nmap (nécessaire pour simuler la personnalité),
- **-x** est pour spécifier le fichier d'empreinte de format Xprobe2 (nécessaire pour simuler la personnalité),
- **-a** pour spécifier le fichier d'association entre Nmap et Xprobe2,
- **-0** est pour spécifier le fichier d'empreinte passive d'OS (passive OS fingerprinting),
- **-l** est pour spécifier le fichier log,

Maintenant, pour lancer *arpd* et *honeyd* il suffit d'accéder à leur répertoire à partir d'une console, et d'écrire les deux commandes : *./start-arpd.sh >& resultat&* et *./start-honeyd.sh >& resultat2&*, comme il est indiqué dans la figure Fig. A.1 suivante :

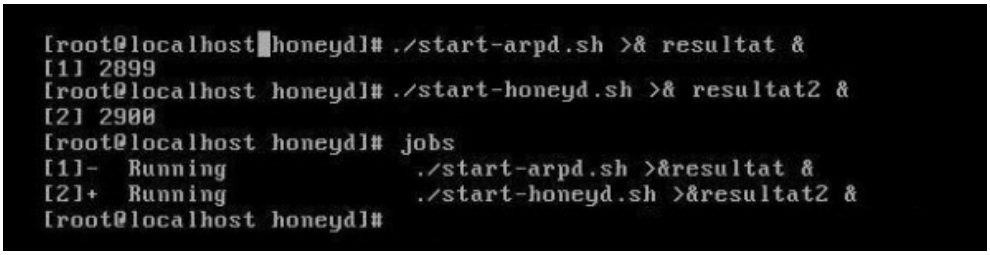


FIG. A.1 – Lancement de *arpd* et *honeyd*

le & est utilisé pour exécuter *arpd* et *honeyd* en arrière plan, *resultat* et *resultat*2 sont des fichiers vers lesquels on va rediriger les informations de debug affichées par *arpd* et *honeyd* respectivement. Lorsqu'on exécute la commande *jobs* on peut voir que *arpd* et *honeyd* sont en cours d'exécution.

A.2.3 Test de fonctionnement de *honeyd*

Une fois l'installation et la configuration de *honeyd* terminées, il ne reste que de le mettre en fonctionnement, mais avant de faire ça on doit d'abord s'assurer que tout marche bien en effectuant quelques tests sur le fonctionnement de différentes machines simulées. Une des façons les plus efficaces pour tester le fonctionnement d'une machines et ses services, c'est d'utiliser un outil de scan de machines et ports, comme *nmap*. Pour tester par exemple le fonctionnement de la machine virtuelle Linux *Suse*8.0, il suffit de lancer la commande : *nmap -sS 172.16.113.31* [1], cette commande va nous permettre de récupérer toutes les informations nécessaires sur les services et les ports TCP ouverts dans cette machine. Le résultat de cette commande est illustré dans la figure Fig. A.2 suivante :

[1] les adresses IP 172.16.113.11, 172.16.113.12, 172.16.113.13 indiquées dans le fichier de configuration sont changées à : 172.16.113.31, 172.16.113.32, 172.16.113.33 respectivement, après la détection d'un conflit d'adresses IP

FIG. A.2 – Résultat de la commande de scan : *"nmap -sS 172.16.113.31"*

Comme il est montré dans cette figure, les services TCP récupérés sont les même que ceux indiqués dans le fichier de configuration, donc la machine simulée fonctionne bien.

Pour tester les deux autres *honeypots* (Windows XP et Windows 2000), il suffit de lancer les deux commandes : *nmap -sS 172.16.113.32* et *nmap -sS 172.16.113.33*. Les résultats de ces deux commandes sont indiqués dans les deux figures (Fig. A.3 et Fig. A.4) suivantes :

