Mécanisme d'observation d'attaques sur Internet avec rebonds

Éric Alata, Ion Alberdi, Vincent Nicomette, Philippe Owezarski, and Mohamed Kaâniche

LAAS-CNRS
Email : {ealata,ialberdi,nicomett,owe,kaaniche}@laas.fr

Résumé Les pots de miel haute-interaction (*honeypots*) permettent d'observer les attaquants évoluer au sein d'une machine compromise. Ils sont intéressants pour mieux comprendre le fonctionnement de ces derniers. Cependant, l'observation se limite en général au pot de miel lui-même car les tentatives de compromission d'autres machines depuis le pot de miel sont limitées. Il nous semble très instructif de pouvoir suivre le parcours d'un attaquant sur différentes machines. Cet article propose, à cette fin, un mécanisme de redirection dynamique de connexions initiées depuis un pot de miel. Ce mécanisme, dont l'originalité réside dans son aspect dynamique, donne l'illusion à un attaquant qu'il dialogue effectivement depuis notre pot de miel avec une autre machine d'Internet alors qu'il est simplement redigiré vers un autre pot de miel. Ce mécanisme de redirection a été implémenté et testé sur un noyau Linux. Nous en présentons ici les concepts et l'implémentation.

1 Introduction

Les pots de miel sont désormais très utilisés pour essayer de comprendre et d'analyser les activités des attaquants d'Internet. Les différentes implémentations de pots de miel qui existent se différencient essentiellement par le niveau d'interaction qu'elles offrent à l'attaquant. Plus le niveau d'interaction est élevé (c'est-à-dire, plus la simulation est proche de la réalité), plus le pot de miel nous donne la possibilité d'observer une attaque dans son intégralité.

Par exemple, un pot de miel tel que `honeyd` [5] offre à l'attaquant un niveau d'interaction limité puisqu'il ne simule que partiellement certains services qui sont couramment proposés. Cependant même si le niveau d'interaction est faible, et qu'il peut s'avérer être insuffisant, ce genre de pot de miel est utile pour, par exemple, identifier quels services sont plus fréquemment ciblés par les attaquants. Ces services sont probablement ceux qui nous apporteront le plus d'information sur ces attaquants. Leur mise à disposition sur des pots de miel plus interactifs peut alors enrichir nos connaissances sur le comportement de ces attaquants. D'autres implémentations offrent la possibilité d'exécuter des systèmes d'exploitation et des logiciels complets. L'attaquant peut alors se connecter à de vrais services et interagir de façon evoluée avec eux. Les deux types de pots de miel précédents sont complémentaires. Aussi, certains travaux comme [4] et [1] proposent de les utiliser conjointement.

Les pots de miel actuels ont une limitation : les rebonds sont en général interdits pour des problèmes légaux. Cette limitation ne permet pas d'observer l'intégralité d'une attaque en train de se déployer sur Internet. De plus, elle risque de décourager l'attaquant qui risque de ne pas revenir sur le pot de miel. Afin de repousser cette limitation, la restriction du trafic en sortie, ou « rate-limiting », peut être utilisée. Cette technique permet d'accepter les connexions sortantes tant qu'elles restent en deça d'une certaine limite. Appliquée aux pots de miel, chaque attaquant se voit affecter la possibilité de réaliser une certain nombre de connexions sortantes mais pas plus. De cette

manière, plus d'informations sont obtenues. Mais cela reste insuffisant et surtout ne résoud pas les problèmes de légalité.

Dans le cadre de travaux de recherche que nous menons actuellement, nous avons déployé un certain nombre de pots de miel basés sur l'utilisation de systèmes d'exploitation complets installés sur des machines virtuelles, par l'utilisation de VMware [3]. Nous avons mené des expériences avec un pot de miel haute interaction architecturé autour de VMware [2]. Elles nous ont permis d'obtenir des informations intéressantes sur le processus d'intrusion des attaquants d'Internet. Toutefois, le pot de miel utilisé a été limité au niveau des connexions sortantes afin d'éviter les rebonds. Seules les connexions à destination du port 80 ont été autorisées, sous surveillance. Or, les résultats obtenus nous montrent que les attaquants, une fois introduits sur le pot de miel, tentent généralement d'utiliser la machine infectée comme rebond pour attaquer d'autres machines d'Internet.

Il est essentiel que les pots de miel offrent un plus haut niveau d'interaction. Nous présentons dans ce papier une technique permettant de donner à l'attaquant l'illusion qu'il peut effectivement « rebondir » depuis notre pot de miel, grâce à un mécanisme de redirection. L'originalité de notre approche réside dans le fait que cette redirection se fait à la volée, dynamiquement, grâce à un module inséré dans le noyau Linux. Lorsqu'un attaquant, depuis notre pot de miel, se met à la recherche de nouvelles cibles sur Internet (par des scans de réseau par exemple), nous faisons en sorte que certains de ces scans aboutissent en les redirigeant à la volée sur un autre de nos pots de miel. L'attaquant peut alors entreprendre de poursuivre ses activités.

Ce papier adopte la structure suivante. La section 2 présente le principe de la redirection que nous avons implémenté dans le noyau Linux (nous présentons l'architecture et les concepts) afin d'accroître ce niveau d'interaction. La section 3 est consacrée à la présentation de l'implémentation proprement dite. La section 4 présente les performances du mécanisme de redirection proposé, en termes de transparence. En particulier, cette section évalue l'impact du mécanisme de redirection sur les temps d'établissements de sessions qui doivent être aussi proches que possible des temps d'établissements sans le mécanisme de redirection. La section 5 présente des expérimentations en cours effectuées avec le mécanisme de redirection visant à valider l'approche proposée. La dernière section conclut l'article et présente quelques perspectives.

2 Principes de fonctionnement

Nous avons imaginé un mécanisme de redirection sélective à la volée des connexions issues d'un pot de miel et à destination d'Internet. L'utilisation de ce mécanisme est adaptée pour l'observation des attaquants n'ayant aucune connaissance des machines qu'ils attaquent.

Le principe consiste à faire en sorte que les connexions sortantes de notre pot de miel soient possibles en « apparence », mais uniquement si redirigées vers d'autres pots de miel. L'originalité de notre approche est que cette redirection se fait de façon dynamique. Le choix de redirection dépend des pots de miel disponibles et des redirections en cours et passées.

