Plateforme pour l'exécution contrôlée de logiciels malveillants

Ion Alberdi — Vincent Nicomette — Philippe Owezarski

CNRS ; LAAS ;
7 Avenue du colonel Roche, F-31077 Toulouse, France
Université de Toulouse ;
UPS, INSA, INP, ISAE ; LAAS ;F-31077 Toulouse, France
ialberdi@laas.fr

RÉSUMÉ. *L'Internet, fort de son succès populaire, n'est pas un environnement sûr. Il pèse sur l'Internet et ses utilisateurs un risque qui augmente au fur et à mesure que l'économie numérique se développe. Ce risque se présente souvent sous la forme de botnets, de larges ensembles de machines corrompues par des pirates qui peuvent les utiliser à dessein pour lancer des attaques massives, des campagnes de spam, etc. Pour lutter de façon proactive contre ces menaces, il est essentiel d'analyser le trafic malveillant dans l'Internet correspondant à la diffusion et au mode d'action des virus, vers ou autres maliciels qui servent à la constitution des botnets. Cet article décrit donc une plateforme de collecte et d'exécution de ces maliciels qui permet d'observer et analyser l'activité illégitime de l'Internet. Toutefois, la législation en vigueur rend responsable le propriétaire d'une machine de tout le trafic généré. Aussi, cette plateforme doit contrôler le trafic qu'elle génère. Un élément essentiel de cette plateforme est donc un nouveau type de pare-feu adapté à l'analyse de toutes les formes de communications, et qui ne va laisser passer que les paquets ne présentant aucun danger. L'article présente les composants de la plateforme et détaille le pare-feu. Il illustre également son fonctionnement et propose une évaluation de ses performances.*

ABSTRACT. *In spite of its popular success, the Internet is not secure. A big threat exists on the Internet and its users, and it increases with the development of the cyber economy. This threat is often related to botnets, i.e. large sets of corrupted machines that hackers can use for launching massive attacks, spams, etc. To proactively fight against these threats, it is essential to analyze malicious traffic corresponding to the spreading and operating modes of viruses, worms or other kinds of malware which serve for the constitution of botnets. This paper describes a platform for collecting and analyzing such malware which allows us to observe and analyze the illegitimate activity on the Internet. However, by law, the owner of a computer is responsible of all the traffic it generates. Therefore, this platform must control its outgoing traffic. One essential component of this platform is then a new type of firewall suited to the analysis of all kinds of communications, and that let go out only the packets presenting no risk. The paper presents the different platform components and details the firewall. It also illustrates its behavior and proposes an evaluation of its performances.*

MOTS-CLÉS : *trafic malveillant, pare-feu, pot de miels, botnets*

KEY WORDS: *malicious traffic, firewall, honeypots, botnets*

1re soumission à *CFIP 2009*, le 28 août 2009.

1. Introduction

Alors que l'Internet est en train de devenir un réseau multi-services pour différents types d'applications, il est également devenu le vecteur privilégié de propagation de logiciels malveillants. Les *botnets* (réseaux de machines compromises) représentent aujourd'hui une des menaces sensibles pesant sur le réseau Internet. Ils sont à l'origine des multiples attaques sur ce réseau et par la même à l'origine de certains dysfonctionnements : lancement et propagation de vers, envoi massif de pourriels, attaques de déni de service, etc.

Ces botnets sont des réseaux de machines pilotées à distance à l'aide de logiciels malveillants préalablement installés, à l'insu du propriétaire de la machine pour la plupart des cas. Pour observer et analyser ces réseaux, une des possibilités consiste à collecter des instances de logiciels malveillants qui pilotent les machines infectées, puis à observer localement leurs agissements en les exécutant. Il est ensuite nécessaire d'extrapoler l'information obtenue après observation locale pour en déduire des informations sur les phénomènes globaux. Cet article se focalise sur la méthodologie adoptée pour l'observation locale et laisse ouvert le problème de l'extrapolation.

Des plateformes permettant d'exécuter ce type de logiciels malveillants ont vu le jour. Les *bac-à-sable* tels que [BAR 07] choisissent d'exécuter le logiciel malveillant dans un environnement Internet entièrement émulé. Or l'information déduite de ces environnements est dépendante de la capacité qu'a l'Internet émulé à se faire passer pour l'Internet réel. Premièrement cette stratégie implique un problème évident de passage à l'échelle. Enfin, elle présente l'inconvénient majeur de ne pouvoir observer les ordres des maîtres des botnets en temps réel. Or, observer en temps réel le début d'une attaque de déni de service distribuée vers un service réel peut permettre de lever au plus tôt une alarme la concernant et ainsi d'en limiter les dégâts.