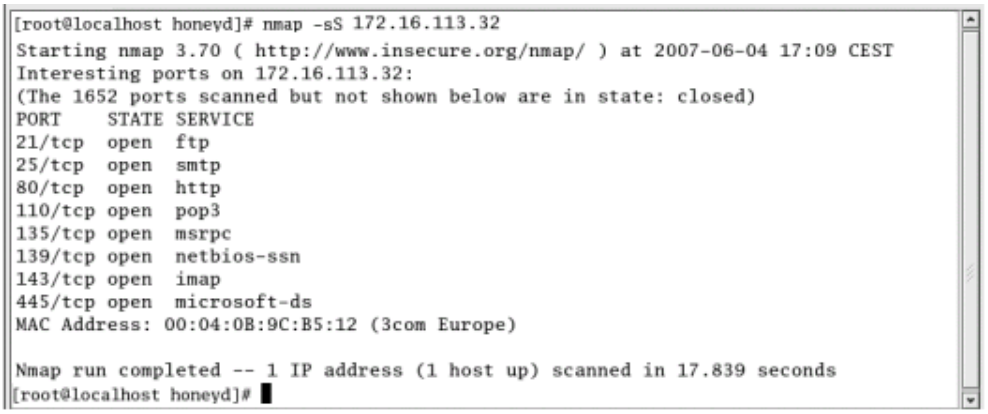


FIG. A.3 – Résultat de la commande de scan : *"nmap -sS 172.16.113.32"*

```
[root@localhost honeyd]# nmap -sS 172.16.113.33
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2007-06-04 17:10 CEST
Interesting ports on 172.16.113.33:
(The 1654 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
80/tcp  open  http
110/tcp open  pop3
137/tcp open  netbios-ns
138/tcp open  netbios-dgm
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:02:B3:54:53:B3 (Intel)

Nmap run completed -- 1 IP address (1 host up) scanned in 44.369 seconds
[root@localhost honeyd]#
```

FIG. A.4 – Résultat de la commande de scan : *"nmap -sS 172.16.113.33"*

La même chose, les services récupérés dans les deux machines sont les mêmes que ceux indiqués dans le fichier de configuration. Donc toutes les machines simulées fonctionnent bien.

Maintenant pour tester le fonctionnement de chaque service à part, il suffit de lancer des requêtes vers chacun de ces services, par exemple :

– Pour tester le service *web* il suffit de lancer un navigateur *web*, et de se connecter ensuite avec la machine virtuelle qu'on veut tester son service *web*, comme le montre la figure Fig. A.5 suivante :

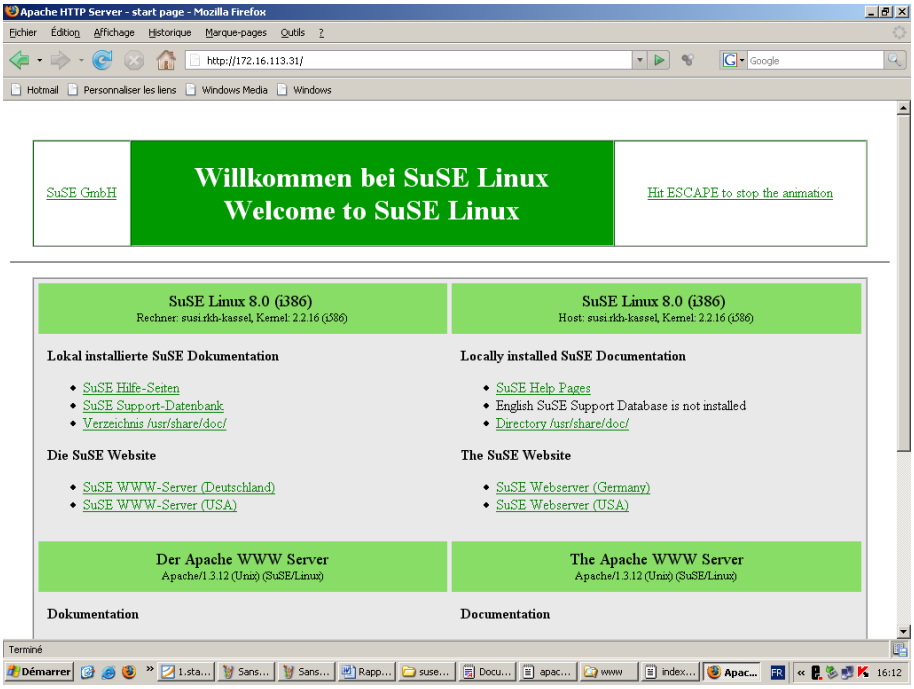


FIG. A.5 – Connexion au serveur *web* de la machine virtuelle *Linux Suse 8.0* via le navigateur *web* firefox