Afin de fixer les idées, la figure 1 présente une mise en situation de ce principe sur un pot de miel constitué de trois machines, b, c et d. La connexion 1 est initiée par l'attaquant depuis la machine a d'Internet, vers la machine b du pot de miel. Cette connexion permet à l'attaquant de prendre le contrôle de la machine b. La machine b constitue le point d'entrée de l'attaquant dans notre mécanisme. Depuis la machine b, l'attaquant tente d'accéder à la machine e d'Internet en initiant la connexion 2. Cette connexion est bloquée par notre mécanisme. L'attaquant tente alors une autre connexion, la connexion 3, vers la machine f d'Internet. Cette connexion est acceptée mais redirigée vers la machine c du pot de miel. Cette connexion qui donne le contrôle de la machine c à

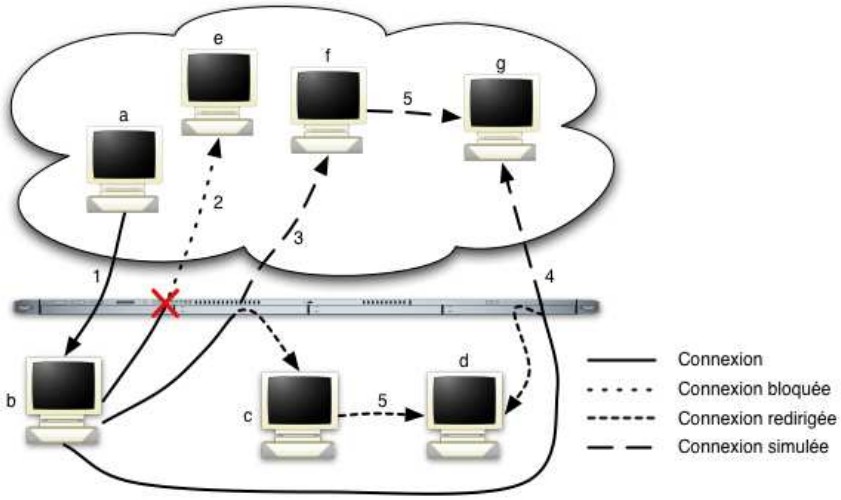

FIG. 1: Le principe de redirection à la volée

l'attaquant, donne aussi l'illusion à ce dernier de contrôler la machine f. En continuant son activité, l'attaquant tente une nouvelle connexion, la connexion 4, depuis la machine a vers la machine g d'Internet. Cette connexion est aussi acceptée, mais, à son tour, redirigée vers la machine d du pot de miel. A présent, depuis la machine c que l'attaquant contrôle depuis la connexion 3, la connexion 5 à destination de la machine g doit aussi être redirigée vers la machine d du pot de miel.

Le mécanisme présenté dans cette section permet d'observer l'activité de l'attaquant, sur les différents rebonds qu'il aurait voulu employer pour réaliser l'attaque. L'intérêt majeur est qu'il donne l'illusion à l'attaquant que ses attaques sur des machines d'Internet ont réussi. En revanche, si l'attaquant possède une connaissance des machines sur lesquelles il souhaite rebondir, cette supercherie est identifiable. Par exemple, dans la figure 1, nous pouvons supposer que l'attaquant contrôle déjà les machines a, e et f. Il peut alors tester, après la connexion 3, si la machine sur laquelle il est connecté est bien la machine f. Cette limitation existe, cependant nous ne savons à ce jour pas comment un attaquant réagit dans cette situation. A l'instar de `honeyd`, qui nous donne beaucoup d'information sur le comportement malicieux d'Internet malgré une furtivité limitée, nous pensons que notre système nous permettra d'en savoir plus. Il nous dira aussi s'il est utile d'augmenter le niveau d'interaction de la simulation du rebond.

La caractéristique essentielle de ce mécanisme de redirection doit évidemment être sa transparence vis-à-vis des attaquants, afin de collecter des données fiables, i.e. non biaisées pour un comportement soupçonneux des attaquants. Pour cela, l'implémentation doit respecter les trois caractéristiques suivantes :

- flexible : offrir la possibilité à un administrateur d'adapter ce principe à ses besoins,
- cohérente : permettre, en fonction des besoins de l'administrateur, de cacher au mieux cette "supercherie" aux yeux des attaquants,

– performante : ne pas apporter trop de latence afin de ne pas eveiller les soupçons des attaquants.

La section suivante présente une implémentation de notre principe de redirection. Tout au long de cette section, une discussion sur les caractéristiques précédentes sera menée.

3 Implémentation

Notre premier choix a porté sur l'environnement cible de notre implémentation. Nous avons opté pour l'emploi du système d'exploitation GNU/Linux. Ce choix a été principalement poussé par notre désir d'intégrer cette implémentation dans notre environnement de pots de miel qui est aussi basé sur le système GNU/Linux[2]. Toutefois, la conception est modulaire facilitant ainsi le portage sur d'autres systèmes.

Trois composants ont été créés (voir Figure 3). Le `module de redirection` mis en œuvre dans le noyau permet l'interception des paquets qui nous intéressent. Le `dialog_tracker` permet de faire le lien entre ce module et les `dialog_handler`. En fonction des caractéristiques du paquet, le `dialog_handler` adéquat à l'obtention de la décision concernant le paquet est sollicité par le `dialog_tracker`. Cette organisation est donc répartie entre l'espace noyau et l'espace utilisateur. Nous aurions pu réaliser tous les traitements seulement au niveau de l'espace utilisateur, ou seulement au niveau de l'espace noyau. La première alternative présente l'inconvénient d'ajouter une latence non négligeable entre la réception d'un paquet et la décision le concernant. Ce comportement peut être détecté lors, par exemple, d'un scan de ports sur une machine. La deuxième alternative n'a pas été choisie pour une raison de flexibilité. Au niveau de l'espace noyau, nous ne pouvons pas profiter des outils de haut niveau tels que les bases de données. Ce choix peut sembler pénalisant en terme de performance, mais nous présenterons une manière simple de pallier ce problème.

Dans la suite, nous présentons chacun des trois composants. Leur fonctionnement et leur emplacement sont expliqués et justifiés. Afin d'illustrer les propos qui suivent, la figure 3 présente le positionnement des trois composants de l'implémentation dans un environnement GNU/Linux.