Une autre approche utilisée dans la rétro-ingénierie des protocoles de communication de ces logiciels, ou du botnet tout entier, consiste à exécuter ces logiciels dans une machine virtuelle sans pour autant filtrer leur communications [HOL 08]. Or lorsque l'on exécute un logiciel malveillant sur une machine, le propriétaire de la machine en question est responsable des éventuels effets de bord liés à son exécution, et en particulier des éventuelles attaques ou tentatives de corruption perpétrées vers des sites extérieurs. En France, comme dans de nombreux pays, un laboratoire de recherche est pénalement responsable du trafic qu'il émet. L'objet de notre travail a pour but principal d'appréhender cette problématique lors de l'exécution de maliciels à des fins d'étude du trafic malveillant.

Notre contribution porte sur la proposition d'un nouveau type de plateforme qui adopte une approche hybride. Cette plateforme émule les services permettant d'observer la propagation des logiciels, tout en surveillant minutieusement l'accès Internet que nous offrons à ces derniers à l'aide d'un nouveau type de filtre de paquets, élément central de notre architecture. L'article est composé comme suit. Nous présentons tout d'abord dans la partie 2 un résumé rapide du principe de fonctionnement des botnets. Nous donnons ensuite dans la partie 3 un aperçu de la plateforme d'exécution de logiciels malveillants que nous avons conçue et dont le filtre de paquets (partie 4) constitue le cœur. La partie 5 en donne quelques exemples d'utilisation et quelques tests de performances. Enfin, la section 6 conclut cet article.

2. Botnets

Un *botnet* est un réseau de machines compromises appelées *bots* ou *zombies*, piloté par une ou plusieurs personnes (appelée(s) maître(s) du botnet). Le pilotage des ces *zombies* se fait à l'aide d'un canal de communication appelé `Command & Control` (`C&C`).

Les stratégies de propagation d'un *zombie* peuvent être nombreuses. Il peut par exemple tenter d'exploiter une faille d'un système d'authentification[1], une corruption de l'espace d'adressage d'une application boguée[2]. Il peut également envoyer des logiciels malveillants en pièce jointe de mail ou publier des logiciels corrompus.

Ces stratégies présentent cependant le même but : réussir à ce que la cible exécute le début d'un code qui à terme, exécutera le logiciel malveillant développé. Enfin, en plus de ces tentatives de propagations, ces logiciels participent à des actions malveillantes distribuées et préprogrammées : envoi de pourriel, déni de service distribué, extraction de données personnelles, etc...

L'étude d'un de ces logiciels, même s'il peut sembler caduc à ce jour, permet d'illustrer assez clairement ces différents concepts. Agobot/Sdbot[3] et ses variantes utilisent des dialectes du protocole IRC comme canal `C&C`, se propagent entre autres à l'aide de failles visant le service SMB du système Windows XP et selon les ordres de son maître, tentent de se propager ou lancent différentes attaques de déni de service distribué. Ils peuvent aussi se mettre à jour ou envoyer du pourriel.

Nous allons à présent illustrer le fonctionnement de notre plateforme d'observation en y exécutant ce type de logiciel.

3. Plateforme d'exécution et d'analyse de logiciel malveillants

Pour étudier ces botnets qui menacent l'Internet, nous avons choisi de nous focaliser sur l'étude et l'analyse des logiciels qui les pilotent. Cette section est ainsi dédiée à une présentation succincte de notre plateforme, avant d'en présenter le pare-feu dans la partie suivante (partie 4). La recherche concernant l'automatisation de la collecte de logiciels malveillants a fourni des méthodologies et outils assez efficaces [BAE 06] que nous utilisons pour collecter certains échantillons.

En ce qui concerne la plateforme d'exécution, nous présentons donc tout d'abord certains aspects méthodologiques puis quelques précisions concernant son implémentation.

3.1. *Méthodologie et conception*

Pour exécuter les logiciels malveillants collectés, puisque nous ne pouvons pas légalement les exécuter sans restriction, nous avons créé un environnement adapté à ces contraintes. Il a pour but d'émuler les communications que nous ne pouvons laisser passer à travers Internet tout en surveillant scrupuleusement les autres. En particulier le trafic du canal `C&C` d'un bot n'est pas en lui même un

1. Comme des mots de passe faibles.
2. Débordement de tampons...
3. `http://www.viruslist.com/en/viruses/encyclopedia?virusid=24977`.