– Pour tester le service *ftp*, il suffit de lancer la commande *ftp* dans une console, comme il est indiqué dans la figure Fig. A.6 suivante :

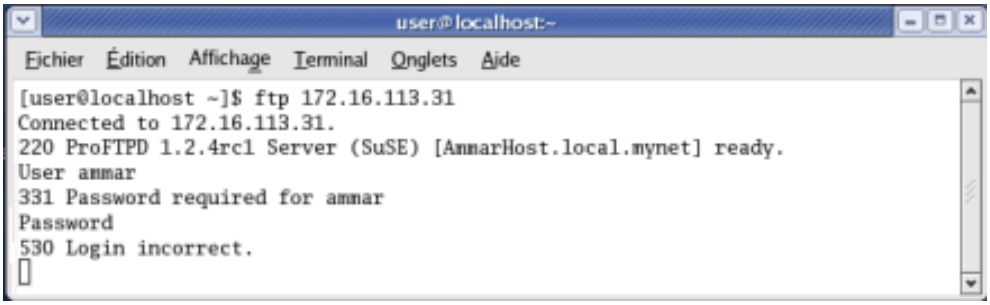


FIG. A.6 – Connexion au serveur *FTP* de la machine virtuelle *Linux Suse 8.0* via une console

– Pour tester le service *telnet*, il suffit de lancer la commande *telnet* dans une console, comme il est indiqué dans la figure Fig. A.7 suivante :

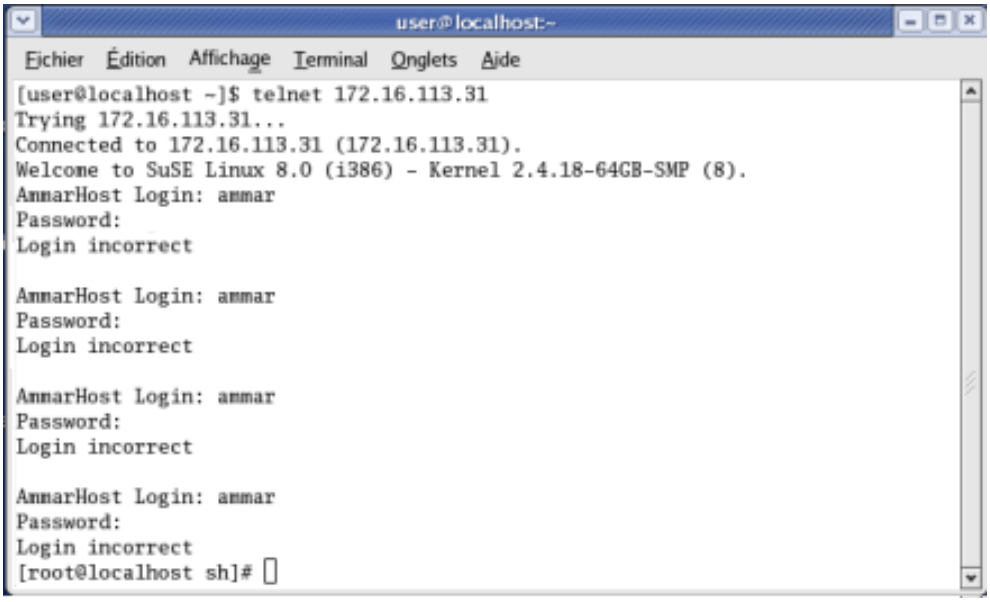


FIG. A.7 – Connexion au serveur *telnet* de la machine virtuelle *Linux Suse 8.0* via une console

Comme il est illustré dans ces trois figures, les services *web*, *ftp* et *telnet* simulés réagissent correctement, ce qui montre leur bon fonctionnement.

La même chose pour les autres services (on ne peut pas mettre les résultats des tests de tous les services simulés dans ce rapport à cause de leur grand nombre).

Bibliographie

[AH07] Argus-Homepage. *"Argus Project"*. http ://qosient.com/argus/argus.8.htm, Last access : March 2007.