3.1 Module de redirection

Le module de redirection doit pouvoir intercepter les paquets afin de décider s'ils doivent être bloqués ou acceptés et vers où les rediriger en cas d'acceptation. Pour ce faire, nous nous sommes appuyés sur le pare-feu `netfilter` [7]. Il est composé de cinq chaînes :

– `INPUT` : traite un paquet à destination d'un processus local,
– `OUTPUT` : traite un paquet généré localement,
– `FORWARD` : traite un paquet routé,
– `PREROUTING` : traite un paquet provenant de l'exterieur,
– `POSTROUTING` : traite un paquet à destination de l'extérieur.

La figure 2 présente le parcours d'un paquet dans les différentes chaînes de `netfilter`. Chaque chaîne est composée de hooks `netfilter`. Un hook est un point de passage d'un paquet dans la pile IP de Linux. Les hooks d'une même chaîne sont ordonnés par priorité. Parmi ces hooks, citons les hooks `conntrack` permettant le suivi des connexions et les hooks `nat` permettant la modification des paquets. Une manière simple d'enrichir `netfilter` est de lui ajouter des hooks par l'insertion d'un module dans le noyau.

Aussi, pour notre implémentation, nous profitons des hooks standards de `netfilter`. Plus précisément, nous utilisons la cible `DNAT` des hooks nat pour rediriger les paquets. Cette cible

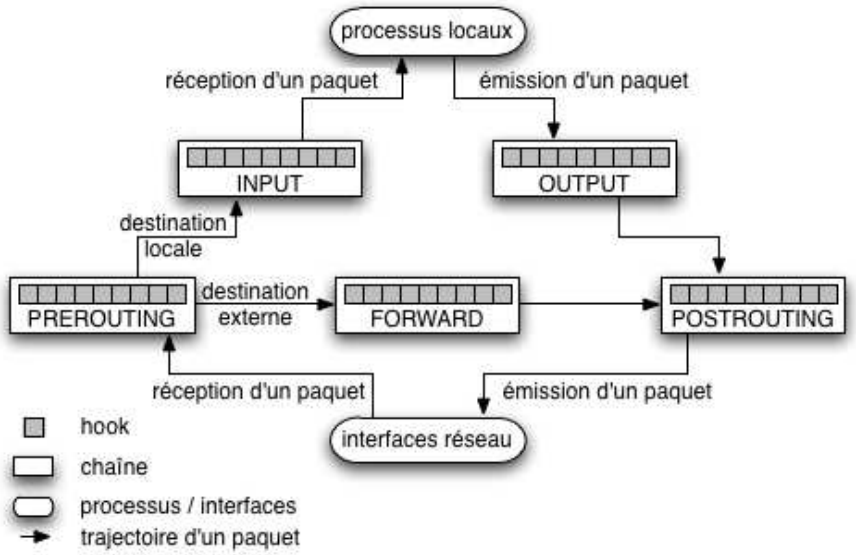

FIG. 2: Parcours des paquets dans les chaines de `netfiter`

est disponible depuis le hook de priorité `NF_IP_PRI_NAT_DST` = -100 de la chaîne `PREROUTING`. Notre module doit donc posséder une priorité inférieure. Ainsi, il peut marquer les paquets avec l'adresse de redirection. Ainsi, le hook permettant le nat réalise la redirection choisie. Ce dernier peut être configuré, par la création d'une règle via l'outil `iptables`, de la manière suivante :

```
iptables -t nat -A PREROUTING -m connmark --mark 0x201A8C0 \
    -j DNAT --to-destination 192.168.1.2
```

A priori, tous les paquets appartenant à une même connexion sont redirigés de la même manière par souci de cohérence. Une nouvelle connexion est identifiée, au niveau du pare-feu `netfilter`, par l'apparition d'un paquet réseau étiqueté `NEW`[1]. Le module `conntrack` est responsable, entre autres, de l'identification des paquets `NEW`, au niveau du hook de priorité `NF_IP_PRI_CONNTRACK` = -200 de la chaîne `PREROUTING`. Nous profitons de cette fonctionnalité du module `conntrack` en nous plaçant après cette priorité. Ainsi, nous pouvons nous contenter de traiter uniquement les premiers paquets de chaque connexion. Les autres paquets d'une même connexion seront redirigés par l'emploi du marquage de connexion.

Notre module de redirection est composé de deux hooks. Le premier, que nous nommerons `HOOKA` permet de décider du devenir du paquet. Rappelons que l'obtention du verdict au niveau de l'espace utilisateur engendre une latence au niveau du temps de réponse, qu'il est important de réduire au maximum. Afin de limiter ces interactions et ainsi accélerer les traitements, nous utilisons un cache

[1] Cette étiquette n'est pas à confondre avec le drapeau `SYN` du protocole TCP. Un paquet étiqueté `NEW` correspond à un paquet IP (TCP, UDP, ICMP ou autre) pour lequel il n'existe aucune connexion en cours.