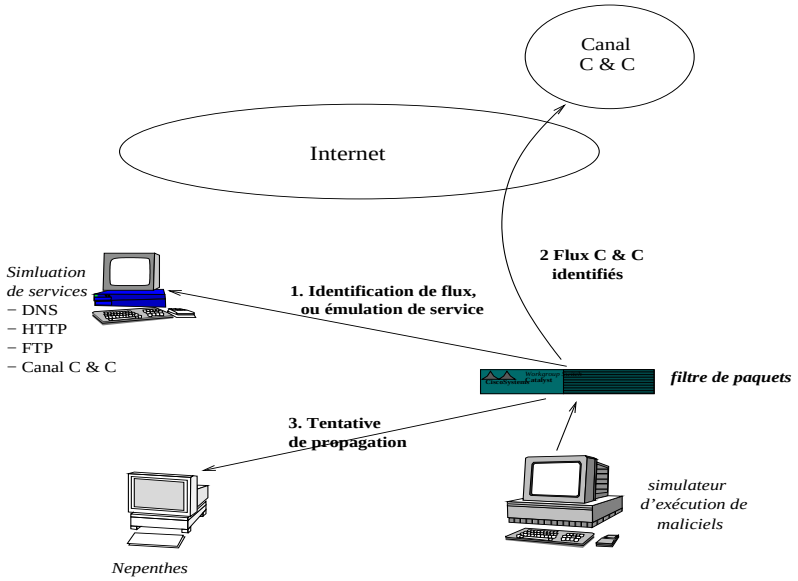

Figure 1. *Plateforme d'exécution*

danger pour le système d'information : il a pour but de communiquer des informations sans pour autant attaquer une tierce partie, il utilise simplement des ressources réseaux. Nous essayons donc de fournir un niveau d'interaction aussi haut que possible sur l'émulation de ces communications : laisser notre `bot` communiquer sur ce canal. Mais pour éviter le cas de figure où une attaque se cache dans un prétendu canal `C&C`, nous inspectons ce trafic à l'aide de notre filtre de paquets.

Il reste à savoir comment identifier les flux, simuler certaines interactions, et surveiller les communications avec l'Internet. Nous utilisons pour cela une approche itérative basée sur des exécutions successives du même logiciel malveillant. Lorsque nous identifions un nouveau flux, nous le redirigeons dans un premier temps vers une machine de notre réseau local qui simule le service demandé par ce flux. Nous observons et analysons ensuite ce flux pour décider s'il y a lieu de le laisser passer sous la vigilence de notre filtre, ou non, lors de la prochaine exécution du logiciel malveillant.

3.2. *Implémentation*

La plateforme globale de simulation est schématisée sur la figure 1.

Elle comprend :

– Un machine qui exécute les logiciels malveillants,

– Un simulateur de services ; cette machine est capable de simuler à la demande les services auxquels fait appel le logiciel malveillant : DNS, HTTP, FTP, etc. Lorsque, lors de l'exécution du lo-

giciel malveillant, celui-ci exécute des requêtes sur des ports bien identifiés et associés à ces services, les flux sont redirigés vers ce simulateur pour tenter de poursuivre l'exécution.

– Pour les services Windows (Netbios, SMB) : l'attaque est redirigée vers une machine exécutant un pot de miel tel que *nepenthes* [BAE 06], dont le but est de simuler une exploitation de faille connue, ou en d'autres termes d'émuler le succès d'une tentative de propagation,

– Pour tout autre service au port inconnu, les flux sont redirigés vers des serveurs TCP ou UDP, qui ne font que lire les octets en entrée.

– Au centre de la plateforme, notre filtre de paquets prend les décisions concernant les communications réseaux. Ce filtre et ses décisions sont cruciaux pour le bon fonctionnement de notre plateforme. Nous reviendrons donc en détail sur la conception et le développement de ce filtre dans la section 4.

3.3. *Un exemple d'exécution*

Pour illustrer le fonctionnement de ce système, l'exemple suivant - correspondant au cas où le logiciel malveillant exécuté a pour objectif de scanner le réseau - détaille toutes les étapes de l'émulation.

Dans un premier temps, le filtre de paquets bloque tous les flux en sortie, mises à part les requêtes DNS qui sont redirigées vers un serveur DNS que nous maîtrisons. Le logiciel malveillant envoie alors une requête DNS pour obtenir l'adresse IP de `URL1`. Supposons que `URL1` existe. Une fois cette adresse obtenue, le bot demande l'ouverture d'une connexion TCP (envoi d'un paquet TCP SYN) vers le port `p1` de la machine hébergeant `URL1`. Notre système va alors rediriger cette requête vers une de nos machines locales (flux 1 de la figure 1). Cette machine va accepter l'ouverture de la connexion TCP. Dès lors, comme la plupart du temps le canal `C&C` du logiciel malveillant est de type IRC, les premières requêtes envoyées après l'établissement de la connexion sont du type :

```
USER user1
NICK nick1
```

Cela nous renseigne également sur la localisation du canal `C&C` du logiciel malveillant qui se trouve à (`URL1`, `port1`). Nous relançons alors le logiciel malveillant en laissant cette fois passer le flux à destination de (`URL1`, `port1`) (flux 2 de la figure 1). Grâce à ces échanges sur le canal IRC `C&C`, notre logiciel malveillant connaît la tâche qu'il doit effectuer : une tâche de scanning d'autres machines de l'Internet (puisque c'est l'exemple que nous considérons). Le logiciel malveillant envoie donc des paquets SYN sur des adresses IP croissantes vers un port donné - le port 445 par exemple. Notre système, qui observe les communications, en déduit qu'il s'agit d'un scan réseau. Le logiciel malveillant est alors relancé après avoir reconfiguré le système pour qu'une des adresses du scan soit redirigée vers une instance de *nepenthes*. On simule ainsi la réussite de propagation du logiciel malveillant (flux 3 de la figure 1).