[BFH07] BackOfficer-Friendly-Homepage. *"BackOfficer Friendly Honeypot Technology"*. http ://www.nfr.com/resource/downloads/back-officer-friendly.tar, Last access : March 2007.

[BKHD06] P. Baecher, M. Koetter, T. Holz, and M. Dornseif. "the nepenthes platform : An efficient approach to collect malware". *In Proceedings of RAID '06.*, 2006.

[Blo04] Michel Blomgren. *(IT Security Consultant) "Introduction to Shellcoding : How to exploit buffer overflows"*. http ://tigerteam.se, 2004.

[Che04] Bill Cheswick. "an evening with berferd in which a cracker is lured, endured, and studied". *Proceedings of the Winter 1992 USENIX Conference : January 20 January 24, 1992, San Francisco, California*, January 2004.

[Com07] G. Combs. *Ethereal Homepage*. http ://www.ethereal.com, Last access : March 2007.

[CSD99] Cisco-Systems-Documents. *"Introduction to Internet"*. http ://-www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/introint.htm, 1999.

[DD07] Gross David and Alias Deimos. *"Dépassement de capacité de la pile : Etude des exploitations avancées de stack overflow, écriture de shellcodes polymorphiques et alphanumériques"*. http ://www.futurezone.biz, avril 2007.

[Det05] Eric Detoisien. "the external attacks". *LinuxFocus article number 282*, January 2005.

[DPD04] M. Dacier, F. Pouget, and H. Debar. "attack processes found on the internet". *NATO Symposium IST-041/RSY-013, Toulouse, France*, April 2004.

[DTH07] Deception-Toolkit-Homepage. *"Deception Toolkit Honeypot Technology"*. http ://www.all.net/dtk, Last access : March 2007.

[EC05] E.Balas and C.Viecco. "towards a third generation data capture architecture for honeynets". *In Proceedings of the 6th Information Assurance Workshop, IEEE*, June 2005.

[HP06] Honeynet-Project. *"Online User Manual"*. http ://www.honeynet.org/tools/, 2006.

[HP07a] Honeynet-Project. *"Honeynet Project Research Alliance"*. http ://www.honeynet.org/alliance/, Last access : March 2007.

[HP07b] Honeynet-Project. *"Know Your Enemy : Defining Virtual Honeynets"*. http ://www.honeynet.org/papers/virtual/index.html, Last access : March 2007.

[HP07c] Honeynet-Project. *"Know Your Enemy : GenII Honeynets"*. http ://www.honeynet.org/papers/gen2/index.html, Last access : March 2007.

[HP07d] Honeynet-Project. *"Know Your Enemy : Honeynets"*. http ://www.honeynet.org/papers/honey-net/index.html, Last access : March 2007.

[HP07e] Honeynet-Project. *"Know Your Enemy : Honeywall CDROM"*. http ://www.honeynet.org/papers/cdrom/index.html, Last access : March 2007.

[HP07f] Honeynet-Project. *"Know Your Enemy : Research"*. http ://www.honeynet.org/research/index.html, Last access : March 2007.

[Hp07g] Honeynet-project. *"Know Your Enemy : Sebek"*. http ://www.honeynet.org/papers/sebek.pdf, Last access : March 2007.

[HP07h] Honeynet-Project. *"Know Your Enemy : Vmware"*. http ://www.honeynet.org/papers/papers/vm-ware/index.html, Last access : March 2007.

[HSBR99] S. Handelman, S. Stibler, N. Brownlee, and G. Ruth. *"RFC 2724 - RTFM : New Attributes for Traffic Flow Measurement"*. http ://www.rfc-editor.org/rfc/rfc2724.txt, October 1999.

[KW00] Poul-Henning Kamp and Robert N. M. Watson. *"Jails : Confining the omnipotent root"*. The FreeBSD Project., http ://docs.freebsd.org/44doc/papers/jail/jail.html., 2000.

[LN97] Paul J. Leach and Dilip C. Naik. *"CIFS/E Browser Protocol - Preliminary Draft"*. Microsoft Technical supports, January 1997.

[L.S03a] L.Spitzner. "honeypots : Catching the insider threat". *IEEE Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003) 1063-9527/03.*, 2003.

[L.S03b] L.Spitzner. "the honeynet project : Trapping the hackers". *IEEE Security and Privacy Magazine*, MARCH/APRIL 2003.