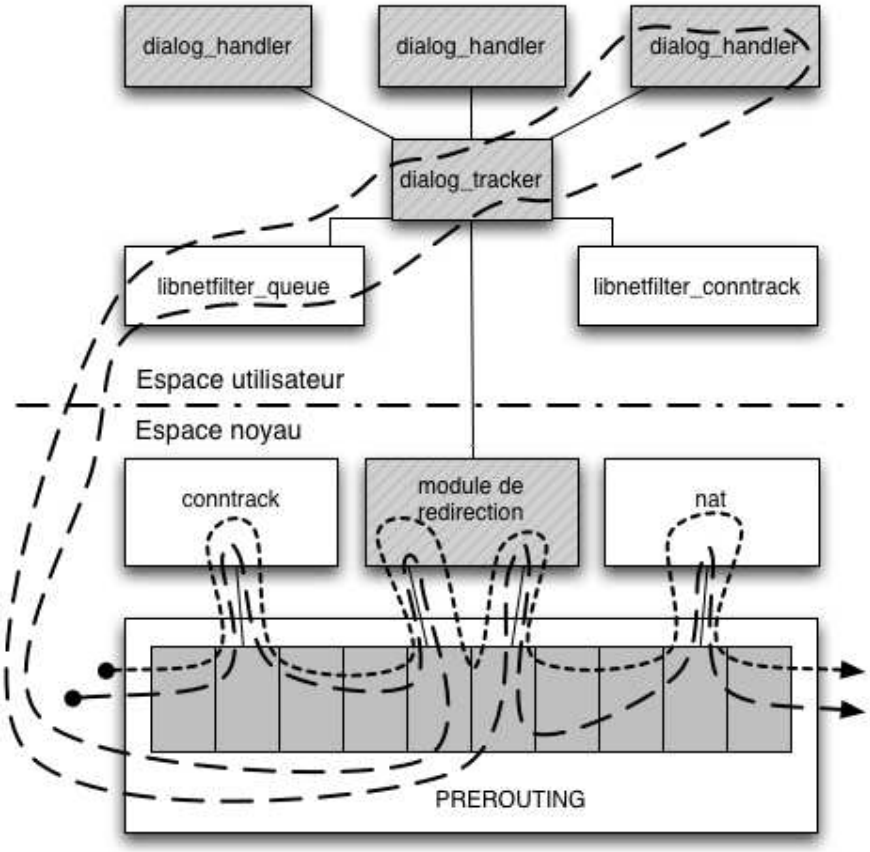

Fig. 3: L'architecture de redirection

au niveau de ce hook. Ce cache mémorise les verdicts obtenus auprès de l'espace utilisateur sous forme de règles. Par exemple, la règle suivante permet de rediriger vers la machine d toutes les connexions depuis le pot de miel vers la machine g :

```
pre :
  protocole   : *
  source      : *
  destination : g
post :
  protocole   : *
  source      : *
  destination : d
```

Ce cache est administré par les composants de l'espace utilisateur. Lorsqu'un paquet `NEW` arrive au niveau de notre module, le cache peut être suffisant pour décider de son devenir. Les verdicts peuvent être `NF_ACCEPT` pour un paquet accepté ou `NF_DROP` pour un paquet rejeté. Le scénario correspondant à l'acceptation d'un paquet est représenté par un trait en pointillé dans la figure 3. S'il n'existe pas de règle dans le cache pour un paquet entrant, le verdict doit être obtenu au niveau de l'espace utilisateur (`NF_STOLEN`). Ce scénario est représenté par un trait discontinu dans la figure 3. Pour solliciter les composants de l'espace utilisateur, nous avons deux possibilités : utilisation de `netlink` ou utilisation de la librairie `libnetfilter_queue`. Nous avons décidé d'utiliser cette librairie. Sans l'utiliser, nous aurions eu à :

- « voler » les paquets à envoyer vers l'espace utilisateur,
- gérer la liste des paquets volés,
- dupliquer le contenu de la fonction `nf_reinject` permettant de réinjecter un paquet précédemment volé, après obtention du verdict le concernant.

Par contre, en utilisant la librairie `libnetfilter_queue`, nous minimisons l'effort d'implémentation. Après obtention du verdict au niveau de l'espace utilisateur, le paquet poursuit son parcours à partir du hook de priorité juste supérieure à celle de `HOOKA`. Le deuxième hook entre en jeu.

Le second hook de notre module, nommé `HOOKB` de priorité juste supérieure à la priorité de `HOOKA`, permet de marquer les connexions afin de choisir la règle `iptables` à utiliser pour le nat. Les marques à appliquer aux connexions sont aussi récupérées auprès du cache du module.

3.2 Le dialog_tracker

Le `dialog_tracker` est le composant permettant de faire le lien entre le module de redirection et le `dialog_handler`. De cette manière, nous rendons le développement des `dialog_handler` indépendant de l'architecture et du système d'exploitation augmentant ainsi la flexibilité. Les interactions entre le `dialog_tracker` et le noyau se font par le biais d'une socket `netlink` et des librairies `libnetfilter_queue` et `libnetfilter_conntrack`. La librairie `libnetfilter_conntrack` permet à un programme au niveau de l'espace utilisateur d'être averti, entre autres, de la création et de la destruction de connexions. Les interactions entre le `dialog_tracker` et les `dialog_handler` se font par le biais d'une socket `AF_INET`.

Une demande de verdict est donc reçue par le biais de la librairie `libnetfilter_queue`. Elle est formattée à ce niveau, pour la rendre compréhensible par les `dialog_handler`. En fonction de son contenu, un `dialog_handler` est selectionné. La requête formattée est alors envoyée au `dialog_handler` selectionné. La réponse du `dialog_handler` correspond au verdict à utiliser pour la requête correspondante. Ce verdict est alors retourné via la librairie `libnetfilter_queue`.

Pour chaque création ou destruction de connexion, le `dialog_tracker` reçoit un évènement de la librairie `libnetfilter_conntrack`. Cet événement est formatté pour le rendre compréhensible par les `dialog_handler`. En fonction de son contenu, un `dialog_handler` est sélectionné. Le `dialog_handler` est alors averti de l'évènement.

A tout moment, un des `dialog_handler` peut décider d'ajouter ou de supprimer des informations du cache du module présent dans le noyau. Cette information est reçue par la socket `AF_INET`. Elle est envoyée au module de redirection par le biais de la socket `netlink`.

Les `dialog_handler` et le `dialog_tracker` peuvent être exécutés sur des machines différentes. Dans ce cas, les paquets permettant à ces composants de communiquer vont arriver dans la chaîne `PREROUTING`. Notre module va les intercepter. Il ne faut pas qu'il les rejète ou les envoie au niveau de l'espace utilisateur sous peine de bloquer le système. Le `dialog_tracker` a été développé de manière à enrichir le module de redirection pour qu'il accepte ces paquets.

3.3 Le dialog_handler

Le `dialog_handler` est le composant responsable, de la prise de décision du devenir des paquets. Plusieurs algorithmes différents peuvent être utilisés pour determiner le choix de redirection. Ils prennent en entrée le paquet à traiter et une base de données. Ce composant est principalement responsable de la cohérence des redirections.

La communication entre le `dialog_tracker` et ce composant étant réalisée par le biais d'une socket `AF_INET`, le `dialog_handler` peut être installé sur des machines distantes, offrant ainsi une grande souplesse. Autrement dit, ce composant possède un comportement indépendant de l'architecture et du système d'exploitation, facilitant ainsi le portage du principe présenté dans ce papier sur d'autres environnements. Cependant, la mise en œuvre du `dialog_tracker` et du `dialog_handler` sur des machines différentes peut être pénalisant du point de vue du temps de réponse.