Cette plateforme est donc fonctionnelle et repose en grande partie sur le filtre de paquets qui permet d'automatiser et de rendre sa configuration plus sûre. Nous présentons ce filtre dans la section suivante.

4. Pare feu

La surveillance de l'accès à Internet lors de l'exécution de logiciels malveillants sur notre plateforme est cruciale. Les outils existants à ce jour, qu'ils soient dénommés systèmes de prévention d'intrusion, pare-feux, ou pare-feu applicatifs sont inadaptés à notre cas d'utilisation.

En effet, le principe de bout-en-bout [GOT 04] qui a contribué au succès du réseau Internet, incite à déléguer autant que possible l'intelligence du réseau à ses extrémités. Une conséquence directe de ce principe est le déséquilibre entre les outils permettant de développer des piles protocolaires aux extrémités, et les outils permettant de développer les inspections de ces piles protocolaires au cœur des réseaux de communications.

Lorsqu'un informaticien se lance dans le développement d'un ensemble de logiciels communiquant selon un protocole, ce dernier peut se baser sur un vaste ensemble de cadriciels constitué d'outils d'encapsulation[4], des différentes librairies socket, de librairies implémentant des protocoles applicatifs[5] et des langages informatiques associés.

Malheureusement ces cadriciels sont pour la plupart inutilisables pour par exemple développer un filtre de paquets. En effet la suite de paquets entrants et sortants vue par les extrémités est différente de celle vue par *un homme du milieu*, Internet pouvant réordonner, perdre ou même dupliquer des paquets [JAI 02]. Dès lors, comme démontré par [ROO , BID 07], les piles protocolaires des nœuds intermédiaires sont différentes des piles protocolaires des extrémités. Négliger l'indéterminisme associé au devenir de la suite de paquets vue par l'homme du milieu peut en effet induire des dysfonctionnements des extrémités.

Les systèmes de détection/prévention d'intrusion et les filtres de paquets actuels se contentent de proposer à un utilisateur la configuration d'un ensemble de compositions protocolaires prédéfinies. La documentation du système de détection d'intrusion Snort [6], nous montre qu'un seul niveau d'encapsulation GRE (*Generic Routing Encapsulation*) est supporté. Une communication du type `Eth/IPv4/GRE/IPv4/GRE/IPv4/TCP/Pile_Applicative` ne peut donc être analysée par Snort[7]. Dès lors, en se contentant, à l'instar de Snort ou de ses semblables [PAX 98] ou des solutions commerciales[8], de proposer la configuration d'un ensemble de compositions de protocoles prédéfinies, les développeurs de logiciels malveillants pourront assez aisément franchir ces barrières. En supposant que la composition IPv4/TCP/HTTP/A/B/C soit la plus profonde des compositions configurables, il leur suffit par exemple de développer un protocole IPv4/TCP/HTTP/A/B/C/M. Une telle composition leur permet alors de réaliser leurs agissements malveillants, tout en étant interopérables avec le pare-feu. Les canaux `C&C`des bots tels que [HOL 08, CHI 07] implémentés au dessus de HTTP ou du protocole pair-à-pair décentralisé Kademlia illustrent ce problème.

---

4. `http://vtun.sourceforge.net/tun/faq.html`, `http://www.openbsd.org/cgi-bin/man.cgi?query=ssh&sektion=1`.
5. `http://www.libssh2.org/wiki/index.php/Main_Page`, `http://curl.haxx.se/libcurl/`.
6. http ://www.snort.org/docs/snort htmanuals/htmanual 2832/node35.html
7. `http://www.snort.org`.
8. Juniper,`http://www.juniper.net/products\_and\_services/intrusion\_prevention\_solutions`. Cisco, `http://www.cisco.com/en/US/netsol/ns785/networking_solutions_package.html`.

Nous avons décidé de diviser notre pare-feu en deux parties, chacune présentant des problématiques particulières :