[Mar05] Jean-Baptiste Marchand. *"Windows network services internals"*. Hervé Schauer Consultants, 2005.

[MB01] D. MacDonald and W. Barkley. *"MS Windows 2000 TCP/IP Implementation Details"*. White Paper, http ://www.microsoft.com/-TechNet/win2000/win2ksrv/technote/tcpipimp.asp ?a=printable, January 2001.

[MFH04] M.Dacier, F.Pouget, and H.Debar. "honeypots : Practical means to validate malicious fault assumptions : Practical experience report". *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, February 2004.

[MH03] M. Mier and T. Holz. *"intrusion detection system list and bibliography"*. http ://www-nks.informatik.tucottbus.de/en/security/ids.html, December 2003.

[MH07] Mantrap-Homepage. *"Mantrap Honeypot Technology"*. http ://-www.mantrap.com, Last access : March 2007.

[NH07a] Nepenthes-Homepage. *"Nepenthes Project"*. http ://nepenthes.sourceforge.net/, Last access : March 2007. finest collection.

[NH07b] Netfilter-Homepage. *"Netfilter Project"*. http ://www.netfilter.org, Last access : March 2007.

[NH07c] Nmap-Homepage. *"Nmap Project"*. http ://www.insecure.org/nmap/, Last access : March 2007.

[PH07] P0f-Homepage. *"P0f Project"*. http ://lcamtuf.coredump.cx/p0f.shtml, Last access : March 2007.

[Pro03] Niels Provos. "honeyd : A virtual honeypot daemon (extended abstract)". *In : Schaumburg, Rolf and Marco Thorbrugge : 10. Security Workshop in networking System, Hamburg, 25./26.*, February 2003.

[Pro04] N. Provos. "a virtual honeypot framework". *In Proceedings of the 12th USENIX Security Symposium*, August 2004.

[Ros03] Jonathan Rose. *"Loadable Kernel Module Rootkits deployed in a honeypot environment"*. 2003.

[SD99] Symantec-Documents. *"Symantec Decoy Server Product Sheet"*. http ://enterprisesecurity.symantec.com.au/Content/displaypdf.cfm ?PDFID=760., 1999.

[SH07a] Snort-Homepage. *"Snort : The open soruce network intrusion detection system"*. http ://www.snort.org, Last access : March 2007.

[SH07b] Swatch-Homepage. *"The Simple WATCHer of Logfiles"*. http ://-swatch.sourceforge.net/, Last access : March 2007.

[SIH07] Snort-Inline-Homepage. *"Snort-inline Project"*. http ://snort-inline.sourceforge.net/, Last access : March 2007.

[Smi97] Nathan P. Smith. *"Stack Smashing Vulnerabilities in the UNIX Operating System"*. http ://millcomm.com/ nate/machines/security/stack-smashing/, May 1997.

[SP03] D. Song and N. Provos. *"Arpd"*. http ://www.honeyd.org/tools.php, February 2003.

[Spi02] L. Spitzner. *"Honeypots : Tracking Hackers"*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, September 2002.

[Spi03] L. Spitzner. *"Honeypots : Definitions and Value of Honeypots"*. http ://www.csd.uoc.gr/ gvasil/stuff/honeypots/honeypots.html, 2003.

[Sto90] Clifford Stoll. *"The Cuckoo's Egg : Tracking a Spy Through the Maze of Computer Espionage"*. NY : Pocket Books, New York, USA, 1990.

[TH07] TCPDump-Homepage. *"TCPDump et Libpcap"*. http ://www.tcpdump.org/, Last access : March 2007.

[THH07] The-Honeyd-Homepage. *"Honeyd Project"*. http ://www.honeyd.org/, Last access : March 2007.

[TSH07] The-Specter-Homepage. *"Specter Project"*. http ://www.specter.com/, Last access : March 2007.

[TSP06] The-Snort-Project. *"Snort Users Manual 2.6.1"*. http ://www.snort.org/docs/snort manual/2.6.1/snort manual.pdf, December 2006.

[UH07] UML-Homepage. *"The user-mode-linux kernel"*. http ://user-mode-linux.sourceforge.net/, Last access : March 2007.