L'activité des attaquants sur l'environnement peut être soutenue. Beaucoup de connexions sortantes peuvent être tentées. Danc ce cas, la totalité des redirections possibles seront activées, assez rapidement. Or, l'intérêt d'un attaquant pour une adresse que nous avons redirigé peut aussi être éphémère. Nous monopolisons des redirections pour des adresses qui n'intéressent plus les attaquants. Afin de ne pas figer l'expérience, une durée de vie est gérée au niveau des redirections en cours. Si la durée de vie d'une redirection tombe à zéro, la redirection correspondante est supprimée. Le `dialog_handler` contacte alors le `dialog_tracker` pour lui indiquer que la redirection est obsolète. Ceci se traduit par la suppression de la règle correspondante dans le cache du module de redirection. Par contre, si un attaquant contacte à nouveau une adresse que nous redirigeons, la durée de vie de la redirection correspondante est mise à jour.

La prise de décision aurait pu être réalisée au sein même du noyau. En la réalisant au niveau de l'espace utilisateur et de manière isolée, nous pouvons exploiter les bases de données existantes pour le stockage des informations. Ceci est important s'il nous semble intéressant de rediriger au même endroit un attaquant qui est venu plusieurs jours auparavant. Conserver en mémoire toutes les redirections afin de mettre en pratique cette idée n'est pas envisageable pour des raisons matérielles. Nous devons par moment stocker les informations sur ces redirections dans une base de données pour libérer les ressources.

La cohérence de l'implémentation dépend fortement des algorithmes employés pour décider du devenir des paquets. Nous présentons ici deux exemples d'implémentations du `dialog_handler`. Ce composant peut être amené à decider du devenir de deux connexions initiées depuis deux adresses d'Internet différentes, vers une même adresse d'Internet. Or, deux adresses d'Internet différentes peuvent être exploitées par un même attaquant. Dans le but d'éviter au maximum d'éveiller les soupçons de l'attaquant, nous avons fait les choix suivants. Toutes les redirections de connexions vers une même machine d'Internet doivent l'être vers le même pot de miel. De plus, des redirections de connexions vers des machines différentes d'Internet doivent être réalisées vers des pots de miel différents. Nous ne pouvons pas non plus nous permettre de rediriger toutes les adresses d'internet par manque de pots de miel. Seulement certaines adresses d'Internet seront redirigées. Une connexion vers une adresse d'Internet est redirigée vers le pot de miel qui lui est déjà associé. Si aucun pot de miel ne lui est associé, le choix de rediriger la connexion vers un pot de miel disponible est effectué aléatoirement. Le choix de la machine vers laquelle la redirection est effectuée peut être deterministe, effectué à priori ou bien aléatoire.

4 Performances

Comme nous l'avons exposé dans la problématique, il est important d'évaluer la transparence du mécanisme de redirection pour les pirates, et donc de savoir de quelle façon le mécanisme de redirection que nous avons mis en place pénalise les connexions réseaux. Il est évident que la modification de `netfilter` que nous avons réalisée, ainsi que le dialogue entre la partie noyau et la partie utilisateur va ralentir les connexions. Nous avons ainsi voulu évaluer la latence engendrée par ce mécanisme de redirection, et vérifier qu'elle est suffisamment faible pour ne pas alerter des pirates forcément attentifs, soupçonneux et en état d'alerte.

La partie la plus délicate pour le mécanisme mis en jeu est la gestion des nouvelles tentatives d'établissement de sessions. En effet lorsque le verdict d'une session a été donné[2], la grande partie du travail restant est faite par `netfilter` en mode noyau, contrairement à la gestion de nouvelles sessions qui se fait entre des processus se trouvant dans l'espace utilisateur et dans l'espace noyau.

Nous avons ainsi voulu éprouver ce mécanisme dans le domaine où il devrait être le moins performant : le scan de plages d'adresses. Nous avons ensuite réfléchi à comment effectuer ce scan suffisamment efficacement pour que la latence engendrée par le mécanisme soit perceptible : si le scan est trop lent les résultats obtenus nous induiraient en erreur.

Nous nous sommes ainsi fixés le contexte suivant : scan de plage d'adresses de $N_{adresse}$ adresses différentes sur un port TCP donné, et simuler n_{hote} hôtes disponibles dans cette plage d'adresses. Nous choisissons de n'envoyer qu'un seul SYN par adresse testée, et considérons que lorsqu'une réponse ne vient pas au bout d'un temps $T_{timeout}$, l'hôte en question est inaccessible.

A moins que n_{hote} soit égal à $N_{adresse}$, au moins un des timeout expirera, ce qui nous donne une borne inférieure concernant le temps d'un tel scan : $b_{inf} = T_{timeout}$.

La question qui vient ensuite est de savoir comment implémenter les tentatives de connexions :

1. Lancer n_{thread} tentatives de connexions en parallèle en attendant le verdict d'une tentative de connexions avant d'en lancer une autre.
2. Lancer les $N_{adresse}$ tentatives consécutivement sans attendre le verdict d'une tentative, puis analyser les paquets lorsqu'ils arrivent ou annuler l'hôte après que $T_{timeout}$ se soit écoulé depuis l'émission du paquet.

Concernant la première méthode, avec n_{thread} tâches parallèles, le b_{inf} peut éventuellement être atteint. Cependant cette solution passe difficilement à l'échelle, car il n'est par exemple pas réaliste d'exécuter 2^{16} threads en parallèle sur une machine personnelle. Il est donc nécessaire de répartir équitablement la plage d'adresse parmi les n_{thread} tâches. En effectuant la division euclidienne suivante nous obtenons :
$N_{adresse} = q \times n_{thread} + r$, $0 \leq r < n_{hread}$.
Ainsi en attribuant $q + 1$ adresses aux r premières tâches[3] puis q aux restantes nous déduisons que cette méthode lors d'un scan où le timeout expire chaque fois, durera au mieux $q \times T_{timeout}$, et que n_{thread} seront nécessaires pour espérer obtenir cette valeur.