Inspecteurs protocolaires du milieu : les équivalents des librairies des extrémités. La conception et le développement d'un protocole à une extrémité d'un réseau se résume plus ou moins à calculer le prochain message à envoyer en fonction des messages estampillés reçus. Le développement d'un inspecteur protocolaire du milieu consiste à calculer quels messages transférer ou éliminer en fonction des messages estampillés que l'extrémité « a pu » recevoir. Un pare-feu correct doit donc, comme démontré par [BID 07, ROO] envisager tous les cas de figure : perte, réordonancement de la suite de paquets observée, etc. L'approche que nous avons adoptée pour le développement de l'inspecteur du milieu de TCP a consisté à utiliser l'approche formelle proposée par [BID 07] pour la synthèse de l'automate de milieu de TCP. Contrairement à d'autres pare-feux tels que `netfilter`[9] qui écrivent leur version de l'automate du milieu à la main, la méthode décrite dans [BID 07] permet, après avoir modélisé les extrémités à l'aide d'automates[10], de synthétiser l'automate du milieu qui prend en compte tous les scénarios possibles (perte de paquets, réordonancement). Cette méthode s'applique bien à la synthèse de l'automate régissant les établissements et fermetures de sessions de TCP. Néanmoins pour surveiller la cohérence des numéros de séquences, acquittements et de la taille des fenêtres de contrôle de flux annoncés, l'espace d'états devient vite trop grand. Dès lors nous avons choisi d'utiliser aussi l'approche analytique proposée par [ROO]. Cette solution modélise un segment TCP comme un quadruplet composé de s (son numéro de séquence), ack (son numéro d'acquittement), n (sa longueur), et de win (la fenêtre de contrôle de flux). La validité des segments allant de $E_0 \rightarrow M \rightarrow E_1$ est vérifiée à l'aide de quatre inégalités :

1) $s + n \leq max_{packets \in E_1 \rightarrow M}\{ack + max(win, 1)\}$

l'index du dernier octet du segment envoyé doit être inférieur à l'index du plus grand octet autorisé par le contrôle de flux du récepteur,

2) $s \geq max_{packets \in E_1 \rightarrow M}\{max(win, 1)\}$

l'index du premier octet du segment envoyé doit être plus grand que l'index du plus grand octet autorisé par le contrôle de flux du récepteur,

3) $ack \leq max_{packets \in E_1 \rightarrow M}\{s + n\}$

le segment ne peut acquitter un octet qu'il n'a pas reçu,

4) $ack \geq max_{packets \in E_1 \rightarrow M}\{s + n\} - max\_ack\_window$

le segment ne doit pas inutilement acquitter des données[11].

Composition de ces inspecteurs : exprime les compositions de protocoles utilisées par les communications des logiciels surveillés. Deux problèmes apparaissent ici :

1) Comment composer (représenté par l'opérateur '/' dans notre filtre de paquets), ces inspecteurs et comment déduire un algorithme de filtrage de ces compositions ? Laisser la

9. http ://www.netfilter.org.
10. Machines de Mealy.
11. max_ack_window est une constante choisie pour être supérieure à la valeur maximum de la fenêtre de contrôle de flux. Cette valeur change donc selon que la « window scale option »(`http://tools.ietf.org/html/rfc1323`) est activée ou non.

possibilité à l'utilisateur du filtre de configurer la composition protocolaire de son choix pose des problèmes. Par exemple, le filtre de paquets devrait pouvoir dire que les compositions telles que UDP/HTTP ou FTP/IPv4 sont invalides. D'un autre côté, comme le filtre de paquets peut être amené à surveiller les communications de plusieurs applications en simultané, il doit pouvoir surveiller différentes compositions protocolaires (représenté par l'opérateur '|' dans notre filtre de paquets). Déduire un algorithme de filtrage d'un ensemble de compositions induit deux problématiques fortement corrélées. Dans un premier temps, notre filtre de paquets doit pouvoir factoriser les inspecteurs protocolaires inutilement dupliqués. Enfin, l'algorithme doit pouvoir savoir comment obtenir un « consensus » entre les décisions prises pour les différentes compositions. Comme montré par un algorithme d'optimisation du ré assemblage de flux TCP [VUT 08] qui choisit parfois d'« attendre le segment suivant », l'ensemble des décisions possibles peut être amené à évoluer. Faire un « consensus » cohérent sur un ensemble de décisions amené *à évoluer* est une autre problématique à résoudre. Nous avons exprimé l'interopérabilité des inspecteurs du milieu en formalisant le type des paquets en entrée et en sortie de l'inspecteur protocolaire du milieu. Par exemple, TCP prend en entrée des « segments TCP », et retourne soit une décision, soit un « flux d'octets, ordonné et sans perte ». UDP, lui prend des datagrammes UDP, et retourne soit une décision, soit un « flux d'octets, désordonné et avec perte ». HTTP lui prend en entrée un « flux d'octets, ordonné et sans perte ». Notre pare-feu peut à présent déduire que contrairement à UDP/HTTP, TCP/HTTP est valide. Notre ensemble de compositions protocolaires est dès lors modélisé à l'aide d'un arbre d'inspecteurs protocolaires du milieu. Ayant en entrée un arbre :

```
[IPv4/UDP/DNS | IPv4/TCP/HTTP]
```

, le filtre de paquets construit l'arbre

```
IPv4/[UDP/DNS|TCP/HTTP]
```

et demande récursivement à chacun de ses fils la décision à prendre sur le paquet en question. Pour assurer la validité de la factorisation, à savoir que par exemple les deux arbres précédents (factorisés et pas) sont équivalents, l'algorithme *consensus* doit avoir la propriété suivante. Cet algorithme prend en entrée un sous-ensemble $ds \in P(DS)$ de l'ensemble des décisions DS et retourne une décision dans DS. La contrainte que l'algorithme doit vérifier est la suivante : $\forall X \in P(P(\mathbf{DS})), consensus(\bigcup_{x \in X} \{consensus(x)\}) = consensus(\bigcup_{x \in X} x)$.