[vH07] vmware Homepage. *"Guide de l'utilisateur"*. http ://www.vmware.com/support/, Last access : March 2007.

[XH07] Xprobe-Homepage. *"Xprobe Project"*. http ://www.sys-security.com/html/projects/X.html, Last access : March 2007.

Résumé

UN "*honeypot*" peut être défini comme un ensemble de pièges permettant de détecter ou de guider des tentatives d'utilisation non autorisées d'un système d'information. Généralement, il se compose d'un ordinateur ou d'un noeud de réseau qui semble faire partie d'un réseau mais qui est isolé et protégé et qui semble contenir de l'information précieuse pour des attaquants. L'utilisation, dans certaines conditions, de leurres informatiques peut s'avérer être d'une très grande utilité pour lutter contre les nouvelles menaces. Cependant, les technologies "*honeypot*" actuelles sont facilement détectables par les intrus et ne jouent donc leur plein rôle comme outil de protection. Donc, par cet axe de recherche, nous souhaitons mettre en place des technologies de leurres informatiques efficaces et non détectables permettant de connaître au plus tôt les nouvelles techniques et méthodes employées par les intrus afin d'anticiper au mieux les nouveaux risques. Ce projet de recherche se propose d'expérimenter les technologies actuelles et d'en proposer une évaluation précise.

Mots clés : Détection d'attaques de systèmes, Leurre informatique, Simulation, Audit.

Abstract

A honeypot can be defined as a set of traps allowing to detect or to drive a not authorized use attempts of a information system. Generally, it is composed of a computer or a network node which seems be a part of a network but which is isolated and protected and seems contain a precious information for attackers. The use of computer baits, in certain conditions, can turn out have a very big utility to fight against new threats. However, current honeypot technologies are easily detectable by the intruders and do not play so their full role as a protection tools. So, by this axis of search, we wish to set up a efficient and not detectable computer lures technologies allowing to know as soon as possible the new techniques and methods used by the intruders to anticipate at best the new risks. This research project suggests experimenting current technologies and proposing it a precise evaluation.

Keywords : Systems attacks detection, computer Bait, Simulation, Audit.

تلخيص

"الهونيبوت" (قدر العسل) هو عبارة عن مصيدة إلكترونية تسمح باكتشاف و مراقبة المحاولات غير الشرعية لاختراق واستعمال الأنظمة الآلية، فهو يتشكل عموما من جهاز كمبيوتر أو أي جهاز آخر مربوط بالشبكة الآلية بحيث يبدو وكأنه جهاز عادي ككل أجهزة الشبكة الأخرى لكنه في الحقيقة معزول و محمي بحيث يظهر و كأنه يحوي معلومات مهمة ومغرية للمخترق. إن استعمال هذه المصايد الإليكترونية داخل الشبكات تحت شروط معينة يكتسي أهمية بالغة في اكتشاف و مواجهة التهديدات الحديثة التي لم يتم الكشف عن وسائلها بعد. لكن أغلب تقنيات "الهونيبوت" الحالية إن لم نقل كلها يمكن اكتشاف وجودها داخل الشبكة بسهولة تامة من طرف المخترقين مما يجعلها تفقد الكثير من أهميتها كوسيلة لحماية الأنظمة الإلكترونية. في موضوع بحثنا هذا سنحاول تقدير تقنيات "هونيبوت" جديدة أكثر فاعلية و غير قابلة للاكتشاف من طرف المخترقين، والتي تسمح بالكشف عن التقنيات والطرق الحديثة التي يستعملها المخترقون في أقصر وقت ممكن مما يمكننا من تجنب كل هذه التهديدات قبل أن توقع أضرارا بليغة داخل الأنظمة المستهدفة. مشروع هذا البحث يتضمن تجريب تقنيات "الهونيبوت" المستعملة حاليا ومن ثم تقدير تقييم مفصل لهذه التقنيات انطلاقا من النتائج المحصل عليها مع محاولة تقدير اقتراحات لسد الثغرات التي تؤدي إلى اكتشاف هذه التقنيات من طرف المخترقين

مفاتيح البحث : اكتشاف الهجمات على الأنظمة، المصايد الآلية، المحاكاة، "الأوديت".