Si on étudie la deuxième méthode, la difficulté réside dans la gestion des expirations des différents timers. Cette gestion engendre le besoin de stocker au pire $N_{adresse}$ structures de données en plus de la complexité de l'algorithme de gestion. Dès lors, il se peut que le temps associé à la gestion de $N_{adresse}$ timers soit supérieur à deux fois le temps associé à la gestion de $\frac{N_{adresse}}{2}$ timers, en

[2] À savoir donner le verdict de rediriger ou non.
[3] Possible car $r < n_{thread}$.

scannant la première moitié de la plage, puis l'autre moitié. Nous avons évalué empiriquement ce phénomène dans l'environnement décrit un peu plus loin et avons obtenu les résultats suivants.

$N_{adresse}$	T_{scan} (secondes)
2^8	4
2^9	4.33
2^{10}	7
2^{11}	12.3
2^{12}	23.5
2^{13}	60.5

Nous voyons que pour $N_{adresse} \leq 2^{12}$ lancer un seul scan est plus efficace que de lancer deux scans consécutifs en coupant la plage d'adresses en deux, car, $T_{scan}(N_{adresse}) < 2 \times T_{scan}(\frac{N_{adresse}}{2})$. Cependant pour $N_{adresse} = 2^{13}$ c'est faux. Nous avons donc choisi de lancer durant notre expérience au maximum 2^{12} scans consécutifs.

Nous avons implémenté les deux stratégies sur un programme en C s'exécutant sous `Windows XP SP1` avec la valeur de registre `MaxTcpRetransmissions` affecté à 0 (pour qu'un seul SYN soit envoyé), un $T_{timeout}$ de 3s (valeur par défaut),$n_{hote} = 3$ et $N_{adresse} \in \{2^8, ..., 2^{16}\}$ en utilisant :

1. L'API threads de `Windows` et les `Winsocket` en mode SOCK_STREAM synchrones pour la stratégie 1.
2. Le select de l'API de `Windows` et les `Winsocket` en mode SOCK_STREAM asynchrones pour la stratégie 2.

Ce système s'est exécuté sur une machine virtuelle QEMU disposant de 128Mo de RAM. Le système hôte était un noyau Linux 2.6.19 avec le module noyau `kqemu`, qui utilisait un `tap device bridgé` pour donner un accès réseau à la machine virtuelle. On a affecté la priorité maximale au processus QEMU à l'aide de la commande nice. La machine quant à elle était un Intel Pentium 4 3.00GHz avec 2Go de RAM. Nous avons obtenus les résultats suivants avec la stratégie 2.

Premièrement le scanner a déduit la disponibilité des trois hôtes simulés et l'indisponibilité des autres pour chaque expérience, ce qui démontre que le scanner mais surtout le module de redirection ont correctement fonctionné. Les performances obtenues sont décrites dans le tableau suivant :

$N_{adresse}$	T_{scan} sans redirection (secondes)	T_{scan} avec redirection (secondes)
2^8	3.0	3.5
2^9	4.5	5.0
2^{10}	7.0	6.5
2^{11}	12.0	13.0
2^{12}	23.5	26.0
2^{13}	48.5	54.5
2^{14}	93.0	114.0
2^{15}	117.5	255.5
2^{16}	219.0	673.5

La pertinence de la méthode 2 est montrée par le cas $N_{adresse} = 2^{16}$. En gardant les même notations que précédemment, nous avions $T_{minMethode1} \geq q * T_{timeout}$ avec $q = E(\frac{N_{adresse}}{n_{thread}})$.
Pour espérer avoir un T_{scan} de 219.0s il nous faudrait donc grossièrement : $n_{threads} \simeq \frac{N_{adresse}}{T_{scan}} \times T_{timeout}$. L'application numérique nous donne 897, en d'autres termes une valeur irréelle pour les conditions de notre expérimentation.

Un premier constat que nous pouvons faire est que pour $N_{adresse} = 2^8$, b_{Inf} est atteinte par notre programme, mais que T_{scan} s'éloigne ensuite de plus en plus de cette valeur[4]. Il nous faudrait cependant évaluer le temps d'envoi T_{envoi} de $N_{adresse}$ segments SYN dans cette plage d'adresse pour estimer plus finement b_{Inf}, car dans le cas où le dernier hôte scanné n'est pas disponible, $b_{Inf} = T_{envoi}(N_{adresse}) + T_{Timeout}$. Cependant l'objectif du programme n'était pas d'atteindre cette valeur, mais d'être suffisamment rapide pour montrer la latence engendrée par le mécanisme de redirection que nous avons implémenté, contrat qui a donc été rempli. En effet jusqu'à $N_{adresse} = 2^{12}$ le mécanisme à l'air de bien fonctionner, cependant nous devons optimiser les algorithmes de gestion d'adresses pour obtenir des valeurs plus raisonnables pour $N_{adresse} > 2^{12}$.

5 Expérimentations

Nous avons mis en œuvre deux expérimentations pour valider la pertinence du mécanisme de redirection présenté dans le cadre de ce papier.

5.1 Observation de l'execution de *malware*

La première expérimentation porte sur l'infiltration des *botnets* en exécutant et observant des *malware* téléchargés à l'aide du pot de miel *Nepenthes*, depuis une sandbox [6]. Les *botnets* sont des réseaux de machines compromises qui peuvent recevoir des ordres d'un contrôleur pour mener, entre autres, des attaques distribuées. Le plus souvent, les machines compromises et le contrôleur communiquent par le biais d'un protocole similaire à *IRC*. Après avoir identifié les flux `Command and Control` associé a une instance de *malware* en redirigeant les flux émis vers une machine que l'on maîtrise [9], nous avons exécuté ce *malware* dans une instance de QEMU. Nous avons aussi ajouté deux pots de miels tournant sur deux autres instances de QEMU :

1. Haute Interaction : *Windows XP SP1*.
2. Basse Interaction : *Nepenthes*.

Ces machines sont connectées à l'aide de `tap devices bridgés`, les adresses étant :

192.168.0.251 Le *malware*.

192.168.0.200 Le haute interaction.

192.168.0.252 Le basse interaction.

Le *malware* est autorisé à accèder à Internet seulement pour le flux `Command and Control` et le `DNS`. Ces flux sont `nat`és avec une adresse publique IP_1. Les flux à destination des ports TCP $\{135, 139, 445\}$ sont gérés par notre mécanisme de redirection, et les flux restants sont bloqués.