2) Surveiller des applications qui à l'instar du canal de données de FTP utilisent des compositions de protocoles temporaires et dynamiquement négociées est problématique. En effet, le seul composant du pare-feu capable d'extraire les paramètres de cette composition est l'inspecteur protocolaire du milieu. Nous devons alors trouver un moyen pour permettre aux inspecteurs du milieu de communiquer ces paramètres à l'algorithme de filtrage global. Pour donner la possibilité à l'utilisateur de configurer une inspection selon ces paramètres négociés dynamiquement, nous lui fournissons un ensemble de variables tel que : $previous\_server, previous\_client, related\_server, related\_client$ que chaque inspecteur protocolaire devra mettre dynamiquement à jour. Imaginons que l'utilisateur configure l'inspection du canal de données d'une session de cette façon :

```
IPv4/TCP/Ftp(data_ch=IPv4/TCP/
TPort(cl_addr=previous_client,sr_addr=related_server,
cl_port=related_client,sr_port=previous_server)/Void)
```

Ici le canal de donnée FTP doit être une session TCP, avec les paramètres suivants : 1) **l'adresse source** : l'adresse du client du canal de contrôle, 2) **l'adresse destination** : l'adresse négociée dans le canal de contrôle à l'aide des commandes PORT ou PASV, 3) **le port source** : le port négocié dans le canal de contrôle et 4) **le port destination** : le port du serveur du canal de contrôle.

Le noeud FTP, après avoir observé la négociation de ces paramètres, va remplacer les valeurs « related » avec celles observées, et envoyer à son noeud père TCP, l'arbre

```
IPv4/TCP/
Tport(cl_addr=previous_client,sr_addr=192.168.1.1,
cl_port=1025,sr_port=previous_server)/Void
```

dont il sera chargé d'affecter les variables « previous » avec leur valeur courante. TCP à son tour communiquera cet arbre à IPv4, qui l'enverra à la racine, et qui rajoutera cet arbre à l'ensemble des arbres temporaires.

Ainsi notre filtre de paquets se présente plus comme un compilateur, qui à partir d'un ensemble de compositions protocolaires, déduit un algorithme de filtrage en fonction de l'ensemble en question. Grâce à cette technique, notre filtre de paquets est donc en mesure de gérer n'importe quel type de composition de protocoles, contrairement aux systèmes de détection/prévention d'intrusions et filtres de paquets actuels.

5. Performances

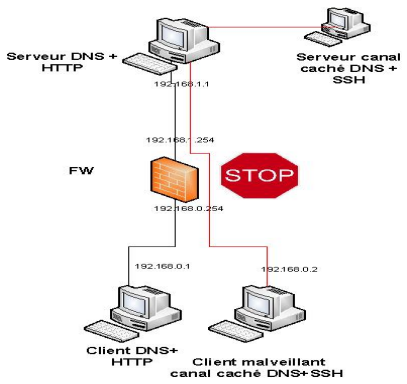

Figure 2. *Canal caché DNS*

Nous avons développé un prototype de ce filtre de paquets en utilisant la libraire *libnetfilter_queue*[12] proposée par le système GNU/Linux qui permet d'implémenter un pare-feu dans l'espace utilisateur. L'environnement dans lequel le filtre de paquets est ammené à être utilisé est hostile. En se référant à l'historique des vulnérabilités de logiciels d'inspection protocolaire tels que Wireshark ou Snort[13], les méthodes de développements standards, en particulier le choix des langages tels que C/C++ est à notre avis discutable. Pour cela nous avons choisi d'utiliser le langage OCaml car il est fortement typé, que les performances des logiciels compilés en code natif sont semblables à celles du C++, et qu'un langage permettant d'obtenir du code de déchiffrement de paquets en OCaml a été développé [MAD 07].

12. http://www.netfilter.org/.
13. http://www.kb.cert.org/vuls/.

Pour illustrer certaines fonctionnalités de notre filtre de paquets, nous avons choisi d'étudier le filtrage de canaux cachés dans le protocole DNS [NUS 08]. Ces canaux cachés exploitent la liberté sémantique offertes par les requêtes TXT de ce protocole[14] pour encoder des paquets TCP ou IP dans ce format, et ainsi encapsuler des protocoles de communications au dessus de DNS. Il est ainsi possible d'utiliser ces canaux pour obtenir gratuitement un accès Internet sur des points d'accès wifi payants. En effet certains points d'accès n'inspectant pas les requêtes DNS entrantes, un client malveillant peut à l'aide de ces canaux cachés surfer gratuitement sur le web. D'un autre côté, un client légitime voulant seulement consulter des sites web peut très bien se contenter des requêtes de type A/AAAA (résolutions de noms). Ainsi une inspection du protocole DNS qui n'autorise que les requêtes de type A/AAAA, et leurs réponses, permet d'empêcher l'établissement de ces canaux cachés. Nous avons dès lors développé l'expérimentation décrite sur la Figure 2 sur un réseau Gigabit Ethernet.