L'expérience menée a produit le scénario suivant :

Après s'être connecté à un serveur *IRC* sur un port TCP non standard(5190), l'instance de *malware* a reçu l'ordre suivant :

```
:hub.24324.com 332 seivNbtC #last:=BGX5tCMl9HMuPlQRIfr7ZDvrWvjsrx3QcTGwmkNACosllT
7o+6BL/FkEl1LzB/AkO7BSNYd1ycZi/zOu/AWHE5fJNT02YoaGogFZbH03O9Z/OVp4bDrWR3gJyIug2Ee
e3JVQHBn/fWG6ANlrYr0mZbtKuh
```

[4] À titre d'information C :¿nmap -n -P0 -T5 -sS ¡x.y.0.0¿/22 -p 22 ($N_{adresse} = 2^{10}$) a mis 53.765s.

Ce canal *IRC* utilise ainsi une certaine technique d'obfuscation pour ne pas être détecté par des approches à base de signatures de commandes *IRC* utilisées par les différents bots [10].

Le *malware* a dès lors lancé un scan de port sur une plage génerée à partir de IP_1 au port TCP 135. Ce dernier ne peut à lui seul connaître cette adresse. Cette information a donc été communiquée dans le message encodé. Notre mécanisme à redirigé les attaques vers nos deux pots de miels avec succès. Ces derniers ont établi la connexion puis ont terminé la session à la demande du client malicieux. Le *malware* a ensuite envoyé un exploit sur ce même port à nos deux pots de miels avec succès et a informé son pirate par le canal de communication de la réussite des attaques, en dénombrant le nombre de réussites associées à ce type d'attaque[5] :

```
PRIVMSG #last :-04dcom2.04c- 1. Raw transfer to 192.168.0.252 complete.
PRIVMSG #last :-04dcom2.04c- 2. Raw transfer to 192.168.0.200 complete.
```

Grâce au mécanisme de redirection, nous avons ainsi pu comprendre une partie d'un message encodé et observer plus en détail leur comportement.

Le lecteur aura cependant remarqué que ce cas peut être louche pour l'administrateur du `botnet` : après un scan d'une plage d'adresse, son instance de *malware* l'informe qu'il a réussit a prendre la main sur deux adresses d'une autre plage[6]. Ici encore notre mécanisme peut être amélioré.

5.2 Observation des activités d'un pirate

Pour la deuxième expérimentation, nous avons déployé sur Internet notre pot de miel haute interaction intégrant le mécanisme de redirection. L'objectif est de collecter des données sur des attaques réelles. Ces données nous permettrons d'enrichir nos connaissances sur le processus d'intrusion des attaquants sur Internet en prennant en compte le phénomène de rebond. De plus, l'analyse détaillée des résultats obtenus et la comparaison avec les observations obtenues dans [2] nous permettrons de confirmer la pertinence du mécanisme proposé et d'illustrer les limitations. La collecte est toujours en cours.

La plateforme que nous avons mise au point est, ainsi que le repréente la figure 4, composée de quatre machines virtuelles s'exécutant sur une machine physique. Nous avons fait le choix d'utiliser QEMU [8] pour exécuter les machines virtuelles. Le noyau de la machine physique a donc été modifié ainsi que nous l'avons présenté dans le papier de façon à implémenter notre mécanisme de rebond dynamique.

Plus précisément, deux des quatres machines virtuelles `M1` et `M2` sont accessibles depuis Internet via une connexion SSH. Les deux autres machines `R1` et `R2` ne sont pas accessibles depuis Internet. Sur chacune de ces machines est installé un système d'exploitation Gnu/Linux, dont le noyau a été modifié, comme décrit dans [2], afin d'observer l'activité des intrus connectés. Nous avons créé sur les machines `M1` et `M2` des comptes avec des mots de passe faibles de façon à ce que des attaquants puissent facilement s'y introduire. Nous avons également configuré le mécanisme de redirection dynamique de telle façon que certaines des tentatives de scan SSH venant des machines `M1` et `M2` soient automatiquement redirigées vers les machines `R1` et `R2`. Ainsi, lorsqu'un attaquant réussit à s'introduire sur la machine `M1` par exemple et que, depuis cette machine, il lance un scan SSH vers une certaine plage d'adresses IP, il y aura forcément une partie des scans qui lui paraitront avoir abouti et qui seront en fait simplement redirigés vers `R1` ou `R2`.

[5] Voir ...last :-<faille_exploitee>- <nb_reussite>.

[6] 192.168.0.252,192.168.0.200 n'étaient pas dans la plage génerée à partir de $IP1$.

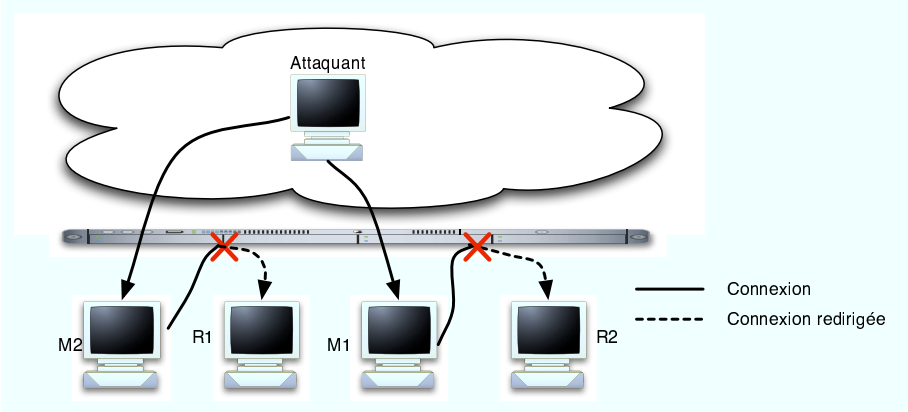

FIG. 4: Plateforme d'observation des activités des pirates.