Le serveur HTTP héberge des fichiers allant de 10ko jusqu'à 500ko dont la taille est générée selon une distribution pareto de paramètre 1.2 [CRO 97]. Le client légitime télécharge, à l'aide de requêtes DNS et HTTP, N_SIM fichiers toutes les 10ms et calcule la moyenne du taux $taille\_fichier/temps\_dl$[15]. Une erreur (ERR) est considérée ssi $temps\_dl > 10s$[16]. Le serveur complice du client malveillant héberge à l'aide du protocole SSH encapsulé dans le canal caché des fichiers d'une taille allant de 10ko à 189ko selon la même distribution pareto. Le client malveillant télécharge à l'aide du canal caché N_SIM fichiers toutes les 10ms, et calcule la moyenne (MOY) du taux $taille\_fichier/temps\_dl$. Une erreur est ici aussi considérée si $temps\_dl > 10s$.

Nous avons effectué cette expérimentation sans filtre de paquets, puis avec le filtre de paquets configuré des manières suivantes :

```
1.IPv4(check_checksum=no)/
  [TCP(tcp_only=yes) |
  UDP(check_checksum=no)/DNS]);;
```

```
2.IPv4(check_checksum=no)/
  [TCP/TPort(sr_port=80)/Void |
  UDP/UPort(sr_port=53)/DNS]);;
```

La configuration 1 laisse passer tous les segments TCP et inspecte le protocole DNS sans vérifier le checksum UDP. La configuration 2 ré-assemble les segments TCP, vérifie que le port du serveur est 80, inspecte le protocole DNS en vérifiant que le port du serveur est 53 (checksum vérifié dans TCP et UDP).

Pour isoler les effets de bords liés aux changements de contextes espace noyau ↔ espace utilisateur, nous avons dans un premier temps évalué les performances de notre filtre de paquets hors-ligne. Pour cela nous avons analysé la capture des traces réseaux de la première expérimentation. Premièrement nous avons constaté que la seconde configuration du filtre, celle avec ré assemblage TCP, consommait 10 fois plus de mémoire que la seconde. Nous avons aussi calculé la différence du temps inter-arrivée des paquets avec le temps de traitement. Une différence négative implique une croissance du tampon d'entrée du filtre et donc une perte de paquet potentielle, qui implique une décroissance de la bande passante de l'application. Les résultats affichés sur la Figure 3 montrent à quel point la première configuration est plus rapide. Nous avons fait quatre expérimentations pour

14. `http://www.isi.edu/in-notes/rfc1464.txt`.
15. Temps de téléchargement.
16. 10s est une estimation subjective du temps d'attente d'affichage toléré d'une page web.

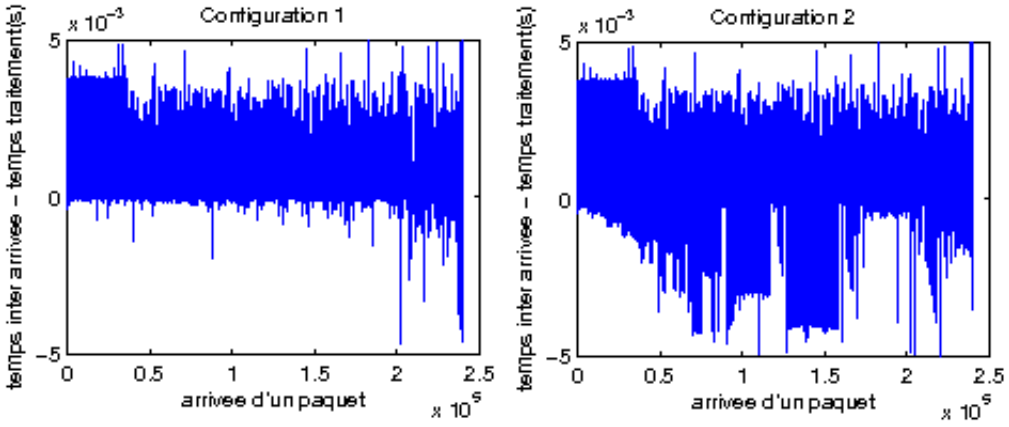

Figure 3. *Performances du filtre de paquets hors ligne selon les configurations 1 et 2*

N_SIM	MOY(Sans filtre)	MOY(Réferentiel)	MOY(Conf 1)	MOY(Conf 2)
1	1.112 Mo/s	0.870 Mo/s	0.753 Mo/s	0.692 Mo/s
5	1.924 Mo/s	0.688 Mo/s	0.582 Mo/s	0.507Mo/s

Figure 4. *Performances du client légitime sans filtre, avec le filtre référentiel, sous la configuration 1, puis avec la configuration 2.*

évaluer les performances du filtre en ligne : sans filtre, avec un filtre dans l'espace utilisateur développé en C qui accepte tous les paquets sans les regarder (nous le nommons filtre référentiel), avec notre filtre avec la configuration 1, puis avec la configuration 2. Nous avons premièrement constaté que les fonctionnalités du filtre sont validées. En effet, alors que le client malveillant atteint une moyenne de 0.003 Mo/s sans filtre, puis 0.001 Mo/s avec le filtre référentiel, il obtient un taux d'erreur de 100% avec notre filtre avec les deux configurations (ce qui montre que le client malveillant est bien bloqué par notre filtre). Dans un second temps, comme montré par la Figure 4, nous voyons ici aussi que le client légitime obtient un meilleur débit avec la configuration 1. Pour finir, notre filtre induit une perte de débit d'environ 14% par rapport au référentiel avec la configuration 1 et de 23% avec la configuration 2.

6. Conclusion

Nous avons proposé dans ce papier, une plateforme permettant l'exécution contrôlée de logiciels malveillants. L'élément central de cette plateforme est un pare-feu qui permet la composition et la parallélisation d'un ensemble prédéfini d'inspecteurs protocolaires du milieu. Ce pare-feu permet

de configurer la surveillance des communications réseaux d'une façon souple et donc de façon plus adaptée à chaque cas de figure. Le développement d'une nouvelle inspection bénéficie aussi de cette souplesse, ce dernier se résumant au développement d'un inspecteur du milieu qui jusqu'à présent faisait défaut. Nous pensons à court-terme automatiser l'exécution et l'observation de ces logiciels malveillants à grande échelle. Selon les résultats observés, nous formaliserons et automatiserons l'envoi des informations obtenues à un superviseur dans un second temps.

7. Bibliographie

[BAE 06] BAECHER P., KOETTER M., DORNSEIF M., FREILING F., « The nepenthes platform : An efficient approach to collect malware », *RAID06*, Springer, 2006, p. 165–184.

[BAR 07] BARFORD P., BLODGETT M., « Toward botnet mesocosms », *HotBots'07*, Berkeley, CA, USA, 2007, USENIX Association, p. 6–6.

[BID 07] BIDDER-SENN D., BASIN D., CARONNI G., « Midpoints Versus Endpoints : From Protocols to Firewalls », *ACNS '07 : Proceedings of the 5th international conference on Applied Cryptography and Network Security*, Berlin, Heidelberg, 2007, Springer-Verlag, p. 46–64.

[CHI 07] CHIANG K., LLOYD L., « A case study of the rustock rootkit and spam bot », *HotBots'07*, Berkeley, CA, USA, 2007, USENIX Association, p. 10–10.

[CRO 97] CROVELLA M. E., BESTAVROS A., « Self-similarity in World Wide Web traffic : evidence and possible causes », *IEEE/ACM Trans. Netw.*, vol. 5, n° 6, 1997, p. 835–846, IEEE Press.

[GOT 04] GOTH G., « When Is a "Little in the Middle" OK ? The Internet's End-to-End Principle Faces More Debate », *IEEE Distributed Systems Online*, vol. 5, n° 3, 2004, page 2, IEEE Educational Activities Department.

[HOL 08] HOLZ T., STEINER M., DAHL F., BIERSACK E., FREILING F., « Measurements and mitigation of peer-to-peer-based botnets : a case study on storm worm », *LEET'08 : Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, Berkeley, CA, USA, 2008, USENIX Association, p. 1–9.

[JAI 02] JAISWAL S., IANNACCONE G., DIOT C., KUROSE J., TOWSLEY D., « Measurement and classification of out-of-sequence packets in a tier-1 IP backbone », *IMW '02 : Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, New York, NY, USA, 2002, ACM, p. 113–114.

[MAD 07] MADHAVAPEDDY A., HO A., DEEGAN T., SCOTT D., SOHAN R., « Melange : creating a functional internet », *EuroSys '07 : Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, New York, NY, USA, 2007, ACM, p. 101–114.

[NUS 08] NUSSBAUM L. R. O., « Prototype de Canal Caché dans le DNS », *CFIP'08 : Colloque Francophone sur l'Ingénierie des Protocoles, Les Arcs, France*, 03 2008.

[PAX 98] PAXSON V., « Bro : a system for detecting network intruders in real-time », *SSYM'98 : Proceedings of the 7th conference on USENIX Security Symposium*, Berkeley, CA, USA, 1998, USENIX Association, p. 3–3.

[ROO] ROOIJ G. V., « Real stateful TCP packet filtering in IP-filter », *Invited talk at the 10th USENIX Security Symposium*, August.

[VUT 08] VUTUKURU M., BALAKRISHNAN H., PAXSON V., « Efficient and Robust TCP Stream Normalization », *SP '08 : Proceedings of the 2008 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2008, IEEE Computer Society, p. 96–110.