A l'heure où nous écrivons cet article, la plateforme est rééllement opérationnelle depuis 1 mois et demi. Nous n'avons pour le moment obtenu que des résultats partiels. Nous avons effectivement pu vérifier que le mécanisme de redirection automatique fonctionne bien mais nous n'avons pas encore vu d'attaquant le subir complètement. En effet, nous avons bien observé des attaquants s'introduire sur la machine `M1` ou `M2` et lancer des scans SSH sur des plages d'adresses IP. Alors que jusqu'à présent, nous n'autorisions pas ce type de scans, la mise en place du mécanisme de redirection nous a permis de leurrer l'attaquant sur le succès de ces scans. Ceci nous a montré que l'implémentation de notre mécanisme fonctionne correctement. Nous l'avons constaté par exemple avec les traces suivantes :

```
james@M1:~/rep_hacker$ ./unix  66..
[+][+][+][+][+] UnixCoD Atack Scanner [+][+][+][+][+]
[+]   SSH Brute force scanner : user & password   [+]
[+]        Undernet Channel : #UnixCoD            [+]
[+][+][+][+][+][+][+] ver 0x10  [+][+][+][+][+][+][+]
[+] Scanam: 66.221.4.* (total: 2) (1.6% done)
66.221.8.* (total: 2) (3.1% done)
66.221.12.* (total: 2) (4.3% done)
66.221.16.* (total: 2) (5.9% done)
66.221.19.* (total: 2) (7.5% done)
66.221.23.* (total: 2) (9.0% done)
66.221.27.* (total: 2) (10.2% done)
66.221.30.* (total: 2) (11.8% done)
66.221.34.* (total: 2) (13.3% done)
66.221.38.* (total: 2) (14.5% done)
66.221.41.* (total: 2) (16.1% done)
66.221.45.* (total: 2) (17.6% done)
66.221.49.* (total: 2) (18.8% done)
66.221.52.* (total: 2) (20.4% done)
```

```
66.221.56.* (total: 2) (22.0% done)
66.221.60.* (total: 2) (23.1% done)
```

L'affichage de `Total :2` montre bien que le mécanisme de redirection fonctionne. En effet, nous avions simplement configuré notre mécanisme pour rediriger de façon dynamique deux adresses IP vers nos deux machines `R1` et `R2`. Nous avons ainsi pu observer ce type de tentative de scans et les redirections associées plusieurs fois depuis la mise en place de la plateforme.

En revanche, et c'est en ce sens que les résultats sont encore partiels, nous n'avons pas vu pour le moment, l'attaquant tenter de se connecter depuis `M1` ou `M2` vers les adresses IP qu'il venait de scanner avec succès. Ainsi dans l'exemple ci-dessus, les deux adresses IP du réseau `66.x.y.z` que l'attaquant croit avoir scanné avec succès sur le port 22 n'ont pas été utilisées par la suite par l'attaquant. On peut proposer plusieurs raisons à cette attitude. Il est possible que l'attaquant considère ne pas disposer d'un nombre d'adresses IP suffisantes pour lancer une attaque sur ce réseau. Il est également possible que l'attaquant que nous avons observé ne soit en charge que de tentatives de scans d'adresses IP mais pas de l'attaque directe des machines scannées avec succès[2]. Dans ce cas, il est probable qu'il va simplement stoquer quelque part la liste des adresses IP scannées et qu'elles seront attaquées plus tard depuis nos machines `M1` et `M2` ou depuis d'autres sites. En l'état actuel, il nous est difficile de répondre à ces questions et seule la poursuite de notre expérimentation nous permettra d'en savoir plus.

6 Conclusion et perspectives

Le degré d'observabilité d'un attaquant qui s'est introduit sur un pot de miel est directement lié au niveau d'interaction qu'il a avec ce pot de miel. L'utilisation habituelle des pots de miel haute interaction est généralement limité au sens où ils ne permettent pas à l'attaquant de poursuivre son attaque sur Internet. Nous avons proposé dans cet article un mécanisme de redirection dynamique qui a pour objectif d'améliorer les possibilités d'observations de processus d'attaques avec rebonds, en tenant compte des contraintes légales.

Le mécanisme a été implémenté pour un environnement GNU/Linux. Néanmoins, ce mécanisme est facilement portable sur d'autres environnements. Deux expérimentations ont été mises en œuvre pour valider la pertinence du mécanisme proposé. Néanmoins, il est nécessaire de continuer les expérimentations en cours pour obtenir des résultats plus significatifs permettant de mieux cerner les limites de notre approche actuelle et de proposer des extensions permettant de les résoudre.

Ces travaux ont ete effectués en partie dans la cadre des projets CADHo de l'ACI Séurité et Informatique, et des projets européens ReSIST (IST 026764) et CRUTIAL (027513).

Références

1. Bailey, M., Cooke, E., Watson, D., Jahanian, F., Provos, N. : A Hybrid Honeypot Architecture for Scalable Network Monitoring. Technical Report CSE-TR-499-04, University of Michigan (2004).
2. Alata, E., Nicomette, V., Kaâniche, M., Dacier, M., Herrb, M. : Lessons learned from the deployment of a high-interaction honeypot. In EDCC'06, 6th European Dependable Computing Conference, October 18-20, 2006, Coimbra, Portugal (2006).
3. Nieh, J., Leonard, O. C. : Examining VMware. *j-DDJ*, 25 (8), pp. 70, 72–74, 76, `http://www.ddj.com/` (2000).

4. Leita, C., Dacier, M., Massicotte, F. : Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In RAID 2006, 9th International Symposium on Recent Advances in Intrusion Detection, September 20-22, 2006, LNCS 4219, Springer Verlag (2006).
5. Provos, N. : Honeyd - A VirtualHoneypot Daemon. In 10th DFN-CERT Workshop, Hamburg, Germany (2003).
6. Freiling, F., Holz, T., Wicherski, G. : Botnet Tracking : Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. Technical Report AIB-2005-07, RWTH Aachen, `http://pi1.informatik.uni-mannheim.de/publications/show/12` (2005).
7. Napier, D. : IPTables/NetFilter – Linux's Next-Generation Stateful Packet Filter. *j-SYS-ADMIN*, 10 (12), pp. 8, 10, 12, 14, 16 (2001).
8. Bellard, F. : QEMU, a Fast and Portable Dynamic Translator. In USENIX 2005 Annual Technical Conference, FREENIX Track, pp. 41–46 (2005).
9. Rajab, M. A., Zarfoss, J., Monrose, F., Terzis, A. : A Multifaceted Approach to Understanding the Botnet Phenomenon. In Proceedings of Internet Measurement Conference 2006 (IMC'06) (2006).
10. Kristoff, J. : Botnets. In 32nd Meeting of the North American Network Operators Group (2004).